

CS352 Project 3 — Reliable UDP (RUDP)

Authors: Brian Zhang : bz271 / Christopher Doss : cad407

Date: November 10, 2025

1. Implementation Summary

Environment & files

- `rudp_server_skeleton.py` (modified)
- `rudp_client_skeleton.py` (modified)
- Assigned UDP port used for testing: **30147**
- Capture tool: `tshark` (pcap files: `project3_handshake.pcap`, `project3_data.pcap`, `project3_teardown.pcap`)

Protocol implemented

- Message types (1 byte): `SYN=1`, `SYN-ACK=2`, `ACK=3`, `DATA=4`, `DATA-ACK=5`, `FIN=6`, `FIN-ACK=7`.
- Header: `type (1B) | seq (4B, big-endian) | len (2B) | payload.`
- Stop-and-wait semantics: the client sends one `DATA` and waits for matching `DATA-ACK(seq)` before sending the next.
- Retransmission: client RTO = 0.5 s, `RETRIES = 10` (configurable). Server inserts a random ACK delay (ms) before replying.

Handshake (three-way)

- `Client -> Server : SYN (seq=0)`
- `Server -> Client : SYN-ACK (seq=0)`
- `Client -> Server : ACK (seq=0)`

After the final ACK the client prints `Connection established` and both sides treat the connection as established.

Data transfer (stop-and-wait)

- Client splits application message into ~200-byte chunks and sends `DATA(seq=n)` for `n=0,1,...`
- After sending `DATA(seq=n)` the client waits for `DATA-ACK(seq=n)`. If no matching ACK arrives within RTO, the client retransmits `DATA(seq=n)`. Retries up to `RETRIES`.
- Server maintains `expect_seq` and on in-order arrival sends `DATA-ACK(seq)` after a random delay. On out-of-order arrival server re-ACKs `expect_seq-1`.

Teardown

- `Client -> Server : FIN`
- `Server -> Client : FIN-ACK`
- Client prints `Connection closed` after receiving `FIN-ACK`.

Server delay used in captures

- For deterministic retransmits during captures the server was run with forced delays: `--min-delay 700 --max-delay 1200` (ms) so delays exceed client RTO.
-

2. Capture Analysis

All captures used the capture filter (loopback) for the assigned port: `udp.port == 30147`.

2.1 project3_handshake.pcap — handshake

`tshark -r project3_handshake.pcap` produced:

```
1 0.000000000 127.0.0.1 → 127.0.0.1 UDP 49 50940 → 30147 Len=7
2 0.000277477 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
3 0.000337059 127.0.0.1 → 127.0.0.1 UDP 49 50940 → 30147 Len=7
```

Interpretation

- Packet 1: client SYN → server.
- Packet 2: server SYN-ACK → client.
- Packet 3: client final ACK → server.
- Handshake completes with no retransmits.

2.2 project3_data.pcap — data and DATA-ACK (retransmissions present)

Partial `tshark -r project3_data.pcap` output (selected frames):

```
2 0.302472406 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
5 1.018251774 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
6 1.018367713 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
10 1.650879975 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
11 1.650993298 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
12 1.651044813 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
25 2.464271864 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
26 2.464395196 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
27 2.464438946 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
28 2.464477621 127.0.0.1 → 127.0.0.1 UDP 49 30147 → 50940 Len=7
```

Interpretation / evidence of retransmits

- Multiple short UDP packets with identical sizes and close timestamps indicate the client retransmitted **DATA** for the same sequence number before receiving the delayed **DATA-ACK** from the server.
- Because the server was forcing ACK delays (700–1200 ms) and the client RTO = 500 ms, the client timed out and retransmitted; the delayed ACKs arrived later.

2.3 project3_teardown.pcap — FIN / FIN-ACK

`tshark -r project3_teardown.pcap` produced:

1 0.000000000	127.0.0.1 → 127.0.0.1	UDP 49 50940 → 30147 Len=7
2 0.000128718	127.0.0.1 → 127.0.0.1	UDP 49 50940 → 30147 Len=7
3 0.000181810	127.0.0.1 → 127.0.0.1	UDP 49 50940 → 30147 Len=7
4 0.000229293	127.0.0.1 → 127.0.0.1	UDP 49 50940 → 30147 Len=7
5 0.000313382	127.0.0.1 → 127.0.0.1	UDP 49 30147 → 50940 Len=7

Interpretation

- Contains the FIN from client and FIN-ACK from server; teardown completes cleanly in this capture.
-

3. Discussion

Why ACK delay triggers retransmission

- The client starts a timer of RTO seconds after sending a **DATA** segment. If the **DATA-ACK** does not arrive before the RTO expires, the client retransmits the DATA.
- By intentionally delaying the server ACK longer than the client RTO (700–1200 ms > 500 ms), the client's timer expires and retransmits. The capture shows duplicate DATA frames before the later-arriving ACK.

How the RUDP implementation ensures reliability and ordering

- Sequence numbers & ACKs: Server accepts only **expect_seq** and sends **DATA-ACK** for that seq; duplicates or out-of-order segments are not accepted as new data.
- Retransmissions: Client retransmits on timeout until ACK is received or retries exhausted.
- Re-ACKing helps recovery: server re-ACKs the last in-order seq when it receives out-of-order packets, telling the sender what it still expects.

Combined, these mechanisms provide reliable, ordered delivery for this simplified protocol.
