Implement *some* TCP functionality using UDP sockets

Q1) How is **your** implementation of data sequencing and retransmission different from traditional TCP? (If there are no apparent differences, you may mention that)

Answer:
a)
The actual TCP protocol is a connection-oriented one. Prior to data transmission, a three-way handshake is necessary to establish a connection. It guarantees the timely and accurate transmission of data between the sender and the recipient.

TCP-like implementation utilising UDP: This implementation makes use of UDP and is connectionless. Before transferring data, there is no need for a handshake or connection establishment. No order of delivery or reliability is guaranteed; each packet is despatched on its own.

b)
Actual TCP: TCP guarantees dependable data delivery. It ensures that data is received in the right order and without errors using acknowledgments (ACKs) and sequence numbers. A section will be retransmitted if it is not acknowledged the first time.

TCP-like implementation utilising UDP: This implementation makes an effort to provide dependability by manually managing acknowledgments and retransmissions. If no ACK is received within a predetermined amount of time, the sender retransmits. The receiver sends ACKs for data received. Also , the manual handling of acknowledgments might need acknowledgments. Hence, this is not the best way to do .

c) Flow Control:

Actual TCP: To prevent a fast sender from overwhelming a slow receiver, TCP has flow control methods. Congestion control techniques and sliding window methods are used for this.

Flow control is not explicitly implemented in the UDP-based TCP-like implementation. It is possible for the sender to send data at a rate that is higher than the receiver's capacity, which could cause congestion and packet loss.

Q2) How can you extend **your** implementation to account for flow control? You may ignore deadlocks

Answer: To extend UDP-based TCP-like implementation to account for flow control, you can implement a simple form of flow control using a sliding window mechanism. This will help prevent the sender from overwhelming the receiver with too much data.

Sender side:
a)Maintain a Sending Window: Establish a sending window that indicates the most chunks that can be delivered by a sender before they are acknowledged.

b)Track Acknowledgments: Keep tabs on which portions the recipient has acknowledged.

c)Adapt sending rate: Change the rate at which you send data in accordance with the acknowledgments you have received and the size of the window that is available.

Receiving side:

a) Keep a Receiving Window Open: Establish a receiving window that indicates the most out-of-order chunks the receiver will accept.

b) Send Window Size in ACKs: Include a "window size" field or similar information in each ACK packet to indicate how much space is available in the receiving window.

c) change Window Size: Depending on the amount of available buffer space, when receiving data, change the receiving window size.

d) Only transmit acknowledgments for chunks that are received within the receiving window.

Implementation

a) Buffer management: The receiver ought to have a buffer to hold chunks that are out of order until they can be processed and sorted correctly.
b) Window management: Depending on the resources that are available and the state of the network, the transmitting and receiving windows should be dynamically modified.
c) Handling Window Updates: The transmitter may need to halt or reduce its broadcast pace if the receiving window gets smaller. The sender may raise its rate if it expands.
d) Timeout handling: Within the parameters of the sending window, the sender should nevertheless implement retransmissions for unacknowledged chunks.