Implement some TCP (?) functionality from scratch

UDP-Based Data Transmission with Sequencing and Retransmission

UDP (User Datagram Protocol) is a connectionless protocol that does not guarantee the order or reliability of data transmission. However, this implementation adds custom features to UDP to provide data sequencing and retransmission, similar to the behavior of TCP (Transmission Control Protocol).

`Client`:

1. The UDP client converts message into fixed size chunks and sends data chunks to the server without establishing a connection.
2. There is no guaranteed delivery or sequencing of packets.
3. The client sends chunks of data, and the server acknowledges each chunk.
4. Retransmission is implemented by resending unacknowledged chunks after 0.1 seconds without waiting for the acknowledgement. This functionality was made using `select()` function.
5. After received all the acknowledgements it sends a final message saying all done.
6. Then it starts receiving from the server and sends acknowledgement for the received chunks.
7. Flow control and congestion control are not explicitly implemented in this UDP implementation.

`Server`:

1. The UDP server listens for incoming data chunks on a predefined port.
2. It acknowledges the receipt of each chunk and processes them independently.
3. The server does not maintain a connection state with clients.
4. Retransmission is handled by the client if it does not receive acknowledgments.
5. After getting all the chunks from the client, the server prints the message and then it sends message by chunks.
6. Retransmission is implemented by resending unacknowledged chunks after 0.1 seconds without waiting for the acknowledgement. This functionality was made using `select()` function.
7. After all the successful sent chunks it sends a final one saying to close.
8. Flow control and congestion control are not implemented.

Questions Asked:

How is your implementation of data sequencing and retransmission different from traditional TCP?

In the provided code, the implementation of data sequencing and retransmission over UDP is different from traditional TCP in the following ways:

`UDP vs TCP`: UDP is a connectionless protocol, while TCP is connection-oriented. In TCP, there is an established connection between the sender and receiver with reliable, ordered, and error-checked data transmission. UDP, on the other hand, lacks these features by design.

`Data Sequencing in UDP`: In the UDP-based implementation, data sequencing is implemented manually using sequence numbers (seq_no). Each data chunk is assigned a unique sequence number to maintain the order of data transmission. This approach allows the sender and receiver to keep track of the order of data chunks.

`Retransmission in UDP`: The code includes a mechanism for retransmitting data chunks if they are not acknowledged by the receiver in 0.1 seconds sending other data not waiting for this. This is achieved through a loop that continually checks for unacknowledged chunks and retransmits them until acknowledgment is received.

`Selective Acknowledgment`: The code uses selective acknowledgment, where the receiver sends an acknowledgment containing the sequence number of the successfully received chunk. This allows the sender to retransmit only the missing or unacknowledged chunks, reducing unnecessary retransmissions.

`Non-blocking Socket`: The UDP socket is set to non-blocking mode (O_NONBLOCK), allowing the sender to continue other operations while waiting for acknowledgments. This differs from TCP, where the sender typically waits for acknowledgments before proceeding.

How can you extend your implementation to account for flow control?

To account for flow control in the current UDP-based implementation, you can implement a mechanism that prevents the sender from overwhelming the receiver with too much data too quickly. Here's how you can extend the implementation for flow control:

`Sender-Based Flow Control`: Implement a mechanism on the sender side to monitor the receiver's buffer space availability. The sender should keep track of the receiver's buffer capacity and adapt its sending rate accordingly.

`Window-Based Flow Control`: Implement a sliding window mechanism where the sender sends data within a dynamically adjusted window size. The window

size indicates the number of unacknowledged chunks allowed in flight at any given time. The sender can adjust the window size based on the receiver's buffer availability and network conditions.

`Receiver's ACKs`: The receiver can send acknowledgments that not only acknowledge the received data but also indicate the available buffer space. For example, the receiver can include the number of free bytes in its acknowledgment.

`Sender's Sending Rate`: The sender should control its sending rate based on the receiver's acknowledgment and available buffer space. It should slow down or pause sending if the receiver's buffer is full and resume when space becomes available.

`Congestion Avoidance`: Implement congestion avoidance mechanisms, such as exponential backoff or congestion window adjustment, to adapt to network congestion and prevent packet loss.

`Error Handling`: Handle situations where the sender receives indications of congestion or buffer overflow from the receiver. This may involve temporarily reducing the sending rate or applying other congestion control techniques.

By implementing these flow control mechanisms, the UDP-based implementation can achieve a more controlled and efficient data transfer, similar to how TCP manages flow control in a reliable manner. However, it's important to note that UDP-based flow control is typically more application-specific and may not provide the same level of reliability and congestion control as TCP.