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?

Write your solution to Problem 1 in this box

(a) There are N * N different combinations.
So, there are N^2 files needed.

(b) N video files ans N audio files are needed.
So, there are 2N files in total

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. . .

Write your solution to Problem 2 in this box

(a) 172.217.4.142

(b) google.com. 179 IN A 172.217.4.142

(c) The second column of ANSWER SECTION represents the TTL of cache. So, the eariliest time to contact the name servers is when TTL=0 which is April 19, 2018 15:39:20

(d) When the cache times out, in 55414 seconds from April 19, 2018 at 15:35:21.

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).

Write your solution to Problem 3 in this box

Yes, it still works, but it would generate more unnecessary traffic. In such circumstances, the sender will keep retransmitting packets until it receives an ACK from receiver. It turns out that the sender will be more likely to time out earlier and retransmit the packets again and again that causes more network traffic.

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?

Write your solution to Problem 4 in this box

No, because it takes too long for receiver to be aware of data packet. The only way for the receiver to realize a packet loss is that it receives a packet without receiving the packet before it. If the sender sends a sequence of packets with seq #s 0, 1, 2, the only way that the receiver knows 1 is lost is that it receives 0 and 2, then it can send a NAK message back to sender. Since the data is sent infrequently, a NAK-only protocol will dramatically increase the response time of a receiver.

Yes. With the same principle in the first case, since now we have a lot of data to send, it doesn't take so long for the receiver to realize a packet loss. And also because the connection experiences few losses, there will not be a lot of NAK messages sent back which is also good.

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.

Write your solution to Problem 5 in this box

(a) The receiver is expecting k means that k-1 has been ACKed by the receiver.
The best case is that, ACK of k-1 is received by the sender so that the sender can move it out of the window. Therefore, the window contains k, k+1, k+2, k+3 if k <= 1020
The worst is that, none of the ACKs of packet k-1 and three packets preceding it were received by sender. Therefore, the window contains k-4, k-3, k-2, k-1 if k <= 1022.
The normal cases are between those two.

(b) Since the receiver is expacting k, k-1 has been ACKed by the receiver.
From the sender side, k-1 is sent. Because the window size is 4, at least k-5 is ACKed and removed out of window by the sender. So, the range of ACK field is k-4 to k-1.