Load Balancing DNS README
Tanvi Wagle (tnw39) and Boning Ding (bnd28)

1. Briefly discuss how you implemented the LS functionality of tracking which TS responded to the query and timing out if neither TS responded.

   We used the select functionality provided by python's socket programming interface. This function determines which sockets are ready to be read from. Therefore, LS can read only the sockets that are ready and have data in the buffer instead of reading from a socket that is not ready causing the program to wait for the socket to send information. We call select in a loop in order to continuously check if the TS sockets have readable information. If any socket does have readable information, then immediately returns to the client because we only need one socket's response. A TS that does not have the information requested by the query sends nothing back. Select also has a timeout functionality. If both sockets do not return anything within 5 seconds, LS returns an error to the client.

2. Are there known issues or functions that aren't working currently in your attached code? If so, explain.

   The program has been tested and works properly. Some tests for connectivity include the following…
   1. Connecting to both servers in different machines/hostnames
      a. Checked by having all machines on different ilab hosts
   2. Connecting to both servers in the same machine/hostname (localhost)
      a. Done this throughout code testing
   3. Different Queries
      a. TS2 returns data
         i. Tested with queries only in TS1
      b. TS1 returns data
         i. Tested with queries only in TS2
      c. TS1 and TS2 timeout
         i. Tested case where none of the requested addresses are in TS1 and TS2. → All Error: HOST NOT FOUND
         ii. Tested different queries
            1. Shuffled TS1 and TS2 order in HNS.txt
            2. TS1 first then TS2 second in order in HNS.txt. Vice versa

3. What problems did you face developing code for this project?

   Setting up communications between each server took a little bit more time since the structure is very different than the first project. Furthermore, there are different cases that this project needed to catch (timeouts in TS1 and TS2 and how they communicate with LS and back to Client). Understanding the difference between enabling the blocking vs nonblocking communication between different socket connections was tricky. We had problems deciding between using blocking recv vs select. The first approach we used was to make the receive call blocking by adding a timeout to each TS socket: ts1Socket.setTimeout(5) and ts2Socket.setTimeout(5). However, this meant that in order for LS to timeout completely it needed to wait a full 10 seconds. It also meant we had to wait for ts1 to timeout first before trying to receive from ts2. The functionality we wanted was a timeout of five seconds between the two sockets. This is why we made the decision to use select(). Lastly, we had issues sending the queries one at a time. Sometimes when queries were sent they were received together instead of individually. To fix this, we only read 200 bytes at a time and made sure each message was 200 bytes long by adding extra trailing spaces.

4. What did you learn by working on this project?

   We learned about the fundamentals of how DNS load balancing works. After completing this project, we have a better understanding of how this DNS server is able to handle server failure and high traffic volume more efficiently than the previous DNS server that we implemented in the first project. For example, instead of returning an error when TS does not find the host in its table we return nothing, thereby creating less traffic between the LS and TS servers. We also learned more about the technicalities of the python socket programming interface and how to read and send data in different ways.