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

Lab 3 Report – 110 Points

##### *Gilad Gabriel Tsehori, Gai Ashkenazy*

***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.

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.lang.\*;

import sun.misc.Signal;

import sun.misc.SignalHandler;

public class DhtServer {

private static int numRoutes; // number of routes in routing table

private static boolean cacheOn; // enables caching when true

private static boolean debug; // enables debug messages when true

private static HashMap<String,String> map; // key/value pairs

private static HashMap<String,String> cache; // cached pairs

private static List<Pair<InetSocketAddress,Integer>> rteTbl;

private static DatagramSocket sock;

private static InetSocketAddress myAdr;

private static InetSocketAddress predecessor; // DHT predecessor

private static Pair<InetSocketAddress,Integer> myInfo;

private static Pair<InetSocketAddress,Integer> predInfo;

private static Pair<InetSocketAddress,Integer> succInfo; // successor

private static Pair<Integer,Integer> hashRange; // my DHT hash range

private static int sendTag; // tag for new outgoing packets

// flag for waiting leave message circle back

private static boolean stopFlag;

/\*\* Main method for DHT server.

\* Processes command line arguments, initializes data, joins DHT,

\* then starts processing requests from clients.

\*/

public static void main(String[] args) {

// process command-line arguments

if (args.length < 3) {

System.err.println("usage: DhtServer myIp numRoutes " +

"cfgFile [debug] [ predFile ] ");

System.exit(1);

}

numRoutes = Integer.parseInt(args[1]);

String cfgFile = args[2];

cacheOn = debug = false;

stopFlag = false;

String predFile = null;

for (int i = 3; i < args.length; i++) {

if (args[i].equals("cache")) cacheOn = true;

else if (args[i].equals("debug")) debug = true;

else predFile = args[i];

}

// open socket for receiving packets

// write ip and port to config file

// read predecessor's ip/port from predFile (if there is one)

InetAddress myIp = null; sock = null; predecessor = null;

try {

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

sock = new DatagramSocket(0 ,myIp);

BufferedWriter cfg =

new BufferedWriter(

new OutputStreamWriter(

new FileOutputStream(cfgFile),

"US-ASCII"));

cfg.write("" + myIp.getHostAddress() + " " +

sock.getLocalPort());

cfg.newLine();

cfg.close();

if (predFile != null) {

BufferedReader pred =

new BufferedReader(

new InputStreamReader(

new FileInputStream(predFile),

"US-ASCII"));

String s = pred.readLine();

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

predecessor = new InetSocketAddress(

chunks[0],Integer.parseInt(chunks[1]));

}

} catch(Exception e) {

System.err.println("usage: DhtServer myIp numRoutes " +

"cfgFile [ cache ] [ debug ] " +

"[ predFile ] ");

System.exit(1);

}

myAdr = new InetSocketAddress(myIp,sock.getLocalPort());

// initialize data structures

map = new HashMap<String,String>();

cache = new HashMap<String,String>();

rteTbl = new LinkedList<Pair<InetSocketAddress,Integer>>();

// join the DHT (if not the first node)

hashRange = new Pair<Integer,Integer>(0,Integer.MAX\_VALUE);

myInfo = null;

succInfo = null;

predInfo = null;

if (predecessor != null) {

join(predecessor);

} else {

myInfo = new Pair<InetSocketAddress,Integer>(myAdr,0);

succInfo = new Pair<InetSocketAddress,Integer>(myAdr,0);

predInfo = new Pair<InetSocketAddress,Integer>(myAdr,0);

}

// start processing requests from clients

Packet p = new Packet();

Packet reply = new Packet();

InetSocketAddress sender = null;

sendTag = 1;

/\* this function will be called if there's a "TERM" or "INT"

\* captured by the signal handler. It simply execute the leave

\* function and leave the program.

\*/

SignalHandler handler = new SignalHandler() {

public void handle(Signal signal) {

leave();

}

};

// Signal.handle(new Signal("KILL"), handler); // capture kill -9 signal

Signal.handle(new Signal("TERM"), handler); // capture kill -15 signal

Signal.handle(new Signal("INT"), handler); // capture ctrl+c

**while (!stopFlag) {**

try {

sender = p.receive(sock,debug);

} catch(Exception e) {

System.err.println("received packet failure");

continue;

}

if (sender == null) {

System.err.println("received packet failure");

continue;

}

if (!p.check()) {

reply.clear();

reply.type = "failure";

reply.reason = p.reason;

reply.tag = p.tag;

reply.ttl = p.ttl;

reply.send(sock,sender,debug);

continue;

}

handlePacket(p,sender);

}

}

/\*\* Hash a string, returning a 32 bit integer.

\* @param s is a string, typically the key from some get/put operation.

\* @return and integer hash value in the interval [0,2^31).

\*/

public static int hashit(String s) {

while (s.length() < 16) s += s;

byte[] sbytes = null;

try { sbytes = s.getBytes("US-ASCII");

} catch(Exception e) {

System.out.println("illegal key string");

System.exit(1);

}

int i = 0;

int h = 0x37ace45d;

while (i+1 < sbytes.length) {

int x = (sbytes[i] << 8) | sbytes[i+1];

h \*= x;

int top = h & 0xffff0000;

int bot = h & 0xffff;

h = top | (bot ^ ((top >> 16)&0xffff));

i += 2;

}

if (h < 0) h = -(h+1);

return h;

}

/\*\* Leave an existing DHT.

\*

\* Send a leave packet to it's successor and wait until stopFlag is

\* set to "true", which means leave packet is circle back.

\*

\* Send an update packet with the new hashRange and succInfo fields to

\* its predecessor, and sends an update packet with the predInfo

\* field to its successor.

\*

\* Transfers all keys and values to predecessor.

\* Clear all the existing cache, map and rteTbl information

\*/

public static void leave() {

**Packet p = new Packet();**

**p.type = "leave";**

**p.senderInfo = myInfo;**

**p.tag = sendTag;**

**// send leave packet to circle around all servers**

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

}

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

\* @param p is the received join packet

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

\*

\* The update message might contains infomation need update,

\* including predInfo, succInfo, and hashRange.

\* And add the new Predecessor/Successor into the routing table.

\* If succInfo is updated, succInfo should be removed from

\* the routing table and the new succInfo should be added

\* into the new routing table.

\*/

public static void handleUpdate(Packet p, InetSocketAddress adr) {

if (p.predInfo != null){

predInfo = p.predInfo;

}

if (p.succInfo != null){

succInfo = p.succInfo;

addRoute(succInfo);

}

if (p.hashRange != null){

hashRange = p.hashRange;

}

}

/\*\* Handle a leave packet from a leaving DHT node.

\* @param p is the received join packet

\* @param adr is the socket address of the host that sent the leave packet

\*

\* If the leave packet is sent by this server, set the stopFlag.

\* Otherwise firstly send the received leave packet to its successor,

\* and then remove the routing entry with the senderInfo of the packet.

\*/

public static void handleLeave(Packet p, InetSocketAddress adr) {

**// leave packet circled back to this server**

**if (p.senderInfo.equals(myInfo)) {**

**stopFlag = true;**

**// transfer all the map from the leaving server to its predecessor**

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

**p.clear();**

**// update the transfer packet**

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

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

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

**p.tag = sendTag;**

**//send transfer packet to predecessor**

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

**}**

**// update predecessor with relevant info**

**p.clear();**

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

**p.hashRange = new Pair(predInfo.right, hashRange.right);**

**p.succInfo = succInfo;**

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

**// update successor with relevant info**

**p.clear();**

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

**p.predInfo = predInfo;**

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

**// clearing everything before leaving**

**cache = null;**

**// remove the routes one by one to enable printing**

**while (!rteTbl.isEmpty()){**

**removeRoute(rteTbl.get(0));**

**}**

**rteTbl = null;**

**map = null;**

**return;**

**}**

**// send the leave message to successor**

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

**//remove the senderInfo from route table**

**removeRoute(p.senderInfo);**

}

/\*\* Join an existing DHT.

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

\*

\* the joining server sends a join packet to the server specified in the cfg file

\* the server joining will become the successor of the server that receives the join packet

\*/

public static void join(InetSocketAddress predAdr) {

**Packet p = new Packet();**

**p.type = "join";**

**p.tag = 1;**

**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)

\*

\* handleJoin is a function that the server that receives the join packet is running

\* first it will send a success packet to the joining server, adding it to the DHT

\* then this server will change his successor info (to the new joining server) and hashRange

\* after that it will send an update packet to original successor, to update its predecessor (joining server)

\* and then transger all the data with hash larger than first hash of new successor to the joining server

\*/

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

**InetSocketAddress originalSuccessorAddress = succInfo.left;**

**int firstHash = ((hashRange.left / 2) + (hashRange.right / 2));**

**//create new packet to joining server**

**p.clear();**

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

**p.predInfo = myInfo;**

**p.succInfo = succInfo;**

**p.senderInfo = myInfo;**

**p.hashRange = new Pair(firstHash, hashRange.right);**

**// update my hash range and the new successor (joining server)**

**hashRange.right = firstHash;**

**succInfo = new Pair(succAdr, firstHash);**

**//add succInfo to this server's routing table**

**addRoute(succInfo);**

**//send packet to new successor**

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

**// update packet to original successor**

**Packet originalSuccessor = new Packet();**

**originalSuccessor.type = "update";**

**originalSuccessor.predInfo = new Pair(succAdr, firstHash);**

**// send packet update to original successor**

**originalSuccessor.send(sock, originalSuccessorAddress, debug);**

**LinkedList<String> keysToRemove = new LinkedList<>();**

**//transfer my data to the new successor**

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

**//if this hash value is in the successors responsibility send him transfer packet**

**if (hashit(entry.getKey()) >= hashRange.right) {**

**// clear the p packet from earlier**

**p.clear();**

**// update the transfer packet**

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

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

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

**p.tag = sendTag;**

**// remove this entry from this map**

**// map.remove(entry.getKey());**

**keysToRemove.add(entry.getKey());**

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

**}**

**}**

**// remove all keys from the table**

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

**map.remove(keysToRemove.get(i));**

**}**

}

/\*\* Handle a get packet.

\* @param p is a get packet

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

\*

\* this function is ran by the server that receives the get request from the client

\* if the hash of the key is under this server return the response to the client

\* else if cache is ON check cache for this value, if it is not in cache then

\* forward the packet in the DHT

\*/

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

InetSocketAddress replyAdr;

// hash values to the check if this server is responsible for this get

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;

}

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 {

// if a server receives a get request for a key it's not

// responsible for, check if it's in the cache

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

**// respond as though it is the responsible server**

**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);**

**return;**

**}**

// hash is not this server's responsibility and

// not in cache: then 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

\*

\* if the value is in the hashRange of this server put the (key, val) in this servers map, if key = ""

\* then remove the key from the map.

\* if cache is ON and key is in cache, remove it to avoid wrong responses

\* if the key is not in this hashRange forward the packet

\*/

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

**InetSocketAddress replyAdr;**

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

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

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

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

**//this server is responsible of this key**

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

**// if this server got the request from the client**

**replyAdr = p.relayAdr;**

**p.senderInfo = myInfo;**

**} else {**

**replyAdr = senderAdr;**

**}**

**if (p.val.equals("")) {**

**//got an empty val, then remove key from map**

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

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

**map.remove(p.key);**

**} else {**

**p.type = "no match";**

**}**

**} else {**

**// put (key, val) in map**

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

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

**}**

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

**} else {**

**// if the server receives a put request for a key it's not**

**// responsible for, remove it from cache**

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

**cache.remove(p.key);**

**}**

**// forward around DHT**

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

**// update this server as relay Server for this packet**

**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

\*

\* this function runs when we need to transfer data from a server to entering server

\* or from leaving server to its predecessor

\* enter the (key, val) pair to this server

\*/

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

\*

\* this function runs on the first server that receives the request from the client

\* it adds the senderInfo of the packet to this servers routing Table

\* and sets the clientAdr, relayAdr, senderInfo to null

\* if the type of packet is "no match" or "failure" it sends it back to the client

\* if the type is success and it is a get or push request add this to the cache, and send success to client

\* and if this is a success from join, add to this joining server the relevant values

\* which are predInfo, succInfo, myInfo and hashRange

\*/

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

**InetSocketAddress clientAddress = p.clientAdr;**

**addRoute(p.senderInfo);**

**p.clientAdr = null; p.relayAdr = null; p.senderInfo = null;**

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

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

**} else if (p.type.equals("failure")) {**

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

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

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

**// store key/val in local cache**

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

**// reply to client's request for either put or get**

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

**} else {**

**// this is the joining server; add new relevant info**

**predInfo = p.predInfo;**

**// update succInfo to this server**

**succInfo = p.succInfo;**

**// when adding succInfo add it to routing table as well**

**addRoute(succInfo);**

**hashRange = p.hashRange;**

**myInfo = new Pair(myAdr, hashRange.left);**

**}**

**}**

}

/\*\* Handle packets received from clients or other servers

\* @param p is a packet

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

\*/

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

if (p.senderInfo != null && !p.type.equals("leave"))

addRoute(p.senderInfo);

if (p.type.equals("get")) {

handleGet(p,senderAdr);

} else if (p.type.equals("put")) {

handlePut(p, senderAdr);

} else if (p.type.equals("transfer")) {

handleXfer(p, senderAdr);

} else if (p.type.equals("success") ||

p.type.equals("no match") ||

p.type.equals("failure")) {

handleReply(p, senderAdr);

} else if (p.type.equals("join")) {

handleJoin(p, senderAdr);

} else if (p.type.equals("update")){

handleUpdate(p, senderAdr);

} else if (p.type.equals("leave")){

handleLeave(p, senderAdr);

}

}

/\*\* Add an entry to the route tabe.

\* @param newRoute is a pair (addr,hash) where addr is the socket

\* address for some server and hash is the first hash in that

\* server's range

\*

\* If the number of entries in the table exceeds the max

\* number allowed, the first entry that does not refer to

\* the successor of this server, is removed.

\* If debug is true and the set of stored routes does change,

\* print the string "rteTbl=" + rteTbl. (IMPORTANT)

\*/

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

**if (rteTbl.size() < numRoutes && !rteTbl.contains(newRoute)) {**

**rteTbl.add(newRoute);**

**} else if (!rteTbl.contains(newRoute)){**

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

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

**rteTbl.remove(i);**

**rteTbl.add(newRoute);**

**break;**

**}**

**}**

**}**

**if (debug) {**

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

**}**

}

/\*\* Remove an entry from the route table.

\* @param rmRoute is the route information for some server

\* need to be removed from route table

\*

\* If the route information exists in current entries, remove it.

\* Otherwise, do nothing.

\* If debug is true and the set of stored routes does change,

\* print the string "rteTbl=" + rteTbl. (IMPORTANT)

\*/

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

**boolean flag = rteTbl.indexOf(rmRoute) != -1;**

**if (flag) {**

**rteTbl.remove(rteTbl.indexOf(rmRoute));**

**if (debug) {**

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

**}**

**}**

}

/\*\* Forward a packet using the local routing table.

\* @param p is a packet to be forwarded

\* @param hash is the hash of the packet's key field

\*

\* This method selects a server from its route table that is

\* "closest" to the target of this packet (based on hash).

\* If firstHash is the first hash in a server's range, then

\* we seek to minimize the difference hash-firstHash, where

\* the difference is interpreted modulo the range of hash values.

\* IMPORTANT POINT - handle "wrap-around" correctly.

\* Once a server is selected, p is sent to that server.

\*/

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

**// if routing table is empty dont forward to routing table, send to successor**

**if (rteTbl.size() == 0) {**

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

**return;**

**}**

**// differe saves the minimum positive difference between hash and first hash**

**// of each entry in the routing table**

**int difference = Integer.MAX\_VALUE, index = -1;**

**// run on all servers in the routing table**

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

**int currentDifference = hash - rteTbl.get(i).right;**

**boolean positive = currentDifference >= 0;**

**if (currentDifference < difference && positive) {**

**difference = currentDifference;**

**index = i;**

**}**

**}**

**if (difference != Integer.MAX\_VALUE){**

**p.send(sock, rteTbl.get(index).left, debug);**

**} else {**

**difference = 0;**

**// all the dufferences were negative, send packet to the router that has the largest difference**

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

**int currentDifference = rteTbl.get(i).right - hash;**

**if (currentDifference >= difference) {**

**difference = currentDifference;**

**index = i;**

**}**

**}**

**p.send(sock, rteTbl.get(index).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.

import java.io.\*;

import java.net.\*;

import java.util.\*;

/\*\* Class for working with DHT packets. \*/

public class Packet {

// packet fields - note: all are public

public String type; // packet type

public int ttl; // time-to-live

public String key; // DHT key string

public String val; // DHT value string

public String reason; // reason for a failure

public InetSocketAddress clientAdr; // address of original client

public InetSocketAddress relayAdr; // address of first DHT server

public int tag; // tag used to identify packet

public Pair<Integer,Integer> hashRange; // range of hash values

public Pair<InetSocketAddress,Integer> senderInfo;// address, first hash

public Pair<InetSocketAddress,Integer> succInfo; // address, first hash

public Pair<InetSocketAddress,Integer> predInfo; // address, first hash

/\*\* Constructor, initializes fields to default values. \*/

public Packet() { clear(); }

/\*\* Initialize all packet fields.

\* Initializes all fields with a standard initial value

\* or makes them undefined.

\*/

public void clear() {

type = null; ttl = 100; key = null; val = null;

reason = null; clientAdr = null; relayAdr = null;

tag = -1; hashRange = null;

senderInfo = null; succInfo = null; predInfo = null;

}

/\*\* Pack attributes defining packet fields into buffer.

\* Fails if the packet type is undefined or if the resulting

\* buffer exceeds the allowed length of 1400 bytes.

\* @return null on failure, otherwise a byte array

\* containing the packet payload.

\*/

public byte[] pack() {

if (type == null) return null;

byte[] buf;

try { buf = toString().getBytes("US-ASCII");

} catch(Exception e) { return null; }

if (buf.length > 1400) return null;

return buf;

}

/\*\* Unpack attributes defining packet fields from buffer.

\* @param buf is a byte array containing the DHT packet

\* (or if you like, the payload of a UDP packet).

\* @param bufLen is the number of valid bytes in buf

\*/

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("value")){**

**val=right;**

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

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

**} 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("hashRange")){**

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

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

**int startRange = Integer.parseInt(chunks[0]);**

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

**hashRange = new**

**Pair<Integer,Integer>(startRange, endRange);**

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

**reason = right;**

**} 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("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 {

// ignore lines that don't match defined field

}

}

return true;

}

/\*\* Basic validity checking for received packets.

\* @return true on success, false on failure;

\* on failure, place an explanatory String in the reason field

\* of the packet

\*/

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;

}

return true;

}

/\*\* Create String representation of packet.

\* The resulting String is produced using the defined

\* attributes and is formatted with one field per line,

\* allowing it to be used as the actual buffer contents.

\*/

public String toString() {

// every packet starts with this line

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

// build the packet by this logic

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 (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");

}

**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 (clientAdr != null) {**

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

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

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

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

**}**

**if (tag != -1) {**

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

**}**

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

**s.append("value:"); s.append(val); s.append("\n");**

**}**

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

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

**}**

if (ttl != -1) {

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

}

return s.toString();

}

/\*\* Send the packet to a specified destination.

\* Packs the various packet fields into a buffer

\* before sending. Does no validity checking.

\* @param sock is the socket on which the packet is sent

\* @param dest is the socket address of the destination

\* debug is a flag; if true, the packet is printed before it is sent

\* @return true on success, false on failure

\*/

public boolean send(DatagramSocket sock, InetSocketAddress dest,

boolean debug) {

if (debug) {

System.out.println("" + sock.getLocalSocketAddress() +

" sending packet to " + dest + "\n" +

toString());

System.out.flush();

}

byte[] buf = this.pack();

if (buf == null) return false;

DatagramPacket pkt = new DatagramPacket(buf, buf.length);

pkt.setSocketAddress(dest);

try { sock.send(pkt); } catch(Exception e) { return false; }

return true;

}

/\*\* Get the next packet on the socket.

\*

\* Receives the next datagram from the socket and

\* unpacks it.

\* @param sock is the socket on which the packet is received

\* @param debug is a flag; if it is true, the received

\* packet is printed

\* @return the sender's socket address on success and null on failure

\*/

public InetSocketAddress receive(DatagramSocket sock, boolean debug) {

clear();

byte[] buf = new byte[2000];

DatagramPacket pkt = new DatagramPacket(buf, buf.length);

try {

sock.receive(pkt);

} catch(Exception e) {

System.out.println("receive exception: " + e);

return null;

}

if (!unpack(buf,pkt.getLength())) {

System.out.println("error while unpacking packet");

return null;

}

ttl--;

if (debug) {

System.out.println(sock.getLocalSocketAddress() +

" received packet from " +

pkt.getSocketAddress() + "\n" + toString());

System.out.flush();

}

if (ttl < 0) {

return null;

}

return (InetSocketAddress) pkt.getSocketAddress();

}

}

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

import java.io.\*;

import java.net.\*;

/\*\*

\* usage: DhtClient hostName configurationFile command [key] [value] [ttl]

\* this is a class of a client that sends UDP packets to the the DHT Server

\* the client reads from cgf file the servers InterAddress and port and sends this specific server

\* in the DHT the request, the server is responsible to search the DHT and send back the correct response

\*\*/

public class DhtClient {

private static boolean debug; // enables debug messages when true

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

//open file to read the servers address and port

BufferedReader pred = new BufferedReader( new InputStreamReader(

new FileInputStream(args[1]), "US-ASCII"));

// get server address and port number

String s = pred.readLine();

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

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

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

// open datagram socket with specified hostName (args[0])

DatagramSocket sock = new DatagramSocket(0, InetAddress.getByName(args[0]));

// Packet class, open new Packet

Packet p = new Packet();

// build packet p

String command = "CSE473 DHTPv0.1\n";

if (args.length < 4) {

System.out.println("Usage: DhtClient serverName configurationFile command key [value] [ttl]");

System.exit(1);

} else if (args.length >= 4 && args[2].equals("get")){

p.type = "get";

p.key = args[3];

p.tag = 1;

if(args.length == 5){

p.ttl = Integer.parseInt(args[4]);

}

} else if (args.length >= 4 && args[2].equals("put")){

String value = (args.length == 5) ? args[4] : "";

p.type = "put";

p.key = args[3];

p.val = value;

p.tag = 1;

if(args.length == 6){

p.ttl = Integer.parseInt(args[5]);

}

} else {

System.out.println("Usage: DhtClient serverName configurationFile command key [value] [ttl]");

System.exit(1);

}

p.send(sock, new InetSocketAddress(serverAdr, serverPort), debug); // send packet to server

// create buffer and packet for reply, then receive it

byte[] inBuf = new byte[1000];

// clear packet

p.clear();

// receive packet from server

p.receive(sock, debug);

// print packet received

System.out.println(p);

//close the socket

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

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:98

put who hah

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:98

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:98

get who

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:98

get goodbye

CSE473 DHTPv0.1

type:no match

key:goodbye

tag:1

ttl:98

get

Usage: DhtClient serverName configurationFile command key [value] [ttl]

get bar

CSE473 DHTPv0.1

type:no match

key:bar

tag:1

ttl:98

put foo toast is tasty

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:toast is tasty

ttl:98

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

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

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:98

put who hah

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:94

put junk mail

CSE473 DHTPv0.1

type:success

key:junk

tag:1

value:mail

ttl:95

put blue moose

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:96

get who

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:95

-----

Until first “get who” operation.

Last 4 operations:

-----

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:95

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:96

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:toast is tasty

ttl:96

get junk

CSE473 DHTPv0.1

type:success

key:junk

tag:1

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?

*After observing both the out1 file and the corresponding log files, we can see that the pair (blue, moose) was initially put in server number 2. Then, this server left the DHT, and the pair was transferred to server number 1 (which joined the DHT “in between” servers 0 and 2).*

*Finally, server number 1 is the one that holds the pair (blue, moose), and the ttl of the packet returned to the client is 96, as observed in out1.*

*By examining the log1\_1 file, we were able to tell that at the time the “get” request was sent to server 1, it used port number 53634 (but this was probably different in previous and later cases, as the DHT uses UDP packets and the port is not a default but rather set to some available local port).*

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.*

*We see that there are 4 requests with different ttls, which are 98 ,96, 95 and 94. We can immediately tell that the server that has 98 ttl is the one that holds the requested pair, while the others are “looking” for this server to respond.*

*In our case, as seen in the previous question, server 2 is the one holding the pair; we see from log1\_2 that this server’s port number is 39261.*

*Now it is important to mention that the reason that we have a “jump” of 1 from 98 to 96, is because any other server than the one holding the pair has to forward the packet to a relay server.*

*Therefore, we can see that the server that is the predecessor to server 2 is server 1 which has a ttl of 96; that is because a “get” packet will be sent to server 1, forwarded to server 2 and then sent back to the relay server (which is server 1 itself in this case) and only then back to the client.  
The port number of server 1, as seen in log1\_1, is 53634.*

*By the same reasoning, we get that the port number of server number 0 (with ttl 95) is 60652 by log1\_0, and the port number of server number 3 (with ttl 94) is 36038 by log1\_3.*

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

*In between these two “get blue” operations, the server that held the pair (server number 2) left the DHT. In consequence, the pair was transferred to its predecessor in the DHT, which is server number 1.*

*Therefore, the path from server number 0 (which received the “get” request) to the destination server was shortened by one edge, and so the ttl increased from 95 to 96.*

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

/127.0.0.1:39261 sending packet to /127.0.0.1:60652

CSE473 DHTPv0.1

type:join

tag:1

ttl:100

/127.0.0.1:39261 received packet from /127.0.0.1:60652

CSE473 DHTPv0.1

type:success

hashRange:1073741823:2147483647

senderInfo:127.0.0.1:60652:0

succInfo:127.0.0.1:60652:0

predInfo:127.0.0.1:60652:0

ttl:99

rteTbl=[(/127.0.0.1:60652,0)]

rteTbl=[(/127.0.0.1:60652,0)]

rteTbl=[(/127.0.0.1:60652,0)]

/127.0.0.1:39261 received packet from /127.0.0.1:60652

CSE473 DHTPv0.1

type:update

predInfo:127.0.0.1:53634:536870911

ttl:99

/127.0.0.1:39261 received packet from /127.0.0.1:36038

CSE473 DHTPv0.1

type:join

tag:1

ttl:99

rteTbl=[(/127.0.0.1:36038,1610612734)]

/127.0.0.1:39261 sending packet to /127.0.0.1:36038

CSE473 DHTPv0.1

type:success

hashRange:1610612734:2147483647

senderInfo:127.0.0.1:39261:1073741823

succInfo:127.0.0.1:60652:0

predInfo:127.0.0.1:39261:1073741823

ttl:100

/127.0.0.1:39261 sending packet to /127.0.0.1:60652

CSE473 DHTPv0.1

type:update

predInfo:127.0.0.1:36038:1610612734

ttl:100

/127.0.0.1:39261 received packet from /127.0.0.1:53634

CSE473 DHTPv0.1

type:put

key:who

relayAdr:127.0.0.1:53634

clientAdr:127.0.0.1:46367

tag:1

value:hah

ttl:98

/127.0.0.1:39261 sending packet to /127.0.0.1:36038

CSE473 DHTPv0.1

type:put

key:who

relayAdr:127.0.0.1:53634

clientAdr:127.0.0.1:46367

tag:1

value:hah

ttl:98

/127.0.0.1:39261 received packet from /127.0.0.1:55868

CSE473 DHTPv0.1

type:put

key:blue

tag:1

value:moose

ttl:99

/127.0.0.1:39261 sending packet to /127.0.0.1:55868

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:99

/127.0.0.1:39261 received packet from /127.0.0.1:49016

CSE473 DHTPv0.1

type:get

key:who

tag:1

ttl:99

/127.0.0.1:39261 sending packet to /127.0.0.1:36038

CSE473 DHTPv0.1

type:get

key:who

relayAdr:127.0.0.1:39261

clientAdr:127.0.0.1:49016

tag:1

ttl:99

/127.0.0.1:39261 received packet from /127.0.0.1:60652

CSE473 DHTPv0.1

type:success

key:who

relayAdr:127.0.0.1:39261

senderInfo:127.0.0.1:60652:0

clientAdr:127.0.0.1:49016

tag:1

value:hah

ttl:96

rteTbl=[(/127.0.0.1:36038,1610612734)]

rteTbl=[(/127.0.0.1:36038,1610612734)]

/127.0.0.1:39261 sending packet to /127.0.0.1:49016

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:96

/127.0.0.1:39261 received packet from /127.0.0.1:53634

CSE473 DHTPv0.1

type:get

key:blue

relayAdr:127.0.0.1:53634

clientAdr:127.0.0.1:45980

tag:1

ttl:98

/127.0.0.1:39261 sending packet to /127.0.0.1:53634

CSE473 DHTPv0.1

type:success

key:blue

relayAdr:127.0.0.1:53634

senderInfo:127.0.0.1:39261:1073741823

clientAdr:127.0.0.1:45980

tag:1

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 server 1 joins the DHT (while servers 0 and 2 are already there), it receives the upper half of the range of server 0 (which had half the values in the entire range). Therefore, server 1 holds approximately values in its range (a quarter of the entire range).*

*After the last server joins the DHT, the number of values in the range of server 1 does not change.*

*After server number 2 leaves the DHT, server number 1 receives its range, and so it will hold half the range: .*

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 60652

127.0.0.1 53634

127.0.0.1 39261

127.0.0.1 36038

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: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 the longest number of hops before returning to the client is “get bar”, which has no match for the key in the DHT.*

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

*The packet is first arriving at server number 3, then forwarded around the DHT as follows: it is sent to server number 0, then server 1, server 2 and finally back to the relay server (server number 3) which responds back to the client “no match”.*

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

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:98

put who hah

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:96

put junk mail

CSE473 DHTPv0.1

type:success

key:junk

tag:1

value:mail

ttl:95

put blue moose

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:96

get who

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:96

----

Until first “get who” operation.

Last 4 operations:

----

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:96

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:96

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:toast is tasty

ttl:96

get junk

CSE473 DHTPv0.1

type:success

key:junk

tag:1

value:mail

ttl:98

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

ttl:98

ttl:96

ttl:95

ttl:98

ttl:96

ttl:96

ttl:98

ttl:96

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 56321

127.0.0.1 38286

127.0.0.1 53062

127.0.0.1 46740

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

log2\_0:rteTbl=[(/127.0.0.1:53062,1073741823)]

log2\_0:rteTbl=[(/127.0.0.1:53062,1073741823), (/127.0.0.1:38286,536870911)]

log2\_0:rteTbl=[(/127.0.0.1:53062,1073741823), (/127.0.0.1:38286,536870911)]

log2\_0:rteTbl=[(/127.0.0.1:53062,1073741823), (/127.0.0.1:38286,536870911)]

log2\_0:rteTbl=[(/127.0.0.1:53062,1073741823), (/127.0.0.1:38286,536870911)]

log2\_0:rteTbl=[(/127.0.0.1:53062,1073741823), (/127.0.0.1:38286,536870911)]

log2\_0:rteTbl= [(/127.0.0.1:38286,536870911)]

log2\_0:rteTbl=[(/127.0.0.1:38286,536870911)]

log2\_0:rteTbl=[(/127.0.0.1:38286,536870911)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_1:rteTbl= [(/127.0.0.1:56321,0)]

log2\_1:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl= [(/127.0.0.1:46740,1610612734)]

log2\_2:rteTbl= []

log2\_3:rteTbl=[(/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:53062,1073741823), (/127.0.0.1:56321,0)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:38286,536870911)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:38286,536870911)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:38286,536870911)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:38286,536870911)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3:rteTbl= [(/127.0.0.1:56321,0)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0)]

log2\_3:rteTbl=[(/127.0.0.1:56321,0)]

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.

*The servers that are still in the DHT after the script terminates are servers 0, 1 and 3. The list of servers and who they still have direct routes to:*

*Server number 0 has a route to server number 1.  
Server number 1 has a route to server number 0.  
Server number 3 has a route to server number 0.*

*We can tell this numbers by the first hash that exists in the pairs in the corresponding routing table.*

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

log2\_1-

log2\_1-/127.0.0.1:38286 received packet from /127.0.0.1:46740

log2\_1-CSE473 DHTPv0.1

log2\_1-type:get

log2\_1:key:bar

log2\_1-relayAdr:127.0.0.1:46740

log2\_1-clientAdr:127.0.0.1:38875

log2\_1-tag:1

log2\_1-ttl:98

log2\_1-

log2\_1-/127.0.0.1:38286 sending packet to /127.0.0.1:53062

log2\_1-CSE473 DHTPv0.1

log2\_1-type:get

log2\_1:key:bar

log2\_1-relayAdr:127.0.0.1:46740

log2\_1-clientAdr:127.0.0.1:38875

log2\_1-tag:1

log2\_1-ttl:98

--

log2\_2-

log2\_2-/127.0.0.1:53062 received packet from /127.0.0.1:38286

log2\_2-CSE473 DHTPv0.1

log2\_2-type:get

log2\_2:key:bar

log2\_2-relayAdr:127.0.0.1:46740

log2\_2-clientAdr:127.0.0.1:38875

log2\_2-tag:1

log2\_2-ttl:97

log2\_2-

log2\_2-/127.0.0.1:53062 sending packet to /127.0.0.1:46740

log2\_2-CSE473 DHTPv0.1

log2\_2-type:no match

log2\_2:key:bar

log2\_2-relayAdr:127.0.0.1:46740

log2\_2-senderInfo:127.0.0.1:53062:1073741823

log2\_2-clientAdr:127.0.0.1:38875

log2\_2-tag:1

--

log2\_3-

log2\_3-/127.0.0.1:46740 received packet from /127.0.0.1:38875

log2\_3-CSE473 DHTPv0.1

log2\_3-type:get

log2\_3:key:bar

log2\_3-tag:1

log2\_3-ttl:99

log2\_3-

log2\_3-/127.0.0.1:46740 sending packet to /127.0.0.1:38286

log2\_3-CSE473 DHTPv0.1

log2\_3-type:get

log2\_3:key:bar

log2\_3-relayAdr:127.0.0.1:46740

log2\_3-clientAdr:127.0.0.1:38875

log2\_3-tag:1

log2\_3-ttl:99

log2\_3-

log2\_3-/127.0.0.1:46740 received packet from /127.0.0.1:53062

log2\_3-CSE473 DHTPv0.1

log2\_3-type:no match

log2\_3:key:bar

log2\_3-relayAdr:127.0.0.1:46740

log2\_3-senderInfo:127.0.0.1:53062:1073741823

log2\_3-clientAdr:127.0.0.1:38875

log2\_3-tag:1

--

log2\_3-rteTbl=[(/127.0.0.1:56321,0), (/127.0.0.1:53062,1073741823)]

log2\_3-/127.0.0.1:46740 sending packet to /127.0.0.1:38875

log2\_3-CSE473 DHTPv0.1

log2\_3-type:no match

log2\_3:key:bar

log2\_3-tag:1

log2\_3-ttl:96

log2\_3-

log2\_3-/127.0.0.1:46740 received packet from /127.0.0.1:56262

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.

*Analyzing the output above, we can tell that the client first addressed server number 3 with the request “get bar” (which no server holds). Then, the packet with the request is passed from server number 3 to server number 1, then server number 2.*

*We can compute (using the provided hashit function) that the hash of the key “bar” belongs to the jurisdiction of server number 2. Therefore, it replies back to the relay address (server number 3) that there is no match for this key, and server 3 returns the response back to the client.*

*Overall, the path is: client -> server 3 -> server 1 -> server 2 -> server 3 -> 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

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:toast is tasty

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:96

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:96

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get blue

CSE473 DHTPv0.1

type:success

key:blue

tag:1

value:moose

ttl:98

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:98

get junk

CSE473 DHTPv0.1

type:success

key:junk

tag:1

value:mail

ttl:98

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

ttl:98

ttl:96

ttl:95

ttl:98

ttl:96

ttl:96

ttl:98

ttl:96

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?

*By analyzing the code of script1 and the output of the command* “grep ttl:9 out2c”*, we can see that right before server number 2 starts to leave the DHT, each of the 4 servers in the network receives a “get blue” request which is answered by the server immediately (that is, if for example “get blue” is sent to server number 1, it retrieves it from its cache and replies to the client directly).*

*We can see that from the output of the command above; each of this 4 “get blue” requests corresponds to a line where ttl:98, which means that all servers hold the pair (blue, moose) in their cache (with the obvious exception of server number 2, which holds the pair in its map rather than in its cache).****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 49768

192.168.6.1 41671

192.168.3.2 33357

192.168.2.5 35184

192.168.2.4 35603

192.168.2.3 41960

192.168.1.1 55136

192.168.5.2 60190

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

log1\_0:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_0:rteTbl=[(/192.168.6.1:41671,268435455)]

log1\_1:rteTbl=[(/192.168.7.1:49768,0)]

log1\_1:rteTbl=[(/192.168.7.1:49768,0)]

log1\_1:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_2:rteTbl=[(/192.168.7.1:49768,0)]

log1\_2:rteTbl=[(/192.168.7.1:49768,0)]

log1\_2:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_2:rteTbl=[(/192.168.2.5:35184,805306366)]

log1\_3:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_3:rteTbl=[(/192.168.3.2:33357,536870911)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_3:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_4:rteTbl=[(/192.168.7.1:49768,0)]

log1\_4:rteTbl=[(/192.168.7.1:49768,0)]

log1\_4:rteTbl=[(/192.168.7.1:49768,0)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_4:rteTbl=[(/192.168.2.3:41960,1342177278)]

log1\_5:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_5:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_5:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_5:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_5:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_6:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_6:rteTbl=[(/192.168.2.4:35603,1073741823)]

log1\_6:rteTbl=[(/192.168.7.1:49768,0)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_6:rteTbl=[(/192.168.5.2:60190,1879048190)]

log1\_7:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_7:rteTbl=[(/192.168.1.1:55136,1610612734)]

log1\_7:rteTbl=[(/192.168.7.1:49768,0)]

log1\_7:rteTbl=[(/192.168.7.1:49768,0)]

log1\_7:rteTbl=[(/192.168.7.1:49768,0)]

log1\_7:rteTbl=[(/192.168.7.1:49768,0)]

log1\_7:rteTbl=[(/192.168.7.1:49768,0)]

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.

*The final route values (the last line for each of the log files in the output from the command* “grep rteTbl log1\_[0-7]” *above) are indeed consistent with the contents of the configuration files.*

*Since we ran script2 with an allowed number of routes of 1, it must be the case that each server has only its current successor in the routing table, and this is the case here. For example, we can see that the server associated with cfg0 (which is 192.168.7.1) has in its final route table the address of the server associated with cfg1 (which is 192.168.6.1).*

*This pattern goes the same for every two sequential servers, that is, for every , the server associated with has the address of the server associated with in its final routing table.*

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

ttl:98

ttl:92

ttl:95

ttl:93

ttl:96

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?

*By the output above we can see that there was no get/put request the was routed to all 8 servers. The reason is as follows: first notice that whenever ttl:98, the request was fulfilled by the first server that the client turned to. Next, we won’t see ttl:97, but rather ttl:96; this is because of the relay server used.*

*Therefore, the lowest number is ttl:91, which means that the largest number of servers to handle any request was 7.*

*To count how many were handled by three or more servers, we count the number of lines where , which is 20.*

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

get flip

CSE473 DHTPv0.1

type:success

key:flip

tag:1

value:flop

ttl:98

get slim

CSE473 DHTPv0.1

type:success

key:slim

tag:1

value:jim

ttl:91

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

/192.168.7.1:49768 sending packet to /192.168.6.1:41671

CSE473 DHTPv0.1

type:transfer

key:flip

tag:1

value:flop

ttl:100

/192.168.7.1:49768 sending packet to /192.168.6.1:41671

CSE473 DHTPv0.1

type:transfer

key:who

tag:1

value:hah

ttl:100

Explain the output.

*When server 1 joined the DHT, it received the upper half of the hash range of server 0. The two pairs (flip, flop) and (who, hah) belonged to server 0, and after server 1 joined, they were transferred to it.*

*We can also see that the ttl field of both packets is 100; this is because they were created and sent from server 0 to server 1.*

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 55982

192.168.6.1 41636

192.168.3.2 40451

192.168.2.5 40380

192.168.2.4 55532

192.168.2.3 52181

192.168.1.1 43692

192.168.5.2 41137

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

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911)]

log3\_0:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.3.2:40451,536870911), (/192.168.6.1:41636,268435455)]

log3\_1:rteTbl=[(/192.168.7.1:55982,0)]

log3\_1:rteTbl=[(/192.168.7.1:55982,0)]

log3\_1:rteTbl=[(/192.168.7.1:55982,0), (/192.168.3.2:40451,536870911)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0), (/192.168.2.4:55532,1073741823)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0), (/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0), (/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0), (/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0), (/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366)]

log3\_2:rteTbl=[(/192.168.7.1:55982,0), (/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190)]

log3\_2:rteTbl=[(/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190), (/192.168.1.1:43692,1610612734)]

log3\_2:rteTbl=[(/192.168.2.5:40380,805306366), (/192.168.5.2:41137,1879048190), (/192.168.1.1:43692,1610612734)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.3.2:40451,536870911), (/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_3:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_3:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_3:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_3:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734)]

log3\_4:rteTbl=[(/192.168.7.1:55982,0), (/192.168.1.1:43692,1610612734), (/192.168.2.3:52181,1342177278)]

log3\_5:rteTbl=[(/192.168.2.4:55532,1073741823)]

log3\_5:rteTbl=[(/192.168.2.4:55532,1073741823)]

log3\_5:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.1.1:43692,1610612734)]

log3\_5:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

log3\_5:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_6:rteTbl=[(/192.168.2.4:55532,1073741823), (/192.168.7.1:55982,0), (/192.168.5.2:41137,1879048190)]

log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734)]

log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734)]

log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

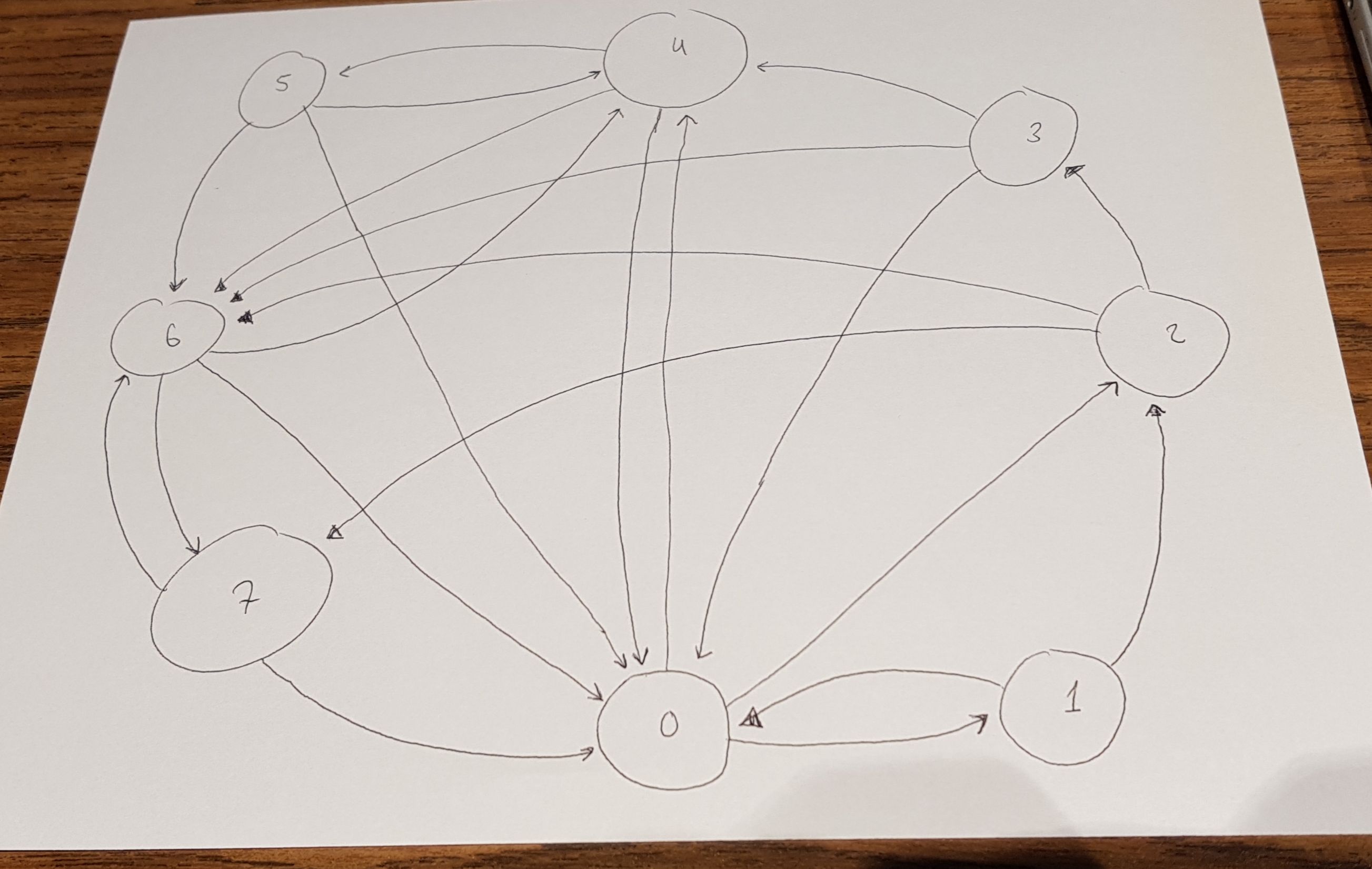
log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

log3\_7:rteTbl=[(/192.168.1.1:43692,1610612734), (/192.168.7.1:55982,0)]

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.

*A server gets more incoming routes when it gets more traffic, that it: it is addressed more by other servers. Therefore, severs like server 0 gets a lot more incoming get/put requests and more servers have a shortcut to server 0.*

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

ttl:98

ttl:96

ttl:96

ttl:95

ttl:96

ttl:96

ttl:96

ttl:96

ttl:96

ttl:96

ttl:96

ttl:94

ttl:96

ttl:96

ttl:96

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.

*Unlike before (when the lowest ttl was 91 so the largest number of servers to handle a request was 7), now the lowest ttl is 94; therefore, the largest number of servers to handle a any single request was 4.*

*This time, only 5 requests were handled by three or more servers.*

*The reason behind the differences is that this time we allowed a routing table for each server (bigger than 1) so each server can hold shortcuts to different servers.*

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

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:96

get who

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:95

--

get political

CSE473 DHTPv0.1

type:success

key:political

tag:1

value:follies

ttl:96

get fantasy

CSE473 DHTPv0.1

type:success

key:fantasy

tag:1

value:football

ttl:95

--

get abra

CSE473 DHTPv0.1

type:success

key:abra

tag:1

value:cadabra

ttl:98

get chocolate

CSE473 DHTPv0.1

type:success

key:chocolate

tag:1

value:fudge

ttl:95

--

get political

CSE473 DHTPv0.1

type:success

key:political

tag:1

value:follies

ttl:98

get fantasy

CSE473 DHTPv0.1

type:success

key:fantasy

tag:1

value:football

ttl:95

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

*It was sent to server number 2 (with the address 192.168.3.2).*

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

*The first server, as mentioned, is server number 2. Then, using the routing table, there is a shortcut from server number 2 to server number 4. Server number 4 sends the packet to server number 6, which has the pair in its map; then a success packet is created by server number 6, which is sent to the relay server (server 2) and sent back to the 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.

*No; this is because the routing table for server number 2 has changed after the last get request. Before this get request, the routing table had 3 entries; to server numbers 3, 4 and 7, and after the success packet got back through server number 2, its routing table was changed in the following way: server number 4 was “kicked” out of the routing table (otherwise it would’ve exceeded its limit of 3), and server number 6 replaced it.*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: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 of servers to handle any single request was 5. There were 14 requests that were handled by three servers or more.*

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

*We can see that the results for cache with no routing were worse than the results for 3 routes, but better than only 1 route with no caching; that is, the largest number of servers that handled a single request was smaller than the number with no caching and no routing, but larger than the one with 3 routes, and the number of requests handled by 3 servers or more is lower than no caching and no routing but higher than 3 routes.*

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

put who hah

CSE473 DHTPv0.1

type:success

key:who

tag:1

value:hah

ttl:92

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:bar

ttl:95

--

get foo

CSE473 DHTPv0.1

type:success

key:foo

tag:1

value:toast is tasty

ttl:96

put flim flam

CSE473 DHTPv0.1

type:success

key:flim

tag:1

value:flam

ttl:95

--

put dungeons dragons

CSE473 DHTPv0.1

type:success

key:dungeons

tag:1

value:dragons

ttl:96

put political follies

CSE473 DHTPv0.1

type:success

key:political

tag:1

value:follies

ttl:95

--

get flim

CSE473 DHTPv0.1

type:success

key:flim

tag:1

value:flam

ttl:96

get flip

CSE473 DHTPv0.1

type:success

key:flip

tag:1

value:flop

ttl:95

--

get abra

CSE473 DHTPv0.1

type:success

key:abra

tag:1

value:cadabra

ttl:98

get chocolate

CSE473 DHTPv0.1

type:success

key:chocolate

tag:1

value:fudge

ttl:95

--

get political

CSE473 DHTPv0.1

type:success

key:political

tag:1

value:follies

ttl:98

get fantasy

CSE473 DHTPv0.1

type:success

key:fantasy

tag:1

value:football

ttl:95

--

get flip

CSE473 DHTPv0.1

type:success

key:flip

tag:1

value:flop

ttl:98

get slim

CSE473 DHTPv0.1

type:success

key:slim

tag:1

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.

*After looking at the log files log1c\_2 and log1c\_3, we can tell that the packet came initially from the client to server number 2, then passed to server number 3, where the pair was stored in the cache. So the pair was returned from server number 3 to the relay server (server 2), and from there back to the 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?

*As we’ve seen in the question above and the answer to the 3 routes case, with caching the result came back to the client sooner this time; this is because the pair requested was saved in a cache on an intermediate server (namely server number 3). Server number 3 responded to the request to the client, and thus decreased the number of servers that handled the request (increased the ttl of the packet overall), compared to the configuration that used no caching with 3 routes. This has improved the performance in the case of dealing with this request.*