COMPUTER NETWORKS

Final Project

Seif Eldin Mahmoud,6773
Zeyad Ashraf,6758
Nour Bassem,6971

Introduction:

This report describes the design and implementation of a system that simulates TCP packets using a UDP connection while maintaining reliability and supporting the HTTP protocol on top of UDP. The task involves extending the user space of an application to include mechanisms for error detection and correction, packet retransmission, and flow control. The system should be able to handle packet loss, duplication, and reordering while maintaining a reasonable level of performance.

The report discusses the trade-offs between performance and reliability when designing the solution to achieve these goals. The system implements error detection and correction using checksums, packet retransmission using acknowledgments and timeouts, and flow control using stop and wait control mechanisms. The report also describes how the system parses HTTP requests and responses and handles the various methods and headers defined by the protocol.

Discussion:

1st Server.py file

The file is separated into two classes TCP Server and HTTP Request as each class is handling certain orders as we will consider in the following:

- **TCP Server**

This is a Python code that sets up a TCP server that can handle HTTP requests. The server is initialized with a default IP address and port number, which can be changed if needed. The class constructor creates a socket object that uses the AF_INET protocol and SOCK_STREAM for TCP communication. The IP address and port number are bound to the socket object using the bind method. The sequence number is also initialized to 0.

The method handle_client is used to handle incoming client requests. It is designed to run in an infinite loop until the connection is closed. It reads incoming data from the client, checks the validity of the received data using a checksum, and sends an acknowledgment message back to the client. If the received message is a GET or POST HTTP request, it creates an HTTP Request object and calls the appropriate handler method. The handler methods parse the HTTP request and prepare a response that is sent back to the client. If the request contains a 'keep-alive' header, the connection timeout is set according to the header value. If the request contains a 'connection: close' header, the connection is closed.

The method starts listening for incoming connections from clients. When a connection is established, it receives a SYN message from the client. It sends a SYN-ACK message back to the client and waits for an ACK message. If the client sends an ACK message, the connection is established, and a new thread is created to handle the client's request.

If any of the steps in the connection establishment process fails, the connection is terminated, and the socket is closed. In case of an unexpected socket timeout, the connection is also terminated, and the socket is closed.

Overall, this code implements a reliable TCP server that can handle HTTP requests using checksums for error detection and correction, packet retransmission, and flow control mechanisms. It also supports persistent connections by handling the keep-alive and connection headers in HTTP requests. The code follows best practices for network programming, such as running the client request handling code in a separate thread, and properly closing sockets when the connection is terminated.

- **HttpRequest:**

The class has an __init__ method that initializes the following instance variables: method, URI, version, headers, body, and time. The headers parameter has a default value of None, which gets set to an empty dictionary if it is not provided. The parse_request method takes in a request_string argument and uses it to set the method, URI, version, headers, and body instance variables.

The class also has two main methods: GetHandler and POSTHandler. GetHandler handles GET requests and checks if the file exists in the server directory. If the file exists, it reads the file, gets the modified time, and returns a response with status code 200, the data, modified time, keep-alive, connection, and content length. If the file does not exist, it returns a response with the status code 404.

The POSTHandler handles POST requests and appends the body instance variable to the specified file. If the file exists, it returns a response with status code 200 and the modified time, keep-alive, connection, and content length. If the file does not exist, it returns a response with the status code 404.

2nd Client.py

It is a file consisting of one class "TCP Client" and the main method for sending and receiving data.

- TCP Client

It has an initialization method, __init__, which takes the IP address and port number as input arguments. The default IP address is set to localhost and the default port number is set to 5005. It also initializes a socket object using AF_INET and SOCK_STREAM flags for the Internet Protocol (IP) address family and stream-oriented communication, respectively. Finally, it sets the initial sequence number to zero.

The connect method is used to establish a connection with the server. It first sends an SYN message to the server and waits for an SYN-ACK message in response. If the received message is SYNACK, it sends an ACK message to the server, indicating that the connection is established. If the response is not SYNACK, it indicates that the connection could not be established.

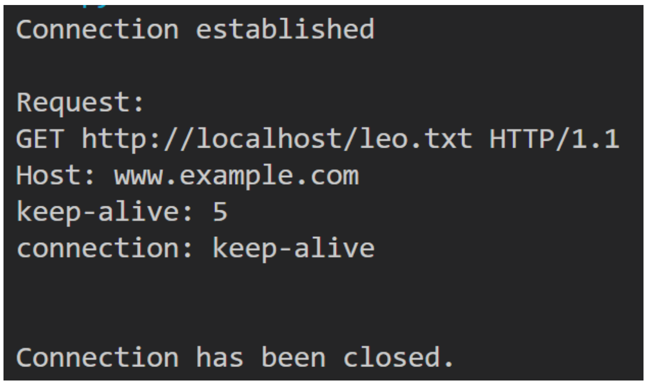
The send method is used to send a message to the server. Before sending the message, it calculates a checksum for the message by summing up the ASCII codes of all characters in the message along with the sequence number modulo 256. The checksum is appended to the message, and the message is sent to the server. It then waits for an ACK message from the server. If the ACK message is received and the checksum is correct, it toggles the sequence number and receives the data sent by the server. If the data is not empty, it prints the data. If the received message is not an ACK message or the checksum is incorrect, it resends the message.

The close method sends a FIN message to the server indicating that the client has finished sending data and is ready to close the connection. It then prints a message indicating that the connection has been closed and closes the socket object.

Test Cases:

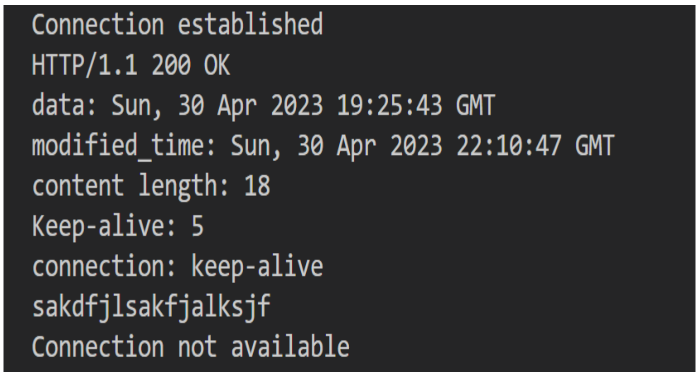
- Case 1: "In case of a timeout"

Request:



```
Connection established

Request:
GET http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive


Connection has been closed.
```

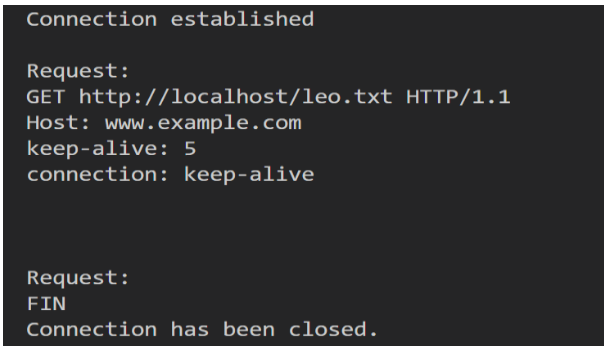
Response:



```
Connection established
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:25:43 GMT
modified_time: Sun, 30 Apr 2023 22:10:47 GMT
content length: 18
Keep-alive: 5
connection: keep-alive
sakdfjlsakfjalksjf
Connection not available
```

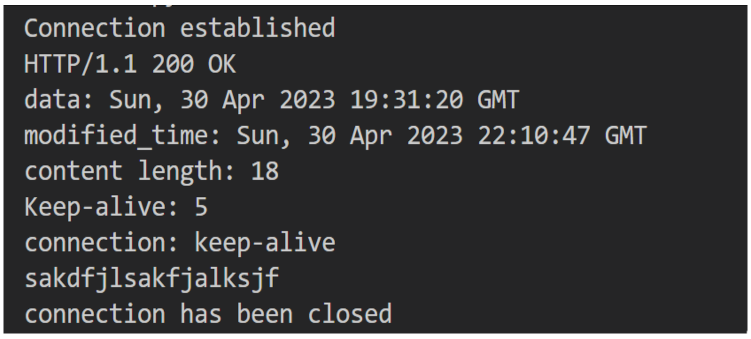
- Case 2: “in case of closing the connection by setting the flag FIN (Get Request)”

Request:



```
Connection established

Request:
GET http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive



Request:
FIN
Connection has been closed.
```

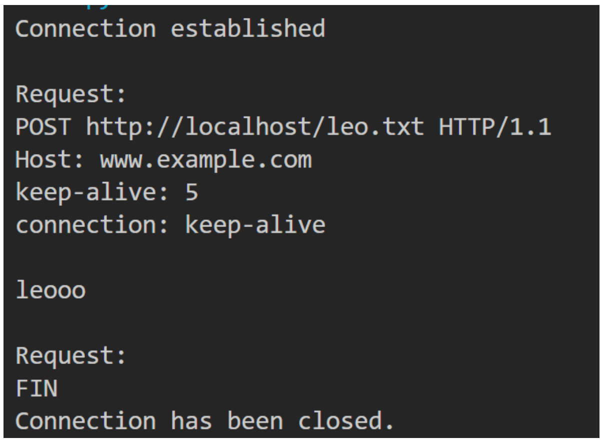
Response:



```
Connection established
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:31:20 GMT
modified_time: Sun, 30 Apr 2023 22:10:47 GMT
content length: 18
Keep-alive: 5
connection: keep-alive
sakdfjlsakfjalksjf
connection has been closed
```

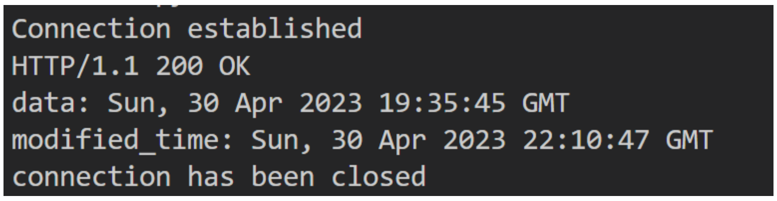
- Case 3: “In case of POST request”

Request:



```
Connection established

Request:
POST http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive

leooo

Request:
FIN
Connection has been closed.
```

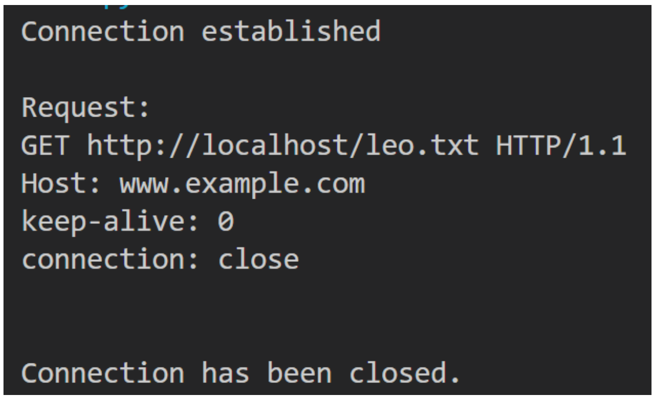
Response:



```
Connection established
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:35:45 GMT
modified_time: Sun, 30 Apr 2023 22:10:47 GMT
connection has been closed
```

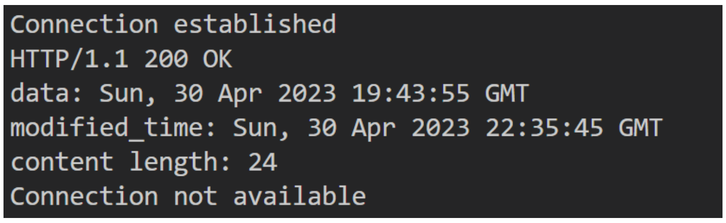
- Case 4: “in case of connection header = close”

Request:



```
Connection established

Request:
GET http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 0
connection: close


Connection has been closed.
```

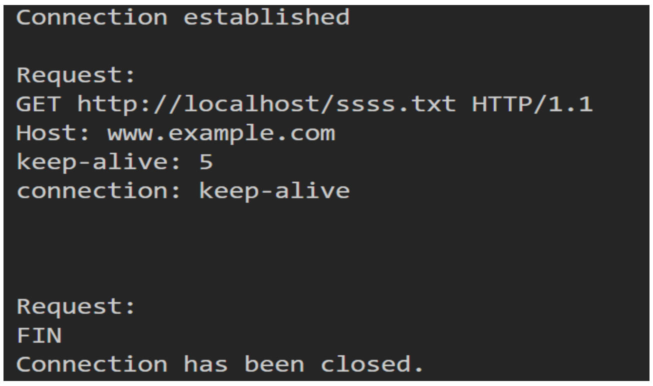
Response:



```
Connection established
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:43:55 GMT
modified_time: Sun, 30 Apr 2023 22:35:45 GMT
content length: 24
Connection not available
```

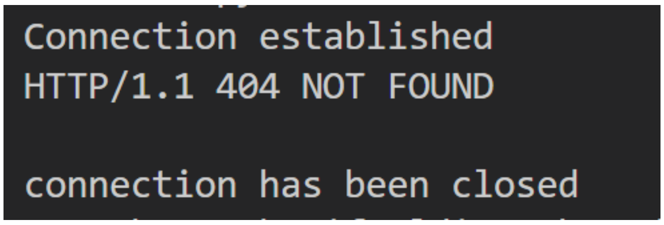
- Case 5: “File doesn't exist”

Request:



```
Connection established

Request:
GET http://localhost/ssss.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive



Request:
FIN
Connection has been closed.
```

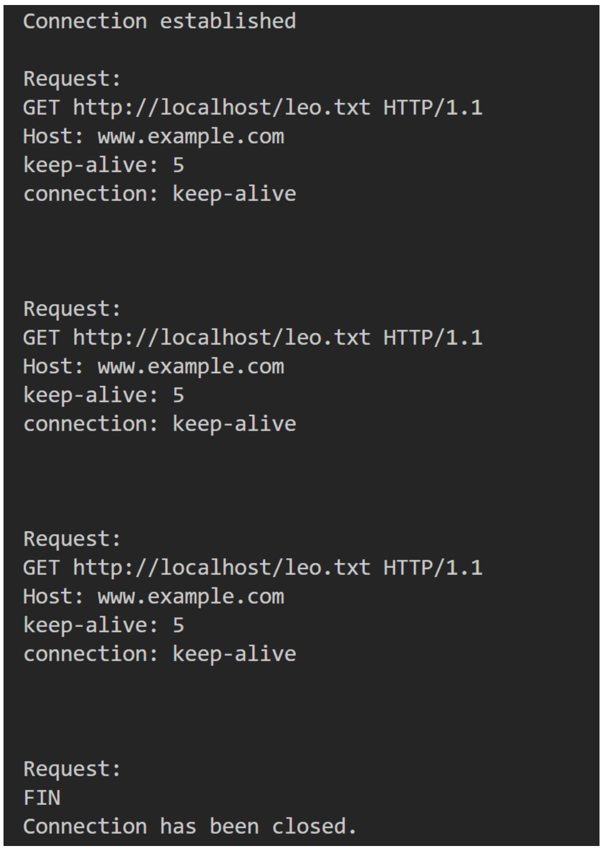
Response:



```
Connection established
HTTP/1.1 404 NOT FOUND

connection has been closed
```

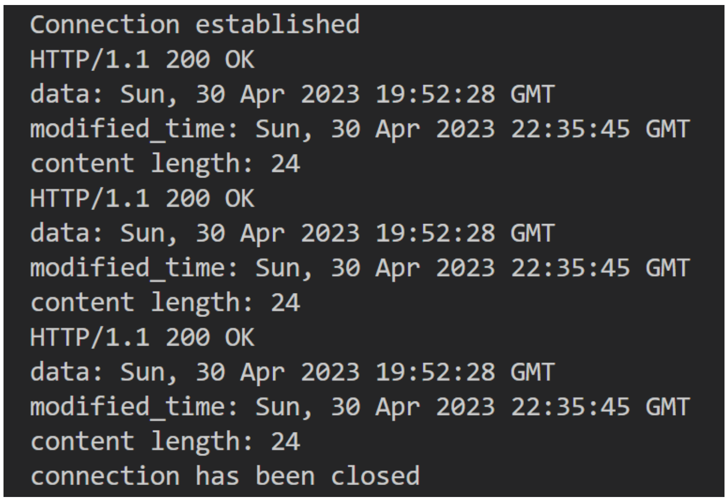
- Case 6: “Sending multiple requests”

Request:



```
Connection established

Request:
GET http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive



Request:
GET http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive



Request:
GET http://localhost/leo.txt HTTP/1.1
Host: www.example.com
keep-alive: 5
connection: keep-alive



Request:
FIN
Connection has been closed.
```

Response:



```
Connection established
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:52:28 GMT
modified_time: Sun, 30 Apr 2023 22:35:45 GMT
content length: 24
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:52:28 GMT
modified_time: Sun, 30 Apr 2023 22:35:45 GMT
content length: 24
HTTP/1.1 200 OK
data: Sun, 30 Apr 2023 19:52:28 GMT
modified_time: Sun, 30 Apr 2023 22:35:45 GMT
content length: 24
connection has been closed
```