

# MP2 Report

---

## Overview

---

This project implements a connection-oriented, reliable, pipelined transport protocol similar to TCP. The protocol supports reliable data transfer between a sender and receiver using **flow control**, **congestion control**, **packet sequencing**, and **error handling**.

## Specification of protocol

---

The protocol includes key features of TCP, such as **reliability**, **flow control**, **congestion control**, and **connection management**. We use **UDP sockets** to build these features from scratch, as UDP is a lightweight, connectionless protocol.

### Connection-oriented

- Three-Way Handshake: The exchange of `SYN`, `SYN-ACK`, and `ACK` in the sender (`Sender.py`) and receiver (`Reveive.py`) ensures a connection is established before data transfer begins.

### Reliable

- ACKs and Sequence ID: For every packet received, teh receievr sends an `ACK:<>sequence_number`. This confirms the succesful receipt pf the package.
- Timeout: If the sender doesn't receive an ACK within a specified timeout (`TIMEOUT_INTERVAL`), it will retransmits the unacknowledged packet.

### Pipline (Flow / Congestion Control):

- Using Go-back-N: We have a dynamic window size in this case. We use `cwnd`, `sshtthresh` to control our window size.
- Receiver Advertised Window: The receiver advertises its available buffer space, and the sender adjusts the number of packets accordingly.
- Using Congestion Window `cwnd`: Starts with a value of 1 and grows exponetially during the slow start phase.
- Using `sshtthresh`: Reduece the congestion window size upon detecting a timeout, resetting to a slower start phase for better network adaptability.

### Performance:

- The sender leverages multi-threading for packet transmission to parallelize operations.

- Our sender will wait for a specified period to receive all potential ACKs after transmitting a batch of packages.

## Connection Teardown

- The use of `FIN` and `FIN-ACK` ensures that the connection is closed, indicating that all data has been delivered reliably.

## Test procedures

---

### Simulation of loss and errors

- Set `LOSS_PROBABILITY = 0.2`: We use random function to drop packets randomly with 20% probability.

### Packets Set Up:

- Prepare a long string message.
- Divide the message into 14 packages. Each packet has a length of 128 bytes.

### Step:

- Start receiver first `python Receive.py`
- Then Start the Sender `python Sender.py`

## Traffic Capture:

Apply a display filter ... <17>
traffic\_part2\_withloss.pcapng

No.	Time	Source	Destination	Protocol	Length	Time to Live	Info
1	2024-11-28 20:17:25.208075911	127.0.0.1	127.0.0.1	UDP	45	64	44400 → 12345 Len=3
2	2024-11-28 20:17:25.208153019	127.0.0.1	127.0.0.1	UDP	49	64	12345 → 44400 Len=7
3	2024-11-28 20:17:25.208217864	127.0.0.1	127.0.0.1	UDP	45	64	44400 → 12345 Len=3
4	2024-11-28 20:17:25.208281367	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
5	2024-11-28 20:17:25.208320650	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
6	2024-11-28 20:17:27.210839410	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
7	2024-11-28 20:17:27.211039398	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
8	2024-11-28 20:17:29.213557801	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
9	2024-11-28 20:17:29.213634498	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
10	2024-11-28 20:17:29.213842069	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
11	2024-11-28 20:17:29.213936426	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
12	2024-11-28 20:17:29.214003939	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
13	2024-11-28 20:17:29.214052571	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
14	2024-11-28 20:17:29.214201375	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
15	2024-11-28 20:17:29.214281530	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
16	2024-11-28 20:17:31.216786952	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
17	2024-11-28 20:17:31.216883658	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
18	2024-11-28 20:17:31.217089565	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
19	2024-11-28 20:17:31.217153552	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
20	2024-11-28 20:17:31.217451514	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
21	2024-11-28 20:17:31.217529196	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
22	2024-11-28 20:17:31.217573367	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
23	2024-11-28 20:17:31.217671953	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
24	2024-11-28 20:17:31.217797413	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
25	2024-11-28 20:17:31.217826059	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
26	2024-11-28 20:17:31.218086406	127.0.0.1	127.0.0.1	UDP	172	64	44400 → 12345 Len=130
27	2024-11-28 20:17:31.218161601	127.0.0.1	127.0.0.1	UDP	173	64	44400 → 12345 Len=131
28	2024-11-28 20:17:31.218205217	127.0.0.1	127.0.0.1	UDP	173	64	44400 → 12345 Len=131
29	2024-11-28 20:17:31.218248056	127.0.0.1	127.0.0.1	UDP	173	64	44400 → 12345 Len=131
30	2024-11-28 20:17:31.218297764	127.0.0.1	127.0.0.1	UDP	47	64	12345 → 44400 Len=5
31	2024-11-28 20:17:31.218368130	127.0.0.1	127.0.0.1	UDP	48	64	12345 → 44400 Len=6

The first three packets are used for the Three-Way Handshake.

After completing the Three-Way Handshake, the sender begins transmitting the first message.

If no errors or losses occur, the Congestion Window ( `cwnd` ) will increase with each iteration (e.g., 20-22, 26-29).

## Error Handling

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	45	43841 → 12345 Len=3
2	0.000134291	127.0.0.1	127.0.0.1	UDP	49	12345 → 43841 Len=7
3	0.000165045	127.0.0.1	127.0.0.1	UDP	45	43841 → 12345 Len=3
4	0.000188070	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
5	0.000270139	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
6	0.000295348	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
7	0.000305872	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
8	0.000378773	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
9	0.000390652	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
10	0.000405436	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
11	0.000415747	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
12	0.000423841	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
13	0.000495029	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
14	0.000504971	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
15	0.000509561	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
16	4.004697219	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
17	4.004886705	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
18	4.005233460	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
19	4.005307604	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130
20	4.005342661	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
21	4.005400841	127.0.0.1	127.0.0.1	UDP	47	12345 → 43841 Len=5
22	4.005527595	127.0.0.1	127.0.0.1	UDP	172	43841 → 12345 Len=130

In this case (e.g., 10–16), the receiver expected to receive 4 packets, but only 3 packets were sent. As a result, the Congestion Window ( `cwnd` ) is reset to 1, and the sender will retransmit the missing packet based on the ACKs received.

### Sender output:

```
1  Sender: Setting up connection. Sending SYN
2  Sender: Connection established. Send ACK
3  Sender: Sent packet: 0
4
5  Sender: Received ACK: 0
6
7  Sender: Sent packet: 1
8  Sender: Sent packet: 2
9
10 Sender: Received ACK: 1
11 Sender: Received ACK: 2
12
13 Sender: Sent packet: 3
14 Sender: Packet 4 lost
15 Sender: Sent packet: 5
16 Sender: Sent packet: 6
17
18 Sender: Received ACK: 3
19 Sender: Received ACK: 3
20 Sender: Received ACK: 3
21 Sender: Timeout occurred, retransmitting...
22
```

```
23 Sender: Packet 4 lost
24
25 Sender: Timeout occurred, retransmitting...
26
27 Sender: Sent packet: 4
28
29 Sender: Received ACK: 4
30
31 Sender: Sent packet: 5
32 Sender: Sent packet: 6
33
34 Sender: Received ACK: 5
35 Sender: Received ACK: 6
36
37 Sender: Sent packet: 7
38 Sender: Sent packet: 8
39 Sender: Sent packet: 9
40
41 Sender: Received ACK: 7
42 Sender: Received ACK: 8
43 Sender: Received ACK: 9
44
45 Sender: Packet 10 lost
46 Sender: Packet 11 lost
47 Sender: Sent packet: 12
48 Sender: Sent packet: 13
49
50 Sender: Received ACK: 9
51 Sender: Received ACK: 9
52 Sender: Timeout occurred, retransmitting...
53
54 Sender: Sent packet: 10
55
56 Sender: Received ACK: 10
57
58 Sender: Sent packet: 11
59 Sender: Sent packet: 12
60
61 Sender: Received ACK: 11
62 Sender: Received ACK: 12
63
64 Sender: Sent packet: 13
65 Sender: Sent packet: 14
66
67 Sender: Received ACK: 13
```

```
68 | Sender: Received ACK: 14
69 |
70 | All packets acknowledged.
71 | Sender: Tearing down connection...
72 | Sender: Connection closed.
```

## Receiver Output

```
1 | Receiver: Waiting for connection...
2 | Receiver: Received SYN, sending SYN-ACK...
3 | Receiver: Connection established.
4 | Receiver: Received packet: 0
5 | Receiver: Received packet: 1
6 | Receiver: Received packet: 2
7 | Receiver: Received packet: 3
8 | Receiver: Received packet: 4
9 | Receiver: Received packet: 5
10 | Receiver: Received packet: 6
11 | Receiver: Received packet: 7
12 | Receiver: Received packet: 8
13 | Receiver: Received packet: 9
14 | Receiver: Received packet: 10
15 | Receiver: Received packet: 11
16 | Receiver: Received packet: 12
17 | Receiver: Received packet: 13
18 | Receiver: Received packet: 14
19 | Receiver: connection teardown initiated by sender.
20 | Receiver: connection closed.
```