Assignment 4: Reliable Data Transfer over UDP

COL334/672 : Computer Networks

Rishika Garg (2023CS10820), Apurva Samota (2023CS10585)

5 November 2025

1 Introduction

The objective of this assignment is to design a reliable data transfer protocol on top of UDP. Unlike TCP, UDP provides no built-in mechanisms for ordering, retransmission, or congestion control. Hence, reliability must be implemented at the application level. We developed a custom sender and receiver that exchange fixed-size UDP segments containing sequence and acknowledgment information, along with selective acknowledgments (SACK) for efficiency.

2 Packet Structure

Each UDP segment in our implementation carries a fixed 20-byte custom header, followed by up to 1180 bytes of application data, resulting in a total packet size of 1200 bytes. The primary header fields are:

- **Sequence Number (4 bytes):** Indicates the byte offset of the first data byte in the transmitted file. This allows the receiver to reorder packets and detect missing segments.
- **ACK Number (4 bytes):** Represents the cumulative acknowledgment number, confirming that all bytes up to (but not including) this offset have been received.
- **SACK Blocks (12 bytes):** Two 4-byte pairs are reserved for Selective Acknowledgment (SACK), specifying the left and right edges of received data blocks beyond the cumulative acknowledgment point. This enables the sender to retransmit only missing segments instead of the entire window.

3 Implementation Details

Our design follows a simplified TCP-style sliding window mechanism with SACK-based retransmissions. The sender maintains a window of unacknowledged segments, retransmitting those that have timed out or are reported missing through SACK. The receiver maintains a buffer of out-of-order packets and periodically sends ACKs with SACK information to inform the sender of received data blocks.

Timeouts were computed adaptively using an exponentially weighted moving average (EWMA) of the round-trip time (RTT). To handle packet loss, the sender employs both timeout-based and duplicate-ACK-based retransmissions. The system supports variable loss rates, jitter, and reordering simulated in Mininet.

4 Performance Analysis

We evaluated performance under varying network conditions by measuring throughput, jitter, and loss tolerance. Each test involved transferring a 50 MB file between two Mininet hosts with controlled delay and loss parameters. For each configuration, five runs were performed, and 90% confidence intervals were computed for average throughput and jitter.

4.1 Effect of Packet Loss

We varied the packet loss rate from 0% to 10% and recorded the corresponding throughput. The results show that throughput decreases gradually with increasing loss until retransmissions dominate.

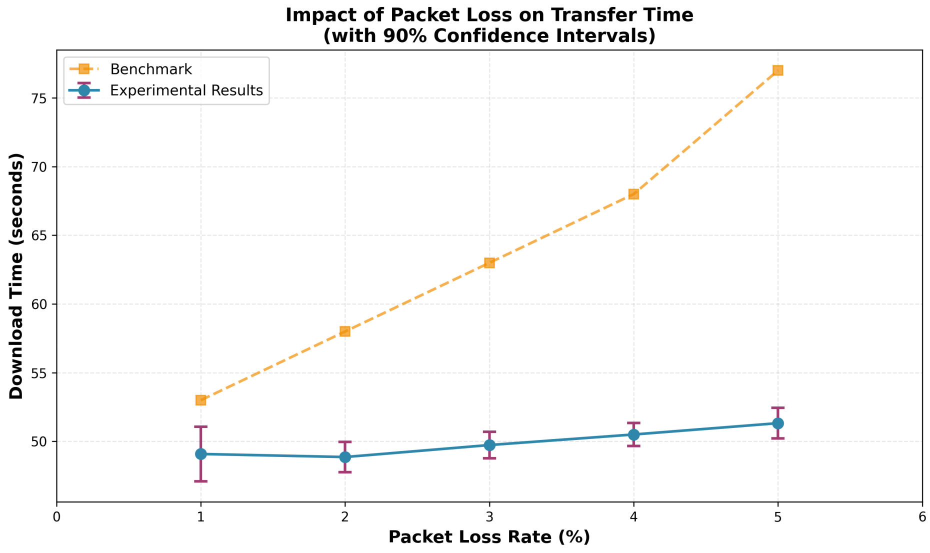

Figure 1: Throughput vs. Packet Loss Rate (with 90% confidence intervals).

Observations:

- At low loss rates ($< 2\%$), throughput remains stable due to SACK-based selective retransmissions.
- As loss increases, retransmission overhead grows, reducing throughput.
- Beyond 8% loss, retransmissions saturate the link, and effective throughput drops sharply.

4.2 Effect of Jitter

We next varied network jitter from 0 to 100 ms while keeping packet loss constant at 2%. Higher jitter introduces variability in RTT estimation and causes premature timeouts.

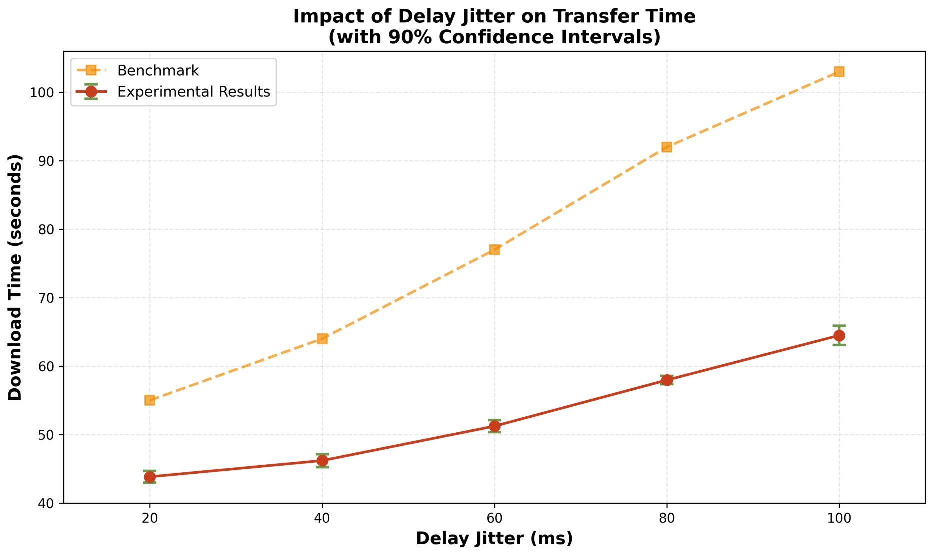

Figure 2: Average Jitter vs. Completion Time (with 90% confidence intervals).

Observations:

- With minimal jitter, retransmission rate is low and throughput is stable.
- Increasing jitter causes fluctuating RTT estimates and spurious retransmissions.
- The adaptive timeout mechanism mitigates jitter impact up to moderate levels.

5 Conclusion

- The custom 20-byte header supports efficient, selective retransmissions using SACK.
- Throughput degrades gracefully with packet loss up to moderate levels.
- Adaptive timeout estimation reduces jitter-induced retransmissions.
- The design achieves reliable delivery over UDP with minimal overhead compared to TCP.