Project 1: DNS Resolver and Name Server

27 February 2017

Submitting your project

Requirements about the delivery of this project:

- Submit via Blackboard (http://blackboard.ru.nl);
- Upload one single .zip archive with the structure as described in the document.

Deadline:

Project Milestone (Non-Caching DNS Resolver): Friday, 24 March, 20:00;

Complete Project (DNS Resolver and Name Server): Friday, 21 April, 20:00.

Late submissions are not accepted. In case you haven't provided anything for the milestone, or your submission was deemed insufficient, you are still allowed to submit the complete project, but the maximum achievable number of points drops from 100 to 80. Your milestone submission is considered *sufficient* if all of the following hold:

1. the DNS resolver can resolve simple domain names (e.g. yahoo.com, bol.com);
2. tests for the 2 non-caching scenarios are provided.

Organization: It is strongly encouraged that you work in pairs of two. You can also work individually. Groups of more than two members are not allowed.

Marks: You can score a total of 100 points. In case you did not meet the milestone requirement, the number of points is multiplied by 0.8. The points are distributed as follows:

- 30 points for a non-caching DNS Resolver;
- 20 points for implementing caching and TTL;
- 25 points for implementing a DNS Name Server;
- 10 points for the set of tests;
- 10 points for documentation;
- 5 points for interface and structure.

Implementation: You must use the latest Python 3.x version (3.6). A Python framework compatible with this version is provided here. You are strongly encouraged to build your project on top of this framework. The framework is designed to simplify and streamline construction of the resolver functions, and also to give an example of how a test can be implemented.

We suggest you use versioning. In particular, we recommend Git. In case you are unfamiliar with it, a good presentation on Git can be found on Giso Dal's page at: http://www.cs.ru.nl/~gdal/files/gittutorial.pdf. The faculty is hosting a GitLab server, accessible with your science login, at https://gitlab.science.ru.nl/. For more information on it, see http://wiki.science.ru.nl/cncz/GitLab.

DNS Resolver and Name Server

This project consists of two parts. You first have to build a caching DNS Resolver capable of solving FQDNs (Fully Qualified Domain Names) to their IP address by accessing its cache, and eventually sending iterative queries to known name servers. You then have to embed the DNS Resolver within a Name Server, which manages a zone file and can respond authoritatively to queries for names within its zone. Moreover, you will have to implement a set of tests for common scenarios. Finally, you will need to document your implementation. Implementing the DNS Resolver marks the **project milestone**. Caching support is not required for the milestone and only required for the complete project. The only tests required for the milestone are those for the non-caching case (valid and invalid FQDN). Documentation is required for both the milestone and the complete project.

The RFC documents that you will have to consider are RFC 1034 and RFC 1035. The DNS Resolver and DNS Name Server should implement the algorithms in Section 5.3.3 of RFC 1034 and in Section 4.3.2 of RFC 1034 respectively, with the simplifications mentioned in the project description. DNS query and reply messages comprise a header and 4 main sections. Among others, the header contains flags for determining whether the message is a query/reply, whether the replier is authoritative or whether recursion is enabled. Each section contains *Resource Records* where each *Resource Record* is a tuple made from a Type (A, CNAME, ...) a Class (IN) and *Resource Record*-specific information. In your implementation, you should concern yourselves only with *Resource Records* of Type A, CNAME and NS, with Class IN. Figure 1 gives an overview of an encapsulated DNS message in the context of a network datagram.

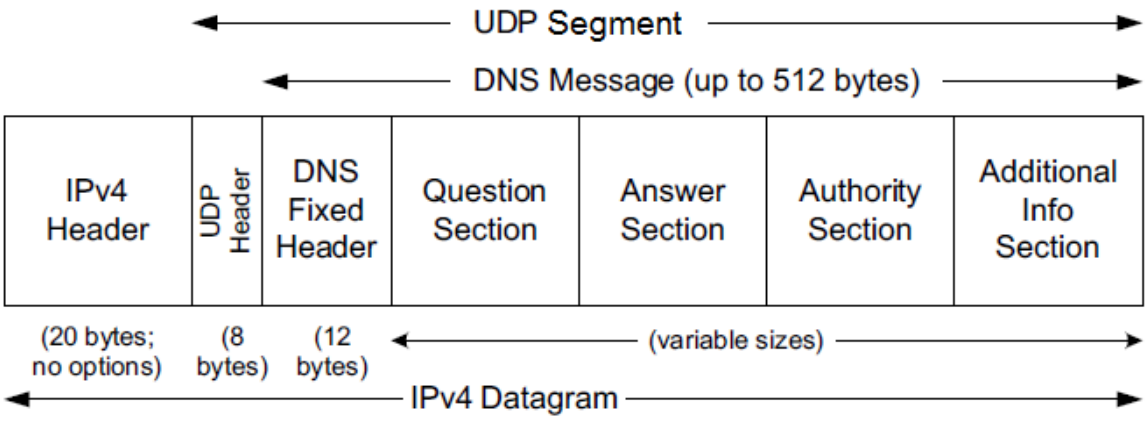

Figure 1: Stack with DNS (adapted from 'TCP-IP Illustrated, Vol 1')

DNS Resolver

Your DNS Resolver, given a FQDN string and a list of name server addresses (we will name them *hints*), will have to roughly (1) consult its cache and resolve the FQDN address immediately if a response for the query is found, (2) based on the FQDN, cache and the hints pick the best name server to query, (3) build an **iterative** DNS query (4) send the query to the server over a UDP channel, and (5) process the response, if it is an answer for FQDN output and return, if not update the hints using the answer and return to step (2).

Functionality: The DNS Resolver should resolve FQDNs to their corresponding addresses. A general overview of a resolver is given in Section 5 of RFC 1034. We first consider the case when there is no caching. Your resolver must send an **iterative** query for the FQDN to one of a list of given name servers (hints). As starting hints you should use the IPs of the root name servers found in the root hints. Hence, the resolver sends the query starting from one of these root name servers, continuing on with a TLD server and eventually with a sequence of authoritative name servers, until it receives an answer for the FQDN or it concludes that the name cannot be resolved.

In case of a positive answer, the DNS Resolver generates an output, which is a tuple containing: (1) the FQDN; (2) the list of IP addresses found; and (3) the list of aliases for the FQDN. This matches the output interface of Python's gethostbyname_ex socket command.

This output has empty lists if the name could not be resolved or an error was encountered. The policy for selecting a name server out of a list of possibilities (hints) can be selecting the first in this list and removing it. If the name server is unreachable or responds with an error message/no answer, you should try the next name server in the list, and so on. If the FQDN can be resolved from the response (the response is a positive answer), resolve it and exit. Otherwise, the answer might contain referrals to better name servers, which you should add at the start of the list of lists (so they are selected as you loop back). If you exhaust the list of hints, you should return an empty output.

At a technical level, the query sent has a corresponding *A-Resource Record* for the FQDN in the Question-Section. A positive answer will contain *A and CNAME-Resource Records* records in the Answer-Section from which the FQDN can be solved. Otherwise, the answer might refer to other name servers, referred by *A and NS-Resource Records*. You use these name servers further on in your resolution.

Caching: To implement caching, for every response received you cache all *A and CNAME-Resource Records*. Each *Resource Record* contains a time to live (TTL) value. The TTL of an entry in the cache is set to either a positive value given as parameter, or, in case no such value was given or the value given was negative, the TTL value of the record that is to be cached. The TTL value is measured in seconds.

Whenever the TTL of an entry expires, the record is removed. You are free to decide on how you manage and enforce TTL. For example, for each entry you may store a timestamp and check entry expiry against it. What is important is, once the TTL expires, the entry is no longer considered. The cache should be used not only as a means of quickly resolving a FQDN, but also to minimize the number of iterative queries sent for a FQDN not solvable by the cache.

Caching should be made to a cache file in JSON format. The JSON entry for a cached record comprises 5 name value pairs, for `name`, `type`, `class`, `ttl` and `rdata` respectively. The cache file is loaded at start and updated. Note that, this does not necessarily apply to real resolvers. As a simplification, you do not have to implement negative caching (caching for negative responses).

DNS Name Server

The DNS Name Server manages a local zone file and uses it to answer A Type queries. It also utilizes the DNS Resolver described in the previous section. Your Name Server should follow the algorithm in Section

4.3.2 of RFC 1034, that is it should: (1) load the local zone file to memory, (2) listen to port 53 for incoming queries, (3) on receiving a query it should consult its zone, and try to answer from its zone (we will call this *zone resolution*), if it cannot and the recursion is enabled (4) it should use the DNS Resolver to answer the query. Finally, it should (5) send the answer back to the originator of the query.

To keep things simple, steps (3) and (4) don't interleave, first zone resolution tries to give a conclusive response, if it fails, the DNS Resolver is used to solve the entire FQDN, and zone resolution is never summoned again for this query. Moreover, no information has to be passed from zone resolution to the DNS Resolver, the DNS Resolver starts from scratch when called to solve the FQDN. This is an accepted deviation from Step 3.b of Section 4.3.2 of RFC 1034.

It is important to clarify that the zone is always consulted first(the specification can be confusing in this regard). The interface differs from that of a DNS Resolver, as its inputs are now DNS queries instead of strings, while outputs are DNS replies sent over a UDP socket. In step (4), the Name Server accesses the Resolver by supplying it the FQDN from the query. It is not a problem that the Resolver will then have to rebuild this query (instead of receiving it directly from the Name Server).

The zone file (or master file) should be structured as described in Section 5 of RFC 1035. Your zone file can contain records of type NS, A and CNAME, and only for these records do you need to provide corresponding handling. An example of such a zone file is the root hints. A major simplification is that you don't have to consider zone maintenance and transfers, as described in Section 4.3.5 of RFC 1034. Moreover, there is no SOA record describing the zone. The zone file is loaded at the start and never updated. How to respond from zone information is described in the RFC 1034 algorithm.

The DNS Name Server is aware of DNS flags, namely it should use its DNS Resolver only if recursion is enabled in the query. Moreover, if it is authoritative over the requested FQDN in the query, it should then send an authoritative response.

Concurrency: Your DNS Name Server should be able to handle multiple queries at the same time. To implement this feature, you should use the DNS Transaction ID to match responses to their corresponding queries. You can optionally employ some (basic) means of guarding the cache against concurrent access. Not doing so will not result in a penalty however, as concurrency is not the aim of the project.

Tests

Along with the DNS Resolver and Name Server, you should also supply a set of tests. We suggest you use FQDNs with relatively stable addresses in your tests, such as `http://gaia.cs.umass.edu/`.

For the DNS Resolver:

- with no caching (required for the project milestone):
  - ◇ solve a FQDN, output with corresponding IP/CNAME/authoritative status generated
  - ◇ look up a FQDN that does not exist, empty output generated
- with caching
  - ◇ solve an invalid cached FQDN (for example invalid_address.com), output corresponds to cache
  - ◇ start your server and wait configured TTL + 1 time for an invalid cached FQDN to expire, an empty output should be generated

For the DNS Name Server:

◇ solve a query for a FQDN for which your server has direct authority

◇ solve a query for a FQDN for which your server does not have direct authority, yet there is a name server within your zone which does

◇ solve a query for a FQDN which points outside your zone

◇ solve parallel requests for different FQDN, their servicing should be made in parallel and correct responses should be generated

Structure and documentation

The project should be structured as follows:

```
proj1_sn1_sn2:
        dns_server.py
        dns_tests.py
        dns
        dns/resolver.py
        cache
        zone
        documentation.*
        (other support files)
```

Where:

- *sn1* and *sn2* are your student numbers (for example, s123456);
- *dns_server.py* is the python program implementing the DNS Name Server;
- *dns_tests.py* is the python program implementing the tests;
- *dns* is a folder where you store all support files;
- *dns/resolver.py* contains the implementation of the resolver;
- *cache* is the cache file used by the DNS Resolver. It should be readable text;
- *zone* is the zone file used by the DNS Name Server. It should also be readable text;
- *documentation.** should explain the implementation.

Interface and Documentation

Your DNS Resolver should be implemented as a library class providing the `gethostbyname` function. This function receives a string and returns a tuple of form : (FQDN, list_of_IP_addresses, list_of_aliases). Your python programs should be run:

```
# running the DNS Server
python dns_server.py [-c|--caching][-t|--ttl time][-p|--port portNum]

# running the set of tests (in case you have implemented runnable tests)
python run_tests.py [-s|--server dns_server][-p|--port portNum]}
```

Where:

- *c* enables caching, by default it is disabled
- *t* sets the TTL that is applied to all cached entries

- s is the IP address in string format of the name server
- p is the port number at which the name server listens

You do not have to implement to long versions of these parameters (though with frameworks such as argparse it is easy to do). The framework provides *dns_client.py*, a wrapper program over a mock DNS Resolver, which you can use to test your Resolver. To test your Name Server you can use `dig` or `nslookup`.

You need to provide a brief documentation of the software provided for both the project milestone and complete project. For the milestone, the documentation should very briefly explain how name resolution is done as well as any challenges encountered. Optionally, you can mention some FQDNs for which the implementation was tested against, as well as special considerations (if any) one must take to run your implementation. For the complete project you should present a general flow of how the DNS Resolver and DNS Name Server operate mentioning the relevant flags for authority, query/reply and recursion. You should also describe how you crafted DNS messages (what libraries you used), how you implemented concurrency, how you generate Transaction IDs, what format you use for storing the cache and what the policy is for removing expired entries from the cache. Also mention any problems you have encountered. Some of these questions can be answered directly by checking the framework provided (should you use it).

Implementation guidance

You are provided a framework which implements a significant portion of the parsing message parsing, as required by RFC 1035. You are referred to RFC 1034 and RFC 1035, in particular the algorithms for the Resolver and Name Server. A very good high level overview on how DNS works is found on technet. You could start off with that, then dive into the RFC documents.

Schematic requirements

Table 1 gives an overview on what you are expected to deliver for the milestone and complete project, respectively. We also review the key simplifications/deviations from the standards. You don't have to implement/can omit: (1) negative caching (or caching for negative responses); (2) Step 3.b of Section 4.3.2 of RFC 1034, moreover information needs not be passed from zone resolution to the DNS Resolver; (3) zone maintenance and transfers, as described in Section 4.3.5 of RFC 1034.

Requirement		Project Milestone	Complete Project
Implement	Name resolution	✓	✓
	Caching	X	✓
	Concurrency	X	✓
	Zone resolution	X	✓
	Interfaces	X	✓
Test	Name resolution	✓	✓
	Caching and concurrency	X	✓
	Zone resolution	X	✓
Structure	Naming and placement	✓	✓
Interface	Short form arguments	X	✓
Document	Brief document	✓	X
	Extensive document	X	✓

Table 1: Requirements for Project Milestone and Complete Project