Problem 1

Suppose you have a new computer just set up. `dig` is one of the most useful DNS lookup tool. You can check out the manual of `dig` at `http://linux.die.net/man/1/dig`. A typical invocation of `dig` looks like: `dig @server name type`.

Suppose that on April 19, 2017 at 15:35:21, you have issued "`dig google.com a`" to get an IPv4 address for `google.com` domain from your caching resolver and got the following result: (If a user just types "dig google.com" the default is type=A)

```
; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17779
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; QUESTION SECTION:
;google.com.                IN     A

;; ANSWER SECTION:
google.com.           239    IN     A      172.217.4.142

;; AUTHORITY SECTION:
google.com.           55414  IN     NS     ns4.google.com.
google.com.           55414  IN     NS     ns2.google.com.
google.com.           55414  IN     NS     ns1.google.com.
google.com.           55414  IN     NS     ns3.google.com.

;; ADDITIONAL SECTION:
ns1.google.com.       145521 IN     A      216.239.32.10
ns2.google.com.       215983 IN     A      216.239.34.10
ns3.google.com.       215983 IN     A      216.239.36.10
ns4.google.com.       215983 IN     A      216.239.38.10

;; Query time: 81 msec
;; SERVER: 128.97.128.1#53(128.97.128.1)
;; WHEN: Wed Apr 19 15:35:21 2017
;; MSG SIZE rcvd: 180
```

1. What is the discovered IPv4 address of `google.com` domain?
2. If you issue the same command 1 minute later, how would "ANSWER SECTION" look like?
3. When would be the earliest (absolute) time the caching resolver would contact one of the `google.com` name servers again? (for issuing the same command "dig google.com a")
4. When would be the earliest (absolute) time the caching resolver would contact one of the `.com` name servers? (for issuing the same command "dig google.com a")

Problem 1 continued on next page...

1. The discovered IPv4 address of the `google.com` domain is 172.217.4.142
2. google.com. 179 IN A 172.217.4.142 (Subtracting 60s from 239)
3. Wed Apr 19 15:39:20 2017 (Adding +239s to the WHEN timestamp)
4. Thu Apr 20 06:58:55 2017 (Adding +55414s to the WHEN timestamp)

Problem 2

Suppose that you walked into Boelter Hall and get connected to `CSD` WiFi network, which automatically gave you IP address of the local caching resolver. However, initially, it doesn't allow you to do anything unless you type your username and password in a popup window (or if you try to go to any website in your browser).

1. Explain a mechanism of how does the "`CSD`" network achieve this / which features of DNS/HTTP make it possible.

The "`CSD`" network achieves this by running a software program on the local caching resolver at the given IP address given by the WiFi network. The caching resolver is a server process and thus has the capability to run any program, including one that can monitor who has access to the WiFi. Because all end users must go through the local caching resolver in order to fetch data from other servers, the caching resolver can choose to only service the stub resolver requests from authenticated users. The server must keep the state of who is logged in and only perform requests to the authoritative servers on behalf of authenticated users. Where HTTP comes in is the mechanism that uniquely identifies each end user. Each user or stub resolver can be uniquely identified with cookies. Clients have a cookie file that attaches an ID for the user in the HTTP request. Thus, the caching resolver now can link authentication successes or failures with cookie IDs for each user. If a user authenticates successfully, the caching resolver can mark that user as authenticated in its own server process and from then on perform DNS queries on behalf of the user, allowing them to access the rest of the Internet. If the user is not authenticated, the caching resolver can continually prompt the user to login and will not perform any DNS queries for this user. Note that the small details of authenticating the user need not run on the caching resolver; instead, the caching resolver can simply pass along the information to a trusted server that runs the authentication software which responds back to the caching resolver with an authentication success or failure.

Problem 3

Same context as Problem 2. After you successfully logged in, you can start using the Internet. Suppose the caching resolver has just rebooted and its cache is completely empty; RTT between your computer and the caching resolver is $10ms$ and RTT between the caching resolver and any authoritative name server is $100ms$; all responses have TTL 12 hours.

1. If you try to go to `ucla.edu`, what would be minimum amount of time you will need to wait before your web browser will be able to initiate connect to the UCLA's web server?
2. What would be the time, if a minute later you will decide to go to `ccle.ucla.edu`?
3. What would be the time, if another minute later you will decide to go to `piazza.com`?
4. What would be the time, if another minute later you will decide to go to `gradescope.com`?

1. Assuming the cache is completely empty, there will be a couple of round trips between servers to find the IP address of `ucla.edu`. Here is each step needed starting from the user's initial request:

- stub resolver requests IP address of `www.ucla.edu` from the caching resolver (5 ms, half of initial RTT)
- caching resolver requests from root server and root server responds with pointers to the `.edu` server (100 ms)
- caching resolver requests from `.edu` server and `.edu` server responds with pointers to the `ucla.edu` server (100 ms)
- caching resolver requests IP address from `ucla.edu` server and `ucla.edu` server responds with the final IP address for `www.ucla.edu` (100 ms)
- caching resolver finally returns the IP address for `www.ucla.edu` to your computer (5 ms, other half of initial RTT)
  Total time required $= 5 + 100 + 100 + 100 + 5 = 310$ ms

2. Because 1 minute $<$ TTL time of 12 hours, the IP address of `ucla.edu` is cached. The caching resolver can query the `ucla.edu` server directly for pointers to `ccle.ucla.edu` because `ccle.ucla.edu` is a subdomain of `ucla.edu` due to the delegation property of the general DNS structure. Here we assume that `ccle.ucla.edu` is its own nameserver.

- stub resolver requests IP address of `www.ccle.ucla.edu` from the caching resolver (5 ms, half of initial RTT)
- caching resolver requests IP address from `ucla.edu` server (cached) which responds with pointers to the `ccle.ucla.edu` server (100 ms)
- caching resolver requests IP address from the `ccle.ucla.edu` server and the server responds with the final IP address for `www.ccle.ucla.edu` (100 ms)
- caching resolver returns IP address for `www.ccle.ucla.edu` to your computer (5 ms, other half of initial RTT)
  Total time required $= 5 + 100 + 100 + 5 = 210$ ms

3. Although we've cached the IP address of the `.edu` server, we now need to find the `.com` server. Thus, we have to talk to the root server.

- stub resolver requests IP address of `www.piazza.com` from the caching resolver (5 ms, half of initial RTT)
- caching resolver requests from root server and root server responds with pointers to the `.com` server (100 ms)
- caching resolver requests from `.com` server and `.com` server responds with pointers to the `piazza.com` server (100 ms)
- caching resolver requests IP address from `piazza.com` server and `piazza.com` server responds with the final IP address for `www.piazza.com` (100 ms)
- caching resolver finally returns the IP address for `www.piazza.com` to your computer (5 ms, other half of initial RTT)
  Total time required $= 5 + 100 + 100 + 100 + 5 = 310$ ms

4. Because we've cached the IP address of the authoritative `.com` server, the caching resolver can immediately query the server to avoid querying the root server.

- stub resolver requests IP address of `www.gradescope.com` from the caching resolver (5 ms, half of initial RTT)
- caching resolver requests IP address from `.com` server (cached) which responds with pointers to the `gradescope.com` server (100 ms)
- caching resolver requests IP address from the `gradescope.com` and the server responds with the final IP address for `www.gradescope.com` (100 ms)
- caching resolver returns IP address for `www.gradescope.com` to your computer (5 ms, other half of initial RTT)
  Total time required $= 5 + 100 + 100 + 5 = 210$ ms

Problem 4

How does SMTP mark the end of a message body? How about HTTP? Can HTTP use the same method as SMTP to mark the end of a message body? Explain.

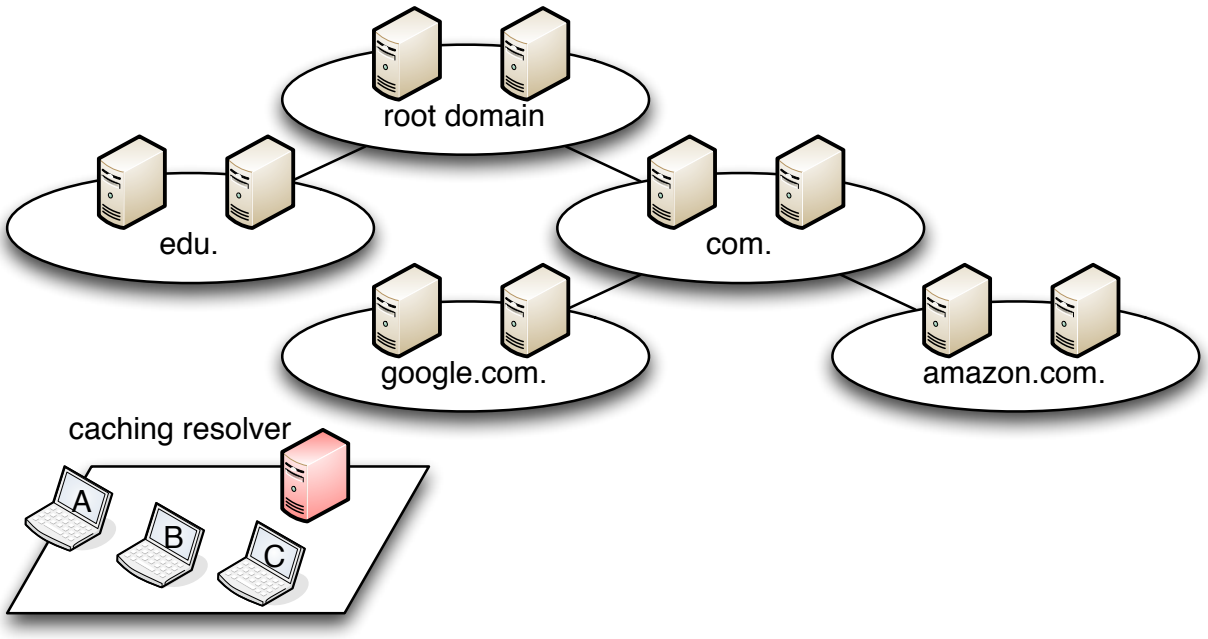
SMTP marks the end of a message body with $< CR >< LF > . < CR >< LF >$, which is a line containing only a period.
HTTP marks the end of a message body with a Content-Length header field that specifies the length of the message body.
HTTP can't use the same method as SMTP to mark the end of a message body because an HTTP message could be written in binary, while SMTP must be written in a strict, 7-bit ASCII textual format. While $< CR >< LF > . < CR >< LF >$ is enough to specify an unambiguous end of an SMTP message body, HTTP may have trouble representing this or may run into ambiguous representations of attempting to end the message body. Thus, HTTP chooses to attach extra header information to eliminate any ambiguities about the length of its message body.

Problem 5

Consider the following environment with a local DNS caching resolver and a set of authoritative DNS name servers.

Assume that initially,

- the caching resolver cache is empty,
- TTL values for all records is 1 hour,
- RTT between stub resolvers (hosts A, B, and C) and the caching resolver is 20 ms,
- RTT between the caching resolver and any of the authoritative name servers is 150 ms,
- There are no packet losses,
- All processing delays are 0 ms



1. At T=0 min, Host-A sends a query for A record for amazon.com, and after receiving the answer sends a query for A record for www.amazon.com. How long did it take to receive all the answers?
2. At T=40 min, Host-B sends a query for MX record for google.com that returns

```
google.com.           3600    IN      MX      10 primary.google.com.
google.com.           3600    IN      MX      30 backup.google.com.
primary.google.com.   3600    IN      A       74.125.28.27
backup.google.com.    3600    IN      A       173.194.211.27
```

   (Similar to NS records, the DNS server may return glue A/AAAA records in addition to the requested MX records.) How long did it take to get the answer?
3. At T=70 min, Host-C sends a query for AAAA (IPv6) record for mail.google.com, following at T=75 mins with a query for AAAA (IPv6) record for hangout.google.com. How long did it take for Host-C to receive each of the answers (i.e., relative to T=70min for the first, and relative to T=75 mins for the second)?
4. List DNS records that the caching resolver has at T=90 minutes

Problem 5 continued on next page...

1. Sum up each RTT value associated with each step of the DNS lookup for the A record for `amazon.com` and the A record for `www.amazon.com`

   - host-A requests A record for `amazon.com` from caching resolver (10 ms, half of initial RTT)
   - caching resolver requests from root server and root server responds with pointers to the `.com` server (150 ms)
   - caching resolver requests from `.com` and this server responds with pointers to the `amazon.com` server (150 ms)
   - caching resolver returns the IP address of `amazon.com` to host-A (10 ms, half of initial RTT)
   - host-A requests the A record for `www.amazon.com` from caching resolver (10 ms, half of initial RTT)
   - since the IP address of `amazon.com` is cached, a query for the A record for `www.amazon.com` is obtained by having the cache resolver directly request the IP address for `www.amazon.com` from the `amazon.com` authoritative server (150 ms)
   - finally, the caching resolver returns the final A record for `www.amazon.com` to host-A (10 ms, half of initial RTT)
   - total time = 490 ms

2. Since 40 min < TTL of 1 hour, the IP address for the `.com` server is cached. Here is each step of host-B sending a query for the MX record for google.com:

   - host-B requests MX record for `google.com` from caching resolver (10 ms, half of initial RTT)
   - caching resolver requests from `.com` server and server responds with pointers to the `google.com` server (150 ms)
   - caching resolver requests the MX record from the `google.com` server which, as specified in the question, returns 4 records including the MX records which are all cached in the caching resolver at around T = 40 min (150 ms)
   - finally, the caching resolver returns the MX record for `google.com` to host-B (10 ms, half of initial RTT)
   - total time = 320 ms

3. Since (70 - 40) < TTL of 1 hour and (75 - 40) < 1 TTL of hour, the IP address for the `google.com` server is cached during both of host-C's queries. Here we assume that the `google.com` server has records for `mail.google.com` and `hangout.google.com` because the picture does not show any separate nameservers below `google.com`.

   (a) Here is each step of host-C sending a query for `mail.google.com`:

   - host-C requests AAAA record for `mail.google.com` from caching resolver (10 ms, half of initial RTT)
   - caching resolver requests from the `google.com` server and the server responds with the AAAA record for `mail.google.com` (150 ms)
   - caching resolver returns the AAAA record to host-C (10 ms, half of initial RTT)
   - total time = 170ms

   (b) Here is each step of host-C sending a query for `hangout.google.com`:

   - host-C requests AAAA record for `hangout.google.com` from caching resolver (10 ms, half of initial RTT)
   - caching resolver requests from the `google.com` server and the server responds with the AAAA record for `hangout.google.com` (150 ms)
   - caching resolver returns the AAAA record to host-C (10 ms, half of initial RTT)
   - total time = 170ms

Problem 5 continued on next page...

4. DNS records in caching resolver at T = 90 minutes:

- google.com. 600 IN A xxx.xxx.x.xxx
  - TTL = 600 because at T = 90 min, there are only 10 minutes left and $10 * 60 = 600$
  - The IP address for the `google.com` server is arbitrary
- google.com. 600 IN MX 10 primary.google.com.
  - TTL = 600 because at T = 90 min, there are only 10 minutes left and $10 * 60 = 600$
- google.com. 600 IN MX 30 backup.google.com.
- primary.google.com. 600 IN A 74.125.28.27
- backup.google.com. 600 IN A 173.194.211.27
- mail.google.com. 2400 IN A xxx.xxx.x.xxx
  - TTL = 2400 because at T = 90 min, there are 40 minutes left and $40 * 60 = 2400$
- hangout.google.com. 2700 IN A xxx.xxx.x.xxx
  - TTL = 2700 because at T = 90 min, there are 45 minutes left and $45 * 60 = 2700$