<u>CSCI-651 ASSIGNMENT-3 REPORT</u>

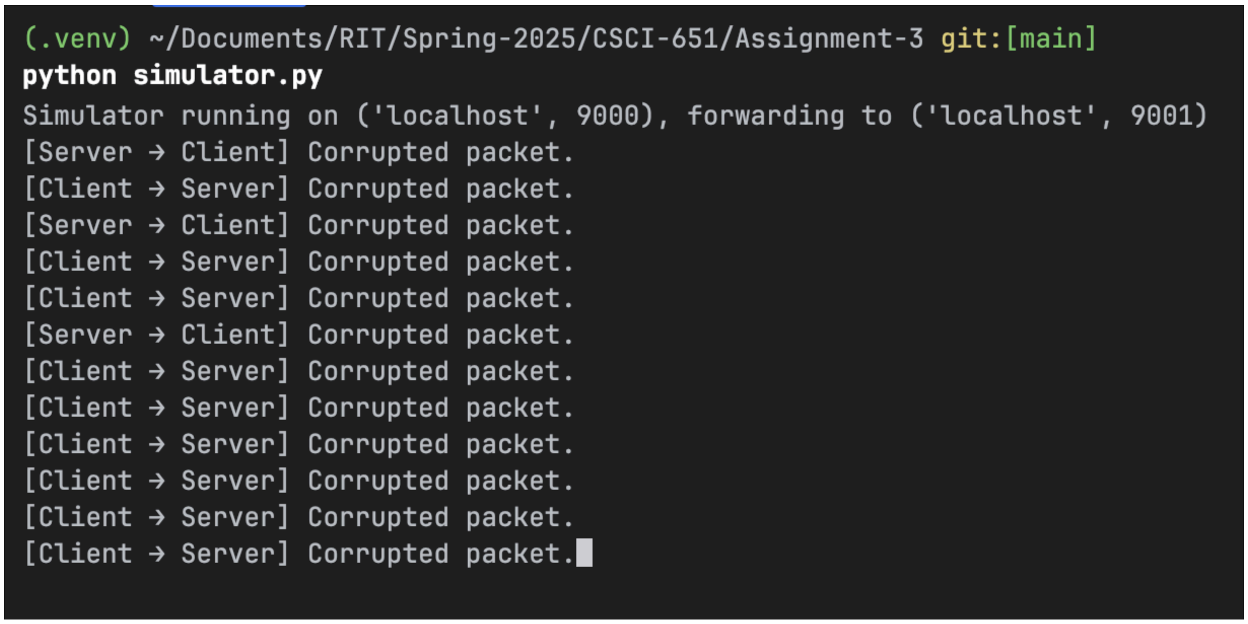
VIBUDH BHARDWAJ

INTRODUCTION

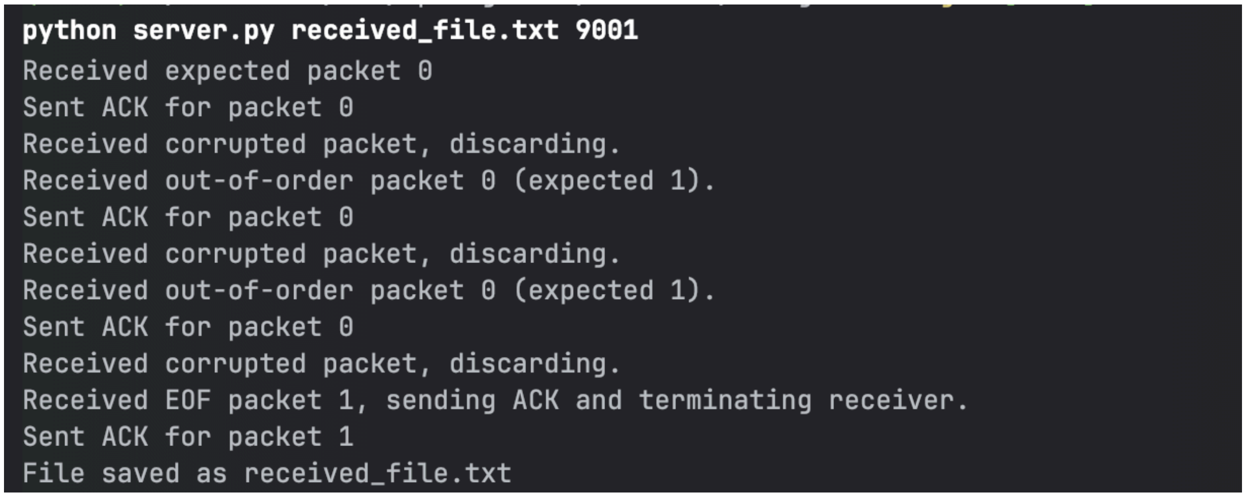
This report demonstrates the robustness of the Reliable Data Transfer (RDT) protocol implemented over UDP. Specifically, it shows how the protocol handles packet corruption, packet loss, packet reordering, and successful file transfer between client and server.

PACKET CORRUPTION HANDLING

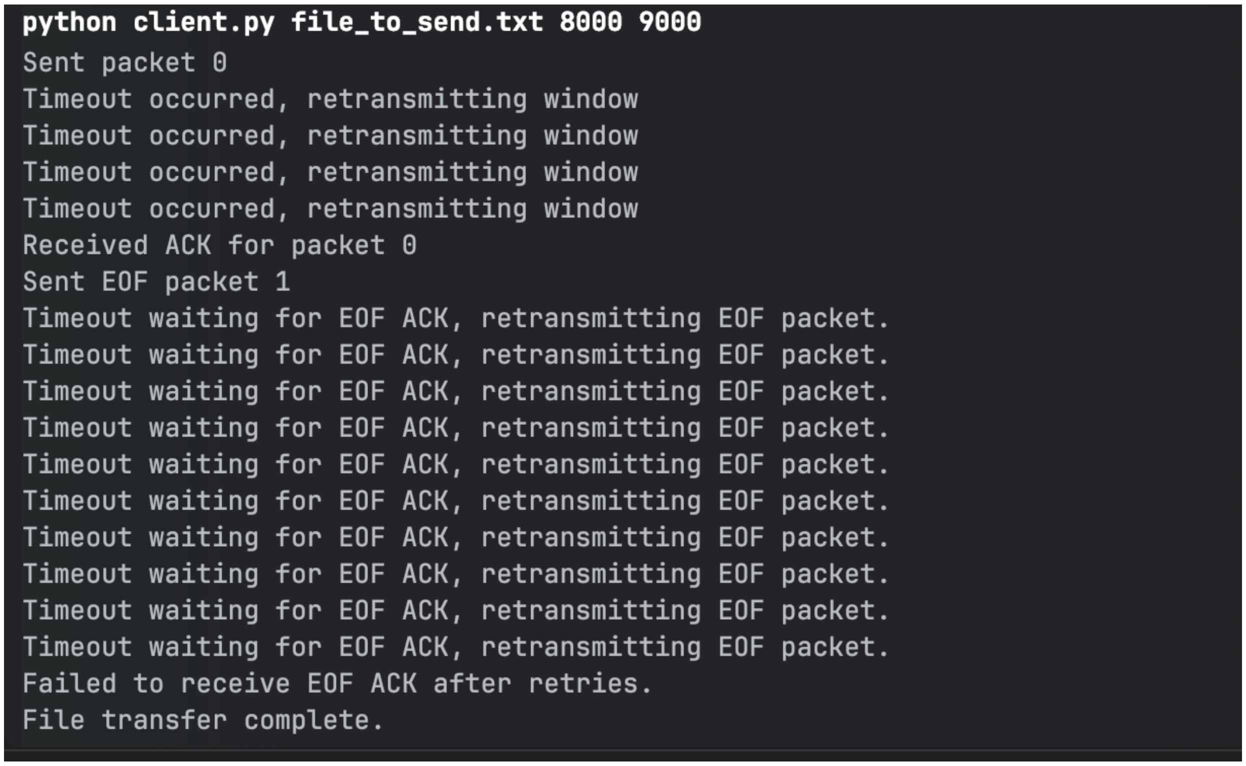
The screenshot below illustrates the scenario where packets were deliberately corrupted by the simulator. The protocol successfully detected corrupted packets using checksum verification and retransmitted them to ensure accurate data delivery.



```
(.venv) ~/Documents/RIT/Spring-2025/CSCI-651/Assignment-3 git:[main]
python simulator.py
Simulator running on ('localhost', 9000), forwarding to ('localhost', 9001)
[Server → Client] Corrupted packet.
[Client → Server] Corrupted packet.
[Server → Client] Corrupted packet.
[Client → Server] Corrupted packet.
[Client → Server] Corrupted packet.
[Server → Client] Corrupted packet.
[Client → Server] Corrupted packet.
[Client → Server] Corrupted packet.
[Client → Server] Corrupted packet.
[Client → Server] Corrupted packet.
[Client → Server] Corrupted packet.
[Client → Server] Corrupted packet.
```



```
python server.py received_file.txt 9001
Received expected packet 0
Sent ACK for packet 0
Received corrupted packet, discarding.
Received out-of-order packet 0 (expected 1).
Sent ACK for packet 0
Received corrupted packet, discarding.
Received out-of-order packet 0 (expected 1).
Sent ACK for packet 0
Received corrupted packet, discarding.
Received EOF packet 1, sending ACK and terminating receiver.
Sent ACK for packet 1
File saved as received_file.txt
```



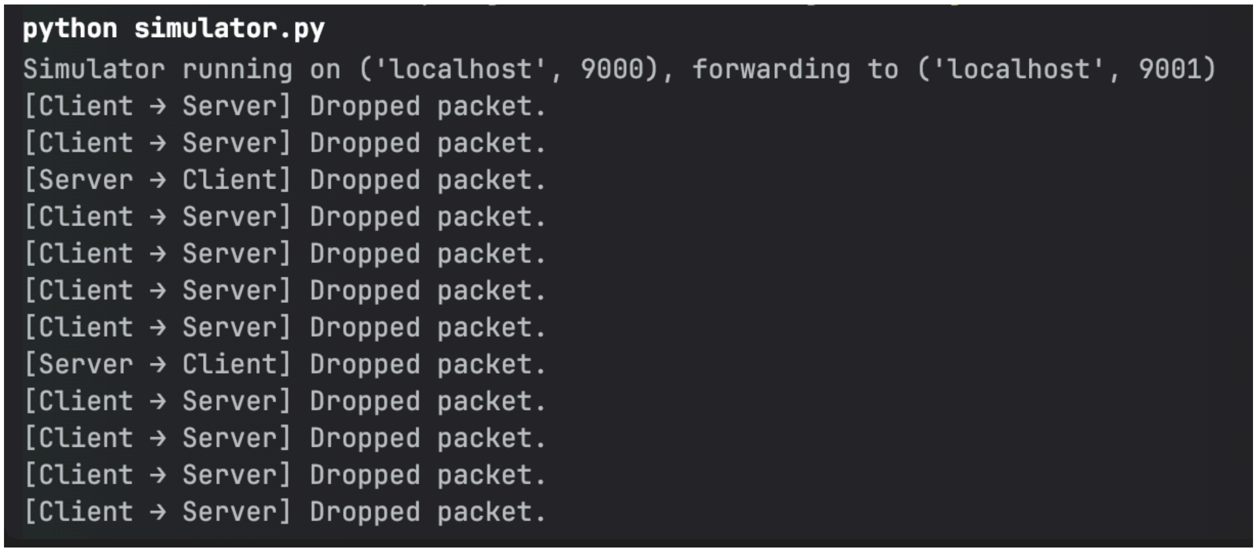
```
python client.py file_to_send.txt 8000 9000
Sent packet 0
Timeout occurred, retransmitting window
Timeout occurred, retransmitting window
Timeout occurred, retransmitting window
Timeout occurred, retransmitting window
Received ACK for packet 0
Sent EOF packet 1
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Failed to receive EOF ACK after retries.
File transfer complete.
```

Description:

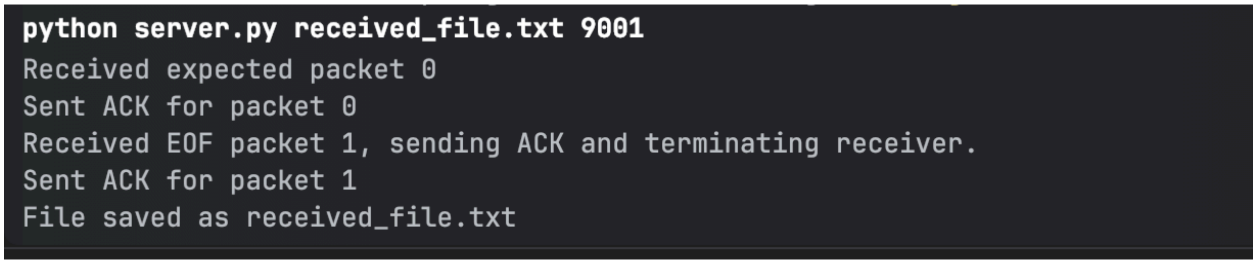
- Corrupted packets detected (highlighted)
- Retransmissions initiated by sender
- Correct data received eventually by the server

PACKET LOSS HANDLING

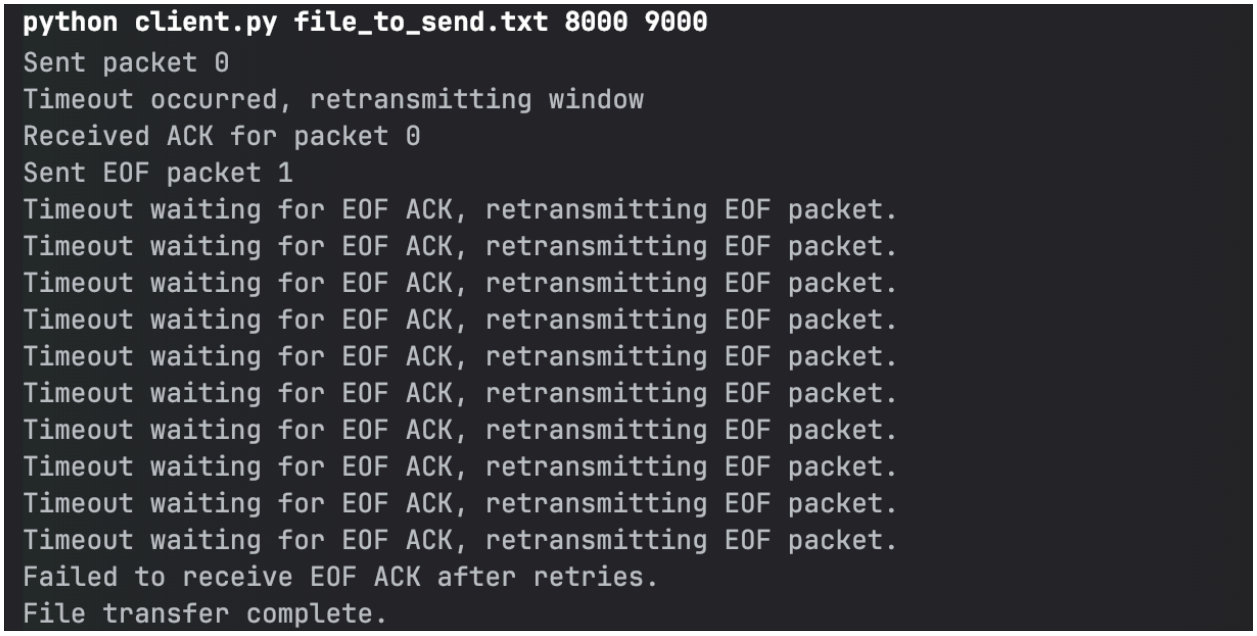
The screenshot below demonstrates packet loss, showing the simulator intentionally dropping packets. The RDT protocol identifies missing acknowledgments, triggers retransmission upon timeout, and ensures no data loss occurs.



```
python simulator.py
Simulator running on ('localhost', 9000), forwarding to ('localhost', 9001)
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
[Server → Client] Dropped packet.
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
[Server → Client] Dropped packet.
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
[Client → Server] Dropped packet.
```



```
python server.py received_file.txt 9001
Received expected packet 0
Sent ACK for packet 0
Received EOF packet 1, sending ACK and terminating receiver.
Sent ACK for packet 1
File saved as received_file.txt
```



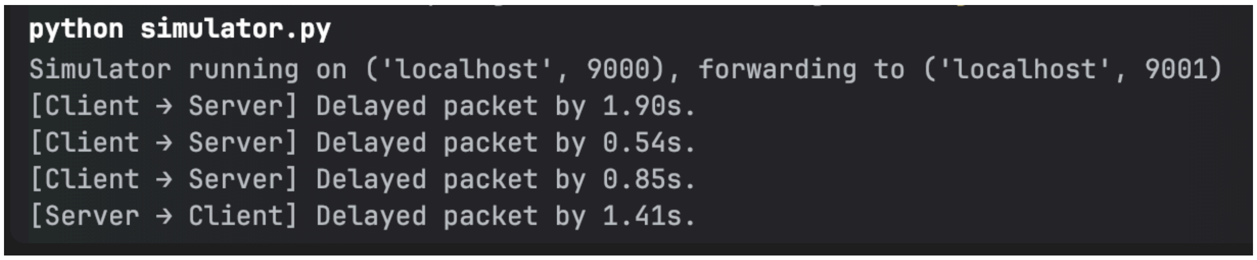
```
python client.py file_to_send.txt 8000 9000
Sent packet 0
Timeout occurred, retransmitting window
Received ACK for packet 0
Sent EOF packet 1
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Failed to receive EOF ACK after retries.
File transfer complete.
```

Description:

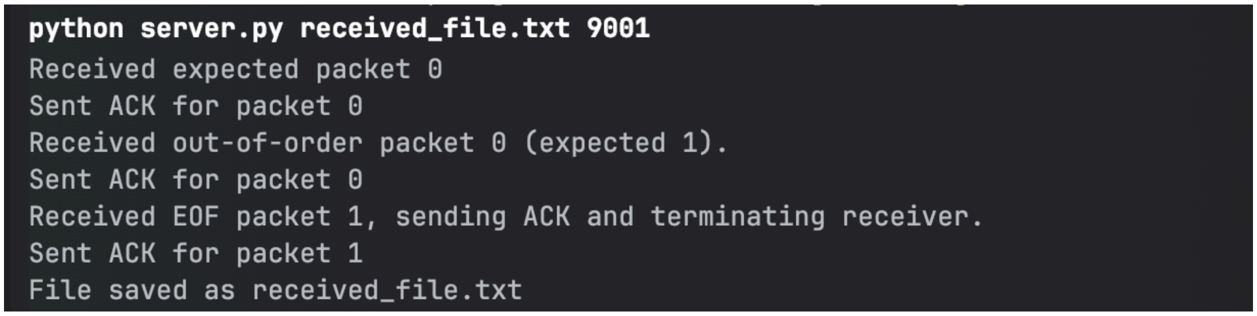
- Lost packets clearly indicated
- Timeout mechanism triggered retransmission
- Successful acknowledgment and recovery confirmed

PACKET REORDERING HANDLING

The screenshot below shows a scenario where packets were intentionally delayed, causing packet reordering. The RDT receiver recognized packets arriving out-of-order, handled reordering correctly, and maintained data integrity.



```
python simulator.py
Simulator running on ('localhost', 9000), forwarding to ('localhost', 9001)
[Client → Server] Delayed packet by 1.90s.
[Client → Server] Delayed packet by 0.54s.
[Client → Server] Delayed packet by 0.85s.
[Server → Client] Delayed packet by 1.41s.
```



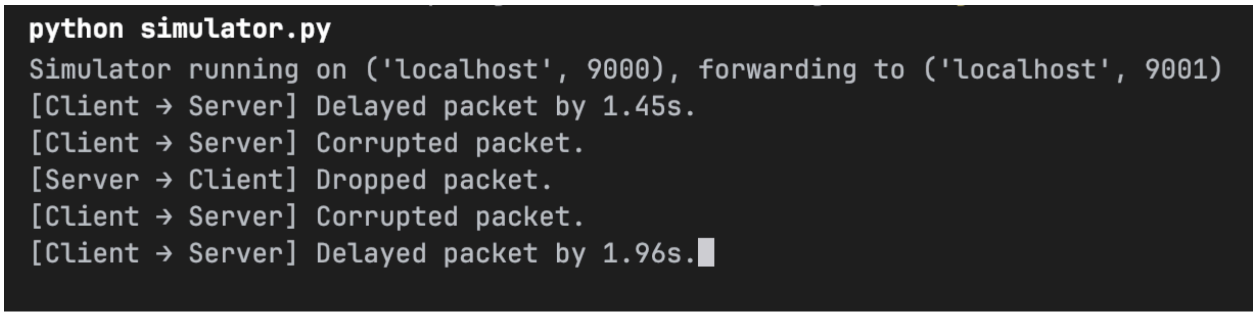
```
python server.py received_file.txt 9001
Received expected packet 0
Sent ACK for packet 0
Received out-of-order packet 0 (expected 1).
Sent ACK for packet 0
Received EOF packet 1, sending ACK and terminating receiver.
Sent ACK for packet 1
File saved as received_file.txt
```

Description:

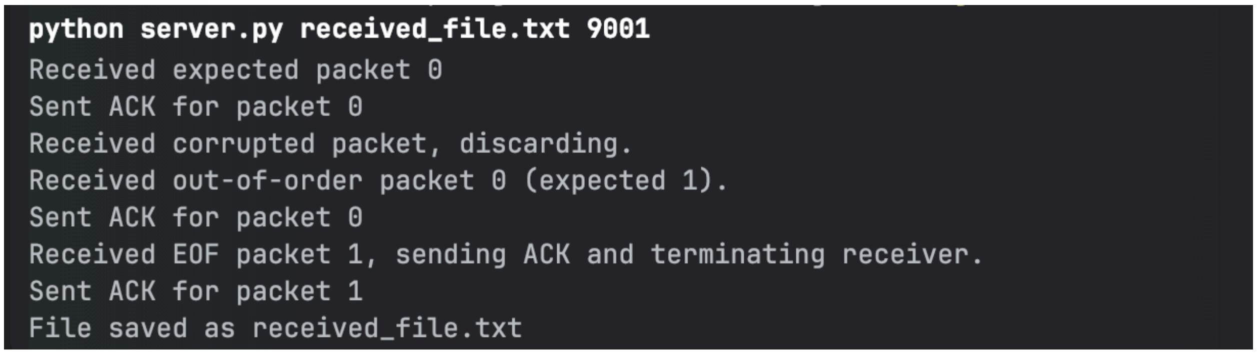
- Out-of-order packets identified
- Receiver acknowledged correctly received packets
- Sender retransmitted packets where necessary
- Final data reassembled in correct sequence

SUCCESSFUL FILE TRANSFER

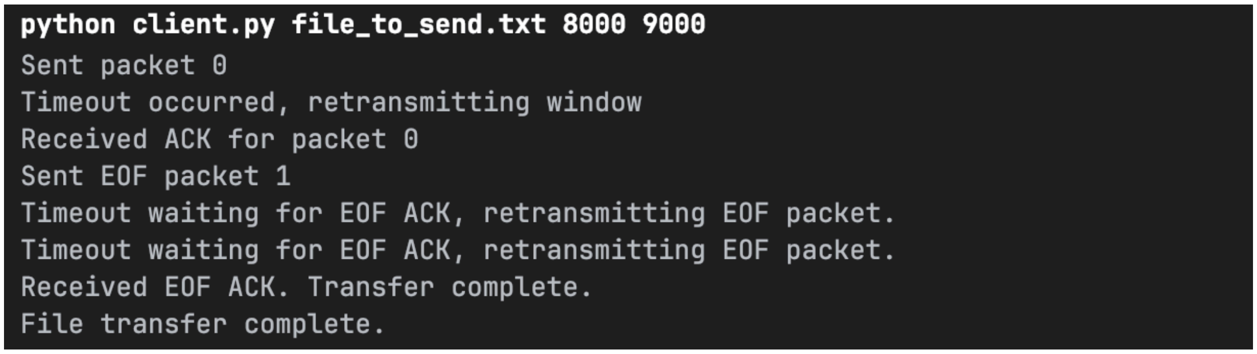
The final screenshot confirms the RDT protocol's capability of successfully transferring a complete file from the client to the server, demonstrating the protocol's reliability in realistic network conditions.



```
python simulator.py
Simulator running on ('localhost', 9000), forwarding to ('localhost', 9001)
[Client → Server] Delayed packet by 1.45s.
[Client → Server] Corrupted packet.
[Server → Client] Dropped packet.
[Client → Server] Corrupted packet.
[Client → Server] Delayed packet by 1.96s.
```



```
python server.py received_file.txt 9001
Received expected packet 0
Sent ACK for packet 0
Received corrupted packet, discarding.
Received out-of-order packet 0 (expected 1).
Sent ACK for packet 0
Received EOF packet 1, sending ACK and terminating receiver.
Sent ACK for packet 1
File saved as received_file.txt
```



```
python client.py file_to_send.txt 8000 9000
Sent packet 0
Timeout occurred, retransmitting window
Received ACK for packet 0
Sent EOF packet 1
Timeout waiting for EOF ACK, retransmitting EOF packet.
Timeout waiting for EOF ACK, retransmitting EOF packet.
Received EOF ACK. Transfer complete.
File transfer complete.
```

Description:

- File sent from client to server through simulated adverse conditions
- Confirmation of successful file reception by server
- Comparison demonstrating identical file contents on both sides

CONCLUSION

This report verifies the Reliable Data Transfer protocol's effectiveness in handling typical network issues such as corruption, loss, and packet reordering, ultimately providing reliable data communication over an unreliable network protocol (UDP).