CS 352 – Internet Technology

Project 3: Reliable UDP (RUDP) Implementation and Packet Capture

Released: October 23, 2025 **Due:** November 10, 2025
Total Points: 100 **Mode:** Individual or Group of 2

1. Objective

In this project, you will design and implement a simplified **Reliable UDP (RUDP)** protocol on top of UDP sockets. Your implementation will demonstrate how TCP achieves reliable, ordered delivery through: A **three-way handshake** (connection setup) **Stop-and-wait reliability** (sequence numbers, ACKs, timeouts, and retries) A **FIN / FIN-ACK teardown** (connection close) You will verify correctness by recording **Wireshark captures (.pcap)** of your implementation. Your **server must intentionally introduce random ACK delays** so that the client experiences timeouts and retransmits, mimicking real network latency.

2. Assigned UDP Port

Each student or team is assigned a unique UDP port (e.g., 30001–30100). Use this same port in both client and server. ASSIGNED_PORT = 30077 SERVER = ("127.0.0.1", ASSIGNED_PORT) ■■ **Use only your assigned port** — using another team's port may cause conflicts during grading.

3. Provided Files

You are provided exactly two skeleton files: **rudp_client_skeleton.py rudp_server_skeleton.py** These include constants for message types and helper functions (**pack_msg()**, **unpack_msg()**). You must complete all **TODO sections** to implement: Three-way handshake Reliable stop-and-wait data transfer Connection teardown (FIN / FIN-ACK) **Random ACK delay logic on the server** to simulate network jitter

4. Protocol Specification

Type	Code	Purpose
SYN	1	Client initiates connection
SYN-ACK	2	Server acknowledges SYN
ACK	3	Client confirms connection
DATA	4	Data segment with sequence number
DATA-ACK	5	ACK for received DATA
FIN	6	Client requests close
FIN-ACK	7	Server confirms closure

5. Implementation Tasks

A. Three-Way Handshake

Implement: Client → Server : SYN Server → Client : SYN-ACK Client → Server : ACK After successful setup: [CLIENT] Connection established [SERVER] Connection established **B. Reliable Data Transfer**

(Stop-and-Wait with Random Delay)
Split your message into ~200-byte chunks (seq = 0, 1, 2, …). Client sends DATA(seq=n) and waits for DATA-ACK(seq=n). If no ACK arrives within timeout (**RTO** ≈ **0.5s**), client retransmits. Server must insert a **random ACK delay** before replying: import random, time delay = random.randint(100, 1000) # milliseconds time.sleep(delay / 1000.0) Set client retry limit high (e.g., **RETRIES = 10**) so retransmissions occur under long delays. **C. Connection Teardown**
Implement clean closing: Client → Server : FIN Server → Client : FIN-ACK After completion: [CLIENT] Connection closed [SERVER] Connection closed

6. Wireshark Capture Requirements

Capture actual packets using Wireshark. Apply filter **udp.port == <your_assigned_port>** and save: **project3_handshake.pcap** – SYN, SYN-ACK, ACK **project3_data.pcap** – DATA and DATA-ACK (showing retries) **project3_teardown.pcap** – FIN and FIN-ACK Ensure your captures show retransmissions caused by delayed ACKs.

7. Report Requirements

Your **report.pdf** must include: **Implementation Summary:** explain handshake, data transfer (with delay), and teardown logic. **Capture Analysis:** describe what each .pcap file shows and how retransmissions appear. **Discussion:** why ACK delay triggers retransmission and how your RUDP ensures reliable delivery. Only .pcap captures are accepted; screenshots are not required.

8. Submission

Submit a ZIP file named **<netid>_project3.zip** containing: rudp_client.py rudp_server.py project3_handshake.pcap project3_data.pcap project3_teardown.pcap report.pdf If working in a **group of 2**, submit only one ZIP file with both **names and NetIDs** clearly mentioned in the report. Submit on Canvas by **11:59 PM, November 10, 2025.**