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

Lab 3 Solution

##### *Your name here*

***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.

***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.

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

***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 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.**

***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.**

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?

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.*

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

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

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?

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.

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.

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?

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

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**.

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

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

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

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.

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

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.

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**.

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

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?

***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**.

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

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.

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

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 four or more servers?

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

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

Explain the output.

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.

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

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.

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

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.

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

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

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

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.

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**.

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

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

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.

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