Project #2: RU-DNS: Build your own DNS

CS 352 Internet Technology

Due: March 14, 2025

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, generative AI (e.g., ChatGPT), or other resources. Do not post this project or parts of it to question-answering services like Chegg, academic hosting sites such as CourseHero, or generative AI. 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 this project specification or your solutions to the project on your personal GitHub page or any other public-facing services.
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 references and 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. Use of generative AI is not permitted in this course. 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. 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 holding back your most significant efforts until the submission date is too close.
7. There is a due date/time but no "time limit" on projects. That is, you may take as long as you need to work on the project, as long as you submit it on time. Further, you may make an unlimited number of submissions on Canvas.
8. If you have any questions or need clarifications on the project, please post them on Piazza or contact the course staff. We are here to help.

Overview

In project 2, you will design and implement RU-DNS, a simple protocol mimicking the Domain Name System (DNS) we discussed in class. Rather than relying on the Internet's DNS, you will build a set of DNS resolvers with private databases and a client program that resolves queries against this (distributed and hierarchical) database. Overall, there will be 4 programs in this project: a client, a root DNS server (RS), and two top-level domain servers (TS1 and TS2). Based on this simple DNS system, you will implement iterative and recursive query resolution. We do not consider caching or query failures in this project.

1 RU-DNS message format

Rather than using the Internet's standardized DNS protocol message format, we will use a simplified protocol message format for this project. RU-DNS protocol *request* messages have the following message format (there are four space-separated items):

```
0 DomainName identification flags
```

where `0` represents that this is a protocol request (not response) message, `DomainName` is a domain name that we want to resolve (e.g., www.google.com), `identification` is an integer incremented by the requesting client for each fresh request (starting from 1), and `flags` is one of two possible strings representing the following special processing:

- the flag `rd` (for "recursion desired") represents that the recipient DNS server should recursively resolve this query; and
- the flag `it` (for "iterative") represents that the client wants to perform iterative resolution.

Exactly one of the flags above will be provided in each RU-DNS request (their behaviors are mutually exclusive and complete).

(Aside: The real Internet's DNS system uses the absence of a "recursion desired" flag to indicate iterative resolution. Instead, we use a dedicated flag to simplify parsing of RU-DNS *requests. Further, real root DNS servers do not support recursion.)*

Here are examples of request messages in `RU-DNS`:

```
0 www.google.com 5 rd
0 princeton.edu 10 it
0 njit.edu 16 it
0 x.ai 23 it
```

RU-DNS protocol *response* messages have the format (five space-separated values):

```
1 DomainName IPAddress identification flags
```

where `1` indicates that this is a protocol response (not request) message, `DomainName` is the domain whose IP address was requested in a prior `RU-DNS` request, `identification` is the identification on that request, and `flags` indicate conditions related to the DNS response:

- the flag `aa` (for "authoritative") indicates that the response arrived directly from an authoritative name server. When any DNS server (`RS`, `TS1` or `TS2`) responds directly to the client with an IP address in its own local database, that response is considered authoritative. However, when the root server `RS` merely asks the client to contact a different DNS server (one of the TLD servers for iterative resolution) or forwards a response from one of the TLD servers (under recursive resolution), those responses are not considered authoritative, and do not include the `aa` flag.
- the flag `ra` (for "recursion available") indicates that the response was constructed through a recursive resolution by `RS`.
- the flag `ns` (for "name server") indicates that the DNS response did not fully resolve the domain requested, but instead is directing the client to the TLD server for iterative resolution. For such a response, the returned `IPAddress` isn't an IP address, but the hostname of the TLD server to whom the client is being redirected to.
- the flag `nx` (for "non-existent") indicates that the IP address for the requested domain name did not exist in the local database of the final DNS server which ultimately completed the resolution process (in the appropriate manner specified below). The `IPAddress` returned for a non-existent domain should be set to `0.0.0.0`

Exactly one flag is provided in each DNS response. Iterative queries (`it` flag set in the request) which are successfully resolved should result in the `aa` flag set in the response arriving at the client. Recursive queries (`rd` flag in the request) which are successfully resolved should result in the `ra` flag set in the response arriving at the client. Queries that failed resolution (no entry for the requested domain name in the appropriate database) should result in the `nx` flag set in the response, with the `IPAddress 0.0.0.0`. A responses that redirects the client to a TLD server must use the `ns` flag with the corresponding TS's hostname filled in the place of the `IPAddress`. The `identification` in the response must match the `identification` of the query.

Here are examples of DNS response messages in `RU-DNS`:

```
1 www.google.com 9.7.5.6 5 ra
1 princeton.edu 0.0.0.0 10 nx
1 njit.edu cheese.cs.rutgers.edu 16 ns
1 njit.edu 10.5.6.7 17 aa
1 x.ai 45.67.89.103 23 aa
```

2 Designing the DNS servers

Each DNS server is a separate program named either `rs.py, ts1.py,` or `ts2.py,` corresponding to `RS, TS1,` and `TS2` respectively. There is a client program (`client.py`).

Each DNS server has a local database of mappings from domain names to IP addresses. Each server must first read this database (presented as an input file with a fixed name, one per server) and store the mappings in an easily-searchable data structure in memory. You must write code to look up this in-memory database upon each RU-DNS query.

Here is an example of the contents of a `TS2` database file when `TS2` is in charge of the `.edu` top-level domain (one mapping per line):

```
rutgers.edu 128.1.1.4
njit.edu 10.5.6.7
```

Here is an example of the contents of the `TS1` database file when `TS1` is in charge of the `.com` top-level domain:

```
princeton.com 192.1.1.7
www.google.com 9.7.5.6
```

DNS lookups are case-insensitive. If there is a hit in the local DNS database, the server must respond with the version of the domain name that is in the local DNS database. The lookup algorithm must find a mapping for the exact (case-insensitive) domain name that is requested. For example, `google.com` and `www.google.com` are considered two different domain names.

The database of the root DNS server will specify the top-level domain that is managed by each TLD server, and also contain mappings resolving domain names which are managed by neither of the TLD servers. Here is an example of the contents of an RS database file:

```
com java.cs.rutgers.edu
edu cheese.cs.rutgers.edu
github.io 25.6.7.1
x.ai 45.67.89.103
bit.ly 1.2.3.4
```

The `RS` database always starts with two lines which specify the top-level domains managed by `TS1` and `TS2` respectively. In the example above, `TS1` running at `java.cs.rutgers.edu` manages the `.com` top-level domain, and `TS2` running at `cheese.cs.rutgers.edu` manages the `.edu` top-level domain. The remainder of the lines are mappings for domain names which can be resolved directly at the root DNS server, and are not under the top-level domains managed by `TS1` or `TS2`.

The client works to resolve hostnames listed in order in an input file with a fixed name. The mode of resolution (iterative or recursive) is also specified along with the domain name to be resolved. Here is an example of an input to the client (one query per line):

```
www.google.com rd
princeton.edu it
njit.edu it
x.ai it
```

The client always contacts the root DNS server `RS` first for query resolution (not `TS1` or `TS2`). Each new DNS query receives a unique `identification` field, which is one more than the `identification` in the previous query. The first query uses an `identification` value `1`. When the client sends an RU-DNS query to `RS`, the root server `RS` first checks if the requested domain name falls under one of the two top-level domains specified in the first two lines of the root database:

1. If the requested domain name is under one of the two top-level domains in the root database, and the query is flagged **iterative (it)**, `RS` returns a DNS response with the `ns` flag and the hostname of the corresponding TLD server (noted in the corresponding line of the root database) to the client. At this point, the client should create a new DNS query, with an `identification` value that is one more than the `identification` of the prior query. This new query should be directed to the TLD server provided in the prior (`ns`-flagged) response.
2. If the requested domain name is under one of the two top-level domains in the root database, and the query is flagged **recursive (rd)**, `RS` should relay the query *as is* to the corresponding TLD server, and relay the response it receives from that `TS` to the client. If the original response is authoritative (`aa` flag set), the `RS` should *modify the* `flags` *in the response to* `ra`. Otherwise (i.e., if the response has an `nx` flag set), `RS` must relay the response *as is* to the client.
3. If the requested domain name is under neither of the two top-level domains listed in the root database, `RS` should attempt to resolve it using the mappings in the root database. If a mapping is found, `RS` should send a response to the client with the authoritative reply flag (`aa`) set.
4. If the requested domain name is under neither of the two top-level domains listed in the root database, and `RS`'s attempt to resolve it using the root database fails, `RS` should send a response to the client with the non-existent domain flag (`nx`) set.
5. If either the client or the RS contacts the TLD server, the `TS` should attempt to resolve the requested domain name using its local (TS) database. If the resolution succeeds, the `TS` should send a response to the requester with the authoritative reply flag (`aa`) set. If the resolution fails, the `TS` should send a response with the non-existent domain flag (`nx`) set. The requester can be either the client (iterative resolution) or the RS (recursive resolution) resolving a domain name under the top-level domain managed by the `TS`.

   Pay attention to how each field in the DNS response is populated under each scenario.

The `TS` will use the same listening port to receive connections and queries from both the `RS` and the client. For simplicity, we keep the behavior of the `TS` exactly the same regardless of whether the request arrives from the client or the `RS`. Typically, the `TS` cannot tell which other program contacts it (i.e., the client or the RS) using the 4-tuple of the connection over which the RU-DNS query message is received.

3 How we will run and test your code

The `RS` server listens on a fixed port for messages from the client. Each `TS` server listens on the same fixed port both to receive messages directly from the client (iterative queries) and to receive messages from `RS` (recursive queries). The client and `RS` may contact one or both `TS` servers during resolutions. We will run the programs with command line parameters that provide information on the relevant listening ports to each program, as follows (your program must parse these command line arguments correctly):

```
python3 ts1.py rudns_port
python3 ts2.py rudns_port
python3 rs.py  rudns_port
```

```
python3 client.py rs_hostname rudns_port
```

The argument `rudns_port` denotes the port on which all the server programs listen to connections carrying RU-DNS queries. We will not reuse the same listening port on the same machine across different programs, so each server program must be running on a different machine. The client must know the root DNS server's hostname to contact it (this is similar to how a local DNS resolver must be configured with the list of root DNS servers). This information is available through the `rs_hostname` command line argument. The root DNS server will get to know the TS hostnames through the entries in the root database.

An example. Suppose:

- `client.py` runs on ilab1.cs.rutgers.edu;
- `rs.py` runs on cheese.cs.rutgers.edu;
- `ts1.py` runs on java.cs.rutgers.edu;
- `ts2.py` runs on ilab4.cs.rutgers.edu; and
- the RU-DNS port is set to 45000.

We will execute your programs as follows:

```
(on java.cs.rutgers.edu)
python3 ts1.py 45000

(on ilab4.cs.rutgers.edu)
python3 ts2.py 45000

(on cheese.cs.rutgers.edu)
python3 rs.py 45000

(on ilab1.cs.rutgers.edu)
python3 client.py cheese.cs.rutgers.edu 45000
```

The databases for the three DNS servers will be provided in the files named `ts1database.txt`, `ts2database.txt`, and `rsdatabase.txt`. Their formats were described earlier in this document. The client will read a list of domain names to resolve from a file named `hostnames.txt` and issue corresponding RU-DNS queries in order. Examples of these files are provided in the project archive.

Your DNS server programs must log each RU-DNS response sent out over any connection, in the same order in which they were sent out, in files named `rsresponses.txt, ts1responses.txt,` and `ts2responses.txt` (respectively at `RS, TS1,` and `TS2`). Your client program must log each RU-DNS response it receives in order (including responses with the `NS` flag for iterative queries) in a file named `resolved.txt`. Example outputs are provided in the project archive.

The order of the outputs must be consistent with the order in which the domain names are queried and that in which RU-DNS messages are sent between the client/servers.

4 What you must submit

Please turn in four programs `rs.py`, `ts1.py`, `ts2.py`, `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 3 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 machine. 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. We will adapt the input files (.txt) accordingly when we do this.

You may simplify the initial design of your programs by testing all programs on one machine first. However, the design of the project asks all servers to listen on the same port, which cannot happen on one machine. Hence, we suggest that you start your development by running the servers on different ports on the same machine first (change the first two lines of `rsdatabase` to use `localhost` as the host names of the `TS`es), with the client assuming a fixed mapping from a top-level domain to the specific `TS` and corresponding port (e.g., say, assuming that `edu` queries go to `TS1` on port `p1` and `com` queries go to `TS2` on port `p2` $\neq$ `p1`). Once this functionality is tested to work correctly, you can move the servers to different machines and run the servers on the same fixed port provided through the command line. In your final submission, the client cannot assume a fixed mapping between a top-level domain and a specific `TS`. Your final submission must work when we run the three servers on distinct machines.

There are example input files (`hostnames.txt`, `rsdatabase.txt`, `ts1database.txt`, `ts2database.txt`) available for testing. We will use test cases not provided in these examples when grading your programs. You must populate the output files (`rsresponses.txt`, `ts1responses.txt`, `ts2responses.txt`, and `resolved.txt`). You will be graded based on the correctness of the outputs in these files.

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. Briefly discuss how you implemented recursive and iterative query functionality. Please be clear and specific.
4. Is there any portion of your code that does not work as required 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. Please be specific and technical in your response.

Notes

1. Start your programs by first running the two TS programs, then the RS program, and finally the client program, so that all listening sockets are in place whenever a `connect()` call is made.
2. RU-DNS lookups are case-insensitive. The response must contain the domain name that is in the local database of the server that generated the response.
3. You may assume that each domain name and each line containing a mapping (domain name to IP address) is smaller than 300 characters.
4. You may either use long-lived (persistent) or individual (non-persistent) connections to send RU-DNS request messages between the different programs.
5. Your program should not crash or hang on wellformed inputs.
6. *Please start this project early* to allow plenty of time for questions on Piazza should you run into difficulties.