CSE 473 – Introduction to Computer Networks

Lab 3 Report – 110 Points

Your name here: Richard Wu; Jiaming Qiu

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.

```
/**
 * 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() {
      // your code here
      // a random generator
      Random r = new Random();
      // send leave packet to successor
      Packet p = new Packet();
      p.type = "leave";
      p.tag = r.nextInt(100000);
      p.senderInfo = myInfo;
      p.send(sock, succInfo.left, debug);
      // wait until stopFlag is set to "true"
      while(!stopFlag);
      // send update packet to pred and succ
      p.clear();
      p.type = "update";
      p.tag = r.nextInt(100000);
      p.succInfo = succInfo;
      p.hashRange = new Pair<Integer, Integer>(predInfo.right,hashRange.right);
      p.send(sock,predInfo.left,debug);
      p.clear();
      p.type = "update";
      p.tag = r.nextInt(100000);
      p.predInfo = predInfo;
      p.send(sock,succInfo.left,debug);
      // transfer all keys and empty data structures
      p.clear();
      p.type = "transfer";
      for (Map.Entry<String, String> entry : map.entrySet()) {
            p.tag = r.nextInt(100000);
            p.key = entry.getKey();
            p.val = entry.getValue();
```

```
            p.send(sock,predInfo.left,debug);
        }
        map.clear();
        cache.clear();
        rteTbl.clear();
}
/**
 * Join an existing DHT.
 *
 * @param predAdr is the socket address of a server in the DHT,
 *                  Send a join packet to the predecessor, and then wait
 *                  for reply. The reply should typically be success
 *                  packet which contains succInfo, predInfo, and
 *                  hashRange. All the information is set and the sucInfo
 *                  is added to the route table.
 */
public static void join(InetSocketAddress predAdr) {
        // your code here
        // send join packet
        Packet p = new Packet();
        p.tag = new Random().nextInt(100000);
        p.type = "join";
        p.send(sock,predAdr,debug);
        // wait for receiving information
        p.clear();
        p.receive(sock,debug);
        succInfo = p.succInfo;
        predInfo = p.predInfo;
        hashRange = p.hashRange;
        myInfo = new Pair<InetSocketAddress, Integer>(myAdr, hashRange.left);
        addRoute(succInfo);
}

/**
 * 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)
 *                  This function handles join request. It divides its hash
 *                  range by half and send the top half to the join server.
 *                  It also sends succInfo and predInfo to the server to
 *                  set up the links. It then sends update packet to its
 *                  original successor to update its predecessor. It
 *                  finally updates its own information and send transfer
 *                  packets to the new server.
 */
public static void handleJoin(Packet p, InetSocketAddress succAdr) {
        // your code here
        // send success packet to new server
        int left = hashRange.left.intValue();
        int right = hashRange.right.intValue();
        int mid = left + (right-left)/2;
        p.type = "success";
        p.hashRange = new Pair<Integer, Integer>(mid+1, right);
        p.succInfo = succInfo;
        p.predInfo = myInfo;
        p.senderInfo = myInfo;
        p.send(sock,succAdr,debug);
        // send update packet to original successor
        p.clear();
```

```
            p.type = "update";
            p.tag = new Random().nextInt(100000);
            p.senderInfo = myInfo;
            Pair<InetSocketAddress, Integer> joinInfo = new Pair<InetSocketAddress,
Integer>(succAdr, mid+1);
            p.predInfo = joinInfo;
            p.send(sock,succInfo.left,debug);
            // update some information
            succInfo = joinInfo;
            addRoute(succInfo);
            hashRange = new Pair<Integer, Integer>(left, mid);
            // send transfer packets to new server
            p.clear();
            p.type = "transfer";
            for(Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
it.hasNext(); ) {
                Map.Entry<String, String> entry = it.next();
                int hashValue = hashit(entry.getKey());
                if(mid+1 <= hashValue && hashValue <= right) {
                   p.key = entry.getKey();
                   p.val = entry.getValue();
                   p.tag = new Random().nextInt(100000);
                   p.send(sock,succAdr,debug);
                     it.remove();
                }
            }
      }

      /**
       * Handle a get packet.
       *
       * @param p         is a get packet
       * @param senderAdr is the socket address of the sender
       *                  This function handles get packet. If the hash is in its
       *                  range. It sends the information back either to the client
       *                  or to the relay server. If it's not in its range, it will
       *                  first look up the information in cache. If it cannot find
       *                  the entry, it will forward the request to another server.
       */
      public static void handleGet(Packet p, InetSocketAddress senderAdr) {
            // this version is incomplete; you will have to extend
            // it to support caching
            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;
                  }
                  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(cacheOn){
                // Iterate through cache
                for (Map.Entry<String, String> entry : cache.entrySet()) {
                    if(entry.getKey().equals(p.key)) {
                        if (p.relayAdr != null) {
                            replyAdr = p.relayAdr;
                            p.senderInfo = myInfo;
                        } else {
                            replyAdr = senderAdr;
                        }
                        p.type = "success";
                        p.val = entry.getValue();
                        p.send(sock, replyAdr, debug);
                        return;
                    }
                }
            }
            // 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
 *                  This function handles put packet. If the hash is in its
 *                  range, it handles the request and sends information back
 *                  either to the client or to the relay server. If it's
 *                  not in its range, it will first look up the information
 *                  in cache. If it finds the entry, it will delete it and
 *                  then forward request to another server.
 */
public static void handlePut(Packet p, InetSocketAddress senderAdr) {
        // your code here
        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;
            }
            if (p.val != null) {
                map.put(p.key, p.val); //put or update
                p.type = "success";
            } else {
                // remove instruction
                if(map.remove(p.key) != null) {
```

```
                        p.type = "success";
                    } else {
                        int ttl = p.ttl; // Use the original information
                        int tag = p.tag;
                        p.clear();
                        p.type = "failure";
                        p.reason = "no corresponding (key, value) pair";
                        p.tag = tag;
                        p.ttl = ttl;
                        p.senderInfo = myInfo;
                    }
                }
                p.send(sock, replyAdr, debug);
        } else {
                if(cacheOn){
                    // Iterate through cache
                    for (Map.Entry<String, String> entry : cache.entrySet()) {
                        if(entry.getKey().equals(p.key)) {
                            cache.remove(p.key);
                            break;
                        }
                    }
                }
                // 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
 *
 *                  This function handls a transfer packet.
 *                  It silently puts the entry into map and then
 *                  return.
 */
public static void handleXfer(Packet p, InetSocketAddress senderAdr) {
        // your code here
        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 handles success, failure, and no match
 *                  packet. It assumes it's the relay server and clears
 *                  all the server information and then send the packet
 *                  back to the client. If the packet is success and has
 *                  key/value, it will put the entry into cache.
 */
public static void handleReply(Packet p, InetSocketAddress senderAdr) {
```

```
            // your code here
            p.relayAdr = null;
            InetSocketAddress client = p.clientAdr;
            p.clientAdr = null;
            p.senderInfo = null;
            // Add into cache
            if(cacheOn && p.type.equals("success") && !cache.containsKey(p.key)
&& p.val != null) {
                  cache.put(p.key, p.val);
            }
            p.send(sock,client,debug);
      }

      /**
       * 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) {
            // your code here
            if (!rteTbl.contains(newRoute)) {
                  if (rteTbl.size() >= numRoutes) {
                        for (Iterator<Pair<InetSocketAddress, Integer>> it =
rteTbl.iterator(); it.hasNext();) {
                           if (!it.next().equals(succInfo)) {
                              it.remove();
                              break;
                           }
                        }
                  }
                  if(rteTbl.size() < numRoutes) {
                        rteTbl.add(newRoute);
                        if (debug) {
                              System.out.println("rteTbl=" + rteTbl + "\n");
                        }
                  }
            }
      }

      /**
       * Remove an entry from the route tabe.
       *
       * @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) {
            // your code here
            if (rteTbl.contains(rmRoute)) {
                  rteTbl.remove(rmRoute);
```

```
                if (debug) {
                        System.out.println("rteTbl=" + rteTbl + "\n");
                }
        }
    }

    /**
     * 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) {
        // your code here
        int minDiff = Integer.MAX_VALUE;
        InetSocketAddress fwdAdr = null;
        for (Iterator<Pair<InetSocketAddress, Integer>> it = rteTbl.iterator();
it.hasNext();) {
            Pair<InetSocketAddress, Integer> srvInfo = it.next();
            int firstHash = srvInfo.right.intValue();
            int diff = hash - firstHash;
            int mod = Math.floorMod(diff,Integer.MAX_VALUE);
          if (mod < minDiff) {
            minDiff = mod;
            fwdAdr = srvInfo.left;
          }
        }
        p.send(sock,fwdAdr,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.

```
/** 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() {

      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");
      }
      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 (val != null) {
      s.append("value:"); s.append(val); s.append("\n");
   }
   if (tag != -1) {
      s.append("tag:"); s.append(tag); s.append("\n");
   }
   if (ttl != -1) {
      s.append("ttl:"); s.append(ttl); s.append("\n");
   }
   if (reason != null) {
      s.append("reason:"); s.append(reason); s.append("\n");
   }
   return s.toString();
}

/** 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("relayAdr")) {
         chunks = right.split(":");
         if (chunks.length != 2) return false;
         relayAdr = 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("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 if (left.equals("hashRange")) {
            chunks = right.split(":");
            if (chunks.length != 2) return false;
            int leftR = Integer.parseInt(chunks[0]);
            int rightR = Integer.parseInt(chunks[1]);
            hashRange = new Pair<Integer,Integer>(leftR,rightR);
        } else if (left.equals("tag")) {
            tag = Integer.parseInt(right);
        } else if (left.equals("key")) {
            key = right;
        } else if (left.equals("value")) {
            val = right;
        } else if (left.equals("reason")) {
            reason = right;
        } else {
            // ignore lines that don't match defined field
        }
    }
    return true;
}
```

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

```
import java.io.BufferedReader;
import java.net.*;
import java.io.*;
import java.util.*;
/*
 * DhtClient.java
 * This is the DhtClient. This program will take from 4 or 5
 * command line arguments. The first is the IP address of the
 * socket that the client will bind to. The second is the
 * name of a configuration file containing the IP address and
 * port number of one of the DhtServers. The third is an
 * operation like "get" or "put" and the remaining arguments
 * specify the key and/or value for the operation.
 */
public class DhtClient {

   public static void main(String args[]) throws Exception {
      // Check the argument number
      if (args.length != 4 && args.length != 5) {
         System.err.println("usage: DhtClient myIp " +
                  "serverFile put/get [ key ] [ value ] ");
         System.exit(1);
      }
      InetAddress myIp = null;
      DatagramSocket sock = null;
      InetSocketAddress server = null;
      // Read arguments
      try {
         myIp = InetAddress.getByName(args[0]);
         sock = new DatagramSocket(0,myIp);
         BufferedReader serv =
               new BufferedReader(
                   new InputStreamReader(
                  new FileInputStream(args[1]),
                  "US-ASCII"));
         String s = serv.readLine();
         serv.close();
         String[] chunks = s.split(" ");
         server = new InetSocketAddress(
               chunks[0],Integer.parseInt(chunks[1]));
      } catch(Exception e) {
         System.err.println("usage: DhtClient myIp " +
                  "cfgFile put/get [ key ] [ value ] ");
         System.exit(1);
      }
      // Construct the packet
      Packet request = new Packet();
      if(args[2].equals("get")) {
         request.type = "get";
      }else if(args[2].equals("put")) {
```

```
            request.type = "put";
            if(args.length == 5)
               request.val = args[4];
         }else {
            System.err.println("usage: DhtClient myIp " +
                     "cfgFile put/get [ key ] [ value ] ");
            System.exit(1);
         }
         request.key = args[3];
         request.tag = new Random().nextInt(100000);
         // Send the request
         request.send(sock,server,true);
         // Receive the reply and quit
         Packet reply = new Packet();
         if(reply.receive(sock,true) == null) {
            System.err.println("received packet failure");
            System.exit(2);
         }
         return;
      }
   }
```

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
/127.0.0.1:50068 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:put
key:foo
value:bar
tag:44820
ttl:100

/127.0.0.1:50068 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:44820
ttl:98

put who hah
/127.0.0.1:60735 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:put
key:who
value:hah
tag:82765
ttl:100

/127.0.0.1:60735 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:82765
ttl:98

get foo
/127.0.0.1:61313 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:get
```

```
key:foo
tag:74119
ttl:100

/127.0.0.1:61313 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:74119
ttl:98

get who
/127.0.0.1:57229 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:get
key:who
tag:65831
ttl:100

/127.0.0.1:57229 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:65831
ttl:98

get goodbye
/127.0.0.1:51255 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:get
key:goodbye
tag:59906
ttl:100

/127.0.0.1:51255 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:no match
key:goodbye
tag:59906
ttl:98

get
get bar
/127.0.0.1:53481 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:get
key:bar
tag:21377
ttl:100

/127.0.0.1:53481 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:no match
key:bar
tag:21377
ttl:98

put foo toast is tasty
/127.0.0.1:64155 sending packet to /127.0.0.1:56460
```

```
CSE473 DHTPv0.1
type:put
key:foo
value:toast is tasty
tag:15219
ttl:100

/127.0.0.1:64155 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:success
key:foo
value:toast is tasty
tag:15219
ttl:98

get foo
/127.0.0.1:61947 sending packet to /127.0.0.1:56460
CSE473 DHTPv0.1
type:get
key:foo
tag:82305
ttl:100

/127.0.0.1:61947 received packet from /127.0.0.1:56460
CSE473 DHTPv0.1
type:success
key:foo
value:toast is tasty
tag:82305
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.**

Initial portion:

```
put foo bar
/127.0.0.1:49753 sending packet to /127.0.0.1:65289
CSE473 DHTPv0.1
type:put
key:foo
value:bar
tag:10069
ttl:100

/127.0.0.1:49753 received packet from /127.0.0.1:65289
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:10069
ttl:98

put who hah
/127.0.0.1:57040 sending packet to /127.0.0.1:57086
CSE473 DHTPv0.1
type:put
key:who
value:hah
tag:76198
ttl:100

/127.0.0.1:57040 received packet from /127.0.0.1:57086
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:76198
ttl:94

put junk mail
/127.0.0.1:54349 sending packet to /127.0.0.1:49337
```

```
CSE473 DHTPv0.1
type:put
key:junk
value:mail
tag:54408
ttl:100

/127.0.0.1:54349 received packet from /127.0.0.1:49337
CSE473 DHTPv0.1
type:success
key:junk
value:mail
tag:54408
ttl:95

put blue moose
/127.0.0.1:64021 sending packet to /127.0.0.1:58831
CSE473 DHTPv0.1
type:put
key:blue
value:moose
tag:57985
ttl:100

/127.0.0.1:64021 received packet from /127.0.0.1:58831
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:57985
ttl:98

get foo
/127.0.0.1:65236 sending packet to /127.0.0.1:49337
CSE473 DHTPv0.1
type:get
key:foo
tag:64235
ttl:100

/127.0.0.1:65236 received packet from /127.0.0.1:49337
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:64235
ttl:96

get who
/127.0.0.1:58745 sending packet to /127.0.0.1:58831
CSE473 DHTPv0.1
type:get
key:who
tag:88968
```

```
ttl:100

/127.0.0.1:58745 received packet from /127.0.0.1:58831
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:88968
ttl:95
```

Last four operations:

```
get blue
/127.0.0.1:64312 sending packet to /127.0.0.1:65289
CSE473 DHTPv0.1
type:get
key:blue
tag:69583
ttl:100

/127.0.0.1:64312 received packet from /127.0.0.1:65289
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:69583
ttl:95

get blue
/127.0.0.1:57775 sending packet to /127.0.0.1:65289
CSE473 DHTPv0.1
type:get
key:blue
tag:44707
ttl:100

/127.0.0.1:57775 received packet from /127.0.0.1:65289
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:44707
ttl:96

get foo
/127.0.0.1:49621 sending packet to /127.0.0.1:49337
CSE473 DHTPv0.1
type:get
key:foo
tag:69312
ttl:100

/127.0.0.1:49621 received packet from /127.0.0.1:49337
```

```
CSE473 DHTPv0.1
type:success
key:foo
value:toast is tasty
tag:69312
ttl:96

get junk
/127.0.0.1:50154 sending packet to /127.0.0.1:57086
CSE473 DHTPv0.1
type:get
key:junk
tag:52526
ttl:100

/127.0.0.1:50154 received packet from /127.0.0.1:57086
CSE473 DHTPv0.1
type:success
key:junk
value:mail
tag:52526
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?

58831.

For `put blue moose`, *the ttl of the packet returned to the client is 98. So we can conclude that the entry is just stored in the server which we send data to.*

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.*

server 0 -> server 1 -> server 2 -> server 3 ->server 0 (a ring)
65289->57086->58831->49337->65289

When "get blue" is sent to 49337, ttl received by the client is 94,
because the packet flows through client->server 3->server 0->server 1->server 2->server 3->client
When "get blue" is sent to 57086, ttl received by the client is 96,
because the packet flows through client->server 1->server 2->server 1->client
When "get blue" is sent to 58831, ttl received by the client is 98,
because the packet flows through client->server 2->client
When "get blue" is sent to 65289, ttl received by the client is 95,
because the packet flows through client->server 0->server 1->server 2->server 0->client

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

Because for the second "get blue" operation, the server 2 (with port number 58831) is removed from the DHT, and this server is the one who stores "blue". When it's removed, its data will be transferred to server 1. When a client wants to "get blue" by sending the request to server 0 (with port number 65289). Before it's removed, the ttl is 95 (client->server 0->server 1->server 2->server 0->client); after it's removed, the ttl is 96 (client->server 0->server 1->server 0->client). The result makes perfect sense.

Paste the initial portion of the *log1_2* file below (everything up through the first "*get blue*" operation and response).

```
/127.0.0.1:58831 sending packet to /127.0.0.1:65289
CSE473 DHTPv0.1
type:join
tag:9994
ttl:100

/127.0.0.1:58831 received packet from /127.0.0.1:65289
CSE473 DHTPv0.1
type:success
hashRange:1073741824:2147483647
senderInfo:127.0.0.1:65289:0
succInfo:127.0.0.1:65289:0
predInfo:127.0.0.1:65289:0
tag:9994
ttl:98

rteTbl=[(/127.0.0.1:65289,0)]

/127.0.0.1:58831 received packet from /127.0.0.1:65289
CSE473 DHTPv0.1
type:update
senderInfo:127.0.0.1:65289:0
predInfo:127.0.0.1:57086:536870912
tag:69884
ttl:99

/127.0.0.1:58831 received packet from /127.0.0.1:49337
CSE473 DHTPv0.1
type:join
tag:43546
ttl:99

/127.0.0.1:58831 sending packet to /127.0.0.1:49337
CSE473 DHTPv0.1
type:success
hashRange:1610612736:2147483647
senderInfo:127.0.0.1:58831:1073741824
succInfo:127.0.0.1:65289:0
predInfo:127.0.0.1:58831:1073741824
tag:43546
ttl:99

/127.0.0.1:58831 sending packet to /127.0.0.1:65289
CSE473 DHTPv0.1
type:update
senderInfo:127.0.0.1:58831:1073741824
predInfo:127.0.0.1:49337:1610612736
tag:10015
ttl:100

rteTbl=[(/127.0.0.1:49337,1610612736)]
```

```
/127.0.0.1:58831 received packet from /127.0.0.1:57086
CSE473 DHTPv0.1
type:put
key:who
relayAdr:127.0.0.1:57086
clientAdr:127.0.0.1:57040
value:hah
tag:76198
ttl:98

/127.0.0.1:58831 sending packet to /127.0.0.1:49337
CSE473 DHTPv0.1
type:put
key:who
relayAdr:127.0.0.1:57086
clientAdr:127.0.0.1:57040
value:hah
tag:76198
ttl:98

/127.0.0.1:58831 received packet from /127.0.0.1:64021
CSE473 DHTPv0.1
type:put
key:blue
value:moose
tag:57985
ttl:99

/127.0.0.1:58831 sending packet to /127.0.0.1:64021
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:57985
ttl:99

/127.0.0.1:58831 received packet from /127.0.0.1:58745
CSE473 DHTPv0.1
type:get
key:who
tag:88968
ttl:99

/127.0.0.1:58831 sending packet to /127.0.0.1:49337
CSE473 DHTPv0.1
type:get
key:who
relayAdr:127.0.0.1:58831
clientAdr:127.0.0.1:58745
tag:88968
ttl:99

/127.0.0.1:58831 received packet from /127.0.0.1:65289
```

```
CSE473 DHTPv0.1
type:success
key:who
relayAdr:127.0.0.1:58831
clientAdr:127.0.0.1:58745
senderInfo:127.0.0.1:65289:0
value:hah
tag:88968
ttl:96

/127.0.0.1:58831 sending packet to /127.0.0.1:58745
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:88968
ttl:96

/127.0.0.1:58831 received packet from /127.0.0.1:57086
CSE473 DHTPv0.1
type:get
key:blue
relayAdr:127.0.0.1:57086
clientAdr:127.0.0.1:56150
tag:60511
ttl:98

/127.0.0.1:58831 sending packet to /127.0.0.1:57086
CSE473 DHTPv0.1
type:success
key:blue
relayAdr:127.0.0.1:57086
clientAdr:127.0.0.1:56150
senderInfo:127.0.0.1:58831:1073741824
value:moose
tag:60511
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, its hash range is from 536870912 to 1073741823, which has 536870912 values. After the last server has joined the DHT, server 1's hash range remains unchanged. (i.e. still from 536870912 to 1073741823). When server 2 leaves the DHT, all the hash range of server 2 is transferred to server 1. Therefore, server 1's hash range then becomes from 536870912 to 1610612735, which has 1073741824 values.

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 65289
127.0.0.1 57086
127.0.0.1 58831
127.0.0.1 49337
```

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?

key: bar

The returned result has type "no match"

The returned packet's ttl is 94.

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

client->server 3->server 0->server 1->server 2->server 3->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**.

Initial portion:

```
put foo bar
/127.0.0.1:34787 sending packet to /127.0.0.1:41116
CSE473 DHTPv0.1
type:put
key:foo
value:bar
tag:64371
ttl:100

/127.0.0.1:34787 received packet from /127.0.0.1:41116
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:64371
ttl:98

put who hah
/127.0.0.1:39114 sending packet to /127.0.0.1:35715
CSE473 DHTPv0.1
type:put
key:who
value:hah
tag:40728
ttl:100

/127.0.0.1:39114 received packet from /127.0.0.1:35715
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:40728
ttl:95

put junk mail
/127.0.0.1:44640 sending packet to /127.0.0.1:42346
CSE473 DHTPv0.1
type:put
key:junk
value:mail
tag:17527
ttl:100

/127.0.0.1:44640 received packet from /127.0.0.1:42346
```

```
CSE473 DHTPv0.1
type:success
key:junk
value:mail
tag:17527
ttl:95

put blue moose
/127.0.0.1:33427 sending packet to /127.0.0.1:42711
CSE473 DHTPv0.1
type:put
key:blue
value:moose
tag:65176
ttl:100

/127.0.0.1:33427 received packet from /127.0.0.1:42711
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:65176
ttl:98

get foo
/127.0.0.1:49596 sending packet to /127.0.0.1:42346
CSE473 DHTPv0.1
type:get
key:foo
tag:97547
ttl:100

/127.0.0.1:49596 received packet from /127.0.0.1:42346
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:97547
ttl:96

get who
/127.0.0.1:36524 sending packet to /127.0.0.1:42711
CSE473 DHTPv0.1
type:get
key:who
tag:15386
ttl:100

/127.0.0.1:36524 received packet from /127.0.0.1:42711
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:15386
```

```
ttl:96
```

Last four operations:

```
get blue
/127.0.0.1:51972 sending packet to /127.0.0.1:41116
CSE473 DHTPv0.1
type:get
key:blue
tag:2775
ttl:100

/127.0.0.1:51972 received packet from /127.0.0.1:41116
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:2775
ttl:96

get blue
/127.0.0.1:47240 sending packet to /127.0.0.1:41116
CSE473 DHTPv0.1
type:get
key:blue
tag:34796
ttl:100

/127.0.0.1:47240 received packet from /127.0.0.1:41116
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:34796
ttl:96

get foo
/127.0.0.1:37840 sending packet to /127.0.0.1:42346
CSE473 DHTPv0.1
type:get
key:foo
tag:99892
ttl:100

/127.0.0.1:37840 received packet from /127.0.0.1:42346
CSE473 DHTPv0.1
type:success
key:foo
value:toast is tasty
tag:99892
ttl:96

get junk
/127.0.0.1:32967 sending packet to /127.0.0.1:35715
```

```
CSE473 DHTPv0.1
type:get
key:junk
tag:80060
ttl:100

/127.0.0.1:32967 received packet from /127.0.0.1:35715
CSE473 DHTPv0.1
type:success
key:junk
value:mail
tag:80060
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: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 41116
127.0.0.1 35715
127.0.0.1 42711
127.0.0.1 42346
```

Type the command "`grep rteTbl log2_[0-3]`" and paste the output below.

```
log2_0:rteTbl=[(/127.0.0.1:42711,1073741824)]
log2_0:rteTbl=[(/127.0.0.1:42711,1073741824), (/127.0.0.1:41116,0)]
log2_0:rteTbl=[(/127.0.0.1:41116,0), (/127.0.0.1:35715,536870912)]
log2_0:rteTbl=[(/127.0.0.1:35715,536870912), (/127.0.0.1:42711,1073741824)]
log2_0:rteTbl=[(/127.0.0.1:35715,536870912)]
log2_1:rteTbl=[(/127.0.0.1:42711,1073741824)]
log2_1:rteTbl=[(/127.0.0.1:42711,1073741824), (/127.0.0.1:41116,0)]
log2_1:rteTbl=[(/127.0.0.1:41116,0)]
log2_1:rteTbl=[(/127.0.0.1:41116,0), (/127.0.0.1:42346,1610612736)]
log2_2:rteTbl=[(/127.0.0.1:41116,0)]
log2_2:rteTbl=[(/127.0.0.1:41116,0), (/127.0.0.1:42346,1610612736)]
log2_3:rteTbl=[(/127.0.0.1:41116,0)]
log2_3:rteTbl=[(/127.0.0.1:41116,0), (/127.0.0.1:35715,536870912)]
log2_3:rteTbl=[(/127.0.0.1:41116,0), (/127.0.0.1:42711,1073741824)]
log2_3:rteTbl=[(/127.0.0.1:41116,0)]
```

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.

server 0: [server 1]

server 1: [server 0, server 3]

server 3: [server 0]

Type the command "`grep -B4 -A4 key:bar log2_[0-3]`" and paste the output below.

```
log2_1-
log2_1-/127.0.0.1:35715 received packet from /127.0.0.1:42346
log2_1-CSE473 DHTPv0.1
log2_1-type:get
log2_1:key:bar
log2_1-relayAdr:127.0.0.1:42346
log2_1-clientAdr:127.0.0.1:45829
log2_1-tag:70967
log2_1-ttl:98
log2_1-
log2_1-/127.0.0.1:35715 sending packet to /127.0.0.1:42711
log2_1-CSE473 DHTPv0.1
log2_1-type:get
log2_1:key:bar
log2_1-relayAdr:127.0.0.1:42346
log2_1-clientAdr:127.0.0.1:45829
log2_1-tag:70967
log2_1-ttl:98
--
log2_2-
log2_2-/127.0.0.1:42711 received packet from /127.0.0.1:35715
log2_2-CSE473 DHTPv0.1
log2_2-type:get
log2_2:key:bar
log2_2-relayAdr:127.0.0.1:42346
log2_2-clientAdr:127.0.0.1:45829
log2_2-tag:70967
log2_2-ttl:97
log2_2-
log2_2-/127.0.0.1:42711 sending packet to /127.0.0.1:42346
log2_2-CSE473 DHTPv0.1
log2_2-type:no match
log2_2:key:bar
log2_2-relayAdr:127.0.0.1:42346
log2_2-clientAdr:127.0.0.1:45829
log2_2-senderInfo:127.0.0.1:42711:1073741824
log2_2-tag:70967
--
log2_3-
```

```
log2_3-/127.0.0.1:42346 received packet from /127.0.0.1:45829
log2_3-CSE473 DHTPv0.1
log2_3-type:get
log2_3:key:bar
log2_3-tag:70967
log2_3-ttl:99
log2_3-
log2_3-/127.0.0.1:42346 sending packet to /127.0.0.1:35715
log2_3-CSE473 DHTPv0.1
log2_3-type:get
log2_3:key:bar
log2_3-relayAdr:127.0.0.1:42346
log2_3-clientAdr:127.0.0.1:45829
log2_3-tag:70967
log2_3-ttl:99
log2_3-
log2_3-/127.0.0.1:42346 received packet from /127.0.0.1:42711
log2_3-CSE473 DHTPv0.1
log2_3-type:no match
log2_3:key:bar
log2_3-relayAdr:127.0.0.1:42346
log2_3-clientAdr:127.0.0.1:45829
log2_3-senderInfo:127.0.0.1:42711:1073741824
log2_3-tag:70967
--
log2_3-
log2_3-/127.0.0.1:42346 sending packet to /127.0.0.1:45829
log2_3-CSE473 DHTPv0.1
log2_3-type:no match
log2_3:key:bar
log2_3-tag:70967
log2_3-ttl:96
log2_3-
log2_3-/127.0.0.1:42346 received packet from /127.0.0.1:38343
```

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.

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
/127.0.0.1:50913 sending packet to /127.0.0.1:33250
CSE473 DHTPv0.1
type:get
key:foo
tag:15622
ttl:100

/127.0.0.1:50913 received packet from /127.0.0.1:33250
CSE473 DHTPv0.1
type:success
key:foo
value:toast is tasty
tag:15622
ttl:98

get blue
/127.0.0.1:33207 sending packet to /127.0.0.1:57447
CSE473 DHTPv0.1
type:get
key:blue
tag:63323
ttl:100

/127.0.0.1:33207 received packet from /127.0.0.1:57447
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:63323
ttl:96

get blue
/127.0.0.1:38732 sending packet to /127.0.0.1:45728
CSE473 DHTPv0.1
type:get
key:blue
tag:30954
```

```
ttl:100

/127.0.0.1:38732 received packet from /127.0.0.1:45728
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:30954
ttl:98

get blue
/127.0.0.1:42132 sending packet to /127.0.0.1:35592
CSE473 DHTPv0.1
type:get
key:blue
tag:3837
ttl:100

/127.0.0.1:42132 received packet from /127.0.0.1:35592
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:3837
ttl:98

get blue
/127.0.0.1:53844 sending packet to /127.0.0.1:33250
CSE473 DHTPv0.1
type:get
key:blue
tag:99347
ttl:100

/127.0.0.1:53844 received packet from /127.0.0.1:33250
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:99347
ttl:96

get blue
/127.0.0.1:43942 sending packet to /127.0.0.1:57447
CSE473 DHTPv0.1
type:get
key:blue
tag:9942
ttl:100

/127.0.0.1:43942 received packet from /127.0.0.1:57447
CSE473 DHTPv0.1
type:success
key:blue
```

```
value:moose
tag:9942
ttl:98

get blue
/127.0.0.1:56742 sending packet to /127.0.0.1:45728
CSE473 DHTPv0.1
type:get
key:blue
tag:66208
ttl:100

/127.0.0.1:56742 received packet from /127.0.0.1:45728
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:66208
ttl:98

get blue
/127.0.0.1:58449 sending packet to /127.0.0.1:35592
CSE473 DHTPv0.1
type:get
key:blue
tag:61197
ttl:100

/127.0.0.1:58449 received packet from /127.0.0.1:35592
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:61197
ttl:98

get blue
/127.0.0.1:37939 sending packet to /127.0.0.1:33250
CSE473 DHTPv0.1
type:get
key:blue
tag:25560
ttl:100

/127.0.0.1:37939 received packet from /127.0.0.1:33250
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:25560
ttl:98

get blue
/127.0.0.1:55870 sending packet to /127.0.0.1:33250
```

```
CSE473 DHTPv0.1
type:get
key:blue
tag:85702
ttl:100

/127.0.0.1:55870 received packet from /127.0.0.1:33250
CSE473 DHTPv0.1
type:success
key:blue
value:moose
tag:85702
ttl:98

get foo
/127.0.0.1:57670 sending packet to /127.0.0.1:57447
CSE473 DHTPv0.1
type:get
key:foo
tag:19118
ttl:100

/127.0.0.1:57670 received packet from /127.0.0.1:57447
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:19118
ttl:98

get junk
/127.0.0.1:58081 sending packet to /127.0.0.1:45728
CSE473 DHTPv0.1
type:get
key:junk
tag:80601
ttl:100

/127.0.0.1:58081 received packet from /127.0.0.1:45728
CSE473 DHTPv0.1
type:success
key:junk
value:mail
tag:80601
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: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?

No, every server has a copy of (blue, moose) in its cache. Because the last four "get blue" requests before server 2 start to leave are forwarded to the four severs respectively, and they all come back successfully with ttl=98, indicating that all the four servers have access to this record and can return the result directly.

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 39152
192.168.6.1 59222
192.168.3.2 38761
192.168.2.5 40439
192.168.2.4 57500
192.168.2.3 46752
192.168.1.1 54645
192.168.5.2 58745
```

Now, type "`grep rteTbl log1_[0-7]`" and paste the output below.

```
log1_0:rteTbl=[(/192.168.2.4:57500,1073741824)]
log1_0:rteTbl=[(/192.168.3.2:38761,536870912)]
log1_0:rteTbl=[(/192.168.6.1:59222,268435456)]
log1_1:rteTbl=[(/192.168.3.2:38761,536870912)]
log1_2:rteTbl=[(/192.168.2.4:57500,1073741824)]
log1_2:rteTbl=[(/192.168.2.5:40439,805306368)]
log1_3:rteTbl=[(/192.168.2.4:57500,1073741824)]
log1_4:rteTbl=[(/192.168.7.1:39152,0)]
log1_4:rteTbl=[(/192.168.1.1:54645,1610612736)]
log1_4:rteTbl=[(/192.168.2.3:46752,1342177280)]
log1_5:rteTbl=[(/192.168.1.1:54645,1610612736)]
log1_6:rteTbl=[(/192.168.7.1:39152,0)]
log1_6:rteTbl=[(/192.168.5.2:58745,1879048192)]
log1_7:rteTbl=[(/192.168.7.1:39152,0)]
```

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.

Yes, they are consistent. Because in this case, we set numRoutes=1. Therefore each server is only allowed to store the information of its successor in the route table. After all the servers join the DHT, route table will never be modified anymore unless some server leaves. So the final route values must be consistent with the configuration file that describes the sequence of servers in the ring.

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?

No. The largest number is 7 (because the smallest ttl value is 91). If a request is handled by three or more servers, the ttl value will be at most 95. Therefore, there are 20 such requests.

Type "`grep –B15 ttl.91 out1`" and paste the output below.

```
get slim
/192.168.4.2:38535 sending packet to /192.168.1.1:54645
CSE473 DHTPv0.1
type:get
key:slim
tag:95657
ttl:100

/192.168.4.2:38535 received packet from /192.168.1.1:54645
CSE473 DHTPv0.1
type:success
key:slim
value:jim
tag:95657
ttl:91
```

Type the command "`grep -B3 -A4 transfer log1_0`" and paste the output below.

```
/192.168.7.1:39152 sending packet to /192.168.6.1:59222
CSE473 DHTPv0.1
type:transfer
key:flip
value:flop
tag:43989
ttl:100

/192.168.7.1:39152 sending packet to /192.168.6.1:59222
CSE473 DHTPv0.1
type:transfer
key:who
value:hah
tag:48710
ttl:100
```

Explain the output.

When server 1 joined the DHT, server 0 split the "top" half of its hash range to server 1. There are 2 records whose key falls in that range. Therefore, server 0 sent two consecutive "transfer" packets to server 1, let server 1 store these records, and remove them from its own hash table.

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.

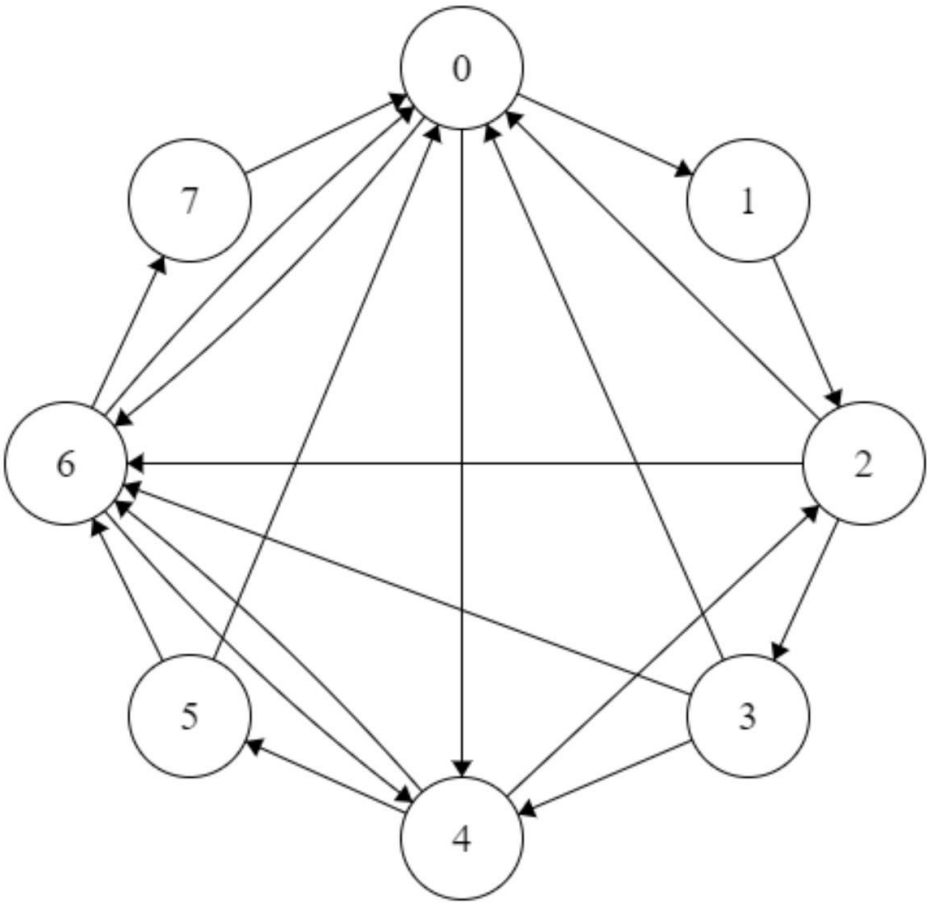
```
192.168.7.1 45084
192.168.6.1 38434
192.168.3.2 40444
192.168.2.5 50267
192.168.2.4 35801
192.168.2.3 41248
192.168.1.1 56794
192.168.5.2 41361
```

Now, type "`grep rteTbl log3_[0-7]`" and paste the output below.

```
log3_0:rteTbl=[(/192.168.2.4:35801,1073741824)]
log3_0:rteTbl=[(/192.168.2.4:35801,1073741824), (/192.168.7.1:45084,0)]
log3_0:rteTbl=[(/192.168.2.4:35801,1073741824), (/192.168.7.1:45084,0),
(/192.168.3.2:40444,536870912)]
log3_0:rteTbl=[(/192.168.7.1:45084,0), (/192.168.3.2:40444,536870912),
(/192.168.1.1:56794,1610612736)]
log3_0:rteTbl=[(/192.168.3.2:40444,536870912),
(/192.168.1.1:56794,1610612736), (/192.168.2.4:35801,1073741824)]
log3_0:rteTbl=[(/192.168.1.1:56794,1610612736),
(/192.168.2.4:35801,1073741824), (/192.168.6.1:38434,268435456)]
log3_1:rteTbl=[(/192.168.3.2:40444,536870912)]
log3_2:rteTbl=[(/192.168.2.4:35801,1073741824)]
log3_2:rteTbl=[(/192.168.2.4:35801,1073741824),
(/192.168.2.5:50267,805306368)]
log3_2:rteTbl=[(/192.168.2.4:35801,1073741824),
(/192.168.2.5:50267,805306368), (/192.168.7.1:45084,0)]
log3_2:rteTbl=[(/192.168.2.5:50267,805306368), (/192.168.7.1:45084,0),
(/192.168.5.2:41361,1879048192)]
log3_2:rteTbl=[(/192.168.2.5:50267,805306368),
(/192.168.5.2:41361,1879048192), (/192.168.2.4:35801,1073741824)]
log3_2:rteTbl=[(/192.168.2.5:50267,805306368),
(/192.168.2.4:35801,1073741824), (/192.168.7.1:45084,0)]
log3_2:rteTbl=[(/192.168.2.5:50267,805306368), (/192.168.7.1:45084,0),
(/192.168.1.1:56794,1610612736)]
log3_3:rteTbl=[(/192.168.2.4:35801,1073741824)]
log3_3:rteTbl=[(/192.168.2.4:35801,1073741824), (/192.168.7.1:45084,0)]
log3_3:rteTbl=[(/192.168.2.4:35801,1073741824), (/192.168.7.1:45084,0),
(/192.168.1.1:56794,1610612736)]
log3_4:rteTbl=[(/192.168.7.1:45084,0)]
log3_4:rteTbl=[(/192.168.7.1:45084,0), (/192.168.3.2:40444,536870912)]
log3_4:rteTbl=[(/192.168.7.1:45084,0), (/192.168.3.2:40444,536870912),
(/192.168.1.1:56794,1610612736)]
```

```
log3_4:rteTbl=[(/192.168.3.2:40444,536870912),
(/192.168.1.1:56794,1610612736), (/192.168.2.3:41248,1342177280)]
log3_5:rteTbl=[(/192.168.1.1:56794,1610612736)]
log3_5:rteTbl=[(/192.168.1.1:56794,1610612736), (/192.168.7.1:45084,0)]
log3_6:rteTbl=[(/192.168.7.1:45084,0)]
log3_6:rteTbl=[(/192.168.7.1:45084,0), (/192.168.5.2:41361,1879048192)]
log3_6:rteTbl=[(/192.168.7.1:45084,0), (/192.168.5.2:41361,1879048192),
(/192.168.2.4:35801,1073741824)]
log3_7:rteTbl=[(/192.168.7.1:45084,0)]
```

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.

It is mainly because server 1 and server 5 did not join the DHT at the beginning. Before these two servers join the DHT, their two predecessors, server 0 and server 4, had to take more responsibility to respond to requests because of their larger hash range. Additionally, the two successors, server 2 and server 6, had more opportunities to be recorded as a successor. Due to such inequality, some servers like server 0 and server 6 have more "incoming routes" after a small number of requests (the number is not big enough to offset the influence of such "inequality").

Next, type "`grep ttl.9 out3`" and paste the output below.

```
ttl:98
ttl:95
ttl:96
ttl:95
ttl:96
ttl:95
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 largest number of servers to handle a single request is 4 (the smallest ttl is 94). 8 requests were handled by three or more servers. We can see that both numbers become much smaller. This is because we are allowing each server to keep a larger number of routes. Requests have a bigger chance to be forwarded to some servers near the one responsible for the given key.

Type "`grep -B15 ttl.95 out3`" and paste the output below.

```
put who hah
/192.168.4.2:44431 sending packet to /192.168.3.2:40444
CSE473 DHTPv0.1
type:put
key:who
value:hah
tag:97418
ttl:100

/192.168.4.2:44431 received packet from /192.168.3.2:40444
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:97418
ttl:95
--

get who
/192.168.4.2:53227 sending packet to /192.168.2.5:50267
CSE473 DHTPv0.1
type:get
key:who
tag:92055
ttl:100

/192.168.4.2:53227 received packet from /192.168.2.5:50267
CSE473 DHTPv0.1
type:success
key:who
value:hah
tag:92055
ttl:95
--

get
get bar
/192.168.4.2:51732 sending packet to /192.168.7.1:45084
```

```
CSE473 DHTPv0.1
type:get
key:bar
tag:98983
ttl:100

/192.168.4.2:51732 received packet from /192.168.7.1:45084
CSE473 DHTPv0.1
type:no match
key:bar
tag:98983
ttl:95
--
put political follies
/192.168.4.2:57983 sending packet to /192.168.3.2:40444
CSE473 DHTPv0.1
type:put
key:political
value:follies
tag:24812
ttl:100

/192.168.4.2:57983 received packet from /192.168.3.2:40444
CSE473 DHTPv0.1
type:success
key:political
value:follies
tag:24812
ttl:95
--

get fantasy
/192.168.4.2:49189 sending packet to /192.168.2.5:50267
CSE473 DHTPv0.1
type:get
key:fantasy
tag:46821
ttl:100

/192.168.4.2:49189 received packet from /192.168.2.5:50267
CSE473 DHTPv0.1
type:success
key:fantasy
value:football
tag:46821
ttl:95
--

get chocolate
/192.168.4.2:37719 sending packet to /192.168.2.3:41248
CSE473 DHTPv0.1
type:get
key:chocolate
tag:48017
```

```
ttl:100

/192.168.4.2:37719 received packet from /192.168.2.3:41248
CSE473 DHTPv0.1
type:success
key:chocolate
value:fudge
tag:48017
ttl:95
--

get fantasy
/192.168.4.2:48726 sending packet to /192.168.3.2:40444
CSE473 DHTPv0.1
type:get
key:fantasy
tag:27670
ttl:100

/192.168.4.2:48726 received packet from /192.168.3.2:40444
CSE473 DHTPv0.1
type:success
key:fantasy
value:football
tag:27670
ttl:95
```

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

server 2

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

client->server 2->server 4->server 6->server 2->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. In the route diagram, server 2 does not have a route to server 4, but instead it has a route directly to server 6. This is because after server 6 found the value to that key and responded to server 2, server 2 removed server 4 from its route table and added server 6 instead.

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 a single request is 6 (the smallest ttl is 92). 14 requests 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.

Both numbers are smaller than the first case (no cache, 1 route), but larger than the second case (no cache, 3 routes). We get the first result because servers are now allowed to cache records before sending the response back to clients. Therefore they can sometimes directly respond to clients by checking cache. We get the second result because the script does not send too many duplicate requests. So more routes turns out to be better than caching in this particular case.

Type "`grep -B15 ttl.95 out1c`" and paste the output below.

```
get foo
/192.168.4.2:52060 sending packet to /192.168.1.1:35755
CSE473 DHTPv0.1
type:get
key:foo
tag:52958
ttl:100

/192.168.4.2:52060 received packet from /192.168.1.1:35755
CSE473 DHTPv0.1
type:success
key:foo
value:bar
tag:52958
ttl:95
--
put flim flam
/192.168.4.2:46668 sending packet to /192.168.3.2:57105
CSE473 DHTPv0.1
type:put
key:flim
value:flam
tag:78409
ttl:100

/192.168.4.2:46668 received packet from /192.168.3.2:57105
CSE473 DHTPv0.1
type:success
key:flim
value:flam
tag:78409
```

```
ttl:95
--
put political follies
/192.168.4.2:52173 sending packet to /192.168.3.2:57105
CSE473 DHTPv0.1
type:put
key:political
value:follies
tag:88780
ttl:100

/192.168.4.2:52173 received packet from /192.168.3.2:57105
CSE473 DHTPv0.1
type:success
key:political
value:follies
tag:88780
ttl:95
--

get flip
/192.168.4.2:52979 sending packet to /192.168.2.4:51189
CSE473 DHTPv0.1
type:get
key:flip
tag:51088
ttl:100

/192.168.4.2:52979 received packet from /192.168.2.4:51189
CSE473 DHTPv0.1
type:success
key:flip
value:flop
tag:51088
ttl:95
--

get chocolate
/192.168.4.2:50873 sending packet to /192.168.1.1:35755
CSE473 DHTPv0.1
type:get
key:chocolate
tag:96356
ttl:100

/192.168.4.2:50873 received packet from /192.168.1.1:35755
CSE473 DHTPv0.1
type:success
key:chocolate
value:fudge
tag:96356
ttl:95
--
```

```
get fantasy
/192.168.4.2:37155 sending packet to /192.168.2.5:58569
CSE473 DHTPv0.1
type:get
key:fantasy
tag:25275
ttl:100

/192.168.4.2:37155 received packet from /192.168.2.5:58569
CSE473 DHTPv0.1
type:success
key:fantasy
value:football
tag:25275
ttl:95
--

get slim
/192.168.4.2:51817 sending packet to /192.168.1.1:35755
CSE473 DHTPv0.1
type:get
key:slim
tag:78881
ttl:100

/192.168.4.2:51817 received packet from /192.168.1.1:35755
CSE473 DHTPv0.1
type:success
key:slim
value:jim
tag:78881
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.

client->server 2-> server 3->server 2->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?

It is clear that this request is responded by some intermediate server (server 3) using its cache.