CS 455: Computer Communications and Networking
(Fall 2024)

PA-1: DNS Client
Due Date: October 14, 11:59 PM EST

Project Description

In this project, you will implement a DNS client that can query an existing DNS server for a domain name to IP address translation. We discussed in class the operations of DNS and the types of messages (query and response) used by DNS protocol. “nslookup” and “dig” are two tools that can be used to achieve the same translation using the command line. In this programming assignment, you will manually handcraft DNS query messages, send them to known DNS servers, and process their response. You have been provided with specifics of the DNS message format (which we also discussed in class) in the following description.

Your DNS client has three big high-level pieces (in order of operations):

1. Prepare the DNS query: the client should read the hostname provided by a user and prepare a query message which adheres to DNS protocol specifications. Note that if your message does not adhere to the specifications, the DNS server will not be able to identify/understand your query.
2. Send the DNS query: the client should create a UDP socket connection to the server and send the DNS query message you prepared above.
3. Receive and process the DNS response: the client should receive a response from the DNS server. It should then process the response based on the format of the DNS response message. It should extract the necessary information (*e.g.*, IP address) and display it to the user on the command line.

Next, we will now look at each of the three above pieces in detail.

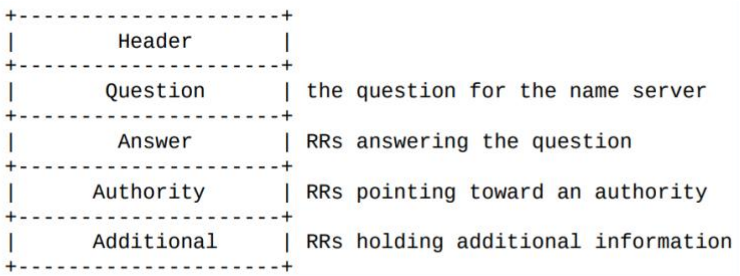
DNS Query

Your client should be able to read the hostname provided by a user. The hostname will be provided as a command-line argument to your client program as follows

```
$> my-dns-client <host-name>
```

This syntax may vary based on your programming language, for example, python my-dns-client <host-name>. Although a typical DNS client can query multiple hostnames at the same time, we will restrict to only one hostname as an argument in this programming assignment.

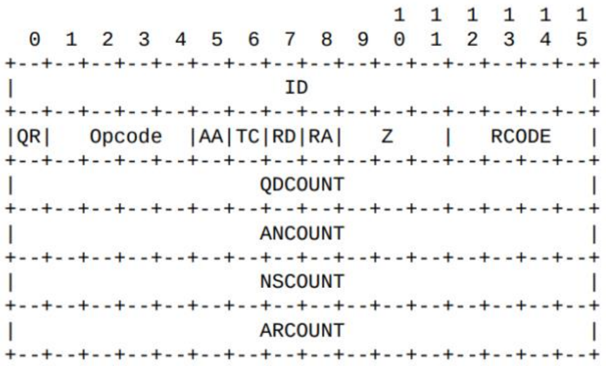
Once your client program reads the hostname, it should prepare a DNS query message. The general format of the DNS query and response message is provided in our textbook (Fig. 2.21). Specific bit-by-bit format of the query and response messages can be found in the official DNS RFC document (https://datatracker.ietf.org/doc/html/rfc1035#section-4 Section 4.1.1-4.1.3). It is highly recommended that you go through Section 4 of the official RFC to clearly understand the role and purpose of each field before starting the code. The query and response messages have 5 sections shown in Fig. 1. Your query message will only have the first two sections.



```
+---------------------+
|        Header       |
+---------------------+
|       Question      | the question for the name server
+---------------------+
|        Answer       | RRs answering the question
+---------------------+
|      Authority      | RRs pointing toward an authority
+---------------------+
|      Additional     | RRs holding additional information
+---------------------+
```

Fig. 1: DNS message sections

The header contains the following fields (refer to Fig.2):



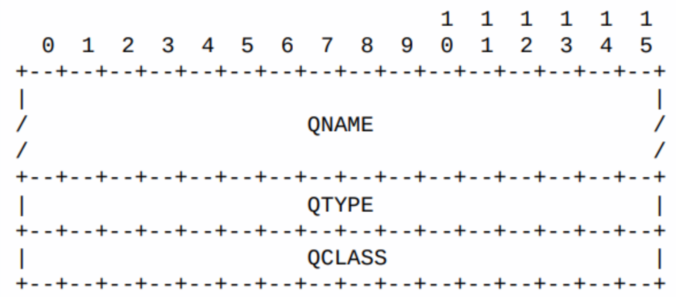
```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Fig. 2: Fields of the Header section

We will describe only the important fields here. You should read the RFC and learn about the specifics of each field.

- ID: A 16-bit identification that can uniquely identify a query message. Note that when you send multiple query messages, you will use this ID to match the responses to queries. The ID can be randomly generated by your client program.
- QR: 0 for query and 1 for a response.
- OPCODE: We are only interested in a standard query (*i.e.*, it is 0).
- RD: This bit is set if the client wants the name server to recursively pursue the query. We will set this to 1.
- QDCOUNT: Since we are sending only one hostname query at a time, we will set this to 1.

The Question section has the following format (more details are available in the RFC draft).



```
                                    1  1  1  1  1  1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                                               |
    /                     QNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                     QTYPE                     |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                     QCLASS                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Fig. 3: Fields of the Question section

QNAME: it contains multiple labels - one for each section of a URL. For example, for gmu.edu, there should be two labels for gmu and edu. Each label consists of a length octet (3 for gmu), followed by ASCII code octets (67 for g, 6D for m, 75 for u). This is repeated for each label of the URL. *The QNAME terminates with the zero-length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.*

QTYPE: Set to 1 because we are interested in only type A records. ***In this assignment, we will create queries with only type A records (NS, MX, and other types are not considered for queries).***

QCLASS: set to 1 for the Internet.

Your client program can first calculate the above fields for both sections and then concatenate them. Note that the data type of each field can be different (binary, integer, string, hex, *etc.*), but they should all be converted to hexadecimal to form a complete message. Make sure to remove any unnecessary white spaces or newline characters that might be part of your strings.

Send Query

Once you have prepared a query message, your client program will establish a socket connection with a DNS server. As we discussed in class, DNS uses UDP for transferring the message and chooses an appropriate port number for communication with the server. We will use Google's public DNS server in the programming assignment. You can learn about public DNS servers at https://www.lifewire.com/free-and-public-dns-servers-2626062. The IP address of Google's public DNS server is 8.8.8.8. You can use any other DNS server (for example, gmu's DNS server) for testing, but your assignment will be graded for only the 8.8.8.8 server.

DNS messages sent over UDP can be lost in the network. Your code should implement a timeout-based retry where it waits for 5 seconds and if the server does not respond within that time, it resends the query message. If no response is received after 3 attempts, it should print out an error message about the timeout.

Receive and Process Response

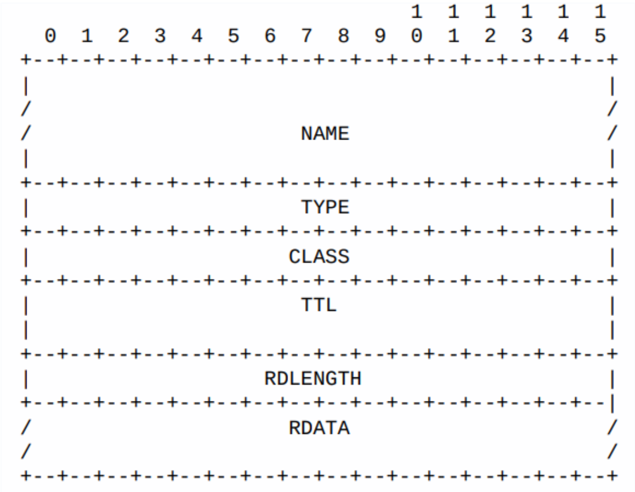
After your client sends the DNS query to the server, the server will reply back. Assuming your query is properly formatted and the server is able to understand your request, the response will follow the standard DNS response message format. If your query message is not properly formatted, you might receive an error message in response or not receive any reply at all! A good starting point here is to receive whatever the server is sending back in a buffer and analyze it.

A response can have all five sections shown in Fig.1. First, let us assume that this is a standard response, in which case, it will have the top three sections (Header, Question, and Answer). You can ignore the Authority and Additional sections when processing the response.

The Header section will be similar to that of the DNS query.

- ID will be the same as the query.
- QR will be set to 1 because it is a response.
- AA will depend on if the server is the authority for the domain or not (most likely it is not, and thus AA is 0).
- RCODE: response code will be 0 if no errors, and >0 otherwise.

As with the query format, make sure to go over the RFC specification (Section 4) to ensure each field is properly interpreted by your client. The Question section will be exactly the same as the query message. The Answer section will include one or more resource records (RRs). An RR has the following format:



```
                                    1  1  1  1  1  1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                                               |
    /                                               /
    /                      NAME                     /
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                      TYPE                     |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                     CLASS                     |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                      TTL                      |
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                   RDLENGTH                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
    /                     RDATA                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Fig. 4: Resource Record (RR) format

Some of the important fields are:

- NAME is the domain for which the IP address is resolved. *It uses a compressed format that can be ignored in your processing.*

- TYPE and CLASS are the same as query messages.
- TTL (Time To Live): specifies the time interval (in seconds) that the RR may be cached before being considered outdated.
- RDATA is the resolved IP address.

After your client program has parsed and processed the request, it should print <field, value> tuple for every field of the response message (shown below).

Your code should process all RRs received in the response. Although our queries are for only TYPE A records, the response may include any type of RR. For example, it is possible that the response includes one CNAME type RR and three type A RRs. Your code should process all of them.

Grading Rubric

Your print output should look something like this:

```
$> my-dns-client gmu.edu

Preparing DNS query ...
DNS query header = "......"
DNS query question section = "......"
Complete DNS query = "......"

Contacting DNS server ...
Sending DNS query ...
DNS response received (attempt 1 of 3)
Processing DNS response ...
----------------------------------------------------------------------------
header.ID = <value>
header.QR = <value>
header.OPCODE = <value>
....
....
question.QNAME = <value>
question.QTYPE = <value>
question.QCLASS = <value>
....
....
answer.NAME = <value>
answer.TYPE = <value>
....
....
answer.RDATA = <value>                    ## resolved IP address ##
```

```
...
(The above is repeated for all RRs in response)
-------------------------------------------------------------------------
```

The <value> can be printed in hex or decimal.

You can use the following five hostnames for testing your code. Your code will be evaluated for 5 additional hostnames. However, as long as your code takes care of processing query and response RRs for the following 5 hosts, no additional cases need to be implemented in your program. This means that your goal can be that your code works for the following 5 hosts exactly as expected (cross-check with dig and Wireshark).

1. www.cnn.com
2. www.gmu.edu
3. www.vt.edu
4. www.youtube.com
5. www.example.com

Note that the 8.8.8.8 DNS server to which you are sending the queries is not authoritative for any domain (*i.e.*, no NS records expected). Thus, you will primarily be concerned about processing A and CNAME type records in response.

If your code correctly implements the following steps and yields the correct/expected print output, you will get the corresponding points shown on the right. The expected print output is shown above. Your code will be tested for 5 randomly selected hostnames.

1. Handing the input hostname argument	[5 pts]
2. Create the DNS query	
a. Create and print the DNS header	[15 pts]
b. Create and print the Question section	[15 pts]
c. Print the entire query after converting to Hex	[15 pts]
3. Open and close socket	
a. Create socket	[5 pts]
b. Send the DNS query	[5 pts]
c. Receive the DNS response	[5 pts]
d. Close the socket	[5 pts]
4. Parse the response message	
a. Parse and print the response header	[15 pts]
b. Print the resolved IP address	[15 pts]

Debugging with Wireshark and Dig

A good way to debug your code is to use Wireshark and Dig to see how queries and responses are (exactly field-by-field) created. You can use >$ dig @8.8.8.8 <hostname> for pointing your dig to the 8.8.8.8 DNS server while collecting the query and response packets on Wireshark.

Policies and Submission

Programming Language

You can implement your client in C or Python. Use of any other language is not recommended. If you really want to use another language for implementation, you need the instructor's approval. You must get approval within one week of the release of this programming assignment. Approval will be provided in only rare cases.

Working with a Partner

This programming assignment can be done alone or in a team of two. A team cannot have more than two students. In fact, it is encouraged that you work on the programming assignments with a partner.

If you choose to do the programming assignments in a team of two, you have to maintain the same team for all programming assignments. This means you cannot choose a different team member for subsequent programming assignments. Hence, a recommended way is to form a team starting this assignment and keep working with the partner for all remaining programming assignments.

Note on Plagiarism

In this class, it is absolutely mandatory that you adhere to the honor code rules of GMU and the Computer Science Department. This also means (1) do not copy code from online resources and (2) your implementation should be purely based on your own thought process and conceived design, and not inspired by any other students or online resources, (3) you cannot copy your code or design from other students in the class.

We reserve the right to check your assignment with other existing assignments (from other students or online resources) for cheating using code-matching programs. Any violation of the honor code will be reported to the GMU honor committee, with a failing grade (F) in this course.

*** Do not put your code online on GitHub or other public repositories ***
Violation of this policy would be considered an honor code violation.

Grading, Submission, and Late Policy

- The standard late policy applies: Late penalty for this PA will be 15% for each day. Submissions that are late by 3 days or more will not be accepted
- This PA accounts for **8%** of your final grade
- You will submit your solution via Blackboard

What to Submit?

- Submit the following four files
    1. Your code in a zip archive file. If you simply include a pre-compiled executable binary and do not include your code files in the archive, no points will be given.
    2. A README.txt that explains how to compile your program. A standard way in which your code will be run and tested is through the "my-dns-client <host-name>" command. Include the command that your compiled code should be run and tested.
    3. An ANSWER.txt file that shows the command line output you got when you run your program for the 5 hosts mentioned above.
    4. A PARTNERS.txt file mentioning the name and GMU IDs of two students who worked on the project as a team.

  Both team members should submit these four pieces separately on Blackboard before the deadline.
- Your assignment will be tested and graded on a standard Linux machine. So, if your code requires any specific libraries, make sure to include that in the compilation instructions.