Networks

Final Project

TCP using UDP

CCE 2026

Submission Date: 18/5/2025

Name	ID
Rowan Ahmed Nazmi	8382
Mohamed Hesham Shabana	8349
Seif Maged Farouk	8387

Introduction:

The task is to design and implement a system that simulates TCP packets using a UDP connection by extending the user space of your application while maintaining reliability and supporting the HTTP protocol on top of UDP. UDP is a connectionless protocol that does not guarantee the delivery or order of packets. This presents several challenges when attempting to transfer data reliably.

Features Implemented:

1. TCPonUDP Class (UDP_to_TCP.py)

This class simulates a TCP connection over UDP. Since UDP is unreliable by design, this class adds essential TCP-like mechanisms:

- **Three-Way Handshake**: Uses SYN, ACK, and FIN flags to establish and terminate a connection between client and server.
- **Checksums**: Ensures data integrity. The receiver recomputes and verifies the checksum for each packet.
- **Packet Loss and Corruption Simulation**: Probabilistically drops or corrupts packets to simulate real network conditions.
- **Retransmission Mechanism**: Implements a stop-and-wait protocol. If an ACK is not received in time (due to loss/corruption), the packet is resent.
- **Graceful Connection Termination**: Mimics TCP's FIN/ACK closing process to safely end sessions.

2. HTTP Server (httpServer.py)

- Built on top of TCPonUDP, the HTTP server receives requests using the serve_connection() method and sends back structured HTTP responses.
- Supports:
    - **GET requests**: for accessing simple content like /index.html.
    - **POST requests**: processes and prints submitted form data.
    - **404 Handling**: returns a 404 Not Found response for unsupported paths.

3. HTTP Client (httpClient.py)

- Establishes a connection using TCPonUDP.connect(), then sends HTTP requests and receives responses reliably.

- Demonstrates both **GET and POST** functionality over the custom UDP-based reliable channel.
- Provides user feedback and prints the full HTTP response received from the server.

Sample Runs:

Sample #1:

Server Side:

```
C:\Users\Seif\Downloads\UDP-to-TCP>python httpServer.py
[SERVER] Listening on 127.0.0.1:8081
[SERVER] Waiting for new connection...
Server listening for handshake...
Received SYN from ('127.0.0.1', 64277)
Sent SYN-ACK to ('127.0.0.1', 64277)
Received ACK from ('127.0.0.1', 64277)
Connection established
Server ready to receive data or close connection...
Sent ACK for GET
Received data: b'GET /index.html HTTP/1.0\r\n        Host: 127.0.0.1\r\n        Connection: close\r\n
   \r\n        '
[SERVER] Received request:
GET /index.html HTTP/1.0
        Host: 127.0.0.1
        Connection: close


ACK received
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

Client Side:

```
C:\Users\Seif\Downloads\UDP-to-TCP>python httpClient.py
[CLIENT] Sending GET request...
Received SYN-ACK
Connection established
ACK received
Server ready to receive data or close connection...
Sent ACK for GET
Received data: b'HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nConnection: close\r\n\r\nOK'
[CLIENT] Received response:
HTTP/1.0 200 OK
Content-Type: text/html
Connection: close

OK
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

In this sample run, there were no losses or corruptions.

Sample #2:

Server Side:

```
C:\Users\Seif\Downloads\UDP-to-TCP>python httpServer.py
[SERVER] Listening on 127.0.0.1:8081
[SERVER] Waiting for new connection...
Server listening for handshake...
Received SYN from ('127.0.0.1', 58447)
Sent SYN-ACK to ('127.0.0.1', 58447)
Received ACK from ('127.0.0.1', 58447)
Connection established
Server ready to receive data or close connection...
Sent ACK for GET
Received data: b'GET /index.html HTTP/1.0\r\n        Host: 127.0.0.1\r\n        Connection: close\r\n
   \r\n        '
[SERVER] Received request:
GET /index.html HTTP/1.0
        Host: 127.0.0.1
        Connection: close


Packet Lost.
Timeout or parse error, retrying… (1/5)
ACK received
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

Client Side:

```
C:\Users\Seif\Downloads\UDP-to-TCP>python httpClient.py
[CLIENT] Sending GET request...
Received SYN-ACK
Connection established
ACK received
Server ready to receive data or close connection...
Sent ACK for GET
Received data: b'HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nConnection: close\r\n\r\nOK'
[CLIENT] Received response:
HTTP/1.0 200 OK
Content-Type: text/html
Connection: close

OK
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

In this run, the client sent a GET request and faced a simulated packet loss but eventually received the data after retransmission.

Sample #3:

Server Side:

```
Received SYN from ('127.0.0.1', 51492)
Sent SYN-ACK to ('127.0.0.1', 51492)
Received ACK from ('127.0.0.1', 51492)
Connection established
Server ready to receive data or close connection...
[SERVER] Packet error: Invalid Packet: Checksum Mismatch.. Ignoring corrupted packet.
Sent ACK for GET
Received data: b'POST /index.html HTTP/1.0\r\n        Host: 127.0.0.1\r\n        Content-Length: 0\r\n
    Connection: close\r\n        \r\n        '
[SERVER] Received request:
POST /index.html HTTP/1.0
        Host: 127.0.0.1
        Content-Length: 0
        Connection: close


[SERVER] POST data received at /index.html:
Packet Lost.
Timeout or parse error, retrying… (1/5)
Packet Lost.
Timeout or parse error, retrying… (2/5)
ACK received
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

Client Side:

```
C:\Users\Seif\Downloads\UDP-to-TCP>python httpClient.py
[CLIENT] Sending GET request...
Received SYN-ACK
Connection established
Packet Corrupted...
Timeout or parse error, retrying… (1/5)
ACK received
Server ready to receive data or close connection...
Sent ACK for GET
Received data: b'HTTP/1.0 200 OK\r\nContent-Length: 0\r\nConnection: close\r\n\r\n'
[CLIENT] Received response:
HTTP/1.0 200 OK
Content-Length: 0
Connection: close


FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

In this run, the client sent a POST request and faced a simulated packet corruption and loss but eventually received the data after retransmission.

Sample #4:

Server Side:

```
[SERVER] Listening on 127.0.0.1:8081
[SERVER] Waiting for new connection...
Server listening for handshake...
Received SYN from ('127.0.0.1', 56714)
Sent SYN-ACK to ('127.0.0.1', 56714)
Received ACK from ('127.0.0.1', 56714)
Connection established
Server ready to receive data or close connection...
Sent ACK
Received data: b'GET /ix.html HTTP/1.0\r\n        Host: 127.0.0.1\r\n
        Connection: close\r\n        \r\n        '
[SERVER] Received request:
GET /ix.html HTTP/1.0
        Host: 127.0.0.1
        Connection: close


ACK received
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

```
C:\Users\Seif\Downloads\UDP-to-TCP>python httpClient.py
[CLIENT] Sending GET request...
Received SYN-ACK
Connection established
ACK received
Server ready to receive data or close connection...
Sent ACK
Received data: b'HTTP/1.0 404 Not Found\r\nContent-Type: text/plain\r\nConnect
ion: close\r\n\r\n404 Not Found.'
[CLIENT] Received response:
HTTP/1.0 404 Not Found
Content-Type: text/plain
Connection: close

404 Not Found.
FIN sent, waiting for peer response...
FIN received from peer.
ACK sent for peer's FIN.
ACK received.
Connection closed.
```

Wireshark: