Q1: Unreliable transport protocol
In the lecture, you have learned how a reliable transport protocol can be built on top of a best-effort delivery network. However, some applications still use an unreliable transport protocol.

1. What are the characteristics of best-effort and of reliable transport?

Solution:
- Best-effort delivery: There is no guarantee for packets to arrive in the correct order, correctly (bit corruption) or even arrive at all.
- Reliable transport: It provides all the above guarantees by making use of sequence numbers, checksums and acknowledgments.

2. What could be advantages of using an unreliable transport protocol?

Solution:
- Better performance/less overhead since you don’t have to wait for ACKs to arrive;
- Lightweight implementation;
- As no connection setup is required (e.g., TCP three-way handshake), you can immediately start sending.

3. What type of applications are suitable to use unreliable transport protocols?

Solution: Applications for which it is more important to have “live” data than to have “complete” data. In voice/video-calls, for example, lost packets lead to lower quality, but delayed packets lead to distorted conversations.
4. As we will later see, the User Datagram Protocol (UDP) only provides unreliable transport. Assume you are forced to use a network which only supports UDP as a transport protocol. You must transmit an important document which eventually should be correctly transmitted. Do you see a way to implement some of the reliable transport mechanisms despite using UDP?

Solution: Yes, the reliable transport mechanisms could be implemented by the application/in the application layer.

Q2. Network unreliability: Name five undesirable effects the network can have on individual packets and a stream of packets when attempting to deliver them to the destination.

Solution: packets can get lost, corrupted, reordered, delayed, or duplicated.

Q3: Timeouts

1. In reliable transmission, what is the effect of setting a too short timeout, and conversely what is the effect of setting a too long timeout?

Solution: too short timeouts will lead to unnecessary transmissions; network capacity is wasted. Moreover, if a fixed action is attached to a timeout occurring (i.e. reducing rate), the transmission will also slow down unnecessarily. Too long timeouts will lead to slow transmission because it will take a long time before the sender discovers a packet has been dropped. The receiver will not accept and forward packets to the application layer during this time, as it forces an in-order stream of data.

2. RTT is a higher-level network property that is mainly used to decide what the re-transmission timeout should be set to. Which four network delays determine the RTT?

Solution:

- Transmission delay: time it takes to send a packet at the link capacity
- Propagation delay: time it takes for a bit to go over the link
- Processing delay: time it takes for the switch to process the packet
- Queuing delay: time a packet spends in the queue

3. Of these four network delays, what changes in the network can you do to reduce the RTT?

Solution:

- Transmission delay: increase the link capacity
• Propagation delay: if possible, use data replication to shorten the client-server distance. Improving routing or physical transmission media also helps.

• Processing delay: processing delays are typically negligible in comparison to other delays, and there is little room for improvement here.

• Queuing delay: better load balancing (i.e. via routing), reduce queue size

Q4: Video Streaming: TRUE or FALSE?

• Video resolution is a synonym of video bitrate. FALSE. The bitrate is the amount of encoded video data sent from the server to the client. The resulting resolution is determined by the quality of encoding and the device context (i.e. screen size, DPI). For example, doubling device screen size while maintaining the same bitrate will not result in the same resolution.

• In video streaming, the entire content is usually provided as a continuous flow of bytes. FALSE. Usually the video is divided into video segments ("chunks") that are provided separately.

• Adaptation algorithms may vary depending on users’ devices. TRUE. (a) Video buffer size can vary across devices. A device with a large buffer may benefit from a different algorithm (or at least a differently tuned algorithm) than one with a very small buffer. (b) Different network patterns occur in different environments, for example a mobile environment is different from a home TV. (c) Depending on the device, there might be different requirements in terms of the so called Quality of Experience (e.g. HDTV users might need a higher video quality in comparison to mobile users for the same QoE).

• Usually, a particular video content is stored on a single server. FALSE. To enable load balancing, redundancy and fault tolerance, video content is replicated across multiple servers.

• Video chunks duration is typically in the order of seconds. TRUE. The video is divided into chunks to allow bitrate adaptation. Network condition changes, and the ability to detect and respond to them, happen on the granularity of seconds.

Q5: How does Buffer Based Adaptation work? What is its main weakness and how could it be solved?

In Buffer Based Adaptation (BBA), the quality of the next chunk to be fetched is decided according to the buffer occupancy of the player. If the buffer is nearly full, the player usually fetch high quality video segments. On the other hand, if the buffer is nearly empty, the player could decide to downgrade the
quality of the playback, in order to speed up the download process and to avoid rebuffering. Its main weakness shows up in the start up phase, where the buffer is empty by definition. If not handled properly, a pure BBA approach would start fetching video segments at the lowest available quality, and then increasing it once the buffer is sufficiently filled. In order to avoid a poor initial playback, the rate could be picked according to the immediate past estimated throughput.

Q6: Reliable Transport Questions

Q6.1: What mechanism is used to detect corrupted packets? Quickly explain.

Checksums. Each packet contains a checksum of its contents in its header, the receiver can then calculate the checksum itself and compare it to the stored checksum. If they don’t match, the packet was corrupted.

Q6.2: How would a reliable transport protocol deal with packets that were duplicated by the network?

Sequence numbers. Each packet (or byte) is numbered, a receiver will only pass packets (or specific bytes within them) to the application, if it hasn’t already accepted the corresponding sequence numbers before.

Q6.3: When a timeout is triggered in the sender, (how) can the sender be sure that the receiver has really not received the packet?

It cannot be sure. Triggered timeouts are only an indicator, but not proof of packet loss. It could just as well be that the ACK got lost or that the ACK would arrive just after we have resent the packet. In these cases the sender will resend the packet even though it was already received, as seen above this is handled in the receiver with sequence numbers.

Q7: Name and explain three ACKing mechanisms and name an advantage and disadvantage for each of them.

- **Individual ACKs**: Each ACK only acknowledges a single packet and gives no other information.
  
  *Advantages:*
  
  – The acknowledgement has a fixed size.

  *Disadvantages:*
  
  – Missing packets are only implicitly signaled to the sender by the absence of an ACK. Sender can’t tell the difference between lost packet and lost ACK.
If an ACK is lost, sender will always need to resend that packet.

- **Full Feedback (aka Selective Acknowledgements):** Each ACK acknowledges all received data.

  **Advantages:**
  
  - With every ACK, the sender knows exactly which packets have not yet been delivered and can resend only those packets. This decreases the number of sent packets.
  
  - Each ACK contains at least all the information as the ACK before it. If an ACK is lost, no information is lost because the next ACK will also contain that info.

  **Disadvantages:**
  
  - Sending information about all received packets uses non-constant space and is more difficult to use in a header. The logic to deal with it in hardware/software is also more complex/slower.

- **Cumulative ACKs:** Each ACK acknowledges that all packets (bytes) up to a given sequence number were received.

  **Advantages:**
  
  - Same as with individual ACKs, the size of this information is constant.
  
  - Similar to full feedback, ACKs contain all the information of the ACKs before. When ACKs are lost, we don’t lose any information because packets will be ACKed in all subsequent ACKs as well.

  **Disadvantages:**
  
  - Missing packets are only implicit, as with individual ACKs. It’s hard to tell for the sender if packets were actually lost. However, this is a bit mitigated by the nature of the ACK messages: If the sender receives the same ACK multiple times, it’s an indication that a packet was lost. Since these duplicate ACKs signal that packets were received but the number of received in-order packets didn’t change.