Project 2: Load-balancing DNS servers

- Overview

In project 2, we will explore a design that implements load balancing among DNS servers by splitting the set of hostnames across multiple DNS servers.

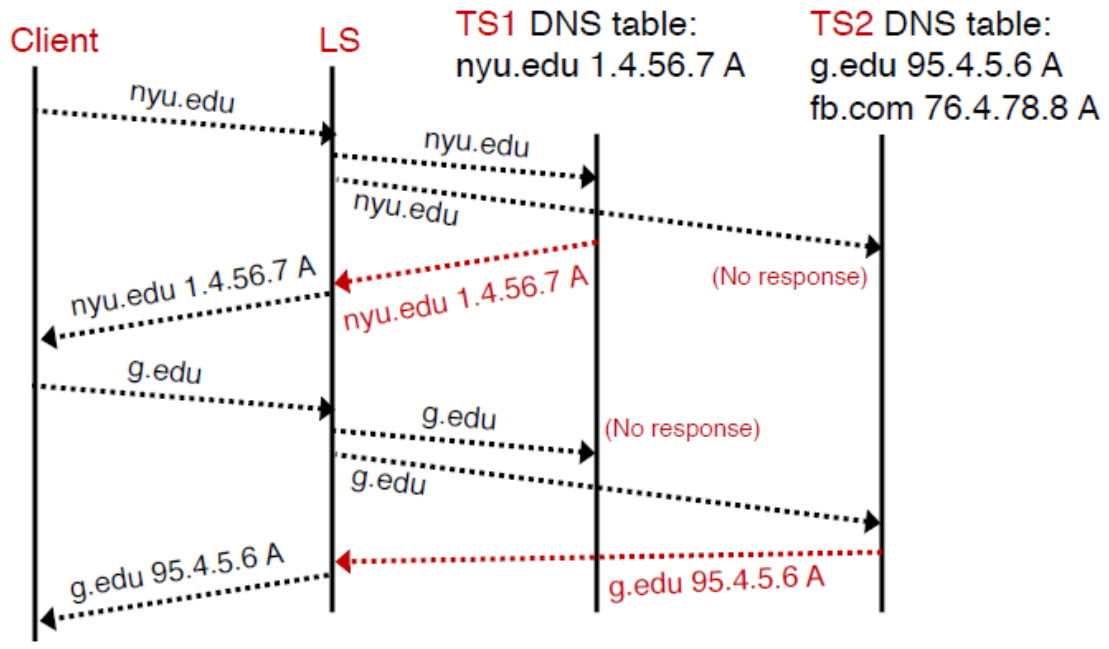
You will change the root server from project 1, RS, into a load-balancing server LS, that interacts with two top-level domain servers, TS1 and TS2. Only the TS servers store mappings from hostnames to IP addresses; the LS does not. Further, the mappings stored by the TS servers do not overlap with each other; as a result, **AT MOST ONE** of the two TS servers will send a response to LS. Overall, you will have four programs: the client, the load-balancing server (LS), and two DNS servers (TS1 and TS2).

Each query proceeds as follows. The client program makes the query (in the form of a hostname) to the LS. LS then forwards the query to _both_ TS1 and TS2. However, at most one of TS1 and TS2 contain the IP address for this hostname. Only when a TS server contains a mapping will it respond to LS; otherwise, that TS sends nothing back.

There are three possibilities. Either (1) LS receives a response from TS1, or (2) LS receives a response from TS2, or (3) LS receives no response from either TS1 or TS2 within a fixed timeout interval (see details below).

If the LS receives a response (cases (1) and (2) above), it forwards the response as is to the client. If it times out waiting for a response (case 3) it sends an error string to the client. More details will follow.

Please see the attached pictures showing interactions among the different programs.



More details on the design

1. TS design: There are two TS servers which each maintain a DNS table consisting of

three fields:

- Hostname

- IP address

- Flag (A only; no NS)

For each query received from the LS, each TS server does a lookup in its DNS table, and if there is a match, sends the DNS entry as a string: Hostname IPaddress A

If the Hostname isn't found in the DNS table, the TS server sends nothing back. A TS server without the hostname in its local DNS table MUST NOT send any data to the LS or the client. DNS tables can be read from PROJ2-DNSTS1.txt and PROJ2-DNSTS2.txt respectively for TS1 and TS2. We will ensure that the two DNS tables have no overlapping hostnames. Note that DNS lookups are case-insensitive. If there is a hit in the local DNS table, the TS programs must respond with the version of the string that is in their local DNS table. Each TS maintains just one connection -- with the LS.

2. LS design: The LS receives queries from the client and forwards them directly to

both TS1 and TS2. Since the DNS tables for TS1 and TS2 have no overlap, at most one will respond to any query. It is possible that neither of them responds. If the LS receives a response from one of the TS servers, it should just forward the response as is to the client. (As shown above, this string will have the format: Hostname IPaddress A as obtained from the TS that just responded.) If the LS does not receive a response from either TS within a time interval of 5 seconds (OK to wait slightly longer), the LS must send the client the message: Hostname - Error:HOST NOT FOUND where the Hostname is the client-requested host name.

The LS maintains three connections (and sockets): one with the client, and one with each TS server.

The tricky part of implementing the LS is figuring out which TS has pushed data into its corresponding socket (if at all one of them has), and timing out when neither has pushed data. Think carefully about how you will achieve this. Just performing recv() calls on the two TS sockets won't do it, since recv() by default is a __blocking call__: if you recv() on the TS1 socket but TS1 hasn't pushed any data, your LS program will indefinitely hang.

3. Client: The client is very simple. The client sends requests only to the LS. The client also directly prints the output it receives from the LS. The client reads hostnames to query from PROJ2-HNS.txt, one query per line. The client must write all the outputs it receives from LS into a file, RESOLVED.txt, one line per query. The client must NOT communicate directly with TS1 or TS2. The client maintains only one connection -- with the LS.

4. How we will test your programs: As part of your submission, you will turn in four programs: ls.py, ts1.py, ts2.py, and client.py, and one README file (more on this below). We will be running the four programs on the ilab machines with **Python 2.7**. Please do not assume that all programs will run on the same machine or that all connections are made to the local host. We reserve the right to test your programs with local and remote socket connections, for example with client.py, ts1.py, ts2.py, and ls.py each running on a different machine. You are welcome to simplify the initial development and debugging of your project and get off the ground by running all programs on one machine first. However, you must eventually ensure that the programs can work across multiple machines.

The programs must work with the following command lines:

python ts1.py ts1ListenPort

python ts2.py ts2ListenPort

python ls.py lsListenPort ts1Hostname ts1ListenPort ts2Hostname ts2ListenPort

python client.py lsHostname lsListenPort

Here:

- ts1ListenPort and ts2ListenPort are ports accepting incoming connections at TS1 and TS2 (resp.) from LS;

- lsListenPort is the port accepting incoming connections from the client at LS;

- lsHostname, ts1Hostname, and ts2Hostname are the hostnames of the machines running LS, TS1, and TS2 (resp.).

We will provide the input files PROJ2-HNS.txt, PROJ2-DNSTS1.txt, and PROJ2-DNSTS2.txt. You must populate RESOLVED.txt from the client. The entries in the DNS tables (one each for TS1 and TS2) will be strings with fields separated by spaces. There will be one entry per line. You can see the format in PROJ2-DNSTS1.txt and PROJ2-DNSTS2.txt. Your server programs should populate the DNS table by reading the entries from the corresponding files.

Your client program should output the results to a file RESOLVED.txt, with one line per result.

See the samples attached in this folder.

We will test your programs by running them with the hostnames and tables in the attached input files (*.txt) as well as with new hostnames and table configurations. You will be graded based on the outputs in RESOLVED.txt. Your programs should not crash on correct inputs.

5. README file

In addition to your programs, you must also submit a README file with clearly delineated sections for the following.

1. Please write down the full names and netids of all your team members.
2. Briefly discuss how you implemented the LS functionality of tracking which TS responded to the query and timing out if neither TS responded.
3. Are there known issues or functions that aren't working currently in your attached code? If so, explain.
4. What problems did you face developing code for this project?
5. What did you learn by working on this project?

Submission:

Turn in your project on Sakai assignments. Only one team member needs to

submit. Please upload a single zip file consisting of client.py, ls.py,

ts1.py, ts2.py, and README.

Some hints

Run your programs by first starting the TS programs, then the LS program, and finally the client program.

There are a few methods to turn a blocking recv() call at LS into a call that will return, possibly after a timeout. One possibility is to use a non-blocking socket. Another is to use the system call select(). Multi-threading may help, but is not necessary. DNS lookups are case-insensitive. It is okay to assume that each DNS entry or hostname is smaller than 200 characters.

START EARLY to allow plenty of time for questions on Piazza should you run into difficulties.