Reliable UDP over UDP with HTTP Support - Project Details

# Introduction

This project simulates TCP packets over UDP by extending the user space of the application while maintaining reliability and supporting the HTTP protocol on top of UDP. UDP is connectionless and does not guarantee delivery or order, which creates challenges in reliable data transfer. This implementation addresses these with error detection, retransmission, flow control, and HTTP support.

# Project Requirements

- Implement TCP-like reliable transmission using stop-and-wait.  
- Support handshake and teardown with TCP flags (SYN, SYNACK, ACK, FIN).  
- Use CRC32 checksum for error detection and simulate corrupted packets.  
- Support configurable packet loss and corruption.  
- Handle retransmissions, duplicates, sequence numbers, timeouts.  
- Support HTTP 1.0 GET and POST methods with basic headers.  
- Return HTTP status 200 OK and 404 Not Found.  
- Write test cases and provide detailed documentation.

# Design & Architecture

The system consists of several modules:  
- A reliable UDP socket layer that handles packet packing/unpacking, checksum, and loss/corruption simulation.  
- An HTTP server that uses the reliable UDP layer to receive requests and send responses with proper handshake.  
- An HTTP client that performs handshake, sends requests, receives responses, and gracefully closes connections.  
- A test suite automating basic GET and POST tests.

# Packet Structure & Flags

Packets follow this structure:  
- Sequence Number (4 bytes, unsigned int)  
- Flags (1 byte): bits for SYN (0x01), ACK (0x02), FIN (0x04)  
- Checksum (4 bytes): CRC32 of the payload  
- Payload: actual data bytes (HTTP request or response)

# Flow Control & Handshake

Stop-and-wait flow control is implemented:  
- Client and server perform a handshake: SYN -> SYN+ACK -> ACK.  
- Sequence numbers increment per packet.  
- Connection teardown uses FIN and ACK flags.  
- The client waits for acknowledgments and retransmits on timeout.

# Checksum & Error Detection

CRC32 is used for checksums on packet payloads.  
Packets with incorrect checksums are discarded, triggering retransmission.  
Corrupted payloads are simulated by randomly flipping bits.

# Packet Loss & Corruption Simulation

Simulation modes include:  
- Clean: no loss or corruption  
- Loss only: packets dropped with configurable probability  
- Corruption only: packets corrupted randomly  
- Loss + Corruption: combined effects  
Users select mode and rates interactively or via config file.

# HTTP 1.0 Support (GET/POST)

Supports minimal HTTP/1.0 GET and POST methods.  
Basic header parsing is done, including Content-Length and Connection.  
Responds with status 200 OK or 404 Not Found depending on request.  
HTTP connections are closed after each request/response as per HTTP/1.0.

# Retransmission & Timeout Handling

If no acknowledgment is received within a timeout, packets are retransmitted.  
Duplicates are detected and ignored using sequence numbers.  
Timeouts and max retry counts are configurable.

# Testing & Usage Instructions

- Run the server: `python rudp\_http\_server.py`  
- Run the client: `python rudp\_http\_client.py`  
- Select simulation mode and rates when prompted.  
- Use `python test\_rudp.py` to run automated GET and POST tests.  
- Use tools like Wireshark or Postman for packet inspection.  
- Adjust loss and corruption rates to test robustness.

# Assumptions & Limitations

- Stop-and-wait flow control only (no sliding window or congestion control).  
- Single client support.  
- No fragmentation of large HTTP messages.  
- Basic HTTP parsing; no advanced headers or features.  
- Simulated network issues only; no real network layer emulation.  
- No TLS or security features.

# Summary

This project demonstrates how reliable communication and HTTP can be layered over UDP using simple stop-and-wait flow control, error detection, and retransmission. It provides a solid foundation for understanding TCP-like protocols and HTTP mechanics in a controlled, simulated environment.