1. My implementation for data sequencing and retransmission in your code is different from traditional TCP in the following ways:

1. ***<u>Using UDP to implement TCO</u>***: My code uses UDP (User Datagram Protocol) for communication, whereas traditional TCP (Transmission Control Protocol) is a connection-oriented protocol. TCP provides reliable, ordered, and error-checked delivery of data, while UDP does not guarantee these features. In my code, I am implementing reliability and ordering at the application layer.
2. ***<u>Sending Data in the form of chunks</u>***: I am sending data in the form of data chunks. Sending data in the form of data chunks is a common technique used in networking and communication systems to efficiently transmit large or structured datasets over a network. Chunks are smaller, manageable units of data that are sent individually and reassembled on the receiving end to reconstruct the original data
3. ***<u>Retransmission:</u>*** In traditional TCP, if the sender does not receive an acknowledgment (ACK) for a segment within a specified timeout, it retransmits that segment. In my code, I have implemented a similar retransmission mechanism but I have done it using data chunks by resending acknowledgments for missing chunks after a timeout. However, this is done at the application layer, whereas TCP's retransmission is managed by the transport layer.
4. ***<u>Ordering:</u>*** TCP ensures that data sent from one side arrives in the same order on the other side. In my code, I have implemented ordering by adding a sequence number to each chunk of data and ensuring that data is received in order on the server-side which is different from the traditional TCP.
5. ***<u>Reliability:</u>*** TCP guarantees reliable delivery of data, while my code relies on manual acknowledgment and retransmission for reliability.

2. To extend my implementation to account for flow control, I can implement a simple flow control mechanism to prevent the sender from overwhelming the receiver with data. One common approach is to use a sliding window mechanism to limit the number of unacknowledged chunks the sender can have in-flight at any given time. Here's how I can modify my code to incorporate basic flow control:

1. ***<u>Define a window size</u>***-We can define a window size to limit the number of unacknowledged chunks that can be sent at once. You can add a constant for this, e.g., **WINDOW_SIZE**.
2. ***<u>Maintain two variables</u>***- We need to maintain two variables to keep track of the sender's window: **base** and **nextSeqNum**.
   - **base**: Represents the sequence number of the oldest unacknowledged chunk.
   - **nextSeqNum**: Represents the sequence number of the next chunk to be sent.
3. ***<u>Modify the sender code to:</u>***
   - Initialize **base** and **nextSeqNum** to 0.
   - Send up to **WINDOW_SIZE** chunks initially.
   - Increment **nextSeqNum** for each sent chunk.
   - When an acknowledgment is received, update **base** to the acknowledged sequence number and slide the window.
4. In this modified code, the sender sends chunks within the defined window size and waits for acknowledgments. When an acknowledgment for the base sequence number is received, it slides the window and sends more chunks as long as they are within the window. This basic flow control mechanism helps prevent overloading the receiver. We can adjust **WINDOW_SIZE** to control the flow more precisely based on our requirements.

5. ***Dynamic Window Sizing:*** To optimize flow control,we can consider implementing dynamic window sizing. The sender and receiver can negotiate and adjust the window size based on network conditions. For example, the receiver might signal the sender to increase or decrease the window size depending on its processing capacity or buffer space.

6. ***Timeout Handling:*** If we have a dynamic window size,we should be prepared to handle timeouts when acknowledgments are not received within a reasonable time. We might need to retransmit chunks or adjust the window size based on timeout events.

7. ***Error Handling*:** Consider how our implementation will handle cases where chunks are lost or corrupted. This could involve retransmission of lost chunks or requesting retransmission of specific chunks that were received with errors.There could be several more errors that we might need to handle.