Gebze Technical University
Computer Engineering

CSE 476 – 2021 Spring
TERM PROJECT
REPORT

Dilara KARAKAŞ

171044010

WEB SERVER

I. Problem Definition

In this part, I will develop a simple Web Server in Python that can only process one request. My web server will do the following:

- When the server receives a request from a client, it will create a socket.
- This will receive the HTTP request using the opened socket.
- To find the requested file, it must meet the incoming request.
- It should open the desired file from the server's location. If it can't find the file, it should throw a "404 not found" error.
- It should generate an HTTP response containing this file.
- It should send the generated HTTP response to the client over the TCP connection.

II. Source Code

I did all the operations using skeleton code. I completed the code, run my server, and then tested my server by sending requests from browsers running on different hosts.

```
from socket import *

serverSocket = socket(AF_INET, SOCK_STREAM)

SERVER_PORT = 8080
BUFFER_SIZE = 1024
# Prepare a server socket
# Fill in start
serverSocket.bind(("", SERVER_PORT))
serverSocket.listen(1)

print(gethostname(), "  ", gethostbyname(gethostname()))
print("Web server is started. Port number : ", SERVER_PORT)
```

• I am import socket module. I am preparing a server socket with localhost or different hosts and port number 8080. This socket is prepared as a listening socket. (Figure 1)

Figure 1

In the while loop, the server will now start waiting for the connection using the accept() function. The accept() function returns an explicit link between the following. Server and client along with the client's address. The connection is actually a different socket on another port. Data, it is read from the connection with recv() . The output data keeps the contents of the HelloWorld.html file in the same directory. The content of this output data, is HellloWorld.html, will be sent to the connected client. A 200-coded HTTP

```
while True:
    # Establish the connection
    print('Ready to serve...')
    connectionSocket, addr = serverSocket.accept()
    try:
        message = connectionSocket.recv(BUFFER_SIZE)
        filename = message.split()[1]
        f = open(filename[1:])
        outputdata = f.read()
        f.close()
        # Send one HTTP header line into socket
        # Fill in start
        connectionSocket.send(bytes('\HTTP/1.0 200 OK\r\n\r\n'.encode()))
        # Fill in end

        # Send the content of the requested file to the client
        for i in range(0, len(outputdata)):
            connectionSocket.send(bytes(outputdata[i].encode()))
        connectionSocket.close()
    except IOError:
        # Send response message for file not found
        # Fill in start
        connectionSocket.send(bytes('\HTTP/1.0 200 404 NOT FOUND\r\n\r\n'.encode()))
        connectionSocket.send(bytes("<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n".encode()))
        # Fill in end

        # Close client socket
        # Fill in start
        connectionSocket.close()
        # Fill in end

serverSocket.close()
```

Figure 2

message is then sent, indicating that the client has successfully connected. When this is done, the file content is returned and the web page is displayed on the client side. (Figure 2)

III. Outputs

Same Machine

Figure 3

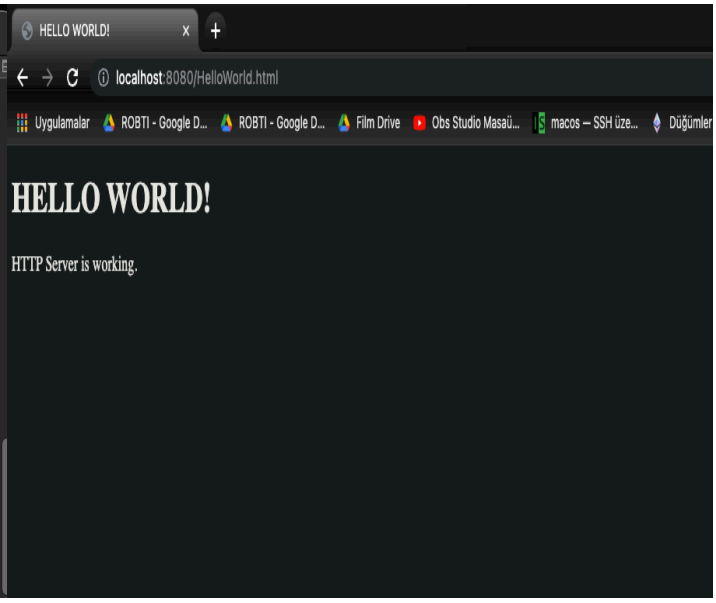

Figure 4

Different Machine

If we try to connect with localhost we will fail.

If we give the IP address of the computer where the server is running, it will be successful.

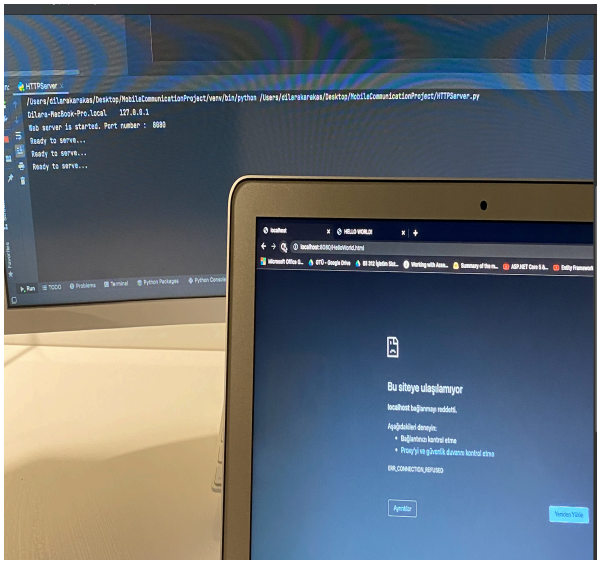

Figure 5

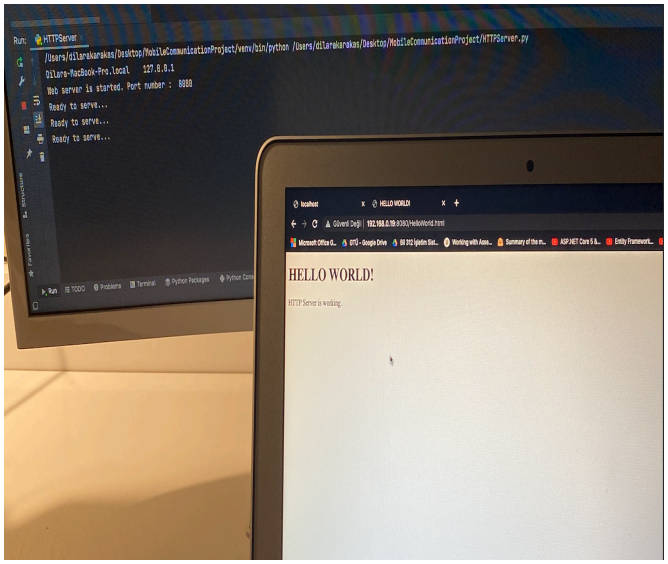

Figure 6

UDP PINGER LAB

I. *Problem Definition*

A client will be written and this client will send 10 consecutive ping messages to a server over UDP. It will keep and print the time from each ping message to the reply pong message. Response pongs exceeding 1 second will print a "request timed out" message, depending on delays or packet losses caused by the nature of UDP.

II. Source Code

No changes have been made to the server code. (Figure 7)

The UDP client is similar the UDPPingerServer, but does not use bind() to attach its socket to an address. It uses sendto() to deliver its message directly to the server, and recvfrom() to receive the response.
The SOCK_DGRAM tag is used as the data will be retrieved from the UDP server. Unlike TCP, there is no bind() operation in UDP client code. If there is no message within one second, it will time out. That's why the settimeout(1) function is used. A socket address is then created. (Figure 8)

```python
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets import random
import random
from socket import *
# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Generate random number in the range of 0 to 10
    rand = random.randint(0, 10)
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)
    # Capitalize the message from the client
    message = message.upper()
    # If rand is less is than 4, we consider the packet lost and do not respond
    if rand < 4:
        continue
    # Otherwise, the server responds
    serverSocket.sendto(message, address)
```

Figure 7

```python
UDP_Port_Number = 12000
UDP_IP_Address = "127.0.0.1"

# Create a UDP socket
clientSocket = socket(AF_INET, SOCK_DGRAM)
# Set a timeout value of 1 second for the created socket
clientSocket.settimeout(1)
ping_number = 1
```

Figure 8

```
while ping_number <= 10:
    start_time = time.time()
    try:
        message = 'Ping ' + str(ping_number) + " " + str(time.strftime("%H:%M:%S"))
        clientSocket.sendto(bytes(message.encode()), (UDP_IP_Address, UDP_Port_Number))
        print("Send to message : " + message)
        data, server = clientSocket.recvfrom(1024)
        finish_time = time.time()
        rtt = finish_time - start_time
        print("Respond : ", data.decode(), " Round Trip Time(RTT): ", rtt)
    except timeout:
        print('REQUEST TIMED OUT')
    ping_number += 1
    if ping_number > 10:
        clientSocket.close()
```

Figure 9

Message format is as shown in the assignment pdf. It is in the form of "Ping sequence_number time". If return is received in less than 1 second, data acquisition is performed. If the process takes longer than 1 second, the desired message is printed on the screen. If the message is received correctly, the round trip time value is printed as requested.
When the ping_number is 10, that is, when 10 pings are made, the socket is closed. (Figure 9)

III. Outputs

Same Machine

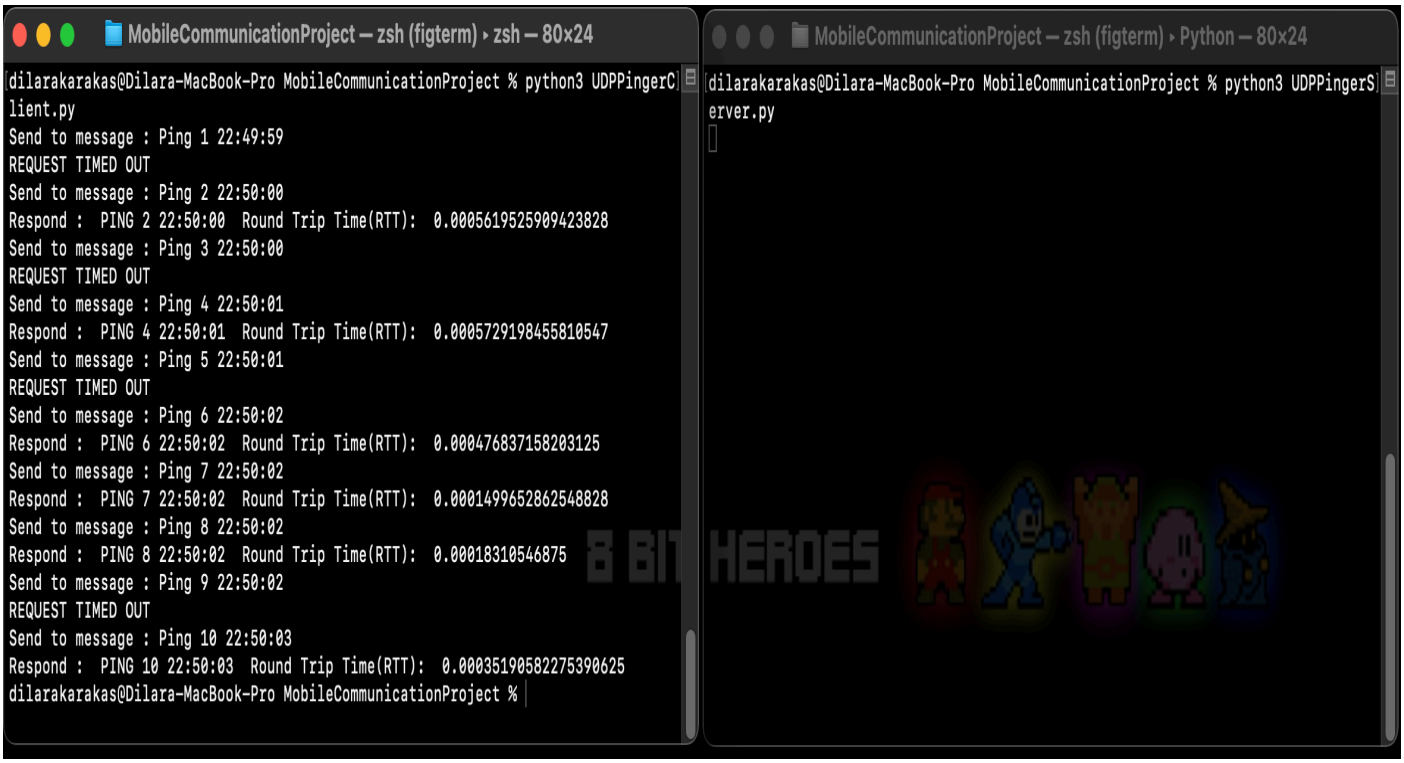

Figure 10

Different Machine

With localhost

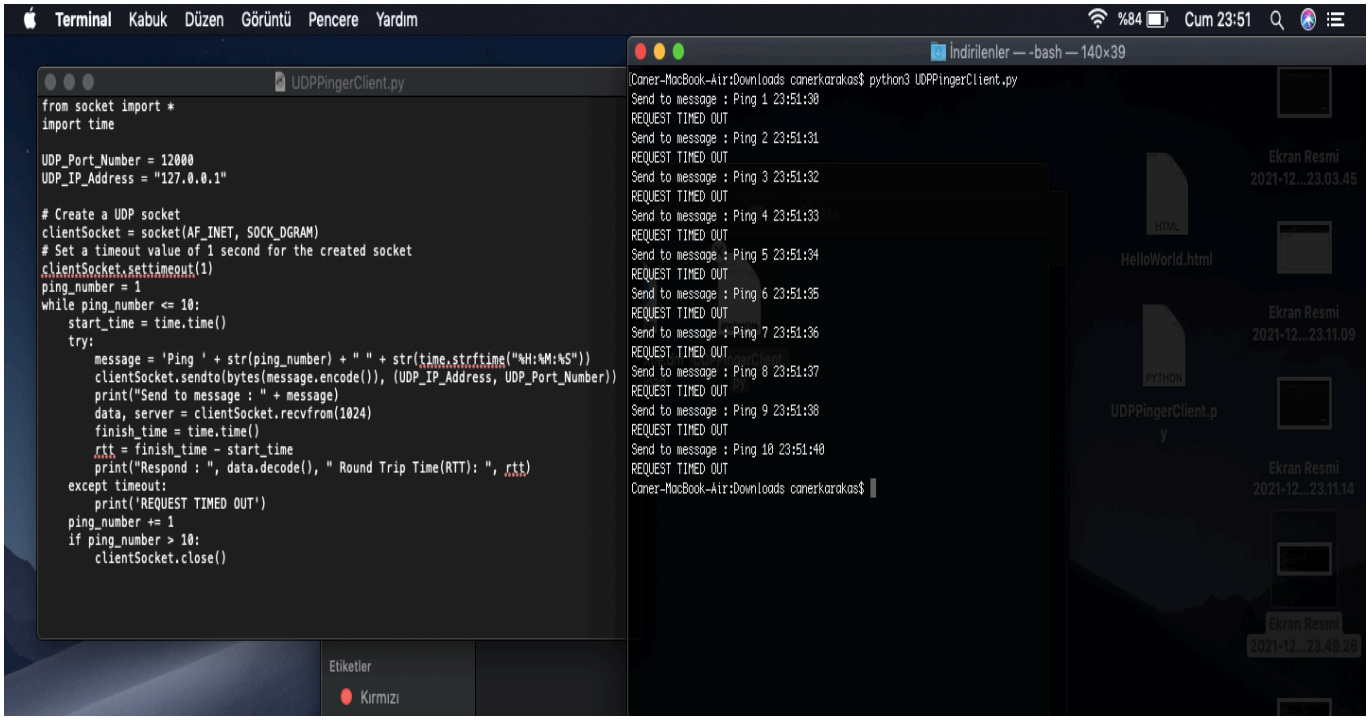

Figure 11

With the ip address of the computer where the server code is running

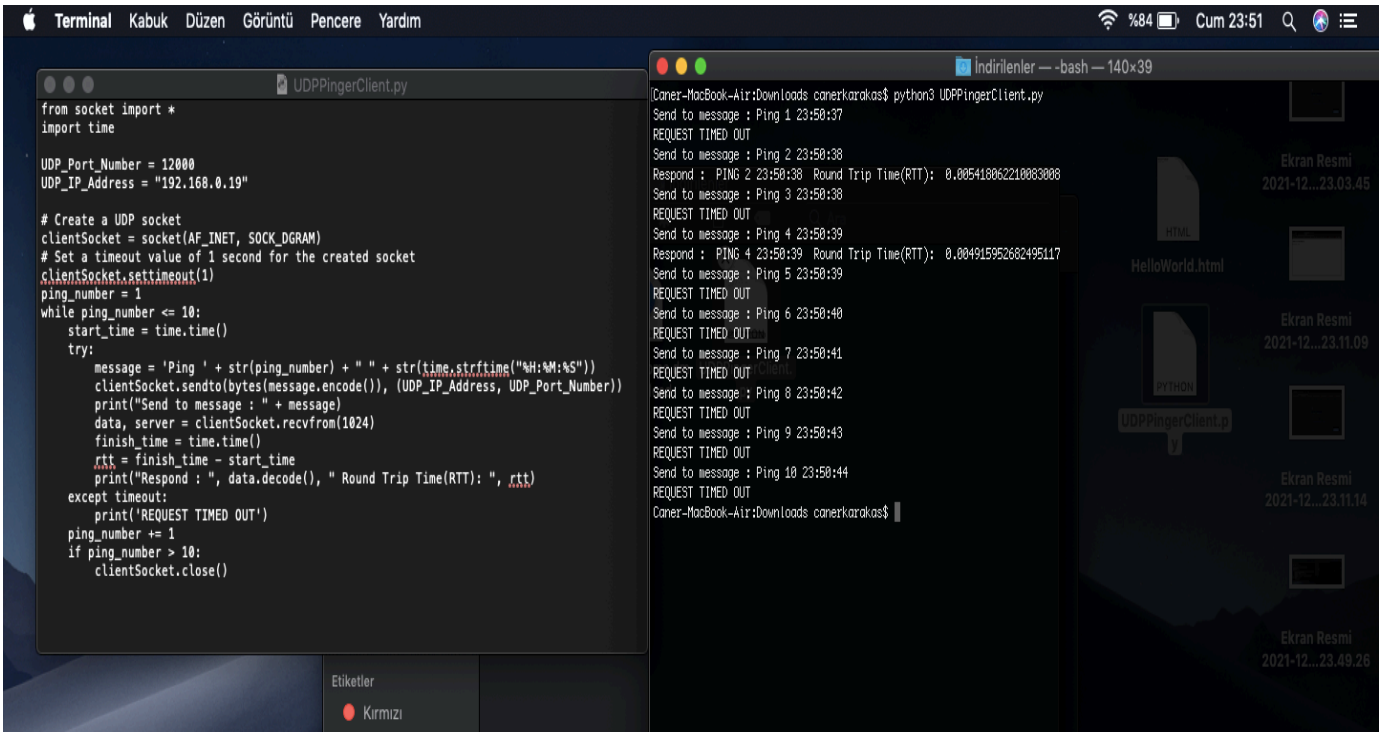

Figure 12

SMTP CLIENT

I. *Problem Definition*

We will create a simple mail client that sends email to any recipient. Our client will need to establish a TCP connection with a mail server (e.g., a Google mail server), dialogue with the mail server using the SMTP protocol, send an email message to a recipient (e.g., our friend) via the mail server, and finally close the TCP connection with the mail server.

II. Source Code

I create mail client with using SMTP protocol and also us- ing Google Mail Server. For being able to use google mail server,I add a Transport Layer Security (TLS) and Secure Sockets Layer (SSL) for authentication and security reasons, before send MAIL FROM command. I choose mail server that is Google Mail Server and I call it with port 587. I create client socket and establish a TCP connection with mail server. I send HELO Command and print server response. Then, I open TLS for Google mail server and print server response. After that, I create SSL for authentication and security reasons. I send MAIL FROM command with using sender mail address and print server response. I send RCPT TO command with using receiver mail address and print server response. I send DATA command and print server response. I send message data. After that, message ends with single period. Then, I send QUIT command and get server response. Last, I close socket.

```
import base64
import ssl
from socket import *

msg = "\r\n I love computer networks!"
endmsg = "\r\n.\r\n"

MailFrom = "mobile.network.proje@gmail.com"
RcptMail = "karakasdilara1@gmail.com"
password = "dilaramobile123"

# Choose a mail server (e.g. Google mail server) and call it mailserver
# Fill in start
mailserver = ("smtp.gmail.com", 587)
# Fill in end

# Create socket called clientSocket and establish a TCP connection with mailserver
# Fill in start
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect(mailserver)
# Fill in end

recv = clientSocket.recv(1024)
print(recv)
if recv[:3] != '220':
    print('220 reply not received from server.')

# Send HELO command then print server response.
heloCommand = 'HELO Alice\r\n'
clientSocket.send(heloCommand.encode())
recv1 = clientSocket.recv(1024)
print(recv1)
if recv1[:3] != '250':
    print('250 reply not received from server.')
```

Figure 13

```
# Start TLS
tls = 'starttls\r\n'
clientSocket.send(tls.encode())
recvTls = clientSocket.recv(1024)
print(recvTls)
if recvTls[:3] != '220':
    print('220 reply not received from server.')

# Secure Sockets Layer (SSL)
WrapClientSocket = ssl.wrap_socket(clientSocket, ssl_version=ssl.PROTOCOL_SSLv23)
# Send Auth login
WrapClientSocket.send(('AUTH LOGIN\r\n').encode())
recvAuthLogin = WrapClientSocket.recv(1024)
print("Recv Auth Login: ", recvAuthLogin.decode())
if recvAuthLogin[:3] != '334':
    print('334 reply not received from server.')

# Send mail
WrapClientSocket.send((base64.b64encode(MailFrom.encode()))+('\r\n').encode())
recvMail = WrapClientSocket.recv(1024)
print("Recv Send Mail: ", recvMail.decode())
if recvMail[:3] != '334':
    print('334 reply not received from server.')

# Send password
WrapClientSocket.send((base64.b64encode(password.encode()))+('\r\n').encode())
recvPassword = WrapClientSocket.recv(1024)
print("Recv Send Password: ", recvPassword.decode())
if recvPassword[:3] != '235':
    print('235 reply not received from server.')
```

Figure 14

```
# Send MAIL FROM command and print server response.
# Fill in start
mailFrom = "MAIL FROM: <"+MailFrom+"> \r\n"
WrapClientSocket.send(mailFrom.encode())
recvMailFrom = WrapClientSocket.recv(1024)
print("Recv Mail From: ", recvMailFrom)
if recvMailFrom[:3] != '250':
    print('250 reply not received from server.')
# Fill in end

# Send RCPT TO command and print server response.
# Fill in start
rcptMail = "RCPT TO: <"+RcptMail+"> \r\n"
WrapClientSocket.send(rcptMail.encode())
recvRCPTMail = WrapClientSocket.recv(1024)
print("Recv RCPT Mail: ", recvRCPTMail)
if recvRCPTMail[:3] != '250':
    print('250 reply not received from server.')
# Fill in end

# Send DATA command and print server response.
# Fill in start
data = "DATA\r\n"
WrapClientSocket.send(data.encode())
recvData = WrapClientSocket.recv(1024)
print("Recv Data: ", recvData)
if recvData[:3] != '354':
    print('354 reply not received from server.')
# Fill in end
```

Figure 15

```
# Send message data.
# Fill in start
message = "SUBJECT: SMTP Mail Client Test \r\n\r\n" + msg
WrapClientSocket.send(("From: "+MailFrom + '\r\n').encode())
WrapClientSocket.send(("To: "+RcptMail + '\r\n').encode())
WrapClientSocket.send(message.encode())
WrapClientSocket.send(endmsg.encode())
recvSendMessage = WrapClientSocket.recv(1024)
print("Recv Send Message: ", recvSendMessage)
if recvSendMessage[:3] != '250':
    print('250 reply not received from server.')
# Fill in end

# Send QUIT command and get server response.
# Fill in start
WrapClientSocket.send("QUIT\r\n".encode())
recvQuit = WrapClientSocket.recv(1024)
print("Recv Quit: ", recvQuit)
if recvQuit[:3] != '221':
    print('221 reply not received from server.')
WrapClientSocket.close()
# Fill in end
```

Figure 16

III. Outputs

```
dilarakarakas@Dilara-MacBook-Pro MobileCommunicationProject % python MailClient.py
220 smtp.gmail.com ESMTP a204sm16513161wmd.39 - gsmtp

250 smtp.gmail.com at your service

220 2.0.0 Ready to start TLS

('Recv Auth Login: ', u'334 VXNlcm5hbWU6\r\n')
('Recv Send Mail: ', u'334 UGFzc3dvcmQ6\r\n')
('Recv Send Password: ', u'235 2.7.0 Accepted\r\n')
('Recv Mail From: ', '250 2.1.0 OK a204sm16513161wmd.39 - gsmtp\r\n')
('Recv RCPT Mail: ', '250 2.1.5 OK a204sm16513161wmd.39 - gsmtp\r\n')
('Recv Data: ', '354  Go ahead a204sm16513161wmd.39 - gsmtp\r\n')
('Recv Send Message: ', '250 2.0.0 OK  1640535234 a204sm16513161wmd.39 - gsmtp\r\n')
('Recv Quit: ', '221 2.0.0 closing connection a204sm16513161wmd.39 - gsmtp\r\n')
dilarakarakas@Dilara-MacBook-Pro MobileCommunicationProject %
```

Figure 17

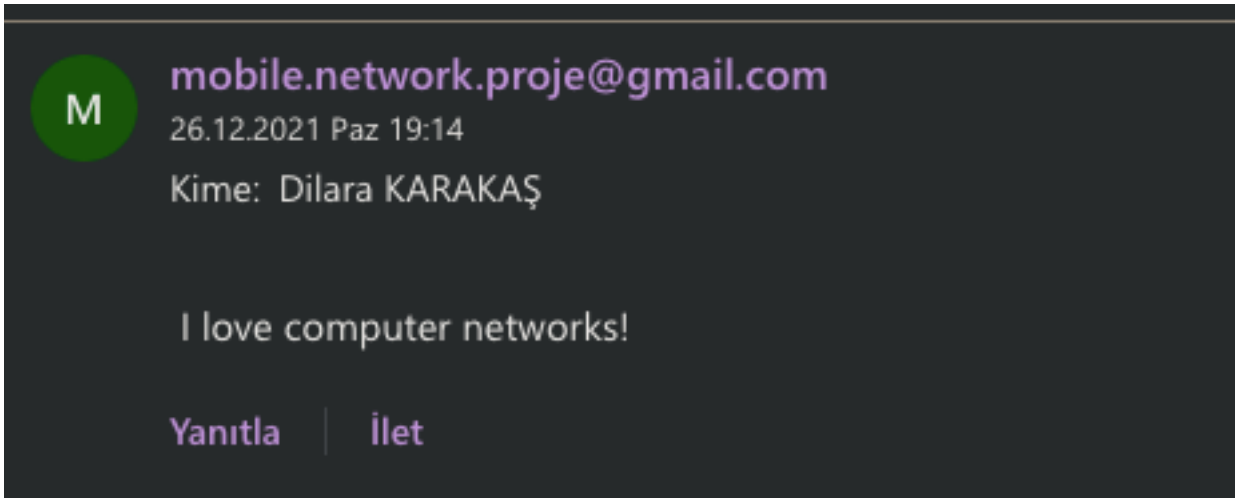

Figure 18

Figure 19

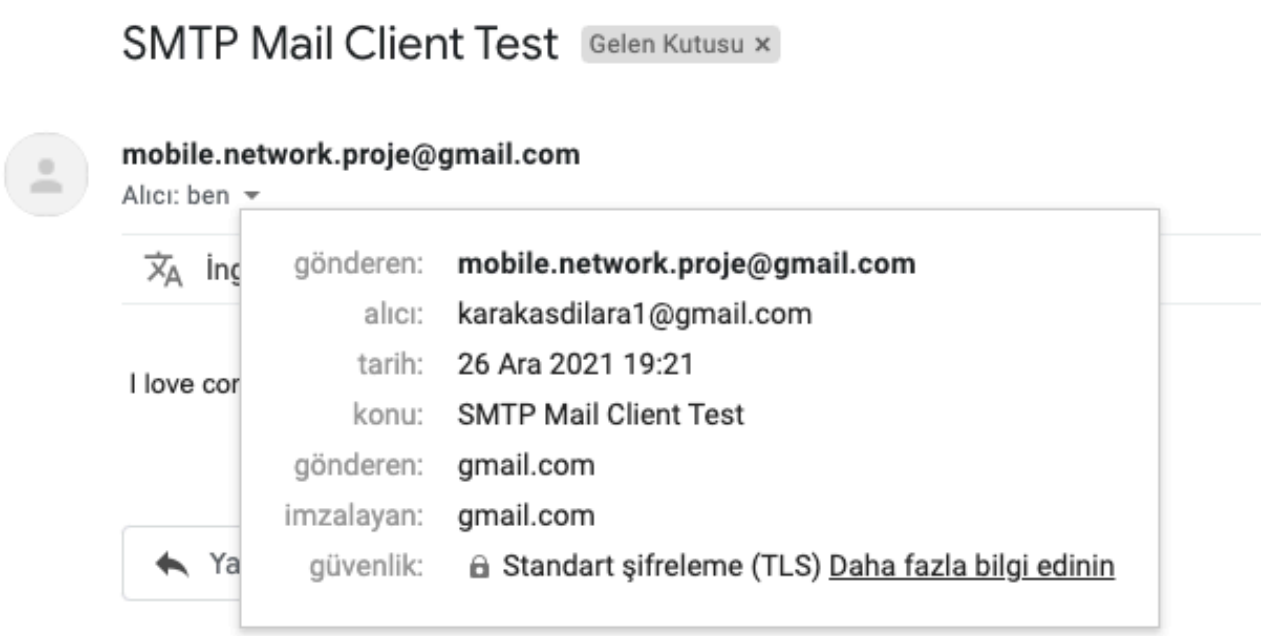

Figure 20

NOTE!!

WEB SERVER and UDP PINGER LAB should be compiled with python3 and SMTP CLIENT should be compiled with python2. Lab2 optinal 1 and lab3 optinal 1 were implemented in the project.