Comparison of Custom UDP-based Sequencing & Retransmission with Traditional TCP:

1. Sequence Numbers:

Mine Approach:

I have divided data into fixed-size chunks.

Each chunk is labeled with a distinct sequence number (e.g., for a 100-byte string divided into two 50-byte chunks, the chunks are labeled 1 and 2).

Traditional TCP:

Sequence numbers are assigned to bytes, not chunks.

The sequence number for a given byte signifies its position within the data stream, allowing for granular tracking and retransmission of data.

2. Acknowledgements (ACKs):

Mine Approach:

The server acknowledges the receipt of each chunk with an ACK.

Traditional TCP:

TCP can use cumulative acknowledgments.

A single ACK can confirm the receipt of multiple segments. For example, an ACK for byte 100 can confirm the receipt of bytes 1 to 100.

3. Retransmission:

Mine Approach:

The server identifies missing chunks by iterating over a table.

Clint also maintains a hash table for which ACK is received. For which it is not received it retranmits that data chunk.

Traditional TCP:

Uses a combination of timers and acknowledgments.

Retransmits segments if an ACK isn't received within the Retransmission Timeout (RTO) period.

4. Flow Control:

Mine Approach:

Received packets are buffered in an array.

Data is processed (concatenated into a string) only when all chunks are received, ensuring ordered processing.

Traditional TCP:

Uses a sliding window mechanism.

The receiver's buffer availability determines the window size, signaling to the sender how many bytes it can transmit before needing an acknowledgment.

Allows dynamic adjustment to match the receiver's processing rate.

5. Ordered Data Delivery:

Mine Approach:

Data is divided into chunks and can be reached in any order but I maintain an array of chunks so I place every chunk in its respective place . When all chunks get received I simply iterate over this array and concatenate string.

Traditional TCP:

Guarantees that data is presented to the application layer in the correct order, regardless of the order in which segments were received.

Has built-in mechanisms to handle out-of-order segments.

8. Error Handling:

Mine Approach:

Relies on ACKs and retransmission for error recovery.

Traditional TCP:

Each TCP segment carries a checksum for error detection.

Erroneous segments are discarded, prompting retransmission due to the absence of an ACK.

Conclusion:

While Mine approach captures some of the basic principles behind reliable data transfer, the traditional TCP protocol offers a more comprehensive and robust solution. It handles variable network conditions, congestion, flow control, and error scenarios with sophisticated mechanisms built over years of research and real-world testing. Mine method may serve specific use-cases, especially where data naturally divides into chunks, but for general-purpose reliable data transfer, TCP's in-built mechanisms provide a well-rounded solution.

Flow Control Mechanism in a UDP-Based System:

1. Introduction of Receiver Window:

Concept: The receiver window is a buffer space that the receiver allocates for incoming packets. It essentially tells the sender how much data the receiver can accept without overwhelming its processing capability.

Server-Side:

Allocate a fixed buffer size that will store incoming packets. This buffer represents the maximum window size.

Initially, communicate the entire buffer size to the client as the receiver window, signaling the client that it can send that many packets.

Client-Side:

On initiation or reset, wait for the server to send its initial window size.

Send packets only up to the size of the receiver window. Postpone the sending of additional packets until the window is updated.

2. Feedback Mechanism for Window Updates:

Concept: As the server processes received packets and frees up buffer space, it needs to communicate back to the client about how much new data it can receive.

Server-Side:

For every packet processed and removed from the buffer, update the available window size.

Periodically (or after processing a certain number of packets), send this updated window size back to the client.

Client-Side:

After sending packets, listen for window size updates from the server.

Adjust the rate of data transmission based on the latest window size communicated.