Question 1:

Block	Arrival at destination	Availability
1	9	yes
2	10	-
3	11	-
4	12	yes
5	13	-
6	14	-
7	15	-
8	16	-
9	17	-
10	18	-
11	19	-

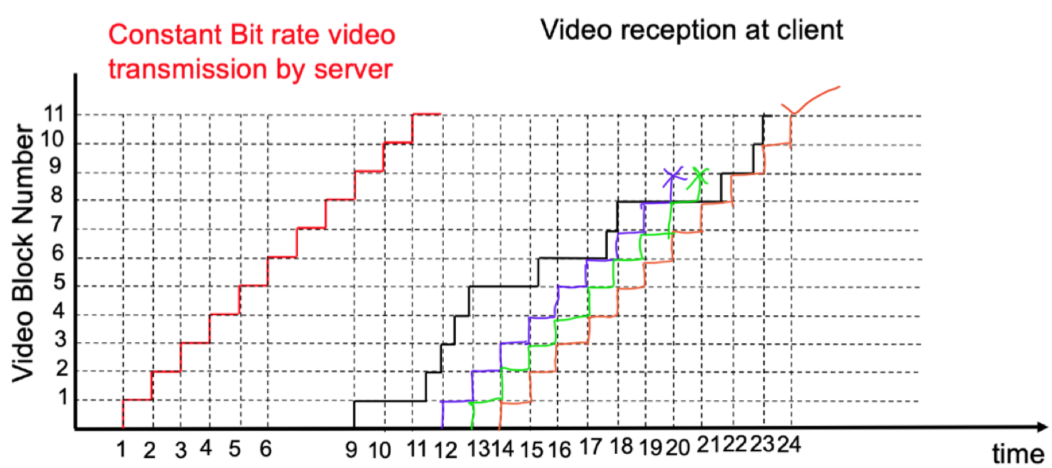
(a.) 1st and 5th blocks will be available for playout.

Block	Arrival at destination	Availability
1	12	yes
2	13	yes
3	14	yes
4	15	yes

5	16	yes
6	17	yes
7	18	yes
8	19	yes
9	20	-
10	21	-
11	22	-

(b.) Only blocks 1, 2, 3, 4, 5, 6, 7, 8 will be available for playout.

(c.)



For delay of 3 time units, it is seen that packet 8 and the following packets are not received in time for playout.

For delay of 4 time units, it is seen that packet 8 and the following packets are not received in time for playout.

But, for delay 5 and above, all packets arrive in time. Therefore, the minimum playout delay such that every block has for playout is 5 time units.

Question 2:

In scheme A, for every 8 chunks of nominal audio, an additional chunk is transmitted as redundancy. So the total number of chunks that needs to be transmitted becomes 9 for every group of 8 chunks.

To calculate the extra bandwidth required, assume each nominal audio chunk is of size n bits, and are transmitted at a rate of R bits per second. Therefore, the nominal bandwidth required would be:

Nominal bandwidth = n x 8 x R

Since 9 chunks need to be transmitted instead of a chunks, the total bandwidth required would be = n x 9 x R

The extra bandwidth required would be = Total bandwidth - Nominal bandwidth

Extra bandwidth = n x 9 x R - n x 8 x R
= n x R

So the extra bandwidth required is equal to the size of one nominal audio chunk multiplied by the transmission rate.
Percentage increase in bandwidth = (n x R / (n x 8 x R)) x 100%
= (1 / 8) x 100%
Percentage increase in bandwidth = 12.5%

In Scheme B, the size of the nth packet equals the size of the nth nominal audio chunk plus the size of the (n-1)th lower quality chunk. Assuming that the nominal audio chunk's size is N, and the lower quality chunk's size is 0.125, the size of the nth packet is the sum of the two sizes i.e. 1.125N. As a consequence, each packet in this scheme requires 1.125 times more bandwidth than a scheme that only uses the nominal audio stream.

Because the receiver must wait for 2 packets to arrive before playback, the minimum playback delay is 2 packet durations. This can be problematic for real-time applications, where low latency is critical. However, the use of a lower quality stream helps reduce the overall bandwidth requirement, which is only **12.5% higher** than the nominal audio stream's bandwidth. This is because the lower quality stream adds only 12.5% extra data to each packet, translating to a **12.5% bandwidth increase** for the entire stream.

Scheme A relies on transmitting an extra redundant packet for every group of 8 packets, but the receiver requires all 9 packets. In this pattern, the first packet of every group of 3 packets is lost, which means that the receiver would receive only 2 packets for every group of 3.

Since this scheme generates an extra redundant packet for every group of 8 packets, it cannot recover the lost packet in a group of 3 packets. Therefore, the receiver will not be able to reconstruct the lost audio data faithfully, and the audio quality would be of poor quality.

In scheme B, each packet depends on the previous packet from the lower quality stream. Therefore, if the first packet of each group of 3 packets is lost, the next two packets in that group cannot be constructed and the effect is propagated further similar to the Domino effect. As a result, the losses may introduce artifacts or distortion in the audio. Furthermore, the scheme requires the receiver to wait for two packets before playback can begin. If the first packet of a group of 3 packets is lost, then the receiver will have to wait for at least 4 packets (i.e., two packets from the current group and two packets from the next group) before playback can begin. This will increase the playback delay and may lead to poor user experience.

Considering the above facts, it can be said that scheme A is better than scheme B.

Question 3:

For L values [1,8]:

L = 1 --> MSE: 45.368011474609375
L = 2 --> MSE: 40.642608642578125
L = 3 --> MSE: 35.45323181152344
L = 4 --> MSE: 31.586135864257812
L = 5 --> MSE: 26.516281127929688
L = 6 --> MSE: 21.207489013671875
L = 7 --> MSE: 13.857742309570312
L = 8 --> MSE: 1.6448211669921875

NOTE: The bytes sent to server for various L values include Python's serialization and de-serialzation memory overheads.

L = 1 --> 946176 bytes
L = 2 --> 7492683 bytes
L = 3 --> 28752454 bytes
L = 4 --> 79062931 bytes
L = 5 --> 176396514 bytes
L = 6 --> 346760076 bytes
L = 7 --> 613843480 bytes
L = 8 --> 801230806 bytes

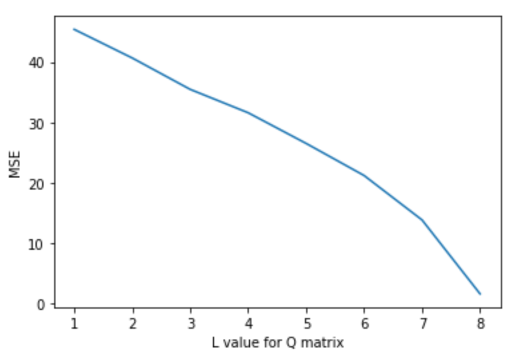


Fig 1. MSE for all L values

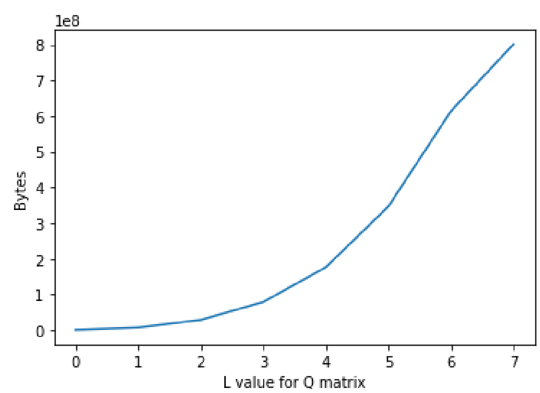


Fig 2. Bytes sent to server for all L values