Project #2: Asynchronous Sockets

CS 352 Internet Technology

Released: February 1, 2022; Due: February 23, 2022

Instructions

Please read these instructions carefully before you begin.

1. You must work on this project in teams of 2.
2. You are free to discuss the project on Piazza or through other means with your peers and the instructors. You may refer to the course materials, textbook, and resources on the Internet for a deeper understanding of the topics. However, you cannot lift solutions from other students or from the web including GitHub, Stack Overflow, or other resources. Do not post this project to question-answering services like Chegg. All written and programmed solutions must be your team's original work. We run sophisticated software to detect plagiarism and carefully monitor student answers. If you are in doubt, please ask us.
3. You cannot post your solutions to this project on your personal GitHub page or on other web services hosting class materials.
4. For each question in the project report, please be clear and concise. Vague and rambling answers will receive zero credit.
5. For the report question on collaboration, please include anyone you discussed the project with, and also any resources you consulted to learn how to solve this project, including URLs of pages visited on the Internet. Please be specific about the aspect of the project that you got help with. You must be thorough and as complete as possible here. It is mandatory to answer this question.
6. This project is due on Feb 23, which is the day after the first mid-term. However, you do get 3 weeks to work on this project, and there is no project or problem set due the week before that of the submission date. We encourage you to *start early* and get the bulk of the work for this project done the week(s) before it is due, rather than keeping it until the submission date.
7. If you have any questions or clarifications on the project, please post them on Piazza or contact the course staff. We are here to help.

Overview

In project 2, we will implement a set of programs that mimic an implementation of load balancing across DNS servers. In many load balancing scenarios, it is common to perform *request duplication*, i.e., sending the same request to two servers, and picking the response that arrives first. At the cost of using more system resources, this allows the overall system to hide the delay caused by a slow server. In this project, we will model a slow server in the extreme by implementing servers that do not send any response at all in some cases.

Overall, you will build four programs in this project: a client, a root server (RS) that implements load balancing, and two DNS servers TS1 and TS2. We will implement load balancing across DNS servers as follows. First, the client sends its DNS query to the root server RS. RS resolves the query *recursively*, by querying two other DNS servers TS1 and TS2, and returning the response or an error to the client. In this project, only TS1 and TS2 store the actual mappings from domain names to IP addresses. RS does not store any name to IP mappings.

The mappings stored by TS1 and TS2 do not overlap with each other. Only when a TS server contains a mapping for the queried domain name will it respond to RS; otherwise, that TS sends nothing back. Hence, at most one TS will respond to any query from RS. It is also possible that neither TS contains a mapping for the queried domain name.

However, RS does not know in advance which TS, if any, will contain the IP address for a given domain name. Hence, RS must query *both* TS1 and TS2. There are three possibilities: (1) TS1 responds; (2) TS2 responds; or (3) neither responds within a fixed timeout. Note that it is never the case that both TS1 and TS2 respond. In cases (1) and (2) above, RS must relay the response as is from the server to the client. In case (3), RS returns an error to the client.

Please note that RS implements recursive query resolution. That is, only RS interacts with the client. The TS servers do not communicate directly with the client. See the attached pictures showing the interactions among the different programs.

Design of the servers and DNS message formats

In lieu of the actual DNS protocol, this project will use a simple message format for name queries and responses, which we define below.

Design of the TS servers

There are two TS servers. The primary function of each TS server is to look up a domain name queried by the RS and return an entry with an IP address, if the lookup succeeds. Each TS maintains just one connection: with the RS. Each TS server maintains a DNS table consisting of three fields:

- Domain name
- IP address
- Type (A only in this project)

For each query received from the RS, each TS server does a lookup of the domain name in its DNS table, and if there is a match, sends a DNS response with the following string:

`DomainName IPaddress A IN`

The four fields represent the name, value, type, and class of the response, respectively. If the queried domain name isn't found in the local DNS table, the TS server sends nothing back. A TS server without the domain name in its local DNS table *must not* send any data to the RS or the client. TS1 (resp. TS2) will read its DNS table from the input file `PROJ2-DNSTS1.txt` (resp. `PROJ2-DNSTS2.txt`). We will ensure that the two DNS tables have no overlapping domain names.

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. You will see examples of this in the samples attached in the project archive.

Design of the RS server

The RS server receives queries from the client and forwards them directly to both TS1 and TS2. If either one responds, RS will relay the response directly to the client, else it will send an error message.

Since the DNS tables for TS1 and TS2 have no overlap, at most one of TS1 or TS2 will respond to any query. It is possible that neither of them responds. If the RS 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 IN` as obtained from the TS that just responded. If the RS does not receive a response from either TS within a time interval of 5 seconds (it's also OK to wait slightly longer), the RS must send the client the message `DomainName - TIMED OUT` where the `DomainName` is the domain name queried by the client.

The RS maintains three connections/sockets: one with the client and one with each TS server. The most tricky part of implementing the RS is figuring out which TS, if any, has pushed data into its corresponding socket, and timing out when neither has pushed data. Think carefully about how you will implement this. Just performing `recv()` calls on the two TS sockets won't achieve the desired result, since `recv()` by default is a *blocking call*: if you `recv()` on the TS1 socket but TS1 hasn't pushed any data, your RS program will hang indefinitely waiting for data from TS1.

Client

The client sends queries to the RS. The client also directly prints the output it receives from the RS. Conceptually, it is the simplest part of this project. The client reads domain names to query from the input file `PROJ2-HNS.txt`, one query per line. The client must write all the outputs it receives from RS into a file, `RESOLVED.txt`, one line per response. The client must NOT communicate directly with TS1 or TS2. The client maintains only one connection: with the RS.

Some helpful notes

(1) Run your programs by first starting the TS programs, then the RS program, and finally the client program.

(2) There are a few methods to turn a blocking `recv()` call at the RS 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 we have found it unnecessary.
(3) DNS lookups are case-insensitive. The DNS response must contain the version of the domain name that is in the DNS table.
(4) It is okay to assume that each DNS entry or hostname is smaller than 200 characters.
(5) *Please start this project early* to allow plenty of time for questions on Piazza should you run into difficulties.

What you must submit and how we will test it

For your project submission on Canvas, please turn in four programs: `rs.py, ts1.py, ts2.py`, and `client.py`, and a project report entitled `report.pdf`. The questions for the report are listed below. We will be running the four programs on the ilab machines with the Python 2 version on `ilab`. Please compress the files into a single Zip archive before uploading to Canvas. Only one team member must submit.

Please do not assume that all programs will run on the same machine or that all connections are made to the local host. We will test your programs with local and remote socket connections, for example with `client.py`, `ts1.py, ts2.py,` and `rs.py` each running on a different machine. You may simplify the initial development/debugging of your programs (and get off the ground) by testing 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 rs.py rsListenPort ts1Hostname ts1ListenPort ts2Hostname ts2ListenPort
python client.py rsHostname rsListenPort
```

where

- `ts1ListenPort` and `ts2ListenPort` are ports accepting incoming connections at TS1 and TS2 (resp.) from RS;
- `rsListenPort` is the port accepting incoming connections from the client at RS;
- `rsHostname, ts1Hostname`, and `ts2Hostname` are the hostnames of the machines running RS, 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 (`PROJ2-DNSTS1.txt` and `PROJ2-DNSTS2.txt` for TS1 and TS2 respectively) will be strings with fields separated by spaces. There will be one DNS entry per line. You can see the format in the samples provided with the project archive. Your server programs TS1 and TS2 should populate their local DNS table by reading the entries from their 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 for questions about formatting. We

will test your code both with these samples and other test cases of our own, and you will be graded based on the outputs in `RESOLVED.txt`. Your program should not crash or hang on correct inputs.

Project report

Please answer the following questions for the project report.

1. Team details: Clearly state the names and netids of your team members (there are 2 of you).
2. Collaboration: Who did you collaborate with on this project? What resources and references did you consult? Please also specify on what aspect of the project you collaborated or consulted.
3. Discuss how you implemented the RS functionality that tracks which TS responded to a given query or timing out if neither TS responded. Please be clear and specific.
4. Is there any portion of your code that does not work as required in the description above? Please explain.
5. Did you encounter any difficulties? If so, explain.
6. What did you learn from working on this project? Add any interesting observations not otherwise covered in the questions above. Be specific and technical in your response.

Contact the course staff on Piazza if you have any questions.