CS 455: Computer Communications and Networking

PA-1: DNS Client

Project description

In this lab, you will implement a DNS client that can query an existing DNS server for domain name to IP address translation. We discussed in class the operations of DNS and 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 command line. However, 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 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. 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 you message does not adhere to the specifications, DNS server will not be able to identify/understand your query.
2. Send 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 DNS response: the client should receive a response from the DNS server. It should then process the response based on the format of DNS response message. It should extract the necessary information (IP address and more) and display it to the user on command line.

We will now look at each of the three above pieces in details.

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>

An example of this would be -

$> my-dns-client gmu.edu

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 message can be found in the official DNS RFC document (https://tools.ietf.org/html/rfc1035#page-25). It is highly recommended that you go through 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 first two sections.

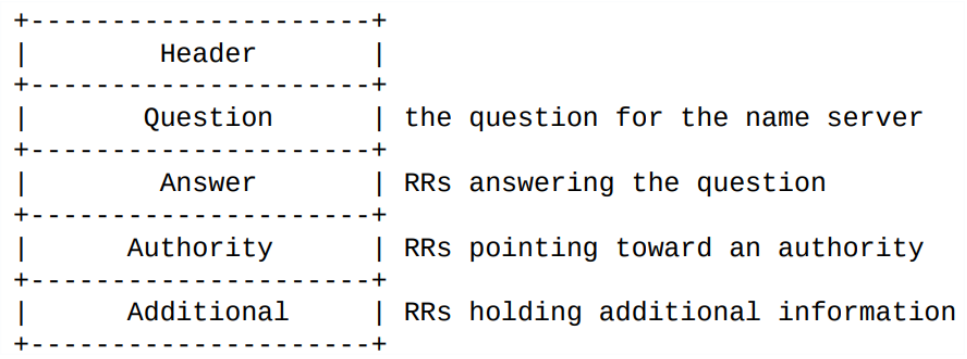

Fig. 1: DNS message sections

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

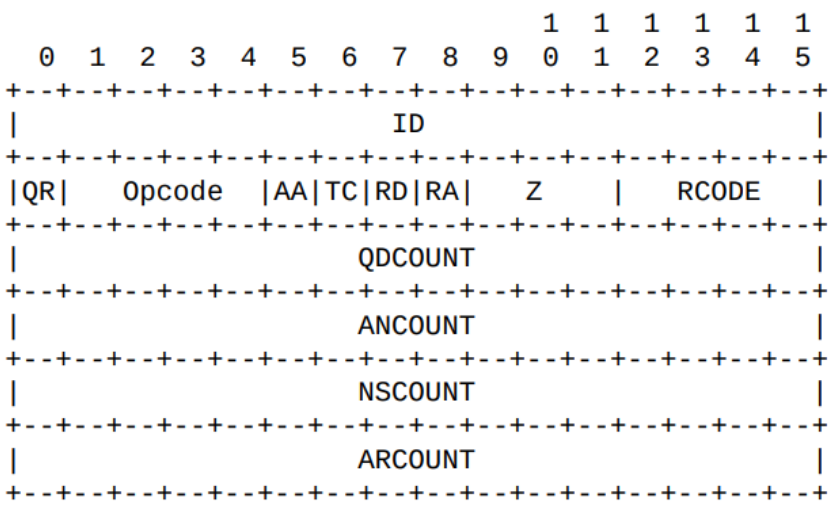

Fig. 2: Fields of Header section

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

- ID: A 16 bit id 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 response.
- OPCODE: We are only interested in standard query.
- RD: This bit is set if client wants the name server to recursively pursue the query. We will set this to 1.
- QDCOUNT: Since we are only sending one hostname query at a time, we will set this to 1.

The question section has the following format (more details available from the RFC draft).

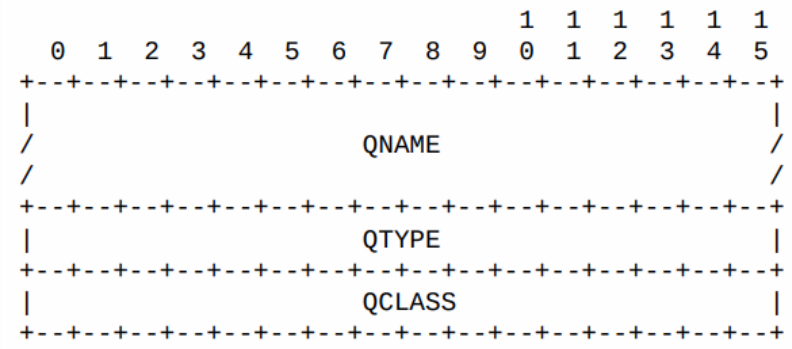

Fig. 3: Fields of question section

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 only interested in A type records. You can ignore NS, MX and CNAME type records for this assignment.

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 new line characters which 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 the class, DNS uses UDP for transferring the message. Choose 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 server at https://www.lifewire.com/free-and-public-dns-servers-2626062 (last updated 9/02/21). 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 only for 8.8.8.8 server.

DNS query 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 resend the query message. If no response is received after 3 attempts, this should print out an error message about timeout.

Receive and process response

After sending the DNS query to 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 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 analyzing 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 top three sections (header, question and answer).

The header section will be similar to that of 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 authority for the domain or not (most likely it is not, so 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 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:

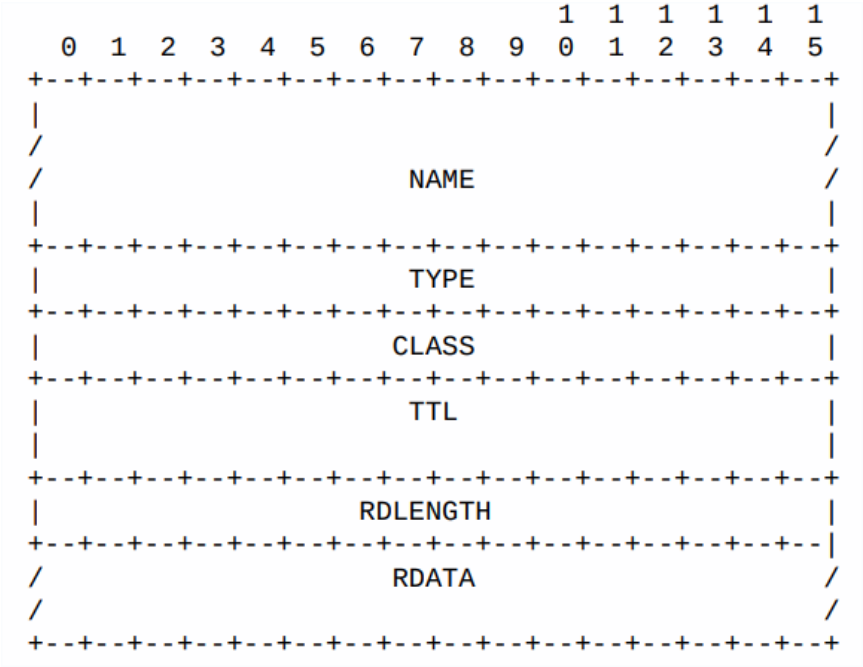

Fig. 3: Resource Record (RR) format

Some of the important fields are:

- NAME is the domain for which the IP address is resolved. It uses a compressed format which 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 considered outdated.
- RDATA is the resolved IP address.

After your client program has parsed and processed the request, your client program should print <field, value> tuple for every field of the response message. This would look something like:

$> my-dns-client gmu.edu
Preparing 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 ##
…. (include any authority or additional RRs received in the response also)

Debugging with wireshark

A good way to debug your client program is to run Wireshark and capture packets sent and received by your program. Wireshark will allow you to see the values of the fields as they are in network for your query and response message. This way, it can help you properly form the messages and debug any issues with formatting of the messages. Also, it can help you debug any error messages that server might be sending to your program in response.

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 other language for implementation, you need instructor's approval. You must get the approval within one week of release of this programming assignment. Approval will only be provided in rare cases.

Working with a partner

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

If you choose to do the programming assignments in 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 GMU and Computer Science Department honor code rules. 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 can 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 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

- Standard late policy applies - Late penalty for this lab will be 15% for each day. Submissions that are late by 2 days or more will not be accepted
- You will submit your solution via Blackboard
- Both team members should individually submit their identical assignments.

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 which explains how to compile your program. A standard way in which your code will be ran and tested is through “my-dns-client <host-name>” command. Include the command using which your compiled code should be ran and tested.
    3. An ANSWER.txt file which shows the command line output you got when you run your program to any three popular hostnames.
    4. A PARTNERS.txt file mentioning the name and GMU IDs of two students worked on the project as a team.

  Both team members should submit these four pieces separately on Blackboard before 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.
