Reliable Data Transfer Protocol Report

Name: Samyak Rajesh Shah
Email: ss4604@rit.edu

1. Introduction

This report presents the implementation of a **Reliable Data Transfer Protocol** built over UDP for CSCI-651 Homework 3.
The project consists of a **server** and **client** designed to achieve reliable file transfer over an unreliable network by implementing:

- Acknowledgments (ACKs)
- Retransmissions and timeouts
- Checksums for data integrity
- Sequence numbers and sliding window
- Simulation of packet **loss, corruption, reordering, and dropped acknowledgments**

The protocol was designed and tested using Python sockets with multi-threading, ensuring robust performance under simulated adverse network conditions.

2. Project Structure

File Name	Description
server.py	Implements the reliable UDP server with sliding window and retransmission logic
client.py	Implements the reliable UDP client that validates packets, sends ACKs, and writes the received file
requirements.txt	Lists Python dependencies
sample.txt	Sample file for transmission
README.md	Explains setup, command-line usage, and example runs
docs/	Sphinx-generated documentation for codebase

3. Environment Setup

- **Python Version:** 3.8 or above
- **Required Libraries:**
  - argparse

- socket
- threading
- hashlib
- json
- base64

Installation

pip install -r requirements.txt

4. Reliable Data Transfer Design

The system was designed with the following reliability features:

Feature	Description
Sliding Window	Allows multiple in-flight packets to improve throughput
Checksum	SHA-256 checksum ensures data integrity
Sequence Numbers	Maintain correct packet order
Timeout & Retransmission	Ensures lost packets are resent
Simulated Impairments	Random packet loss, corruption, and reordering for testing
Threaded Acknowledgment Handling	Concurrent reception of ACKs during transmission

5. Server (server.py)

5.1 Description

The **server** reads a file, divides it into packets, and sends them to the client using UDP.
It employs a **sliding window** for concurrent sending and handles:

- Timeouts and retransmissions
- Packet reordering simulation
- Corruption and loss emulation
- Dropped ACK simulation

5.2 Command-line Arguments

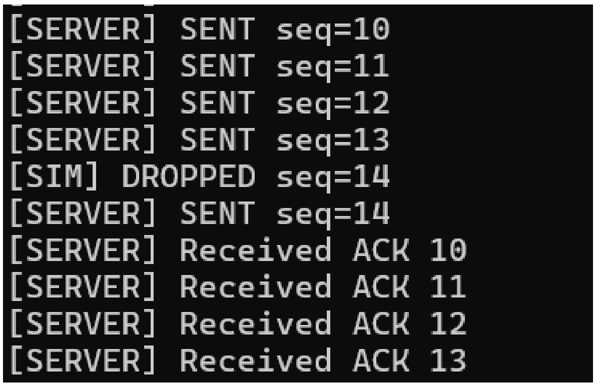
Option	Description
--host	IP address to bind server (default: 127.0.0.1)
--port	UDP port number (default: 9000)
--file	File to be transferred (required)

5.3 Example Usage

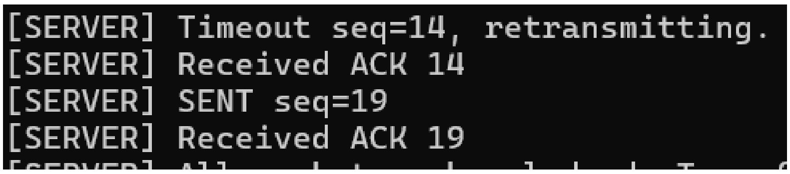
python server.py --host 127.0.0.1 --port 9000 --file sample.txt

5.4 Sample Execution

- **Screenshot 1:** Server showing simulated packet loss and retransmission



- **Screenshot 2:** Server indicating reordered and corrupted packets handled



6. Client (client.py)

6.1 Description

The **client** requests a file, receives packets, validates them via checksum, sends acknowledgments, and reorders if necessary.
After receiving all packets, it reconstructs and writes the final file.

6.2 Command-line Arguments

Option	Description
--server_host	Server IP address (default: 127.0.0.1)
--server_port	Server UDP port (default: 9000)
--file	File name to request from the server (required)

6.3 Example Usage

python client.py --server_host 127.0.0.1 --server_port 9000 --file sample.txt

6.4 Sample Execution

- **Screenshot 3:** Client receiving packets and sending ACKs

```
[CLIENT] Server reports 20 packets.
[CLIENT] Stored packet seq=0
[CLIENT] Sent ACK 0
[CLIENT] Stored packet seq=1
[CLIENT] Sent ACK 1
[CLIENT] Stored packet seq=2
[CLIENT] Sent ACK 2
[CLIENT] Stored packet seq=3
[CLIENT] Sent ACK 3
[CLIENT] Stored packet seq=4
```

- **Screenshot 4:** Client handling out of order packets

```
[CLIENT] Stored packet seq=17
[CLIENT] Sent ACK 17
[CLIENT] Stored packet seq=18
[CLIENT] Sent ACK 18
[CLIENT] Stored packet seq=14
[CLIENT] Sent ACK 14
[CLIENT] Stored packet seq=19
[CLIENT] Sent ACK 19
```

- **Screenshot 5:** Client successfully writing received file

```
[CLIENT] Stored packet seq=19
[CLIENT] Sent ACK 19
[CLIENT] All packets received.
[CLIENT] File successfully saved as received.txt
```

- **Screenshot 6:** Client handling corrupted packets

```
[SERVER] SENT seq=1
[SIM] CORRUPTED seq=2
[SERVER] SENT seq=2
[SERVER] SENT seq=3
[SERVER] SENT seq=4
[SERVER] Received ACK 1
[SERVER] Received ACK 3
```

```
[CLIENT] Corrupt packet seq=2, dropped.
[CLIENT] Stored packet seq=3
[CLIENT] Sent ACK 3
```

- **Screenshot 7:** Client handling duplicate packets

```
[CLIENT] Stored packet seq=11
[CLIENT] Sent ACK 11
[CLIENT] Stored packet seq=12
[CLIENT] Sent ACK 12
[CLIENT] Stored packet seq=13
[CLIENT] Sent ACK 13
[CLIENT] Stored packet seq=14
[CLIENT] Sent ACK 14
[CLIENT] Stored packet seq=15
[CLIENT] Sent ACK 15
[CLIENT] Duplicate seq=11 ignored.
```

7. Conclusion

The project successfully demonstrates a **custom reliable data transfer protocol** over UDP. The implementation meets all key requirements — handling **packet loss**, **corruption**, **reordering**, and **timeouts** gracefully using **ACKs**, **checksums**, and **sequence numbers**.

The resulting protocol can be extended for higher-level applications like **file synchronization** or **streaming**, forming a robust foundation for transport-layer reliability.