Problem 1

Consider a DASH system for which there are N video versions (at N different rates and qualities) and N audio versions (at N different rates and qualities). Suppose we want to allow the player to choose at any time any of the N video versions and any of the N audio versions.

(a) If we create files so that the audio is mixed in with the video, so server sends only one media stream at given time, how many files will the server need to store (Each a different URL)?

(b) If the server instead sends the audio and video streams separately and has the client synchronize the streams, how many files will the server need to store?

(a) N files, under the assumption that we do a one-to-one matching by pairing video versions with audio versions in a decreasing order of quality and rate.

(b) 2N files.

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, 2018 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.          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
```

(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) When would be the earliest (absolute) time the caching resolver 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 caching resolver 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 15:39:20 2018` (`Wed Apr 17 15:35:21 2018` + 239 sec)

(d) `Thu Apr 18 06:58:55 2017` (`Wed Apr 17 15:35:21 2018` + 55414 sec)

Problem 3

The sender side of *rdt3.0* simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the acknum field of an acknowledged packet. Suppose that in such circumstances, *rdt3.0* were simply to retransmit the current data packet. Would the protocol still work? (Hint: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the *nth* packet is sent, in the limit as n approaches infinity).

The protocol would still work, since a retransmission would be what would happen if the packet received with errors has actually been lost (and from the receiver standpoint, it never knows which of these events, if either, will occur).
To get at the more subtle issue behind this question, one has to allow for premature timeouts to occur. In this case, if each extra copy of the packet is ACKed and each received extra ACK causes another extra copy of the current packet to be sent, the number of times packet n is sent will increase without bound as n approaches infinity.

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 4 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-1 to k-1.