Problem 1

Suppose that you walked into Boelter Hall and get connected to CSD WiFi network, which automatically gave you IP address of the local DNS server. Suppose the local DNS server has just rebooted and its cache is completely empty; RTT between your computer and the local DNS serverr is 10ms and RTT between the caching resolver and any authoritative name server is 100ms; all responses have TTL 12 hours. Suppose the DNS lookup follows iterative searching.

(a) 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 UCLAs web server? Suppose the location of UCLAs web server is delegated to ucla.edu name server.

(b) What would be the time, if a minute later you will decide to go to `ccle.ucla.edu`? Suppose `ccle.ucla.edu` happen to be delegated to ucla.edu name server.

(a) 310ms

(b) 110ms

Problem 2

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, 2019 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:

```
; <<>> 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.             12412   IN      NS      ns4.google.com.
google.com.             12412   IN      NS      ns2.google.com.
google.com.             12412   IN      NS      ns1.google.com.
google.com.             12412   IN      NS      ns3.google.com.

;; ADDITIONAL SECTION:
ns1.google.com.         13462   IN      A       216.239.32.10
ns2.google.com.         13462   IN      A       216.239.34.10
ns3.google.com.         13462   IN      A       216.239.36.10
ns4.google.com.         13462   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 2019
;; MSG SIZE rcvd: 180
```

(a) What is the discovered IPv4 address of `google.com` domain?

(b) If you issue the same command 1 minute later, how would "ANSWER SECTION" look like?

(c) If the client keeps issuing `dig google.com A` every second, when would be the earliest (absolute) time the local DNS server would contact one of the `google.com` name servers again?

(d) If the client keeps issuing `dig google.com A` every second, when would be the earliest (absolute) time the local DNS server would contact one of the `.com` name servers?

Problem 2 continued on next page...

(a) `172.217.4.142`

(b) `google.com.  179 IN A 172.217.4.142`

(c) (`Wed Apr 17 15:35:21 2019` + 239 sec)

(d) (`Wed Apr 17 15:35:21 2019` + 12412 sec)

Problem 3

Suppose Bob joins a BitTorrent torrent, but he does not want to upload any data to any other peers (so called free-riding).

(a) Bob claims that he can receive a complete copy of the file that is shared by the swarm. Is Bob's claim possible? Why or why not?

(b) Bob further claims that he can further make his "free-riding" more efficient by using a collection of multiple computers (with distinct IP addresses) in the computer lab in his department. How can he do that?

(a) Yes. His first claim is possible, as long as there are enough peers staying in the swarm for a long enough time. Bob can always receive data through optimistic unchoking by other peers.

(b) His second claim is also true. He can run a client on each host, let each client free-ride, and combine the collected chunks from the different hosts into a single file. He can even write a small scheduling program to make the different hosts ask for different chunks of the file. This is actually a kind of Sybil attack in P2P networks.

Problem 4

Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

In a NAK only protocol, the loss of packet x is only detected by the receiver when packet *x+1* is received. That is, the receivers receives *x-1* and then *x+1*, only when *x+1* is received does the receiver realize that x was missed. If there is a long delay between the transmission of x and the transmission of *x+1*, then it will be a long time until x can be recovered, under a NAK only protocol.
On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent – a significant reduction in feedback in the NAK-only case over the ACK-only case.

Problem 5

Consider the GBN protocol with a sender window size of 6 and a sequence number range of 1,024. Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that the medium does not reorder messages. Answer the following questions:

(a) What are the possible sets of sequence numbers inside the senders window at time t? Justify your answer.

(b) What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time t? Justify your answer.

(a) Here we have a window size of N=4. Suppose the receiver has received packet k-1, and has ACKed that and all other preceding packets. If all of these ACK's have been received by sender, then sender's window is [k, k+N-1]. Suppose next that none of the ACKs have been received at the sender. In this second case, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window is thus [k-N,k-1]. By these arguments, the senders window is of size 4 and begins somewhere in the range [k-N,k].

(b) If the receiver is waiting for packet k, then it has received (and ACKed) packet k-1 and the N-1 packets before that. If none of those N ACKs have been yet received by the sender, then ACK messages with values of [k-N,k-1] may still be propagating back.Because the sender has sent packets [k-N, k-1], it must be the case that the sender has already received an ACK for k-N-1. Once the receiver has sent an ACK for k-N-1 it will never send an ACK that is less that k-N-1. Thus the range of in-flight ACK values can range from k-N to k-1.

Note: In very extreme case where RTT vary from much lower than RTO to much higher than RTO, the ACK k-N-1 could also happen to be propagating back. We accept both answers.