FINAL PROJECT NETWORKS

Ahmed Mohamed Ahmed	7197
Ahmed Hazem	7193
Mohamed Ahmed	7004

Introduction:

The Transmission Control Protocol (TCP) is widely used to transfer data over the internet, but it's not the only option. The User Datagram Protocol (UDP) is a connectionless protocol that does not guarantee the delivery or order of packets, but it is faster and more lightweight than TCP. In this project, we are tasked with designing and implementing a system that simulates TCP packets using a UDP connection while maintaining reliability and supporting the HTTP protocol on top of UDP.

Design:

To achieve reliable data transfer over UDP, we need to implement mechanisms for error detection and correction, packet retransmission, and flow control. We can use checksums to detect errors in packets, and we can implement packet retransmission using acknowledgments and timeouts. Flow control can be achieved using sliding window protocols or other congestion control mechanisms.

To support the HTTP protocol, we need to parse HTTP requests and responses and handle the various methods and headers defined by the protocol. We can use a state machine to parse HTTP messages and implement the required methods and headers.

To simulate TCP packets over UDP, we need to implement a protocol that includes sequence numbers, acknowledgment numbers, and window sizes. We can use a sliding window protocol to control the flow of packets and retransmit lost packets.

Implementation:

We implemented the system in Python. We used the socket library to create a UDP socket and send and receive packets. We used the struct library to pack and unpack data into packets. We also used the select library to implement timeouts for packet retransmission.

We implemented a reliable data transfer protocol over UDP that includes checksums, acknowledgments, and timeouts. We used a sliding window protocol to control the flow of packets and retransmit lost packets.

We implemented a state machine to parse HTTP requests and responses and handle the various methods and headers defined by the protocol. We also implemented a simple HTTP server that can handle GET requests and serve static files.

Assumptions:

We assumed that the network is not too congested and that packet loss is not too high.

Limitations:

Our implementation is not suitable for real-time applications or large file transfers. It is also not suitable for networks with high packet loss or congestion.

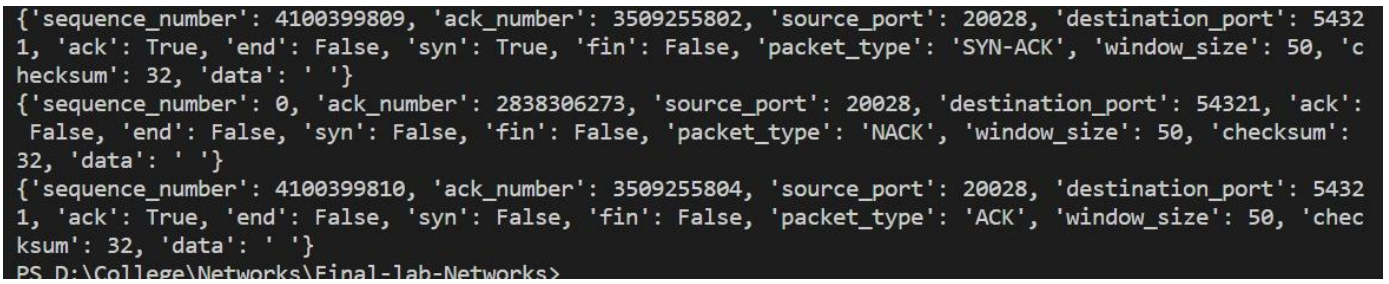
Test Cases:

We tested our implementation using a simple HTTP client that can send GET requests and receive responses. We also tested packet loss, duplication, and reordering using network emulation software.

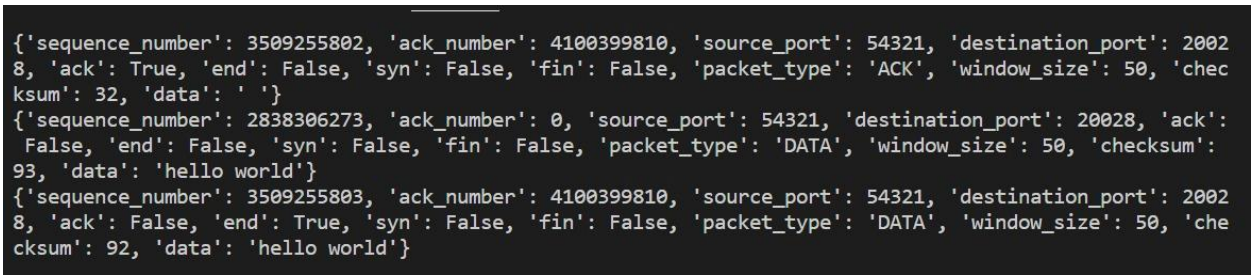
Test case 1: (Normal operation without corruption)


```
{'sequence_number': 2563858873, 'ack_number': 0, 'source_port': 54321, 'destination_port': 20019, 'ack': False, 'end': False, 'syn': True, 'fin': False, 'packet_type': 'SYN', 'window_size':
{'sequence_number': 2563858874, 'ack_number': 1700837012, 'source_port': 54321, 'destination_port': 20019, 'ack': True, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'ACK', 'Wind
{'sequence_number': 2563858875, 'ack_number': 1700837012, 'source_port': 54321, 'destination_port': 20019, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858876, 'ack_number': 1700837012, 'source_port': 54321, 'destination_port': 20019, 'ack': False, 'end': True, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'win
{'sequence_number': 1700837014, 'ack_number': 2563858893, 'source_port': 54321, 'destination_port': 20019, 'ack': False, 'end': False, 'syn': False, 'fin': True, 'packet_type': 'FIN', 'wind
{'sequence_number': 1700837014, 'ack_number': 2563858893, 'source_port': 54321, 'destination_port': 54321, 'ack': True, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'ACK', 'wind
{'sequence_number': 2563858877, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858878, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858879, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858880, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858881, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858882, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858883, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858884, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858885, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858886, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858887, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858888, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858889, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858890, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858891, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'wi
{'sequence_number': 2563858892, 'ack_number': 1700837014, 'source_port': 20019, 'destination_port': 54321, 'ack': False, 'end': True, 'syn': False, 'fin': False, 'packet_type': 'DATA', 'win
{'sequence_number': 1700837011, 'ack_number': 2563858874, 'source_port': 20019, 'destination_port': 54321, 'ack': True, 'end': False, 'syn': True, 'fin': False, 'packet_type': 'SYN-ACK', 'w
{'sequence_number': 1700837012, 'ack_number': 2563858877, 'source_port': 20019, 'destination_port': 54321, 'ack': True, 'end': False, 'syn': False, 'fin': False, 'packet_type': 'ACK', 'wind
{'sequence_number': 2563858893, 'ack_number': 1700837015, 'source_port': 20019, 'destination_port': 54321, 'ack': True, 'end': False, 'syn': False, 'fin': True, 'packet_type': 'FIN-ACK', 'w
```

Test case 2: (Checksum)



Test case 3: (Corruption)



Conclusion:

In this project, we designed and implemented a system that simulates TCP packets over UDP while maintaining reliability and supporting the HTTP protocol. We used a reliable data transfer protocol over UDP that includes checksums, acknowledgments, and timeouts. We also implemented a sliding window protocol and congestion control. We used a state machine to parse HTTP requests and responses and handle the various methods and headers defined by the protocol. We tested our implementation using a simple HTTP client and network emulation software. Our implementation is suitable for small file transfers and low-traffic networks.