COMPUTER NETWORKS

Final Project

Submitted by: Omar Essam – 7859

Mervat Tamer – 7779

I. Brief Overview:

Aim: Design and Implementation of a Reliable UDP-based TCP Packet Simulation System with HTTP/1.0 Support

The objective of this project is to outline the design and implementation of a system capable of simulating TCP packets over UDP while ensuring reliability and supporting the HTTP/1.0 protocol. This involves addressing challenges such as packet loss, duplication, and reordering, while maintaining a balance between performance and reliability.

System Architecture:
The system utilizes Python's socket library for low-level networking. A newly created class facilitates the transition between UDP and TCP-like behavior. This class handles connection establishment through a handshake mechanism, implements reliable data transfer using acknowledgments and retransmissions, and supports HTTP/1.0 requests and responses.

Reliability Mechanisms:
1. Error Detection and Correction: Checksums, such as CRC-32, are calculated for packets before transmission and verified upon receipt. Incorrect checksums result in packet dropping.
2. Packet Retransmission: Retransmissions are triggered by timeouts when acknowledgments are not received from the receiver.
3. Flow Control: Sliding window protocols or congestion control mechanisms are employed to regulate the flow of data.

HTTP/1.0 Support:
The system parses HTTP requests and responses, supporting methods like GET and POST. It handles HTTP headers and responses with status codes such as OK and NOT FOUND.

Error Handling and Testing:
Error handling is implemented for scenarios like invalid packets, timeouts, and graceful connection termination. Testing involves

simulating packet loss, corruption, and other scenarios to validate the system's robustness.

Covered:
- HTTP 1.0 compliance with special consideration to HTTP headers.
- Utilization of Python UDP socket for implementation.
- Creation of a custom class to manage the transition between UDP and TCP behaviors.
- Implementation of HTTP server and client supporting GET and POST methods.
- Integration of Stop-and-Wait protocol for reliable data transfer.
- Calculation of checksums for packet integrity and handling of false checksums.
- Special methods for simulating packet loss and corruption.
- Consideration of retransmission, duplicate packet handling, sequence numbers, handshake, flags (e.g., SYN, SYNACK, ACK, FIN), and timeouts.

II. Source Code:

a. Server:

```
import time
import socket
class server:
    def __init__(self,PACKET_SIZE = 100,FORMAT = "utf-8",PORT = 5000):
        self.seq_num = 10
        self.ack_num = 0
        self.PACKET_SIZE = PACKET_SIZE
        self.FORMAT = FORMAT
        self.PORT = PORT
        self.IP = socket.gethostbyname(socket.gethostname())
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        server_address = (self.IP,self.PORT)
        self.server_socket.bind(server_address)
        print('UDP server is running on {}:{}'.format(*server_address))

        if self.establish_connection():
            self.check_received()
    def establish_connection(self):
        MAX_RETRIES = 3
        RETRY_TIMEOUT = 2  # seconds
```

```
        retries = 0
        PACKET_SIZE = self.PACKET_SIZE
        while retries < MAX_RETRIES:
            # Listen for incoming SYN packets
            data, client_address = self.server_socket.recvfrom(PACKET_SIZE)
            data = data.decode()
            # f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_window}"
            seq_num = int(data.split(":")[0])
            packet = data.split(":")[2]
            received_checksum = int(data.split(":")[3])
            calculated_checksum = udp_checksum(seq_num,0,0,packet)

            # Check for SYN packet
            if packet == "SYN" and received_checksum == calculated_checksum:
                # Send SYN-ACK packet if SYN received
                ack_num = seq_num+len(packet)
                seq_num = self.seq_num
                data = "SYN-ACK"
                checksum = udp_checksum(seq_num,ack_num,0,data)

                packet = f"{seq_num}:{ack_num}:{data}:{checksum}:{0}"
                self.server_socket.sendto(packet.encode(), client_address)

                # Listen for ACK packet from the client
                self.server_socket.settimeout(RETRY_TIMEOUT)
                try:
                    r_data, client_address =
self.server_socket.recvfrom(PACKET_SIZE)
                    r_data = r_data.decode()
                    ack_num = int(r_data.split(":")[1])
                    r_seq_num = int(r_data.split(":")[0])
                    packet = r_data.split(":")[2]
                    received_checksum = int(r_data.split(":")[3])
                    calculated_checksum =
udp_checksum(r_seq_num,ack_num,0,packet)
                    # Check if the received packet is an ACK packet
                    if packet == "ACK" and ack_num == seq_num+len(data) and
calculated_checksum == received_checksum:
                        # da el haneshta8al beeh ba3d keda
                        self.seq_num = ack_num
                        self.ack_num = r_seq_num+len("ACK")
                        print("Connection established successfully.")
                        print(self.seq_num,self.ack_num)
                        return True
```

```
                except socket.timeout:
                    # Timeout occurred, retry
                    print("Timeout occurred. Retrying...")
                    retries += 1
                    continue
            else :
                print("Error in the message")

        print("Connection establishment failed after {}
retries.".format(MAX_RETRIES))
        return False
    def check_received(self):
        # f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_window}"
        PACKET_SIZE = self.PACKET_SIZE
        RETRY_TIMEOUT = 15
        self.server_socket.settimeout(RETRY_TIMEOUT)
        data, client_address = self.server_socket.recvfrom(PACKET_SIZE)
        data = data.decode()
        seq_num = int(data.split(":")[0])
        ack_num = int(data.split(":")[1])
        request = data.split(':')[2]
        received_checksum = int(data.split(':')[3])
        calc_checksum = udp_checksum(seq_num,ack_num,0,request)
        if calc_checksum == received_checksum and request == "FIN":
            self.seq_num = ack_num
            self.ack_num = seq_num+len("FIN")
            self.connection_termination(client_address)
        else:
            # check if data is corrupted here and send neg ack
            if calc_checksum == received_checksum:
                if request.split()[0] == "GET":
                    print("Received GET request")
                    path = request.split()[1]
                    self.seq_num = ack_num
                    self.ack_num = seq_num+len(request)
                    # to resend if negative ack
                    while True:
                        if path == "/path/to/resource":
                            data = "HTTP/1.0 200 OK\r\nSuccessful GET Request"
                            checksum =
udp_checksum(self.seq_num,self.ack_num,0,data)
                            packet =
f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
                            self.server_socket.sendto(packet.encode(),
client_address)
```

```
                        else :
                            data = "HTTP/1.0 404 Not Found\r\nPath Not Found"
                            checksum =
udp_checksum(self.seq_num,self.ack_num,0,data)
                            packet =
f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
                            self.server_socket.sendto(packet.encode(),
client_address)
                            # Listen for ACK packet from the client
                        self.server_socket.settimeout(RETRY_TIMEOUT)
                        try:
                            r_data, client_address =
self.server_socket.recvfrom(PACKET_SIZE)
                            r_data = r_data.decode()
                            ack_num = int(r_data.split(":")[1])
                            r_seq_num = int(r_data.split(":")[0])
                            packet = r_data.split(":")[2]
                            received_checksum = int(r_data.split(":")[3])
                            calc_checksum =
udp_checksum(r_seq_num,ack_num,0,packet)
                            if calc_checksum==received_checksum:
                                if ack_num == self.seq_num + len(data):
                                    print("Positive ACK received")
                                    break
                            # negative ack
                            # ya3ni el reponse kanet corrupted
                                else :
                                    print("Negative ACK received. Retrying...")
                                    continue
                                # resend
                        except socket.timeout:
                        # Timeout occurred, retry
                            print("Timeout waiting for ACK packet form Client.
Retrying...")


                elif request.split()[0] == "POST":
                    body = " ".join(request.split()[3:])
                    print(f"body of POST request : {body}")
                    self.seq_num = ack_num
                    self.ack_num = seq_num+len(request)
                    while True:
                        data = "HTTP/1.0 200 OK\r\nSuccessful POST Request"
                        checksum = udp_checksum(self.seq_num,self.ack_num,0,data)
```

```
                        packet =
f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
                        self.server_socket.sendto(packet.encode(),
client_address)
                        # Listen for ACK packet from the client
                        self.server_socket.settimeout(RETRY_TIMEOUT)
                        try:
                            r_data, client_address =
self.server_socket.recvfrom(PACKET_SIZE)
                            r_data = r_data.decode()
                            ack_num = int(r_data.split(":")[1])
                            r_seq_num = int(r_data.split(":")[0])
                            packet = r_data.split(":")[2]
                            received_checksum = int(r_data.split(":")[3])
                            calc_checksum =
udp_checksum(r_seq_num,ack_num,0,packet)
                            if calc_checksum==received_checksum:
                                if ack_num == self.seq_num + len(data):
                                    print("Positive ACK received")
                                    break
                                # negative ack
                                # ya3ni el reponse kanet corrupted
                                else :
                                    print("Negative ACK received. Retrying...")
                                    continue
                                # resend
                        except socket.timeout:
                            # Timeout occurred, retry
                            print("Timeout waiting for ACK packet form Client.
Retrying...")


                else:
                    self.seq_num = ack_num
                    self.ack_num = seq_num+len(request)
                    while True:
                        data = "HTTP/1.0 400 Bad Request\r\n"
                        checksum = udp_checksum(self.seq_num,self.ack_num,0,data)
                        packet =
f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
                        self.server_socket.sendto(packet.encode(),
client_address)
                        # Listen for ACK packet from the client
                        self.server_socket.settimeout(RETRY_TIMEOUT)
                        try:
```

```
                            r_data, client_address =
self.server_socket.recvfrom(PACKET_SIZE)
                            r_data = r_data.decode()
                            ack_num = int(r_data.split(":")[1])
                            r_seq_num = int(r_data.split(":")[0])
                            packet = r_data.split(":")[2]
                            received_checksum = int(r_data.split(":")[3])
                            calc_checksum =
udp_checksum(r_seq_num,ack_num,0,packet)
                            if calc_checksum==received_checksum:
                                if ack_num == self.seq_num + len(data):
                                    print("Positive ACK received")
                                    break
                            # negative ack
                            # ya3ni el reponse kanet corrupted
                                else :
                                    print("Negative ACK received. Retrying...")
                                    continue
                                # resend
                        except socket.timeout:
                            # Timeout occurred, retry
                            print("Timeout waiting for ACK packet form Client.
Retrying...")

            else:
                # if packet sent is corrupted
                print("Packet is Corrupted")
            self.check_received()
            return



    # connection terminate
    def connection_termination(self,client_address):
        PACKET_SIZE = self.PACKET_SIZE
        print("Received FIN packet from client.")
        RETRY_TIMEOUT = 20
        data = "ACK"
        checksum = udp_checksum(self.seq_num,self.ack_num,0,data)
        packet = f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
        # Send ACK packet to the client
        self.server_socket.sendto(packet.encode(), client_address)
        print("Sent ACK packet to client.")
        self.seq_num += len(data)
```

```
        data = "FIN"
        checksum = udp_checksum(self.seq_num,self.ack_num,0,data)
        packet = f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
        self.server_socket.sendto(packet.encode(), client_address)
        print("Sent FIN packet to client.")


        while True :
            # Listen for ACK packet from the server
            self.server_socket.settimeout(RETRY_TIMEOUT)
            try:
                data, server_address = self.server_socket.recvfrom(PACKET_SIZE)
                data = data.decode()
                seq_num = int(data.split(":")[0])
                ack_num = int(data.split(":")[1])
                packet = data.split(":")[2]
                received_checksum = int(data.split(":")[3])
                calc_checksum = udp_checksum(seq_num,ack_num,0,packet)
                # Check for ACK packet
                if packet == "ACK" and (received_checksum == calc_checksum):
                    print("Received ACK packet from client.")
                    exit(0)
                    #return True
                # duplicate FIN
                if packet == "FIN":
                    print("FIN duplicate")
                    continue
            except socket.timeout:
                # Timeout occurred, retry
                print("Timeout waiting for ACK packet form Client. Retrying...")

        return False

def udp_checksum(seq_num,ack_num,recv_window,data):
    data = data.encode("utf-8")
    # Pad data if the length is odd
    if len(data) % 2 == 1:
        data += b'\0'

    # add seq number and ack number and recv window and flag
    # Calculate the checksum using the same algorithm as used in the IP header
    sum = 0

    seq1 = (seq_num >> 16) & 0xFFFF # msb 16 bits of seq num
    seq2 = seq_num & 0xFFFF # lsb 16 bits of seq num
```

```
    ack1 = (ack_num >> 16) & 0xFFFF
    ack2 = ack_num & 0xFFFF

    sum += seq1
    sum += seq2
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    sum += ack1
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    sum += ack2
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    sum += recv_window
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    for i in range(0, len(data), 2):
        word = (data[i] << 8) + (data[i+1])
        sum += word

        if sum >> 16:
            sum = (sum & 0xFFFF) + 1
    sum = ~sum & 0xFFFF
    return sum

def main():
    serv = server()
if __name__ == "__main__":
    main()
```

b. Client:

```
import socket
import time
import random
random.seed(1) # un-comment this line to force packet corruption
# choose request type in lines 17-18
# implement packet loss
```

```
class client:
    def __init__(self,PACKET_SIZE = 100,FORMAT = "utf-8",PORT = 5000):
        self.seq_num = 90
        self.ack_num = 0
        self.PACKET_SIZE = PACKET_SIZE
        self.FORMAT = FORMAT
        self.PORT = PORT
        self.IP = socket.gethostbyname(socket.gethostname())
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        if self.establish_connection():
            #self.send_request("GET","/path/to/resource","1.0") # GET
            #self.send_request("POST","/path/to/resource","1.0","helloooo how are
you") # POST
            self.send_request("P","/path/to/resource","1.0","helloooo how are
you") # BAD REQUEST
            self.connection_termination()

    def establish_connection(self):
        seq_num = self.seq_num
        ack_num = self.ack_num
        MAX_RETRIES = 3
        RETRY_TIMEOUT = 2  # seconds
        retries = 0
        server_address = (self.IP,self.PORT)
        PACKET_SIZE = self.PACKET_SIZE
        while retries < MAX_RETRIES:
            # Send SYN packet to the server
            # f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_window}"
            data = "SYN"
            checksum = udp_checksum(seq_num,ack_num,0,data)
            packet = f"{seq_num}:{ack_num}:{data}:{checksum}:{0}"
            self.client_socket.sendto(packet.encode(),server_address)
            print("Sent SYN packet to the server.")

            # Listen for SYN-ACK packet from the server
            self.client_socket.settimeout(RETRY_TIMEOUT)
            try:
                data, server_address = self.client_socket.recvfrom(PACKET_SIZE)
                data = data.decode()
                r_seq_num = int(data.split(":")[0])
                ack_num = int(data.split(":")[1])
                packet = data.split(":")[2]
                received_checksum = int(data.split(":")[3])
                calculated_checksum = udp_checksum(r_seq_num,ack_num,0,packet)
                # Check for SYN-ACK packet
```

```
                if packet == "SYN-ACK" and received_checksum ==
calculated_checksum and ack_num == seq_num+len("SYN"):
                    # Send ACK packet to the server
                    ack_num = r_seq_num+len("SYN-ACK")
                    seq_num = seq_num+len("SYN")
                    data = "ACK"
                    checksum = udp_checksum(seq_num,ack_num,0,data)
                    packet = f"{seq_num}:{ack_num}:{data}:{checksum}:{0}"
                    self.client_socket.sendto(packet.encode(), server_address)

                    print("Connection established successfully.")
                    self.seq_num = seq_num + len("ACK")
                    self.ack_num = ack_num
                    print(self.seq_num,self.ack_num)
                    # keda el seq number el hanebda2 beeh howa
                    return True
                else :
                    #
                    break
            except socket.timeout:
                # Timeout occurred, retry
                print("Timeout occurred. Retrying...")
                retries += 1
                continue

        print("Connection establishment failed after {}
retries.".format(MAX_RETRIES))
        return False

    def connection_termination(self):
        MAX_RETRIES = 5
        RETRY_TIMEOUT = 5  # seconds

        retries = 0
        server_address = (self.IP,self.PORT)
        PACKET_SIZE = self.PACKET_SIZE
        while retries < MAX_RETRIES:
            # Send FIN packet to the server
            # f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_window}"
            data = "FIN"
            checksum = udp_checksum(self.seq_num,self.ack_num,0,data)
            packet_data = f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
            self.client_socket.sendto(packet_data.encode(), server_address)
            print("Sent FIN packet to terminate connection.")
            # Listen for ACK packet from the server
```

```
            self.client_socket.settimeout(RETRY_TIMEOUT)
            try:
                data_received, server_address =
self.client_socket.recvfrom(PACKET_SIZE)
                data_received = data_received.decode()
                #f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_window}"
                r_seq_num = int(data_received.split(":")[0])
                r_ack_num = int(data_received.split(":")[1])
                ack = data_received.split(":")[2]
                r_checksum = int(data_received.split(":")[3])
                recv_window = int(data_received.split(":")[4])
                calc_checksum = udp_checksum(r_seq_num,r_ack_num,recv_window,ack)
                if r_checksum == calc_checksum:
                    if (r_ack_num == self.seq_num + len(data)) and ack == "ACK":
                        self.seq_num = r_ack_num
                        self.ack_num = r_seq_num + len(ack)
                        print("Received ACK packet from server.")
                        # Listen for ACK packet from the server
                        self.client_socket.settimeout(RETRY_TIMEOUT)
                        try:
                            # Listen for FIN packet from the server
                            data_received, server_address =
self.client_socket.recvfrom(PACKET_SIZE)
                            data_received = data_received.decode()
                            #f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_wind
ow}"
                            r_seq_num = int(data_received.split(":")[0])
                            r_ack_num = int(data_received.split(":")[1])
                            fin = data_received.split(":")[2]
                            r_checksum = int(data_received.split(":")[3])
                            recv_window = int(data_received.split(":")[4])
                            calc_checksum =
udp_checksum(r_seq_num,r_ack_num,recv_window,fin)
                            # Check for FIN packet
                            if r_checksum == calc_checksum:
                                if fin == "FIN":
                                    print("Received FIN packet from server.")
                                    seq_num = r_ack_num
                                    ack_num = r_seq_num+len(fin)
                                    data = "ACK"
                                    checksum =
udp_checksum(seq_num,ack_num,0,data)
                                    #f"{seq_num}:{ack_num}:{packet}:{checksum}:{r
ecv_window}"
```

```
                                    packet =
f"{seq_num}:{ack_num}:{data}:{checksum}:{0}"
                                    print(checksum)
                                    self.client_socket.sendto(packet.encode(),
server_address)
                                    self.seq_num = seq_num
                                    self.ack_num = ack_num
                                    print("Sent ACK packet to terminate
connection.")
                                    print("Connection terminated successfully.")
                                    return True
                            except socket.timeout:
                                # Timeout occurred, retry
                                print("Timeout waiting for FIN packet form Server.
Retrying...")
                                retries += 1
                                continue

                except socket.timeout:

                    # Timeout occurred, retry
                    print("Timeout waiting for ACK packet form Server. Retrying...")
                    retries += 1
                    continue

            print("Connection termination failed after {}
retries.".format(MAX_RETRIES))
            return False



    def send_request(self,method,path,version,body=""):
        PACKET_SIZE = self.PACKET_SIZE
        server_address = (self.IP,self.PORT)

        data = f"{method} {path} HTTP/{version}\r\n {body} \r\n"
        TIMEOUT = 5
        if random.randint(0,1) == 0:
            print("will corrupt data")
            # corrupt data
            checksum = 0
        else:
            checksum = udp_checksum(self.seq_num,self.ack_num,0,data)
        # packing data with its checksum
        packet_data = f"{self.seq_num}:{self.ack_num}:{data}:{checksum}:{0}"
```

```
        self.client_socket.sendto(packet_data.encode(), server_address)

        while True:
            self.client_socket.settimeout(TIMEOUT)

            try:
                data_received, _ = self.client_socket.recvfrom(PACKET_SIZE)
                data_received = data_received.decode()
                # feh moshkilla fel response
                #f"{seq_num}:{ack_num}:{packet}:{checksum}:{recv_window}"
                r_seq_num = int(data_received.split(":")[0])
                r_ack_num = int(data_received.split(":")[1])
                response = data_received.split(":")[2]
                r_checksum = int(data_received.split(":")[3])
                #recv_window = int(data_received.split(":")[4])
                calc_checksum = udp_checksum(r_seq_num,r_ack_num,0,response)
                # no error in checksum
                if r_checksum == calc_checksum:
                    # ack or negative ack
                    if r_ack_num == (self.seq_num + len(data)):
                        print(response)
                        # SEND ACK
                        self.seq_num = self.seq_num + len(data)
                        self.ack_num = r_seq_num + len(response)
                        checksum =
udp_checksum(self.seq_num,self.ack_num,0,'ACK')
                        packet_data =
f"{self.seq_num}:{self.ack_num}:{'ACK'}:{checksum}:{0}"
                        self.client_socket.sendto(packet_data.encode(),
server_address)
                        return
                    else:
                        print("Negative ack received")
                        print("Retrying..")
                        self.send_request(method,path,version,body)
                        return
                else:
                    # drop
                    print("Corrupted message!!")
                    #negative ack
                    checksum = udp_checksum(0,self.ack_num,0,"")
                    packet_data = f"{0}:{self.ack_num}:{''}:{checksum}:{0}"
                    self.client_socket.sendto(packet_data.encode(),
server_address)
                    # hane3mil ehh??
```

```
            # implement paccket loss in the serever hena implement packet
corruption
            # el3ab fel checksum beta3et wa7da minhom
            # packet loss
            except socket.timeout:
                print("Time out waiting for http response")
                print("Retrying..")
                self.send_request(method,path,version,body)
                return


def udp_checksum(seq_num,ack_num,recv_window,data):
    data = data.encode("utf-8")
    # Pad data if the length is odd
    if len(data) % 2 == 1:
        data += b'\0'

    # add seq number and ack number and recv window and flag
    # Calculate the checksum using the same algorithm as used in the IP header
    sum = 0

    seq1 = (seq_num >> 16) & 0xFFFF # msb 16 bits of seq num
    seq2 = seq_num & 0xFFFF # lsb 16 bits of seq num
    ack1 = (ack_num >> 16) & 0xFFFF
    ack2 = ack_num & 0xFFFF

    sum += seq1
    sum += seq2
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    sum += ack1
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    sum += ack2
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    sum += recv_window
    if sum >> 16:
        sum = (sum & 0xFFFF) + 1
    for i in range(0, len(data), 2):
        word = (data[i] << 8) + (data[i+1])
        sum += word

        if sum >> 16:
            sum = (sum & 0xFFFF) + 1
```

```
    sum = ~sum & 0xFFFF
    return sum


def main():
    cl = client()
if __name__ == "__main__":
    main()
```

III. Test Cases:

```
if self.establish_connection():
    self.send_request("GET","/path/to/resource","1.0") # GET
    #self.send_request("POST","/path/to/resource","1.0","helloooo how are you") # POST
    #self.send_request("P","/path/to/resource","1.0","helloooo how are you") # BAD REQUEST
    self.connection_termination()
```

a. `self.send_request("GET","/path/to/resource","1.0") # GET`

```
C:\Users\merva>cd C:\Users\merva\OneDrive\Desktop\uni\networks\UDP-Implementation

C:\Users\merva\OneDrive\Desktop\uni\networks\UDP-Implementation>python client.py
Sent SYN packet to the server.
Connection established successfully.
96 17
will corrupt data
Time out waiting for http response
Retrying..
will corrupt data
Time out waiting for http response
Retrying..
HTTP/1.0 200 OK
Successful GET Request
Sent FIN packet to terminate connection.
Received ACK packet from server.
Received FIN packet from server.
29431
Sent ACK packet to terminate connection.
Connection terminated successfully.
```

```
PS C:\Users\merva> & C:/Users/merva/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/merva/OneDriv
py
UDP server is running on 192.168.1.8:5000
Connection established successfully.
17 96
Packet is Corrupted
Packet is Corrupted
Received GET request
Positive ACK received
Received FIN packet from client.
Sent ACK packet to client.
Sent FIN packet to client.
Received ACK packet from client.
PS C:\Users\merva>
```

b. `self.send_request("POST","/path/to/resource","1.0","helloooo how are you") # POST`

```
C:\Users\merva\OneDrive\Desktop\uni\networks\UDP-Implementation>python client.py
Sent SYN packet to the server.
Connection established successfully.
96 17
will corrupt data
Time out waiting for http response
Retrying..
will corrupt data
Time out waiting for http response
Retrying..
HTTP/1.0 200 OK
Successful POST Request
Sent FIN packet to terminate connection.
Received ACK packet from server.
Received FIN packet from server.
29409
Sent ACK packet to terminate connection.
Connection terminated successfully.

C:\Users\merva\OneDrive\Desktop\uni\networks\UDP-Implementation>
```

```
PS C:\Users\merva> & C:/Users/merva/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/merva/OneDrive/Deskt
py
UDP server is running on 192.168.1.8:5000
Connection established successfully.
17 96
Packet is Corrupted
Packet is Corrupted
body of POST request : helloooo how are you
Positive ACK received
Received FIN packet from client.
Sent ACK packet to client.
Sent FIN packet to client.
Received ACK packet from client.
PS C:\Users\merva>
```

c. `self.send_request("P","/path/to/resource","1.0","helloooo how are you") # BAD REQUEST`

```
C:\Users\merva\OneDrive\Desktop\uni\networks\UDP-Implementation>python client.py
Sent SYN packet to the server.
Connection established successfully.
96 17
will corrupt data
Time out waiting for http response
Retrying..
will corrupt data
Time out waiting for http response
Retrying..
HTTP/1.0 400 Bad Request

Sent FIN packet to terminate connection.
Received ACK packet from server.
Received FIN packet from server.
29426
Sent ACK packet to terminate connection.
Connection terminated successfully.
```

```
PS C:\Users\merva> & C:/Users/merva/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/merva/OneDrive/De
py
UDP server is running on 192.168.1.8:5000
Connection established successfully.
17 96
Packet is Corrupted
Packet is Corrupted
Positive ACK received
Received FIN packet from client.
Sent ACK packet to client.
Sent FIN packet to client.
Received ACK packet from client.
PS C:\Users\merva>
```

IV. Conclusion:

The implemented system successfully simulates TCP packets over UDP while ensuring reliability and supporting HTTP/1.0 protocol. By addressing various challenges and requirements, it provides a robust framework for network communication with considerations for both performance and reliability.