Socket Programming: Domain Name System (DNS)

Project 2

(100 points)

Due Date: Friday November 4, 2022 by 5:00 PM

Team: The project is to be done in a team of at most 2 students. You *cannot* discuss your code/data with other classmates (*except* your project partner).

All submissions (including your code) will be checked for ***plagiarism*** against other submissions as well as the public Internet. Plagiarized submissions will be entitled to ***zero*** points.

In this project you will implement **Domain Name System** (DNS) client and server from scratch using the socket API.

Domain Name System (DNS)

There are two ways to identify a host—by a hostname and by an IP address. People prefer the more mnemonic hostname identifier, while routers prefer fixed-length, hierarchically structured IP addresses. In order to reconcile these preferences, we need a directory service that translates hostnames to IP addresses. This is the main task of the Internet's Domain Name System (DNS).

The DNS is:

1. a distributed database implemented in a **hierarchy of DNS servers**, and
2. an application-layer protocol that allows hosts to query the distributed database.

Resolving DNS Queries

In order for a host to be able to send an HTTP request message to a Web server – say www.ucdavis.edu – it must first obtain the IP address of www.ucdavis.edu. This is usually done as follows:

1. An application layer program (such as a web browser which is the client side of the DNS server) contacts a local DNS server known as a *resolver* ((1) in Figure 1). The resolver

maintains a cache of recently translated addresses. If the cache contains the address for the requested hostname, the resolver can return the answer immediately.

2. For hostnames that are not in the cache, the resolver would then need to traverse the distributed DNS infrastructure. The first point of contact is a root DNS server. Root servers typically have a static IP address (e.g., a.root-servers.net currently uses 198.41.0.4). These IP addresses serve as the entry point to the distributed DNS infrastructure. Local DNS resolver sends a DNS query to a root server to get the IP address of the TLD server ((2) in Figure 1).
3. After getting the IP address of the TLD server ((3) in Figure 1), the DNS resolver would then send a query to the TLD DNS server requesting the address of the authoritative DNS server ((4) in Figure 1).
4. After getting the address of the authoritative DNS server ((5) in Figure 1), the DNS resolver would then send a request to the authoritative DNS server ((6) in Figure 1) and in response it would receive the IP address of the initially requested hostname ((7) in Figure 1).
5. The application eventually receives a reply from the local DNS server, which includes the IP address for the requested hostname ((8) in Figure 1).
6. Once the Client DNS receives the IP address from the local DNS server, it can establish a TCP connection to the HTTPs server process located at port 443 at that IP address.

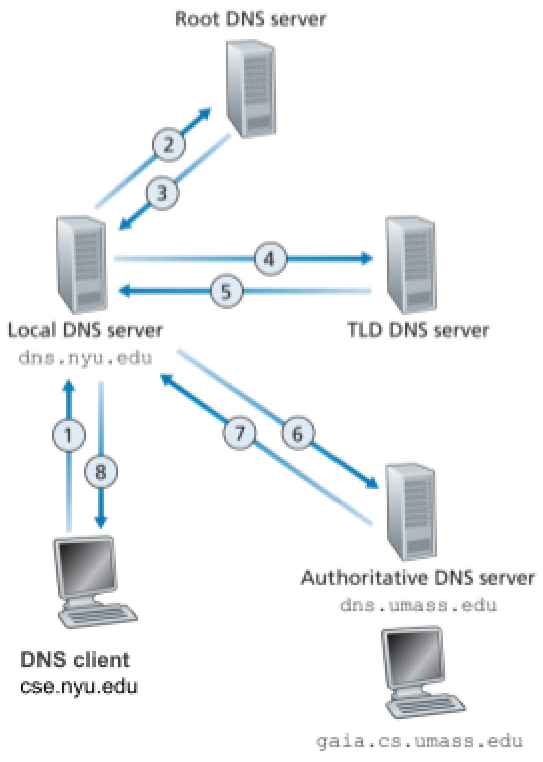
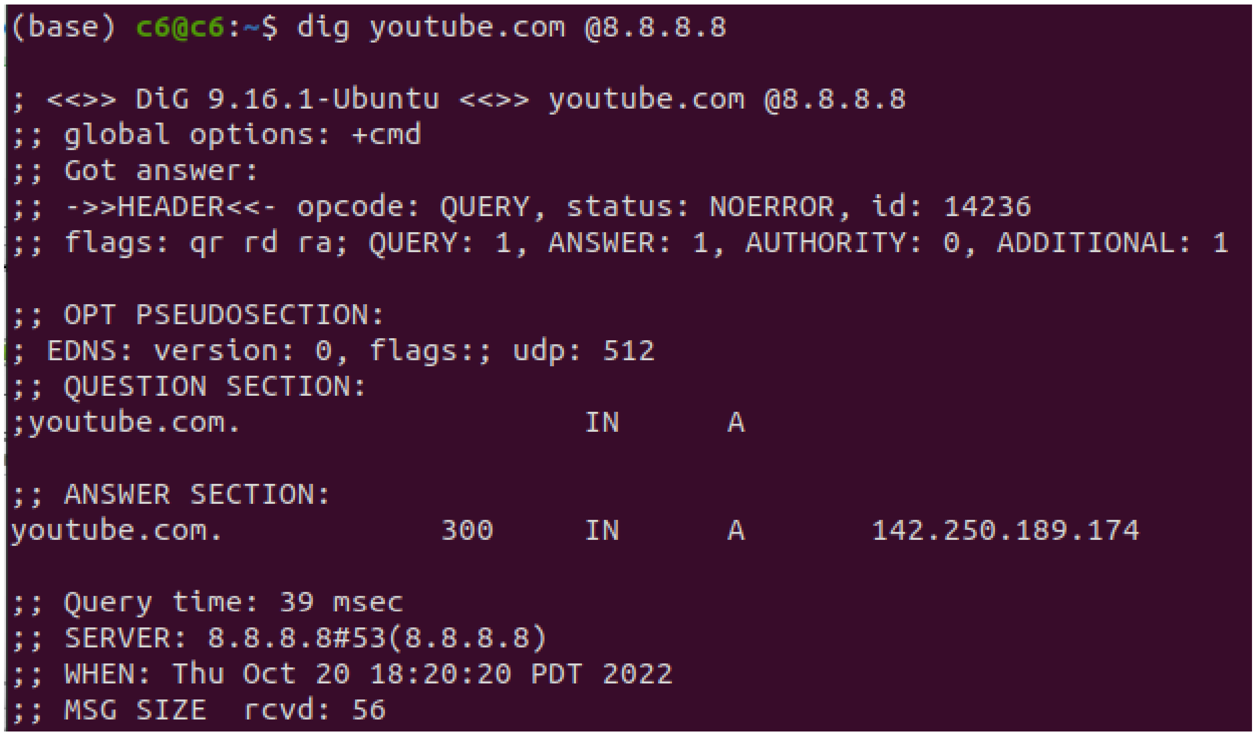


Figure 1: Workflow of a DNS request

Understanding DNS request and response structure using Wireshark

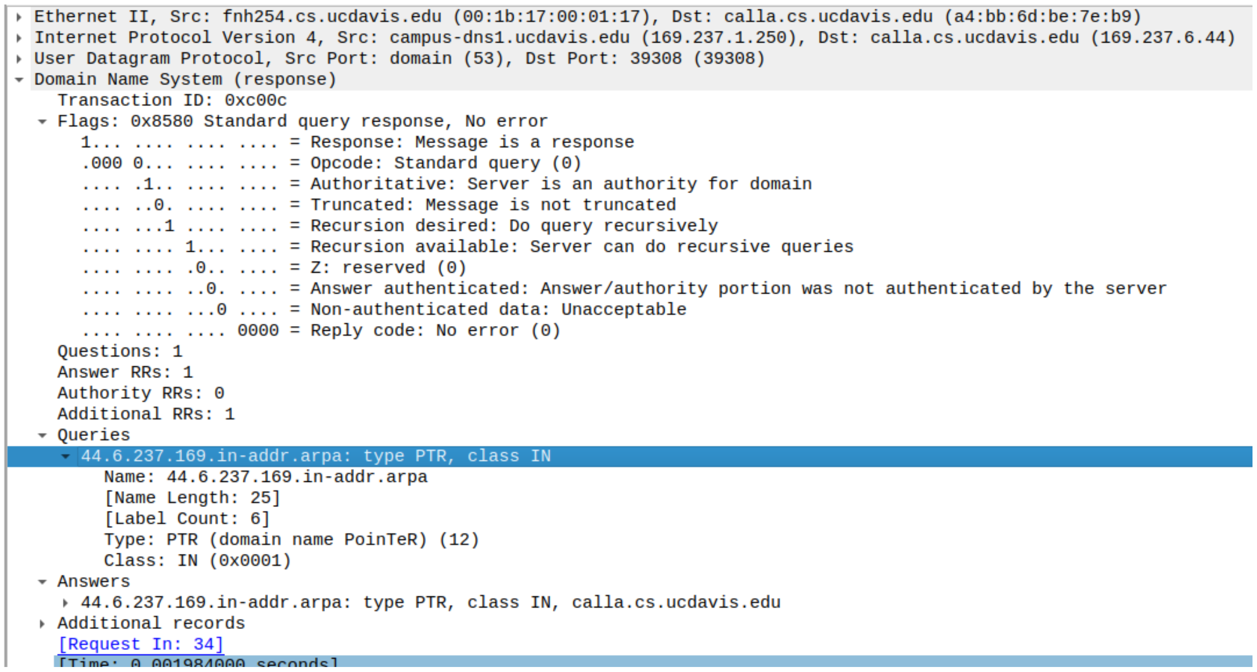
You can use Wireshark to understand the structure of a simple DNS request and response. To see the structure of a DNS request on your local computer, open your wireshark, and start capturing network traffic.
Then, open a terminal and use **dig** to resolve the IP address of a web page. For example:
dig youtube.com @8.8.8.8

This command basically sends a dns query to one of the DNS resolvers with IP address 8.8.8.8. As soon as you push the enter, you will see the resolved IP address in **ANSWER SECTION** which is 142.250.189.174:



```
(base) c6@c6:~$ dig youtube.com @8.8.8.8

; <<>> DiG 9.16.1-Ubuntu <<>> youtube.com @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14236
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;youtube.com.                   IN      A

;; ANSWER SECTION:
youtube.com.            300     IN      A       142.250.189.174

;; Query time: 39 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Oct 20 18:20:20 PDT 2022
;; MSG SIZE  rcvd: 56
```

Now, if you stop the Wireshark and filter DNS packets, you will see the list of DNS requests and responses. The size of the DNS response is usually more than the DNS request because it includes the answers to the query. You can select the DNS packets and investigate their structure as follows.



```
▸ Ethernet II, Src: fnh254.cs.ucdavis.edu (00:1b:17:00:01:17), Dst: calla.cs.ucdavis.edu (a4:bb:6d:be:7e:b9)
▸ Internet Protocol Version 4, Src: campus-dns1.ucdavis.edu (169.237.1.250), Dst: calla.cs.ucdavis.edu (169.237.6.44)
▸ User Datagram Protocol, Src Port: domain (53), Dst Port: 39308 (39308)
▾ Domain Name System (response)
    Transaction ID: 0xc00c
  ▾ Flags: 0x8580 Standard query response, No error
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .1.. .... .... = Authoritative: Server is an authority for domain
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... 1... .... = Recursion available: Server can do recursive queries
      .... .... .0.. .... = Z: reserved (0)
      .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
      .... .... ...0 .... = Non-authenticated data: Unacceptable
      .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▾ 44.6.237.169.in-addr.arpa: type PTR, class IN
        Name: 44.6.237.169.in-addr.arpa
        [Name Length: 25]
        [Label Count: 6]
        Type: PTR (domain name PoinTeR) (12)
        Class: IN (0x0001)
  ▾ Answers
    ▸ 44.6.237.169.in-addr.arpa: type PTR, class IN, calla.cs.ucdavis.edu
  ▸ Additional records
    [Request In: 34]
```

Note how this structure follows the format of a DNS message explained in Section 4 of RFC 1035,

A. Implement DNS Client (30 pints)

In the first part of the project, you will implement a **DNS Client** ((1&8) in Figure 1). The client should send a DNS request to the IP address of three different DNS resolvers provided in Table 1 to query **tmz.com**. After getting the IP address of the tmz.com's web server, we ask that you initiate a TCP connection to the HTTPs server process located at port 442 at that IP address.

IP	**Location**
91.245.229.1 **Alternative**: 46.224.1.42 185.161.112.34	Iran
169.237.229.88 **Alternative**: 168.62.214.68 104.42.159.98	USA
136.159.85.15 **Alternative**: 184.94.80.170 142.103.1.1	Canada

Table 1: IP address of three DNS resolvers located in different regions

Since we want you to implement DNS from scratch, you are NOT allowed to use any libraries/methods that simplify DNS implementation! For example, don't use gethostbyname, getaddrinfo, or other similar methods. If you have any doubt about which libraries/methods you may use, please ask us!

Workflow of your program (15 pints):
Your DNS client should follow this sequence:

1. Build a DNS query.
   a. Build your DNS request, according to RFC 1035, Section 4. Pack DNS header and query in a payload buffer.
2. UDP Socket Calls

   a. After your DNS request payload is built, use sendto() and recvfrom() to send and receive from the public DNS resolvers provided in Table 1. Note that DNS uses UDP port 53.
   b. If you don't get a response within 10 seconds, try another IP address from the alternative section of Table 1 in the same region.
3. Receive the response per request from the DNS server
   a. When you receive your response from the call to recvfrom(), you will have to unpack the response.
      Hint: Since the header is always the first 12 bytes of your response, you should unpack the header first.
4. Parse response message: Now that you have unpacked and parsed the response header, you can parse the rest of the Resource Records (RRs) in the Answer section of the response, to figure out the type of the DNS record and the IP address. Print out this IP address along with the corresponding domain name in the output.
5. Returning the resolved IP address for an A record: Once you find the IP address, initiate a TCP connection to the IP address at port 80 at that using socket API. You need to craft and send a HTTP GET request using socket API. The standard format of the HTTP message is explained in page 101-103 of the textbook. You can see this example of how to create an HTTP request on port 80. After getting the response from the HTTP server, save it as an HTML file. Upload this file along with your code.
   Note: Because most of the sites these days are HTTPS only, sending an HTTP GET request to those sites will result in a 301 redirect page being shown. This is normal and expected behavior.

Report Details:

1. Describe in detail the DNS request and response header format in your implementation. (5 points)
2. Compute the RTT between your DNS client to each of the public DNS resolvers. Do you notice any meaningful differences across different DNS resolvers? Explain. (5 points)
3. Compute the RTT between your HTTP client to the HTTP server of the resolved hostname. (5 points)

B. Implement DNS Server (40 points)

In the second part of the project, you will implement the local DNS resolver using the socket API to find the IP address of tmz.com. You will implement the iterative strategy to resolve input DNS queries by consulting the root name server, the TLD name server, and finally the authoritative DNS server to get the IP address of the requested hostname.

Workflow of your program (30 points):
Roughly, your server should follow this sequence:

1. Build a DNS query.
    a. Build your DNS request, according to RFC 1035, Section 4. Pack DNS header and query in a payload buffer.

3. UDP Socket Calls
    b. After your request is built you will need to use sendto() and recvfrom() to send and receive from the server. DNS uses UDP port number 53.
    c. Send that query to **one of the root servers** (the IP address of root servers are provided in Table 2) and wait for a response. If you wait too long, move to the next root.

4. Receive responses per request from the DNS server hierarchy: When you receive your response from the call to recvfrom(), you will have to unpack the response.

5. Parse response message: Now that you have your response headers, you can parse the rest of the Resource Record in the response, to figure out the type of the DNS record.

6. Continue this process as you work your way down the DNS hierarchy. In the process of parsing responses from root server and TLD server, check the Resource Records (RRs) in the Additional sections of the DNS response. Print out the IP address of each of the DNS servers you are calling.
7. Returning the resolved IP address for an A record: Once your DNS server has received the IP address from the authoritative name server, return it to your DNS client.

Report Details:

1. Compute the RTT from your local DNS server to each of the DNS servers including the root name server, the TLD name server, and the authoritative DNS server of nytimes.com. (10 points)

a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30
b.root-servers.net	199.9.14.201, 2001:500:200::b
c.root-servers.net	192.33.4.12, 2001:500:2::c
d.root-servers.net	199.7.91.13, 2001:500:2d::d
e.root-servers.net	192.203.230.10, 2001:500:a8::e
f.root-servers.net	192.5.5.241, 2001:500:2f::f
g.root-servers.net	192.112.36.4, 2001:500:12::d0d
h.root-servers.net	198.97.190.53, 2001:500:1::53
i.root-servers.net	192.36.148.17, 2001:7fe::53
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30
k.root-servers.net	193.0.14.129, 2001:7fd::1
l.root-servers.net	199.7.83.42, 2001:500:9f::42
m.root-servers.net	202.12.27.33, 2001:dc3::35

Table 2: List of Root Servers

C. Implement DNS Server with Caching (30 points)

In the third part of the project, you are asked to implement a cache for your local DNS server. Before implementing the cache, use your implemented local DNS server to send DNS requests to resolve the following host names.

1. youtube.com
2. facebook.com
3. tmz.com
4. nytimes.com
5. cnn.com

A. Report the time it takes to resolve each of these host names from your local DNS server. (5 points)

When you are parsing the DNS response from the authoritative DNS, there is a field named "TTL" in the DNS Resource record. As described in the RFC 1035, "TTL is a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the IP address only be used for the transaction in progress, and should not be cached."

B. Report the TTL value in the DNS responses to each of these host names. (5 points)

Next, implement a cache in your local DNS server. If a DNS client sends a DNS query to your local DNS server for one of the host names, the local DNS server should check its cache before sending any request to the root DNS server. If the answer is in the cache and the TTL is not expired, return the response directory from the cache. If it is expired, remove it from the cache and contact the root DNS server to start the resolution process. (10 points)

C. Report the time it takes to resolve each of these host names by your DNS client from your local DNS server when it did not implement the cache. (5 points)

D. Report the time it takes to resolve each of these host names by your DNS client from your local DNS server when it did implement the cache (and the answers are already in the cache). (5 points)

Report:

report_[name1]_[student_id1]_[name2]_[student_id2].pdf

At the beginning of the page, specify the following:

1. Full Name of student 1 (Student ID) (Discussion Group)
2. Full name of student 2 (Student ID) (Discussion Group)
3. Name of the submitted code

Uploads:

Please upload the source code for each part separately:

NOTE: please use comments to explain each function in your source code. It will make it easier for TAs to understand your code and give you better points :) Use meaningful function names.

```
PartA_[name1]_[student_id1]_[name2]_[student_id2].py
PartB_[name1]_[student_id1]_[name2]_[student_id2].py
PartC_[name1]_[student_id1]_[name2]_[student_id2].py
```

You need to upload a txt file including the HTTP response for Part A:

```
Partb_http_[name1]_[student_id1]_[name2]_[student_id2].txt
```

We should be able to run your code using the command line by providing the name of the source code file and a domain name. For example to know the IP address of facebook.com using your implemented local DNS resolver we should see the IP address in the output using this **command line**:

```
Python3 PartA_[name1]_[student_id1]_[name2]_[student_id2].py facebook.com
```

The expected output is:

Domain: facebook.com
Root server IP address: 198.41.04
TLD server IP address: X.X.X.X (depends on the response from the root server)
Authoritative server IP address: X.X.X.X (depends on the response from the TLD DNS server)
HTTP Server IP address: X.X.X.X (depends on the response from the Authoritative DNS server)

Testing Environment:

All submissions will be tested on Python 3+.

Late Submission Policy:

No late submissions are allowed. However, if you barely miss the deadline, you can get partial points upto 24 hours. The percentage of points you will lose is given by the equation below. This will give you partial points up to 24 hours after the due date and penalizes you less if you narrowly miss the deadline.

$$Total\ Marks\ you\ get\ =\ (Actual\ Marks\ you\ would\ get\ if\ NOT\ late)\ \times\ \left[1\ -\ \frac{hours\ late}{24}\right]$$

Late Submissions (later than 24 hours from the due date) will result in zero points, *unless you have our prior permission or documented accommodation*.