###### *CSE 473 – Introduction to Computer Networks*

Lab 3 Report – 110 Points

##### *Haiyu Wang, Haodong Huang*

***Part A (30 points).*** Place a copy of the source code of the functions in *DhtServer* to which you added any code or documentation; remember to include the documentation you added for the functions that required it. Highlight your changes by making them **bold**. Remember to also place a complete copy in the repository before you make your final commit. *Your* committed version should have no extraneous *print* statements.

public static void leave() {

**Packet leave\_p = new Packet();**

**leave\_p.type = "leave";**

**leave\_p.tag = sendTag;**

**leave\_p.senderInfo = myInfo;**

**leave\_p.send(sock, succInfo.left, debug);**

**while (!stopFlag);**

**leave\_p.clear();**

**leave\_p.type = "update";**

**leave\_p.succInfo = succInfo;**

**int newleft = predInfo.right;**

**Pair<Integer, Integer> newRange = new Pair<Integer, Integer>(newleft, hashRange.right);**

**leave\_p.hashRange = newRange;**

**leave\_p.tag = sendTag;**

**leave\_p.send(sock, predInfo.left, debug);**

**leave\_p.clear();**

**leave\_p.type = "update";**

**leave\_p.predInfo = predInfo;**

**leave\_p.tag = sendTag;**

**leave\_p.send(sock, succInfo.left, debug);**

**leave\_p.clear();**

**leave\_p.type = "transfer";**

**leave\_p.tag = sendTag;**

**for (Map.Entry<String, String> entry: map.entrySet()) {**

**leave\_p.key = entry.getKey();**

**leave\_p.val = entry.getValue();**

**leave\_p.send(sock, predInfo.left, debug);**

**}**

**map.clear();**

**rteTbl.clear();**

**if(cacheOn) cache.clear();**

}

/\*\* Join an existing DHT.

\* @param predAdr is the socket address of a server in the DHT,

\*

**\* The current server acts as an incoming server, it**

\* **sends a join packet to its predecessor**

\*/

public static void join(InetSocketAddress predAdr) {

**Packet join\_p = new Packet();**

**join\_p.type = "join";**

**join\_p.tag = sendTag;**

**join\_p.send(sock, predAdr, debug);**

}

/\*\* Handle a join packet from a prospective DHT node.

\* @param p is the received join packet

\* @param succAdr is the socket address of the host that

\* sent the join packet (the new successor)

\*

**\* calculate the hashRange for the incoming server**

**\* send an update packet to the successor**

**\* send a success packet to the coming server and add to the route table**

**\* transfer the corresponding (key, value) pairs to the incoming server**

\*/

public static void handleJoin(Packet p, InetSocketAddress succAdr) {

**int len = hashRange.right - hashRange.left;**

**int newleft = hashRange.left + (len / 2 + 1);**

**int tag = p.tag;**

**Pair<Integer, Integer> newRange = new Pair<Integer, Integer>(newleft, hashRange.right);**

**Pair<InetSocketAddress, Integer> newInfo = new Pair<InetSocketAddress, Integer>(succAdr, newleft);**

**p.clear();**

**p.type = "update";**

**p.senderInfo = myInfo;**

**p.predInfo = newInfo;**

**p.tag = sendTag;**

**p.send(sock, succInfo.left, debug);**

**p.clear();**

**p.type = "success";**

**p.succInfo = succInfo;**

**p.predInfo = myInfo;**

**p.hashRange = newRange;**

**p.tag = tag;**

**p.send(sock, succAdr, debug);**

**succInfo = newInfo;**

**hashRange.right = newleft - 1;**

**addRoute(succInfo);**

**p.clear();**

**p.type = "transfer";**

**p.tag = sendTag;**

**List<String> toRemove = new ArrayList<String>();**

**for (Map.Entry<String, String> entry: map.entrySet()) {**

**int hashval = hashit(entry.getKey());**

**if(hashval >= newRange.left && hashval <= newRange.right) {**

**p.key = entry.getKey();**

**p.val = entry.getValue();**

**p.send(sock, succAdr, debug);**

**toRemove.add(p.key);**

**}**

**}**

**for(String key : toRemove) map.remove(key);**

}

/\*\* Handle a get packet.

\* @param p is a get packet

\* @param senderAdr is the the socket address of the sender

\*

**\* check the cacheOn and cache, if it has the key, send back directly**

\* **if the server is responsible for the key, find it and send**

**\* success/no match packet to the relay server**

**\* else forward to the server which has the closest range to the key.**

\*/

public static void handleGet(Packet p, InetSocketAddress senderAdr) {

InetSocketAddress replyAdr;

int hash = hashit(p.key);

int left = hashRange.left.intValue();

int right = hashRange.right.intValue();

**if(cacheOn && cache.containsKey(p.key)) {**

**if (p.relayAdr != null) {**

**replyAdr = p.relayAdr;**

**p.senderInfo = myInfo;**

**} else {**

**replyAdr = senderAdr;**

**}**

**p.type = "success"; p.val = cache.get(p.key);**

**p.send(sock,replyAdr,debug);**

**}**

else {

if (left <= hash && hash <= right) {

// respond to request using map

if (p.relayAdr != null) {

replyAdr = p.relayAdr;

p.senderInfo = myInfo;

} else {

replyAdr = senderAdr;

}

if (map.containsKey(p.key)) {

p.type = "success"; p.val = map.get(p.key);

} else {

p.type = "no match";

}

p.send(sock,replyAdr,debug);

} else {

// forward around DHT

if (p.relayAdr == null) {

p.relayAdr = myAdr; p.clientAdr = senderAdr;

}

forward(p,hash);

}

}

}

/\*\* Handle a put packet.

\* @param p is a put packet

\* @param senderAdr is the the socket address of the sender

\*

**\* check the cacheOn and cache, if containing the key, remove it from the cache**

**\* if the server is responsible for the key, put it into the map**

**\* else forward to another server.**

\*/

public static void handlePut(Packet p, InetSocketAddress senderAdr) {

**if(cacheOn && cache.containsKey(p.key)) cache.remove(p.key);**

**InetSocketAddress replyAdr;**

**int hash = hashit(p.key);**

**int left = hashRange.left.intValue();**

**int right = hashRange.right.intValue();**

**if (left <= hash && hash <= right) {**

**// respond to request using map**

**if (p.relayAdr != null) {**

**replyAdr = p.relayAdr;**

**p.senderInfo = myInfo;**

**} else {**

**replyAdr = senderAdr;**

**}**

**p.type = "success";**

**map.put(p.key,p.val);**

**p.send(sock, replyAdr, debug);**

**} else {**

**// forward around DHT**

**if (p.relayAdr == null) {**

**p.relayAdr = myAdr; p.clientAdr = senderAdr;**

**}**

**forward(p,hash);**

**}**

}

/\*\* Handle a transfer packet.

\* @param p is a transfer packet

\* @param senderAdr is the the address (ip:port) of the sender

\*

**\* when receiving a tranfer packet, put it into the map.**

\*/

public static void handleXfer(Packet p, InetSocketAddress senderAdr) {

**map.put(p.key,p.val);**

}

/\*\* Handle a reply packet.

\* @param p is a reply packet, more specifically, a packet of type

\* "success", "failure" or "no match"

\* @param senderAdr is the the address (ip:port) of the sender

\*

**\* if type is success, there are 2 cases.**

**\* one is that the current server is joining DHT, then update corresponding info**

**\* the other one is success packet for get/put request, then send back to client**

**\* if the type is no match or failure, send back to client.**

\*/

public static void handleReply(Packet p, InetSocketAddress senderAdr) {

**if(p.type.equals("success")) {**

**if(p.hashRange != null) {**

**succInfo = p.succInfo;**

**predInfo = p.predInfo;**

**hashRange = p.hashRange;**

**myInfo.right = hashRange.left;**

**addRoute(succInfo);**

**}**

**else if (p.key != null) {**

**InetSocketAddress cAdr = p.clientAdr;**

**p.senderInfo = null;**

**p.clientAdr = null;**

**p.relayAdr = null;**

**if(cacheOn) cache.put(p.key, p.val);**

**p.send(sock,cAdr,debug);**

**}**

**}**

**else if (p.type.equals("no match") || p.type.equals("failure")) {**

**InetSocketAddress cAdr = p.clientAdr;**

**p.senderInfo = null;**

**p.clientAdr = null;**

**p.relayAdr = null;**

**p.send(sock,cAdr,debug);**

**}**

}

public static void addRoute(Pair<InetSocketAddress,Integer> newRoute){

**// entry equals to myself**

**if(myInfo.equals(newRoute)) return;**

**// entry already exists**

**for(int i = 0; i < rteTbl.size(); i++) {**

**if(rteTbl.get(i).equals(newRoute)) {**

**return;**

**}**

**}**

**boolean can\_add = true;**

**if(rteTbl.size() == numRoutes) {**

**// table is full**

**can\_add = false;**

**for(int i = 0; i < rteTbl.size(); i++) {**

**if(!rteTbl.get(i).equals(succInfo)) {**

**rteTbl.remove(i);**

**can\_add = true;**

**break;**

**}**

**}**

**}**

**if(can\_add) {**

**// table content does change**

**rteTbl.add(newRoute);**

**if(debug) System.out.println("rteTbl=" + rteTbl);**

**}**

}

public static void removeRoute(Pair<InetSocketAddress,Integer> rmRoute){

**for(int i = 0; i < rteTbl.size(); i++) {**

**if(rteTbl.get(i).equals(rmRoute)) {**

**rteTbl.remove(i);**

**if(debug) System.out.println("rteTbl=" + rteTbl);**

**return;**

**}**

**}**

}

public static void forward(Packet p, int hash) {

**// pair<dest\_server\_addr, current\_min\_distance>**

**Pair<InetSocketAddress, Integer> target = new Pair<InetSocketAddress, Integer>(null, Integer.MAX\_VALUE);**

**for(Pair<InetSocketAddress, Integer> route : rteTbl) {**

**int dist = hash - route.right;**

**if(dist < 0) dist += Integer.MAX\_VALUE;**

**if(dist < target.right) {**

**target.left = route.left;**

**target.right = dist;**

**}**

**}**

**p.send(sock, target.left, debug);**

}

***Part B (10 points).*** Place a copy of the source code of the functions in *Packet* where you added code and comments; highlight your changes by making them **bold*.*** Include a complete copy in the repository before you make your final commit. *Your* committed version should have no extraneous *print* statements.

public boolean unpack(byte[] buf, int bufLen) {

// convert buf to a string

String s;

try { s = new String(buf,0,bufLen,"US-ASCII");

} catch(Exception e) { return false; }

// divide into lines and check the first line

String[] lines = s.split("\n");

if (!lines[0].equals("CSE473 DHTPv0.1")) return false;

//process remaining lines

for (int i = 1; i < lines.length; i++) {

String[] chunks = lines[i].split(":",2);

if (chunks.length != 2) return false;

// process the line

String left = chunks[0];

String right = chunks[1];

if (left.equals("type")) {

type = right;

} else if (left.equals("ttl")) {

ttl = Integer.parseInt(right);

} else if (left.equals("clientAdr")) {

chunks = right.split(":");

if (chunks.length != 2) return false;

clientAdr = new InetSocketAddress(chunks[0],

Integer.parseInt(chunks[1]));

} else if (left.equals("succInfo")) {

chunks = right.split(":");

if (chunks.length != 3) return false;

String ip = chunks[0];

int port = Integer.parseInt(chunks[1]);

int hash = Integer.parseInt(chunks[2]);

succInfo = new

Pair<InetSocketAddress,Integer>(

new InetSocketAddress(ip,port),hash);

} **else if (left.equals("key")) {**

**key = right;**

**} else if (left.equals("tag")) {**

**tag = Integer.parseInt(right);**

**} else if (left.equals("value")) {**

**val = right;**

**} else if (left.equals("predInfo")) {**

**chunks = right.split(":");**

**if (chunks.length != 3) return false;**

**String ip = chunks[0];**

**int port = Integer.parseInt(chunks[1]);**

**int hash = Integer.parseInt(chunks[2]);**

**predInfo = new**

**Pair<InetSocketAddress,Integer>(**

**new InetSocketAddress(ip,port),hash);**

**} else if (left.equals("hashRange")) {**

**String[] ranges = right.split(":");**

**if(ranges.length != 2) return false;**

**hashRange = new Pair<Integer, Integer>(Integer.parseInt(ranges[0]), Integer.parseInt(ranges[1]));**

**} else if (left.equals("senderInfo")){**

**chunks = right.split(":");**

**if (chunks.length != 3) return false;**

**String ip = chunks[0];**

**int port = Integer.parseInt(chunks[1]);**

**int hash = Integer.parseInt(chunks[2]);**

**senderInfo = new**

**Pair<InetSocketAddress,Integer>(**

**new InetSocketAddress(ip,port),hash);**

**} else if (left.equals("relayAdr")){**

**chunks = right.split(":");**

**if (chunks.length != 2) return false;**

**relayAdr = new InetSocketAddress(chunks[0],**

**Integer.parseInt(chunks[1]));**

**} else if (left.equals("reason")){**

**reason = right;**

**}**

else {

// ignore lines that don't match defined field

}

}

return true;

}

public boolean check() {

reason = null;

if (type == null) {

reason = "every packet must include a type";

return false;

} else if ((type.equals("get") || type.equals("put")) &&

(key == null || tag == -1)) {

reason = "gets and puts require key and tag";

return false;

}

**else if(ttl < 0) {**

**reason = "ttl < 0";**

**return false;**

**}**

return true;

}

public String toString() {

StringBuffer s = new StringBuffer("CSE473 DHTPv0.1\n");

if (type != null) {

s.append("type:"); s.append(type); s.append("\n");

}

if (key != null) {

s.append("key:"); s.append(key); s.append("\n");

}

if (relayAdr != null) {

s.append("relayAdr:");

s.append(relayAdr.getAddress().getHostAddress());

s.append(":"); s.append(relayAdr.getPort());

s.append("\n");

}

if (clientAdr != null) {

s.append("clientAdr:");

s.append(clientAdr.getAddress().getHostAddress());

s.append(":"); s.append(clientAdr.getPort());

s.append("\n");

}

if (hashRange != null) {

s.append("hashRange:"); s.append(hashRange.left);

s.append(":"); s.append(hashRange.right);

s.append("\n");

}

if (senderInfo != null) {

s.append("senderInfo:");

s.append(senderInfo.left.getAddress().getHostAddress());

s.append(":"); s.append(senderInfo.left.getPort());

s.append(":"); s.append(senderInfo.right);

s.append("\n");

}

//

// add code for missing cases

//

**s.append("tag:");**

**s.append(tag);**

**s.append("\n");**

**if(succInfo != null) {**

**s.append("succInfo:");**

**s.append(succInfo.left.getAddress().getHostAddress());**

**s.append(":"); s.append(succInfo.left.getPort());**

**s.append(":"); s.append(succInfo.right);**

**s.append("\n");**

**}**

**if(predInfo != null) {**

**s.append("predInfo:");**

**s.append(predInfo.left.getAddress().getHostAddress());**

**s.append(":"); s.append(predInfo.left.getPort());**

**s.append(":"); s.append(predInfo.right);**

**s.append("\n");**

**}**

**if (reason != null) {**

**s.append("reason:");**

**s.append(reason);**

**s.append("\n");**

**}**

**if (val != null){**

**s.append("value:");**

**s.append(val);**

**s.append("\n");**

**}**

if (ttl != -1) {

s.append("ttl:"); s.append(ttl); s.append("\n");

}

return s.toString();

}

***Part C (10 points).*** Place a copy of your source code for *DhtClient* here.

import java.io.\*;

import java.net.\*;

import java.util.\*;

/\*\*

\* DhtClient

\* usage: DhtClient clientIP serverCfgFile [operation] [key] [value]

\*

\* Parse server socket address from server's config file

\* Send get/put requests to server

\* Wait for reply packet

\*\*/

public class DhtClient {

private static boolean debug = true;

public static void main(String args[]) throws Exception {

// get server address and port

InetAddress clientAdr = InetAddress.getByName(args[0]);

String cfgFile = args[1];

BufferedReader cfg=

new BufferedReader(

new InputStreamReader(

new FileInputStream(cfgFile),

"US-ASCII"));

String s = cfg.readLine();

String[] chunks = s.split(" ");

cfg.close();

InetAddress serverAdr = InetAddress.getByName(chunks[0]);

int port = Integer.parseInt(chunks[1]);

InetSocketAddress saddr = new InetSocketAddress(serverAdr,port);

// create socket

DatagramSocket sock = new DatagramSocket(0,clientAdr);

/\*\*

\* if operation is get, get the key

\* if operation is put, get the key and value.

\*/

Packet outpkt = new Packet();

outpkt.type = args[2];

if (outpkt.type.equals("get") && args.length > 3) outpkt.key = args[3];

else if (outpkt.type.equals("put")) {

if(args.length > 3) outpkt.key = args[3];

if(args.length > 4) outpkt.val = args[4];

}

outpkt.tag = Math.abs(new Random().nextInt());

outpkt.send(sock,saddr,debug);

outpkt.clear();

outpkt.receive(sock,debug);

sock.close();

}

}

***Part D (10 points).*** Use the provided *script0* to test your client and server on a single computer. Of course, you will first need to compile your java code, *e.g.,*

javac \*.java

in the lab3 directory where your java files are stored. We are using a signal handling API so servers can announce they are leaving before they exit. This will incur some compilation warnings, but you do not need to worry about the ones mentioning “*Signal”* or *“SignalHandler”*. When you test using *script0*, note that this script uses just a single server, so it does not test many of the features of your DHT, but it will allow you to check a significant fraction of the code. You may do this testing on any Unix (including MacOS) or Linux computer (shell.cec.wustl.edu or onl.wustl.edu) or the virtual Linux Lab (linuxlab.seas.wustl.edu). Go to the *test0* directory and read *script0* to make sure you understand what it does, then type

./script0 > out

to run it. Check the output file carefully. When you are satisfied that things are working correctly, paste a copy of the output below. **Commit the output file and the log file in your *test0* directory to your repository.**

put foo bar

/127.0.0.1:44079 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:put

key:foo

tag:475593524

value:bar

ttl:100

/127.0.0.1:44079 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:success

key:foo

tag:475593524

value:bar

ttl:98

put who hah

/127.0.0.1:44358 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:put

key:who

tag:1277247590

value:hah

ttl:100

/127.0.0.1:44358 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:success

key:who

tag:1277247590

value:hah

ttl:98

get foo

/127.0.0.1:56693 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:get

key:foo

tag:269049783

ttl:100

/127.0.0.1:56693 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:success

key:foo

tag:269049783

value:bar

ttl:98

get who

/127.0.0.1:35476 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:get

key:who

tag:1161080623

ttl:100

/127.0.0.1:35476 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:success

key:who

tag:1161080623

value:hah

ttl:98

get goodbye

/127.0.0.1:40500 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:get

key:goodbye

tag:1862999169

ttl:100

/127.0.0.1:40500 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:no match

key:goodbye

tag:1862999169

ttl:98

get

/127.0.0.1:54624 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:get

tag:883625009

ttl:100

/127.0.0.1:54624 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:failure

tag:883625009

reason:gets and puts require key and tag

ttl:98

get bar

/127.0.0.1:40732 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:get

key:bar

tag:1472512661

ttl:100

/127.0.0.1:40732 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:no match

key:bar

tag:1472512661

ttl:98

put foo toast is tasty

/127.0.0.1:53144 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:put

key:foo

tag:1338361457

value:toast is tasty

ttl:100

/127.0.0.1:53144 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:success

key:foo

tag:1338361457

value:toast is tasty

ttl:98

get foo

/127.0.0.1:38841 sending packet to /127.0.0.1:51420

CSE473 DHTPv0.1

type:get

key:foo

tag:642356759

ttl:100

/127.0.0.1:38841 received packet from /127.0.0.1:51420

CSE473 DHTPv0.1

type:success

key:foo

tag:642356759

value:toast is tasty

ttl:98

***Part E (20 points).*** In this part, you are to use the provided *script1* (in the *test1* directory) to test your DHT on a single computer. This script uses four servers, so it will exercise the routing features of your DHT. In the questions that follow, we will refer to the servers by number. The first server that is started is number 0. Its successor in the DHT (after all servers have been started) is number 1. The next is number 2, and so forth. Read the *script1* file and make sure you understand what it does. Notice that each server produces a log file labeled with its number. Now, type

./script1 1 > out1

to run it. Note that this version limits the servers to a single route, so there are no shortcut routes at this point. When you are satisfied that your results are correct, paste the initial and last portion of the *out1* file below. Specifically, include everything up through the first “get who” sequence (including the reply for “get who”) and last four operations. **Commit the output and log files to your repository.**

put foo bar

/127.0.0.1:34593 sending packet to /127.0.0.1:33134

CSE473 DHTPv0.1

type:put

key:foo

tag:418116499

value:bar

ttl:100

/127.0.0.1:34593 received packet from /127.0.0.1:33134

CSE473 DHTPv0.1

type:success

key:foo

tag:418116499

value:bar

ttl:98

put who hah

/127.0.0.1:56246 sending packet to /127.0.0.1:34417

CSE473 DHTPv0.1

type:put

key:who

tag:120991850

value:hah

ttl:100

/127.0.0.1:56246 received packet from /127.0.0.1:34417

CSE473 DHTPv0.1

type:success

key:who

tag:120991850

value:hah

ttl:94

put junk mail

/127.0.0.1:54009 sending packet to /127.0.0.1:51511

CSE473 DHTPv0.1

type:put

key:junk

tag:1380305224

value:mail

ttl:100

/127.0.0.1:54009 received packet from /127.0.0.1:51511

CSE473 DHTPv0.1

type:success

key:junk

tag:1380305224

value:mail

ttl:95

put blue moose

/127.0.0.1:53837 sending packet to /127.0.0.1:35825

CSE473 DHTPv0.1

type:put

key:blue

tag:1920278419

value:moose

ttl:100

/127.0.0.1:53837 received packet from /127.0.0.1:35825

CSE473 DHTPv0.1

type:success

key:blue

tag:1920278419

value:moose

ttl:98

get foo

/127.0.0.1:46234 sending packet to /127.0.0.1:51511

CSE473 DHTPv0.1

type:get

key:foo

tag:1402859749

ttl:100

/127.0.0.1:46234 received packet from /127.0.0.1:51511

CSE473 DHTPv0.1

type:success

key:foo

tag:1402859749

value:bar

ttl:96

get who

/127.0.0.1:51183 sending packet to /127.0.0.1:35825

CSE473 DHTPv0.1

type:get

key:who

tag:1342256528

ttl:100

/127.0.0.1:51183 received packet from /127.0.0.1:35825

CSE473 DHTPv0.1

type:success

key:who

tag:1342256528

value:hah

ttl:95

… …

get blue

/127.0.0.1:58576 sending packet to /127.0.0.1:33134

CSE473 DHTPv0.1

type:get

key:blue

tag:840874796

ttl:100

/127.0.0.1:58576 received packet from /127.0.0.1:33134

CSE473 DHTPv0.1

type:success

key:blue

tag:840874796

value:moose

ttl:95

get blue

/127.0.0.1:52311 sending packet to /127.0.0.1:33134

CSE473 DHTPv0.1

type:get

key:blue

tag:1066675132

ttl:100

/127.0.0.1:52311 received packet from /127.0.0.1:33134

CSE473 DHTPv0.1

type:success

key:blue

tag:1066675132

value:moose

ttl:96

get foo

/127.0.0.1:45577 sending packet to /127.0.0.1:51511

CSE473 DHTPv0.1

type:get

key:foo

tag:1414243973

ttl:100

/127.0.0.1:45577 received packet from /127.0.0.1:51511

CSE473 DHTPv0.1

type:success

key:foo

tag:1414243973

value:toast is tasty

ttl:96

get junk

/127.0.0.1:55441 sending packet to /127.0.0.1:34417

CSE473 DHTPv0.1

type:get

key:junk

tag:1468601509

ttl:100

/127.0.0.1:55441 received packet from /127.0.0.1:34417

CSE473 DHTPv0.1

type:success

key:junk

tag:1468601509

value:mail

ttl:98

By examining the *out1* file, determine the port number used by the server that holds the (*key*,*value*) pair (*blue*, *moose*). What’s the ttl of the packet returned to client?

*The port number used by the server which holds the pair (blue,moose) is 35825.*

*The ttl of the packet is 99.*

Note the last eight *get* operations in the *out1* file before server 2 exits the DHT. Based on the *ttls* of the reply packets, determine each server’s successor. For this question, identify the servers by their port numbers, and also provide the *ttls.*

*Server0’s port number is 33134, successor is Server1.*

*Server1’s port number is 34417, successor is Server2*

*Server2’s port number is 35825, successor is Server3*

*Server3’s port number is 51511, successor is Server0*

*echo get blue*

*java DhtClient localhost cfg3 get blue ttl = 94*

*echo get blue*

*java DhtClient localhost cfg1 get blue ttl = 96*

*echo get blue*

*java DhtClient localhost cfg2 get blue ttl = 98*

*echo get blue*

*java DhtClient localhost cfg0 get blue ttl = 95*

*echo get blue*

*java DhtClient localhost cfg3 get blue ttl = 94*

*echo get blue*

*java DhtClient localhost cfg1 get blue ttl = 96*

*echo get blue*

*java DhtClient localhost cfg2 get blue ttl = 98*

*echo get blue*

*java DhtClient localhost cfg0 get blue ttl = 95*

For the last two “get blue” operations, they are requesting the same server. Why do they get different *ttls*?

*Because for the first “get blue” request, DHT contains Server2, when the Server0 was requested by client for “get blue”, the request was forwarded like Server0 → Server1 → Server2 → Server0 → Client, so the ttl is 95.*

*However, in the second “get blue” request, Server2 had left DHT and its (key, value) pairs had been transferred to Server1. Thus, the request was forwarded like Server0 → Server1 → Server0 → Client, so the ttl is different.*

Paste the initial portion of the *log1\_2* file below (everything up through the first “*get blue*” operation and response).

/127.0.0.1:35825 sending packet to /127.0.0.1:33134

CSE473 DHTPv0.1

type:join

tag:2042409573

ttl:100

/127.0.0.1:35825 received packet from /127.0.0.1:33134

CSE473 DHTPv0.1

type:success

hashRange:1073741824:2147483647

tag:2042409573

succInfo:127.0.0.1:33134:0

predInfo:127.0.0.1:33134:0

ttl:99

rteTbl=[(/127.0.0.1:33134,0)]

/127.0.0.1:35825 received packet from /127.0.0.1:33134

CSE473 DHTPv0.1

type:update

senderInfo:127.0.0.1:33134:0

tag:29098953

predInfo:127.0.0.1:34417:536870912

ttl:99

/127.0.0.1:35825 received packet from /127.0.0.1:51511

CSE473 DHTPv0.1

type:join

tag:311517363

ttl:99

/127.0.0.1:35825 sending packet to /127.0.0.1:33134

CSE473 DHTPv0.1

type:update

senderInfo:127.0.0.1:35825:1073741824

tag:2042409573

predInfo:127.0.0.1:51511:1610612736

ttl:100

/127.0.0.1:35825 sending packet to /127.0.0.1:51511

CSE473 DHTPv0.1

type:success

hashRange:1610612736:2147483647

tag:311517363

succInfo:127.0.0.1:33134:0

predInfo:127.0.0.1:35825:1073741824

ttl:100

rteTbl=[(/127.0.0.1:51511,1610612736)]

/127.0.0.1:35825 received packet from /127.0.0.1:34417

CSE473 DHTPv0.1

type:put

key:who

relayAdr:127.0.0.1:34417

clientAdr:127.0.0.1:56246

tag:120991850

value:hah

ttl:98

/127.0.0.1:35825 sending packet to /127.0.0.1:51511

CSE473 DHTPv0.1

type:put

key:who

relayAdr:127.0.0.1:34417

clientAdr:127.0.0.1:56246

tag:120991850

value:hah

ttl:98

/127.0.0.1:35825 received packet from /127.0.0.1:53837

CSE473 DHTPv0.1

type:put

key:blue

tag:1920278419

value:moose

ttl:99

/127.0.0.1:35825 sending packet to /127.0.0.1:53837

CSE473 DHTPv0.1

type:success

key:blue

tag:1920278419

value:moose

ttl:99

/127.0.0.1:35825 received packet from /127.0.0.1:51183

CSE473 DHTPv0.1

type:get

key:who

tag:1342256528

ttl:99

/127.0.0.1:35825 sending packet to /127.0.0.1:51511

CSE473 DHTPv0.1

type:get

key:who

relayAdr:127.0.0.1:35825

clientAdr:127.0.0.1:51183

tag:1342256528

ttl:99

/127.0.0.1:35825 received packet from /127.0.0.1:33134

CSE473 DHTPv0.1

type:success

key:who

relayAdr:127.0.0.1:35825

clientAdr:127.0.0.1:51183

senderInfo:127.0.0.1:33134:0

tag:1342256528

value:hah

ttl:96

/127.0.0.1:35825 sending packet to /127.0.0.1:51183

CSE473 DHTPv0.1

type:success

key:who

tag:1342256528

value:hah

ttl:96

/127.0.0.1:35825 received packet from /127.0.0.1:34417

CSE473 DHTPv0.1

type:get

key:blue

relayAdr:127.0.0.1:34417

clientAdr:127.0.0.1:56674

tag:834361189

ttl:98

/127.0.0.1:35825 sending packet to /127.0.0.1:34417

CSE473 DHTPv0.1

type:success

key:blue

relayAdr:127.0.0.1:34417

clientAdr:127.0.0.1:56674

senderInfo:127.0.0.1:35825:1073741824

tag:834361189

value:moose

ttl:98

Approximately how many values are in the hash range of server number 1 when it joins the DHT? How many are in its range after the last server has joined the DHT? How many are in its range after server number 2 leaves the DHT?

*When Server1 joins the DHT, its hash range is [536870912, 1073741823], the number of values is 536870912.*

*After the last server has joined the DHT, the hash range of Server1 doesn’t change, the number of values is still 536870912.*

*After Server2 leaves the DHT, its range expands to [536870912, 1610612735], the number of values is 1073741824.*

Type the command “cat ../cfg[0-3]” and paste the output below. Note that the port numbers shown here are those used by your servers in the order 0, 1, 2, 3.

127.0.0.1 33134

127.0.0.1 34417

127.0.0.1 35825

127.0.0.1 51511

Type the command “grep ttl:9 out1” and paste a copy of the output below. Note that this shows the *ttls* in the returned packets, allowing you to infer the number of hops that each packet took on its way through the DHT and back.

ttl:98

ttl:94

ttl:95

ttl:98

ttl:96

ttl:95

ttl:98

ttl:96

ttl:98

ttl:94

ttl:95

ttl:98

ttl:94

ttl:96

ttl:98

ttl:95

ttl:94

ttl:96

ttl:98

ttl:95

ttl:96

ttl:96

ttl:98

Find the first *get* operation that took the longest number of hops before returning to the client. What were the key and value of the returned pair?

*The first get operation with longest number of hops is “get bar” whose ttl is 94. Since there is no key matching “bar”, so the type of the returned packet is “no match”.*

List the servers that the packet passed through, using the server numbers 0, 1, 2, 3.

*The client first connected with Server3, so the way the packet passed through is:*

*Client → Server3 → Server0 → Server1 → Server2 → Server3 → Client.*

Now, re-rerun script1 by typing

./script1 2 > out2

Paste the initial part of the *out2* file below (everything up through the first “*get who*” operation and the last four). Note that this allows shortcut routes, so you should expect that at least some of the packets will require fewer hops to reach the target server. **Commit the output and log files to your repository**.

put foo bar

/127.0.0.1:41471 sending packet to /127.0.0.1:40172

CSE473 DHTPv0.1

type:put

key:foo

tag:174774569

value:bar

ttl:100

/127.0.0.1:41471 received packet from /127.0.0.1:40172

CSE473 DHTPv0.1

type:success

key:foo

tag:174774569

value:bar

ttl:98

put who hah

/127.0.0.1:57464 sending packet to /127.0.0.1:59811

CSE473 DHTPv0.1

type:put

key:who

tag:1246483327

value:hah

ttl:100

/127.0.0.1:57464 received packet from /127.0.0.1:59811

CSE473 DHTPv0.1

type:success

key:who

tag:1246483327

value:hah

ttl:95

put junk mail

/127.0.0.1:53365 sending packet to /127.0.0.1:42395

CSE473 DHTPv0.1

type:put

key:junk

tag:200414795

value:mail

ttl:100

/127.0.0.1:53365 received packet from /127.0.0.1:42395

CSE473 DHTPv0.1

type:success

key:junk

tag:200414795

value:mail

ttl:95

put blue moose

/127.0.0.1:38305 sending packet to /127.0.0.1:56693

CSE473 DHTPv0.1

type:put

key:blue

tag:1550498078

value:moose

ttl:100

/127.0.0.1:38305 received packet from /127.0.0.1:56693

CSE473 DHTPv0.1

type:success

key:blue

tag:1550498078

value:moose

ttl:98

get foo

/127.0.0.1:51753 sending packet to /127.0.0.1:42395

CSE473 DHTPv0.1

type:get

key:foo

tag:1048932275

ttl:100

/127.0.0.1:51753 received packet from /127.0.0.1:42395

CSE473 DHTPv0.1

type:success

key:foo

tag:1048932275

value:bar

ttl:96

get who

/127.0.0.1:33534 sending packet to /127.0.0.1:56693

CSE473 DHTPv0.1

type:get

key:who

tag:1203320191

ttl:100

/127.0.0.1:33534 received packet from /127.0.0.1:56693

CSE473 DHTPv0.1

type:success

key:who

tag:1203320191

value:hah

ttl:96

… …

get blue

/127.0.0.1:48560 sending packet to /127.0.0.1:40172

CSE473 DHTPv0.1

type:get

key:blue

tag:1611471448

ttl:100

/127.0.0.1:48560 received packet from /127.0.0.1:40172

CSE473 DHTPv0.1

type:success

key:blue

tag:1611471448

value:moose

ttl:96

get blue

/127.0.0.1:35325 sending packet to /127.0.0.1:40172

CSE473 DHTPv0.1

type:get

key:blue

tag:1068414313

ttl:100

/127.0.0.1:35325 received packet from /127.0.0.1:40172

CSE473 DHTPv0.1

type:success

key:blue

tag:1068414313

value:moose

ttl:96

get foo

/127.0.0.1:39914 sending packet to /127.0.0.1:42395

CSE473 DHTPv0.1

type:get

key:foo

tag:110471350

ttl:100

/127.0.0.1:39914 received packet from /127.0.0.1:42395

CSE473 DHTPv0.1

type:success

key:foo

tag:110471350

value:toast is tasty

ttl:96

get junk

/127.0.0.1:59532 sending packet to /127.0.0.1:59811

CSE473 DHTPv0.1

type:get

key:junk

tag:387749987

ttl:100

/127.0.0.1:59532 received packet from /127.0.0.1:59811

CSE473 DHTPv0.1

type:success

key:junk

tag:387749987

value:mail

ttl:98

Type the command “grep ttl:9 out2” and paste the output below.

ttl:98

ttl:95

ttl:95

ttl:98

ttl:96

ttl:96

ttl:98

ttl:96

ttl:98

ttl:95

ttl:96

ttl:98

ttl:96

ttl:96

ttl:98

ttl:96

ttl:96

ttl:96

ttl:98

ttl:96

ttl:96

ttl:96

ttl:98

Type the command “cat ../cfg[0-3]” and paste the output below.

127.0.0.1 40172

127.0.0.1 59811

127.0.0.1 56693

127.0.0.1 42395

Type the command “grep rteTbl log2\_[0-3]” and paste the output below.

log2\_0:rteTbl=[(/127.0.0.1:56693,1073741824)]

log2\_0:rteTbl=[(/127.0.0.1:56693,1073741824), (/127.0.0.1:59811,536870912)]

log2\_0:rteTbl=[(/127.0.0.1:59811,536870912)]

log2\_1:rteTbl=[(/127.0.0.1:56693,1073741824)]

log2\_1:rteTbl=[(/127.0.0.1:56693,1073741824), (/127.0.0.1:40172,0)]

log2\_1:rteTbl=[(/127.0.0.1:40172,0)]

log2\_1:rteTbl=[(/127.0.0.1:40172,0), (/127.0.0.1:42395,1610612736)]

log2\_2:rteTbl=[(/127.0.0.1:40172,0)]

log2\_2:rteTbl=[(/127.0.0.1:40172,0), (/127.0.0.1:42395,1610612736)]

log2\_3:rteTbl=[(/127.0.0.1:40172,0)]

log2\_3:rteTbl=[(/127.0.0.1:40172,0), (/127.0.0.1:59811,536870912)]

log2\_3:rteTbl=[(/127.0.0.1:40172,0), (/127.0.0.1:56693,1073741824)]

log2\_3:rteTbl=[(/127.0.0.1:40172,0)]

List each server still in the DHT. For each server, list all of the servers in the DHT it still has routes to when the script finishes.

*Server0, Server1, Server3 are still in the DHT.*

*Server0 has a route to Server1.*

*Server1 has routes to Server0 and Server3.*

*Server3 has a route to Server0.*

Type the command “grep -B4 -A4 key:bar log2\_[0-3]” and paste the output below.

log2\_1-

log2\_1-/127.0.0.1:59811 received packet from /127.0.0.1:42395

log2\_1-CSE473 DHTPv0.1

log2\_1-type:get

log2\_1:key:bar

log2\_1-relayAdr:127.0.0.1:42395

log2\_1-clientAdr:127.0.0.1:39804

log2\_1-tag:213970720

log2\_1-ttl:98

log2\_1-

log2\_1-/127.0.0.1:59811 sending packet to /127.0.0.1:56693

log2\_1-CSE473 DHTPv0.1

log2\_1-type:get

log2\_1:key:bar

log2\_1-relayAdr:127.0.0.1:42395

log2\_1-clientAdr:127.0.0.1:39804

log2\_1-tag:213970720

log2\_1-ttl:98

--

log2\_2-

log2\_2-/127.0.0.1:56693 received packet from /127.0.0.1:59811

log2\_2-CSE473 DHTPv0.1

log2\_2-type:get

log2\_2:key:bar

log2\_2-relayAdr:127.0.0.1:42395

log2\_2-clientAdr:127.0.0.1:39804

log2\_2-tag:213970720

log2\_2-ttl:97

log2\_2-

log2\_2-/127.0.0.1:56693 sending packet to /127.0.0.1:42395

log2\_2-CSE473 DHTPv0.1

log2\_2-type:no match

log2\_2:key:bar

log2\_2-relayAdr:127.0.0.1:42395

log2\_2-clientAdr:127.0.0.1:39804

log2\_2-senderInfo:127.0.0.1:56693:1073741824

log2\_2-tag:213970720

--

log2\_3-

log2\_3-/127.0.0.1:42395 received packet from /127.0.0.1:39804

log2\_3-CSE473 DHTPv0.1

log2\_3-type:get

log2\_3:key:bar

log2\_3-tag:213970720

log2\_3-ttl:99

log2\_3-

log2\_3-/127.0.0.1:42395 sending packet to /127.0.0.1:59811

log2\_3-CSE473 DHTPv0.1

log2\_3-type:get

log2\_3:key:bar

log2\_3-relayAdr:127.0.0.1:42395

log2\_3-clientAdr:127.0.0.1:39804

log2\_3-tag:213970720

log2\_3-ttl:99

log2\_3-

log2\_3-/127.0.0.1:42395 received packet from /127.0.0.1:56693

log2\_3-CSE473 DHTPv0.1

log2\_3-type:no match

log2\_3:key:bar

log2\_3-relayAdr:127.0.0.1:42395

log2\_3-clientAdr:127.0.0.1:39804

log2\_3-senderInfo:127.0.0.1:56693:1073741824

log2\_3-tag:213970720

--

log2\_3-rteTbl=[(/127.0.0.1:40172,0), (/127.0.0.1:56693,1073741824)]

log2\_3-/127.0.0.1:42395 sending packet to /127.0.0.1:39804

log2\_3-CSE473 DHTPv0.1

log2\_3-type:no match

log2\_3:key:bar

log2\_3-tag:213970720

log2\_3-ttl:96

log2\_3-

log2\_3-/127.0.0.1:42395 received packet from /127.0.0.1:48585

Use the output to determine the sequence of servers that the “*get bar*” packet passed through. List them below, in the order that they handled the packet.

*Server3 → Server1 → Server2 → Server3 → Client*

Now, re-rerun script1 once more by typing

./script1 2 cache >out2c

This enables the caching feature. Paste the *final* portion of the *out2c* file below (starting with the second “*get foo*”). **Commit the output and log files to your repository**.

get foo

/127.0.0.1:38494 sending packet to /127.0.0.1:37052

CSE473 DHTPv0.1

type:get

key:foo

tag:1588629950

ttl:100

/127.0.0.1:38494 received packet from /127.0.0.1:37052

CSE473 DHTPv0.1

type:success

key:foo

tag:1588629950

value:toast is tasty

ttl:98

get blue

/127.0.0.1:34402 sending packet to /127.0.0.1:41370

CSE473 DHTPv0.1

type:get

key:blue

tag:1743463338

ttl:100

/127.0.0.1:34402 received packet from /127.0.0.1:41370

CSE473 DHTPv0.1

type:success

key:blue

tag:1743463338

value:moose

ttl:96

get blue

/127.0.0.1:32843 sending packet to /127.0.0.1:43999

CSE473 DHTPv0.1

type:get

key:blue

tag:237952558

ttl:100

/127.0.0.1:32843 received packet from /127.0.0.1:43999

CSE473 DHTPv0.1

type:success

key:blue

tag:237952558

value:moose

ttl:98

get blue

/127.0.0.1:53063 sending packet to /127.0.0.1:52614

CSE473 DHTPv0.1

type:get

key:blue

tag:2077092551

ttl:100

/127.0.0.1:53063 received packet from /127.0.0.1:52614

CSE473 DHTPv0.1

type:success

key:blue

tag:2077092551

value:moose

ttl:98

get blue

/127.0.0.1:45238 sending packet to /127.0.0.1:37052

CSE473 DHTPv0.1

type:get

key:blue

tag:1606520771

ttl:100

/127.0.0.1:45238 received packet from /127.0.0.1:37052

CSE473 DHTPv0.1

type:success

key:blue

tag:1606520771

value:moose

ttl:96

get blue

/127.0.0.1:33008 sending packet to /127.0.0.1:41370

CSE473 DHTPv0.1

type:get

key:blue

tag:2123989019

ttl:100

/127.0.0.1:33008 received packet from /127.0.0.1:41370

CSE473 DHTPv0.1

type:success

key:blue

tag:2123989019

value:moose

ttl:98

get blue

/127.0.0.1:37260 sending packet to /127.0.0.1:43999

CSE473 DHTPv0.1

type:get

key:blue

tag:278246961

ttl:100

/127.0.0.1:37260 received packet from /127.0.0.1:43999

CSE473 DHTPv0.1

type:success

key:blue

tag:278246961

value:moose

ttl:98

get blue

/127.0.0.1:47973 sending packet to /127.0.0.1:52614

CSE473 DHTPv0.1

type:get

key:blue

tag:1362223676

ttl:100

/127.0.0.1:47973 received packet from /127.0.0.1:52614

CSE473 DHTPv0.1

type:success

key:blue

tag:1362223676

value:moose

ttl:98

get blue

/127.0.0.1:55753 sending packet to /127.0.0.1:37052

CSE473 DHTPv0.1

type:get

key:blue

tag:632288375

ttl:100

/127.0.0.1:55753 received packet from /127.0.0.1:37052

CSE473 DHTPv0.1

type:success

key:blue

tag:632288375

value:moose

ttl:98

get blue

/127.0.0.1:47283 sending packet to /127.0.0.1:37052

CSE473 DHTPv0.1

type:get

key:blue

tag:991082643

ttl:100

/127.0.0.1:47283 received packet from /127.0.0.1:37052

CSE473 DHTPv0.1

type:success

key:blue

tag:991082643

value:moose

ttl:98

get foo

/127.0.0.1:56255 sending packet to /127.0.0.1:41370

CSE473 DHTPv0.1

type:get

key:foo

tag:705992160

ttl:100

/127.0.0.1:56255 received packet from /127.0.0.1:41370

CSE473 DHTPv0.1

type:success

key:foo

tag:705992160

value:bar

ttl:98

get junk

/127.0.0.1:54503 sending packet to /127.0.0.1:43999

CSE473 DHTPv0.1

type:get

key:junk

tag:1728126695

ttl:100

/127.0.0.1:54503 received packet from /127.0.0.1:43999

CSE473 DHTPv0.1

type:success

key:junk

tag:1728126695

value:mail

ttl:98

Type the command “grep ttl:9 out2c” and paste the output below.

ttl:98

ttl:95

ttl:95

ttl:98

ttl:96

ttl:96

ttl:98

ttl:96

ttl:98

ttl:95

ttl:96

ttl:98

ttl:96

ttl:98

ttl:98

ttl:96

ttl:98

ttl:98

ttl:98

ttl:98

ttl:98

ttl:98

ttl:98

Just before server 2 starts to leave the DHT network, are there are any servers that do not have the pair (*blue*, *moose*) in their cache? If so, which ones. In either case, how do you know?

*There is no server that doesn’t have the pair (blue,moose) in their cache. We can compare the ttls in this test with those in the former part of the report.*

*In this test, the ttls if the last 4 “get blue” operations before Server2 leaves are all 98, which mean that when the servers receive the requests from the clients, they send back the value directly to the clients, but only Server2 is responsible for the key “blue”. Thus, all of servers have cached this key.*

*Moreover, when we check the ttls of the former test, we can find that the ttls are 96, 96, 98, 96. The numbers of the hops are larger than those in this test, so we can know all the servers cache the pair in this test.*

***Part F (30 points).*** In this part, you will test your DHT in *onl* using multiple servers. Use the provided *onl* configuration file. Create a directory *473/lab3* that contains all the files in the lab3 directory from the repository. It must be this specific directory structure. Also, include copies of all the class files. Go to the *test2* directory, read *script2* to make sure you understand what it does. When you’re ready, type

./script2 1 > out1

Note that it starts eight servers, but that two of the servers are started only after some *puts* and *gets* have been performed. Type “cat ../cfg[0-7]” and paste the output below. **Commit the output and log files to your repository**.

192.168.7.1 54805

192.168.6.1 43037

192.168.3.2 46264

192.168.2.5 52616

192.168.2.4 34889

192.168.2.3 44581

192.168.1.1 54257

192.168.5.2 53634

Now, type “grep rteTbl log1\_[0-7]” and paste the output below.

log1\_0:rteTbl=[(/192.168.2.4:34889,1073741824)]

log1\_0:rteTbl=[(/192.168.3.2:46264,536870912)]

log1\_0:rteTbl=[(/192.168.6.1:43037,268435456)]

log1\_1:rteTbl=[(/192.168.3.2:46264,536870912)]

log1\_2:rteTbl=[(/192.168.2.4:34889,1073741824)]

log1\_2:rteTbl=[(/192.168.2.5:52616,805306368)]

log1\_3:rteTbl=[(/192.168.2.4:34889,1073741824)]

log1\_4:rteTbl=[(/192.168.7.1:54805,0)]

log1\_4:rteTbl=[(/192.168.1.1:54257,1610612736)]

log1\_4:rteTbl=[(/192.168.2.3:44581,1342177280)]

log1\_5:rteTbl=[(/192.168.1.1:54257,1610612736)]

log1\_6:rteTbl=[(/192.168.7.1:54805,0)]

log1\_6:rteTbl=[(/192.168.5.2:53634,1879048192)]

log1\_7:rteTbl=[(/192.168.7.1:54805,0)]

Are the final route values consistent with the contents of the configuration file? Explain why they are consistent, or if they are not, explain any discrepancies.

*Consistent. Since maximum size of route table is 1, the table should change if and only if its successor changes. We can see from the results above that for every server, the changing order of route values is exactly the changing order of its successor.*

Next, type “grep ttl.9 out1” and paste the output below.

ttl:98

ttl:92

ttl:95

ttl:93

ttl:96

ttl:98

ttl:94

ttl:94

ttl:93

ttl:95

ttl:96

ttl:94

ttl:93

ttl:94

ttl:96

ttl:95

ttl:98

ttl:94

ttl:94

ttl:96

ttl:98

ttl:95

ttl:98

ttl:95

ttl:95

ttl:95

ttl:98

ttl:91

ttl:98

ttl:94

ttl:96

ttl:98

ttl:93

Did any of the *get*/*put* requests get routed to all 8 servers? If not, what was the largest number of servers to handle any request? How many were handled by three or more servers?

*The path of a request:*

*Handled by X servers → Client.*

*Thus, a request handled by X servers has a ttl of 100-(x+1).*

*The largest number is 8 which corresponds to the record of “ttl=91”.*

*However, there is no request got routed to all 8 servers because in that case, the path should be like ALL 8 Servers → Relay server → Client, which means the value of ttl should be 90.*

*Handled by 3 or more servers results in a ttl value that is less than or equal to 96. There are totally 25 such requests.*

Type “grep –B15 ttl.91 out1” and paste the output below.

get slim

/192.168.4.2:38091 sending packet to /192.168.1.1:54257

CSE473 DHTPv0.1

type:get

key:slim

tag:1972186811

ttl:100

/192.168.4.2:38091 received packet from /192.168.1.1:54257

CSE473 DHTPv0.1

type:success

key:slim

tag:1972186811

value:jim

ttl:91

Type the command “grep -B3 -A4 transfer log1\_0” and paste the output below.

rteTbl=[(/192.168.6.1:43037,268435456)]

/192.168.7.1:54805 sending packet to /192.168.6.1:43037

CSE473 DHTPv0.1

type:transfer

key:flip

tag:543812369

value:flop

ttl:100

/192.168.7.1:54805 sending packet to /192.168.6.1:43037

CSE473 DHTPv0.1

type:transfer

key:who

tag:543812369

value:hah

ttl:100

Explain the output.

*The transfer events are triggered by the joining of server1(192.168.6.1:43037). After receiving join packet from server1, server0(192.168.7.1:54805) begins executing “handleJoin” function and transferring all (key, value) pairs whose hash values of key are inside the hash range of server1.*

Now, we’re going to re-run script2 using more routes. Type

./script2 3 > out3

Type “cat ../cfg[0-7]” and paste the output below. Commit the output and log files to your repository.

192.168.7.1 57521

192.168.6.1 40459

192.168.3.2 45271

192.168.2.5 42888

192.168.2.4 36742

192.168.2.3 47000

192.168.1.1 33270

192.168.5.2 40444

Now, type “grep rteTbl log3\_[0-7]” and paste the output below.

log3\_0:rteTbl=[(/192.168.2.4:36742,1073741824)]

log3\_0:rteTbl=[(/192.168.2.4:36742,1073741824), (/192.168.3.2:45271,536870912)]

log3\_0:rteTbl=[(/192.168.2.4:36742,1073741824), (/192.168.3.2:45271,536870912), (/192.168.1.1:33270,1610612736)]

log3\_0:rteTbl=[(/192.168.3.2:45271,536870912), (/192.168.1.1:33270,1610612736), (/192.168.6.1:40459,268435456)]

log3\_1:rteTbl=[(/192.168.3.2:45271,536870912)]

log3\_2:rteTbl=[(/192.168.2.4:36742,1073741824)]

log3\_2:rteTbl=[(/192.168.2.4:36742,1073741824), (/192.168.2.5:42888,805306368)]

log3\_2:rteTbl=[(/192.168.2.4:36742,1073741824), (/192.168.2.5:42888,805306368), (/192.168.7.1:57521,0)]

log3\_2:rteTbl=[(/192.168.2.5:42888,805306368), (/192.168.7.1:57521,0), (/192.168.5.2:40444,1879048192)]

log3\_2:rteTbl=[(/192.168.2.5:42888,805306368), (/192.168.5.2:40444,1879048192), (/192.168.2.4:36742,1073741824)]

log3\_2:rteTbl=[(/192.168.2.5:42888,805306368), (/192.168.2.4:36742,1073741824), (/192.168.7.1:57521,0)]

log3\_2:rteTbl=[(/192.168.2.5:42888,805306368), (/192.168.7.1:57521,0), (/192.168.1.1:33270,1610612736)]

log3\_3:rteTbl=[(/192.168.2.4:36742,1073741824)]

log3\_3:rteTbl=[(/192.168.2.4:36742,1073741824), (/192.168.7.1:57521,0)]

log3\_3:rteTbl=[(/192.168.2.4:36742,1073741824), (/192.168.7.1:57521,0), (/192.168.1.1:33270,1610612736)]

log3\_4:rteTbl=[(/192.168.7.1:57521,0)]

log3\_4:rteTbl=[(/192.168.7.1:57521,0), (/192.168.3.2:45271,536870912)]

log3\_4:rteTbl=[(/192.168.7.1:57521,0), (/192.168.3.2:45271,536870912), (/192.168.1.1:33270,1610612736)]

log3\_4:rteTbl=[(/192.168.3.2:45271,536870912), (/192.168.1.1:33270,1610612736), (/192.168.2.3:47000,1342177280)]

log3\_5:rteTbl=[(/192.168.1.1:33270,1610612736)]

log3\_5:rteTbl=[(/192.168.1.1:33270,1610612736), (/192.168.7.1:57521,0)]

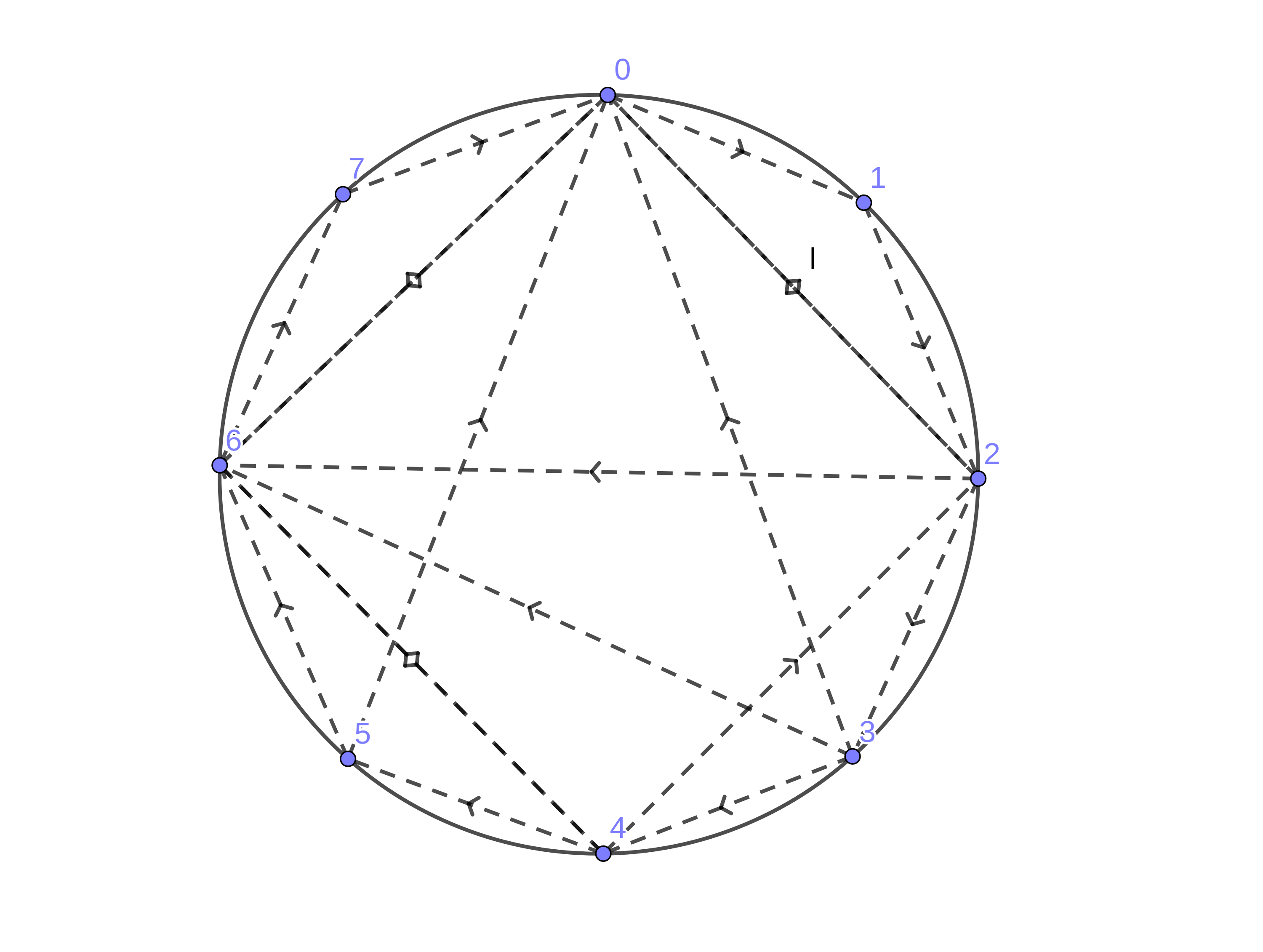
log3\_6:rteTbl=[(/192.168.7.1:57521,0)]

log3\_6:rteTbl=[(/192.168.7.1:57521,0), (/192.168.5.2:40444,1879048192)]

log3\_6:rteTbl=[(/192.168.7.1:57521,0), (/192.168.5.2:40444,1879048192), (/192.168.2.4:36742,1073741824)]

log3\_7:rteTbl=[(/192.168.7.1:57521,0)]

Draw a picture of the eight servers arranged in a circle (label them 0-7). Draw an arrow from server *i* to server *j* if *i* has a direct route to *j* at the end of the run.

**

Note that some servers have more “incoming routes” than others. Explain why this happens.

*Because routing info is added opportunistically. When a server is always responsible for the incoming requests, it responses to the relay server more often than others. Relay servers add the senderInfo to their routing table and thus this kind of server occur more times than others in other servers’ routing table.*

Next, type “grep ttl.9 out3” and paste the output below.

ttl:98

ttl:95

ttl:96

ttl:95

ttl:96

ttl:98

ttl:96

ttl:96

ttl:96

ttl:96

ttl:96

ttl:96

ttl:94

ttl:96

ttl:96

ttl:95

ttl:98

ttl:96

ttl:96

ttl:96

ttl:98

ttl:96

ttl:98

ttl:96

ttl:95

ttl:96

ttl:98

ttl:96

ttl:98

ttl:95

ttl:96

ttl:98

ttl:95

What was the largest number of servers to handle any request? How many were handled by three or more or more servers? Compare these results to those you got earlier and comment on the differences.

*The minimum ttl is 94 thus the largest number is 5.*

*25 requests were handled by three or more servers.*

*Maximum allowed number of routes in routing table causes the difference that requests were handled by less servers in this case. In the previous case, routing table contains only its successor but it can contain more shortcut routes in this case. Thus, the path of a request tends to be shorter.*

Type “grep -B15 ttl.95 out3” and paste the output below.

put who hah

/192.168.4.2:55223 sending packet to /192.168.3.2:45271

CSE473 DHTPv0.1

type:put

key:who

tag:1340853257

value:hah

ttl:100

/192.168.4.2:55223 received packet from /192.168.3.2:45271

CSE473 DHTPv0.1

type:success

key:who

tag:1340853257

value:hah

ttl:95

--

get who

/192.168.4.2:56693 sending packet to /192.168.2.5:42888

CSE473 DHTPv0.1

type:get

key:who

tag:296017435

ttl:100

/192.168.4.2:56693 received packet from /192.168.2.5:42888

CSE473 DHTPv0.1

type:success

key:who

tag:296017435

value:hah

ttl:95

--

put political follies

/192.168.4.2:49984 sending packet to /192.168.3.2:45271

CSE473 DHTPv0.1

type:put

key:political

tag:1888406851

value:follies

ttl:100

/192.168.4.2:49984 received packet from /192.168.3.2:45271

CSE473 DHTPv0.1

type:success

key:political

tag:1888406851

value:follies

ttl:95

--

get fantasy

/192.168.4.2:57361 sending packet to /192.168.2.5:42888

CSE473 DHTPv0.1

type:get

key:fantasy

tag:865119731

ttl:100

/192.168.4.2:57361 received packet from /192.168.2.5:42888

CSE473 DHTPv0.1

type:success

key:fantasy

tag:865119731

value:football

ttl:95

--

get chocolate

/192.168.4.2:48016 sending packet to /192.168.2.3:47000

CSE473 DHTPv0.1

type:get

key:chocolate

tag:77982438

ttl:100

/192.168.4.2:48016 received packet from /192.168.2.3:47000

CSE473 DHTPv0.1

type:success

key:chocolate

tag:77982438

value:fudge

ttl:95

--

get fantasy

/192.168.4.2:40975 sending packet to /192.168.3.2:45271

CSE473 DHTPv0.1

type:get

key:fantasy

tag:2127674336

ttl:100

/192.168.4.2:40975 received packet from /192.168.3.2:45271

CSE473 DHTPv0.1

type:success

key:fantasy

tag:2127674336

value:football

ttl:95

Look at the last *get* operation performed by the script. Which server is the packet sent to by the client?

*Server2(192.168.3.2:45271).*

Use the log files to determine the sequence of servers that this packet passes through. List those servers below, in order.

*Server2 → Server4 → Server6 → Server2 → Client.*

Look at the “route diagram” you made earlier. Is the path used by the packet consistent with your route diagram? If not, explain any discrepancy.

*Not consistent.*

*According to the diagram, Server2 did not have direct route to Server4 but the path showed that it had. This is because before this request, Server2 indeed had direct route to Server4. After Server6 sent response to relay server Server2, Server2 updated its routing table and since the table was full then, it removed Server4(first entry that not equal to its successor) and replaced it with Server6, which resulted in the diagram.*

Now, we are going to re-run script2 with single routes, but with caching enabled. Type

script2 1 cache >out1c

Next, type “grep ttl.9 out1c” and paste the output below. **Commit the output and log files to your repository**.

ttl:98

ttl:92

ttl:95

ttl:93

ttl:96

ttl:98

ttl:94

ttl:94

ttl:96

ttl:95

ttl:96

ttl:94

ttl:93

ttl:94

ttl:96

ttl:95

ttl:98

ttl:96

ttl:95

ttl:96

ttl:98

ttl:95

ttl:98

ttl:98

ttl:95

ttl:98

ttl:98

ttl:95

ttl:98

ttl:96

ttl:98

ttl:98

ttl:96

What was the largest number of servers to handle any request? How many were handled by three or more or more servers?

*The largest number was 7.*

*22 were handled by three or more servers.*

Compare these results to the results for the first two cases (no cache, 1 route and 3 routes) and comment on the differences.

*Obviously, this case is better than the “1 route, no cache” case because cache opportunistically shortens the path of a request.*

*Compared to the “3 routes, no cache” case we can find that the largest number is bigger but the second metric is smaller. This is because after the initial 6 servers joined, there were already shortcut routes in the previous case but caches were empty in this case, which caused initial get/put requests walked along a longer path in this case. However, as more get/put requests had been processed, caches began to work and since 3 routes in the previous case were relatively limited, it had less significant “shortcut impact” than caches. Thus, the metric of “number of requests handled by 3 or more servers” decreases in this case.*

Type “grep -B15 ttl.95 out1c” and paste the output below.

get foo

/192.168.4.2:41497 sending packet to /192.168.1.1:44338

CSE473 DHTPv0.1

type:get

key:foo

tag:687103901

ttl:100

/192.168.4.2:41497 received packet from /192.168.1.1:44338

CSE473 DHTPv0.1

type:success

key:foo

tag:687103901

value:bar

ttl:95

--

put flim flam

/192.168.4.2:45381 sending packet to /192.168.3.2:45124

CSE473 DHTPv0.1

type:put

key:flim

tag:620389723

value:flam

ttl:100

/192.168.4.2:45381 received packet from /192.168.3.2:45124

CSE473 DHTPv0.1

type:success

key:flim

tag:620389723

value:flam

ttl:95

--

put political follies

/192.168.4.2:36284 sending packet to /192.168.3.2:45124

CSE473 DHTPv0.1

type:put

key:political

tag:1138969008

value:follies

ttl:100

/192.168.4.2:36284 received packet from /192.168.3.2:45124

CSE473 DHTPv0.1

type:success

key:political

tag:1138969008

value:follies

ttl:95

--

get flip

/192.168.4.2:34078 sending packet to /192.168.2.4:53825

CSE473 DHTPv0.1

type:get

key:flip

tag:1964286899

ttl:100

/192.168.4.2:34078 received packet from /192.168.2.4:53825

CSE473 DHTPv0.1

type:success

key:flip

tag:1964286899

value:flop

ttl:95

--

get chocolate

/192.168.4.2:39261 sending packet to /192.168.1.1:44338

CSE473 DHTPv0.1

type:get

key:chocolate

tag:2036206643

ttl:100

/192.168.4.2:39261 received packet from /192.168.1.1:44338

CSE473 DHTPv0.1

type:success

key:chocolate

tag:2036206643

value:fudge

ttl:95

--

get fantasy

/192.168.4.2:54100 sending packet to /192.168.2.5:59124

CSE473 DHTPv0.1

type:get

key:fantasy

tag:1445456253

ttl:100

/192.168.4.2:54100 received packet from /192.168.2.5:59124

CSE473 DHTPv0.1

type:success

key:fantasy

tag:1445456253

value:football

ttl:95

--

get slim

/192.168.4.2:52431 sending packet to /192.168.1.1:44338

CSE473 DHTPv0.1

type:get

key:slim

tag:432941669

ttl:100

/192.168.4.2:52431 received packet from /192.168.1.1:44338

CSE473 DHTPv0.1

type:success

key:slim

tag:432941669

value:jim

ttl:95

Look at the last *get* operation performed by the script. Use the log files to determine the sequence of servers that this packet passes through. List those servers below, in order.

*Server2 → Server3 → Server2 → Client.*

Compare this to the result for earlier case of no cache and three routes. Does the request go all the way to the server that is responsible for this (*key*,*value*) pair, or does some intermediate server respond, using the contents of its cache?

*The path of this request for previous case (no cache, 3 routes) is:*

*Server2 → Server4 → Server6 → Server2 → Client.*

*From the path we know that based on hash range, Server6 should be responsible for this key. Comparing it to the current path, we could know that Server3 was an intermediate server and used its cache to respond this request.*