

Gebze Technical University

Term Project Report

CSE476 | Mobile Communication Networks

Abdurrahman Bulut

1901042258

Assignment 1: Web Server

In this assignment, a simple Web Server is developed in Python that is capable of processing only one request. This Web Server creates a connection socket when contacted by a client (browser).

It receives the HTTP request from this connection.

It parses the request to determine the specific file being requested.

It gets the requested file from the server's file system.

It creates an HTTP response message consisting of the requested file preceded by header lines.

It sends the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in the server, It returns a "404 Not Found" error message.

Code Explanation

Server TCP port is selected as 6789 and it creates a socket. It binds to TCP port with using bind() function. After that, socket goes into listening mode with using listen() function.

```
# Prepare a server socket
port = 6789

# Fill in start
serverSocket.bind('', port)

# put the socket into listening mode
serverSocket.listen(1)
# Fill in end
```

It confirm to be ready to serve. With accept() function, the connection is established.

```
# Establish the connection
print("Ready to serve...")
# Fill in start
connectionSocket, addr = serverSocket.accept()
# Fill in end
```

It gets file name and opens that file with open() function. After that, it reads file with using f.read() function. And it close file with using f.close().

```
# Fill in start
message = connectionSocket.recv(1024)
# Fill in end

filename = message.split()[1]
f = open(filename[1:])

# Fill in start
outputdata = f.read()
f.close()
# Fill in end
```

I send one HTTP header line into socket with using send() function. Information is encrypted with encode() function. Read file is send to the client with using send() function. I close socket for client.

```
# Fill in start
# Send one HTTP header line into socket
info = "HTTP/1.1 200 OK\r\n\r\n "
print(info)
connectionSocket.send(info.encode())
# Fill in end

# Send the content of the requested file to the client
for i in range(0, len(outputdata)):
    connectionSocket.send(outputdata[i].encode())
connectionSocket.close()
```

If the file is not found, 404 error will be send to the client and printed with an html page. This messages are sent with using send() function. Server socket is closed with "connectionSocket.close()" statement.

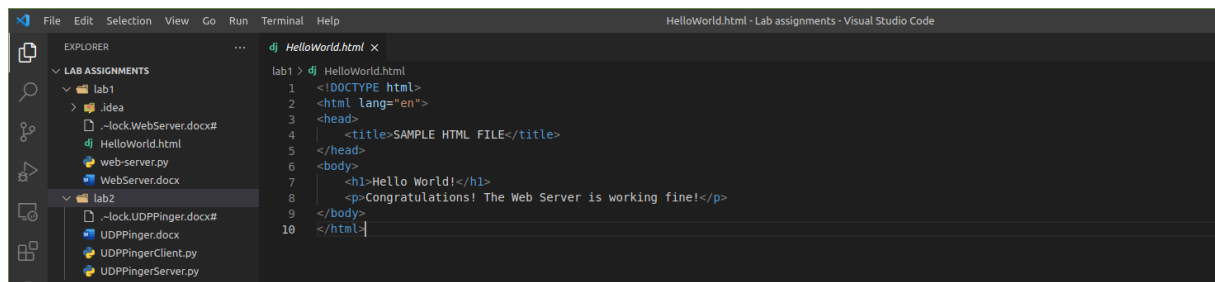
```
except IOError:

    # Send response message for file not found
    # Fill in start
    info = "HTTP/1.1 404 Not Found\r\n\r\n"
    print(info)
    connectionSocket.send(info.encode())

    info = "<html><head><title> 404 </title></head><body><h1>404 Not Found!</h1></body></html>\r\n"
    connectionSocket.send(info.encode())
    # Fill in end

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

This is my basic Html code.



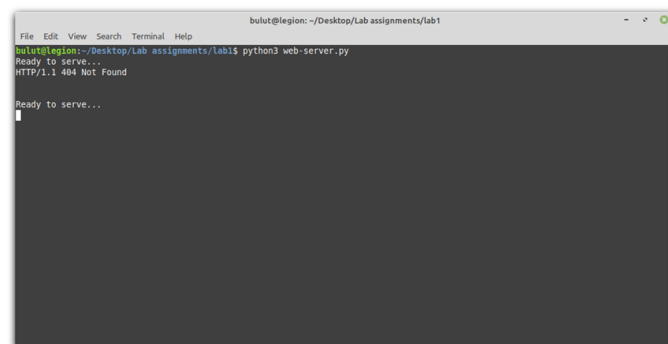
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <title>SAMPLE HTML FILE</title>
5 </head>
6 <body>
7   <h1>Hello World!</h1>
8   <p>Congratulations! The Web Server is working fine!</p>
9 </body>
10 </html>
```

TESTS

If file is not found, it will give 404 error.



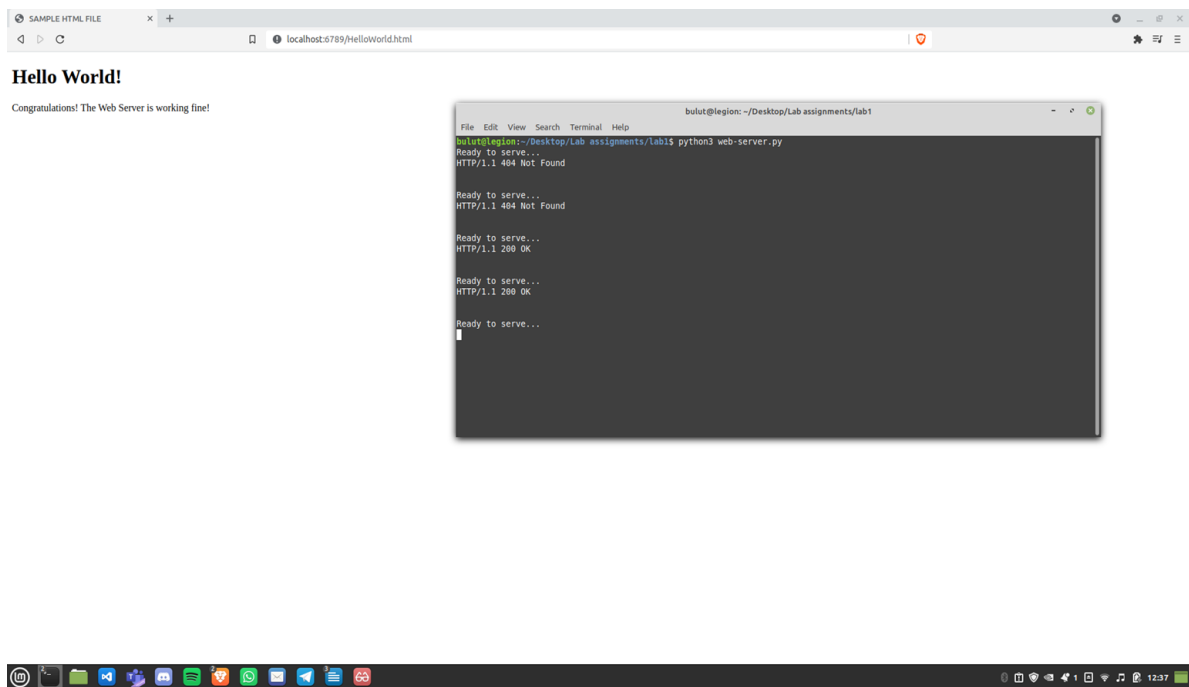
404 Not Found!



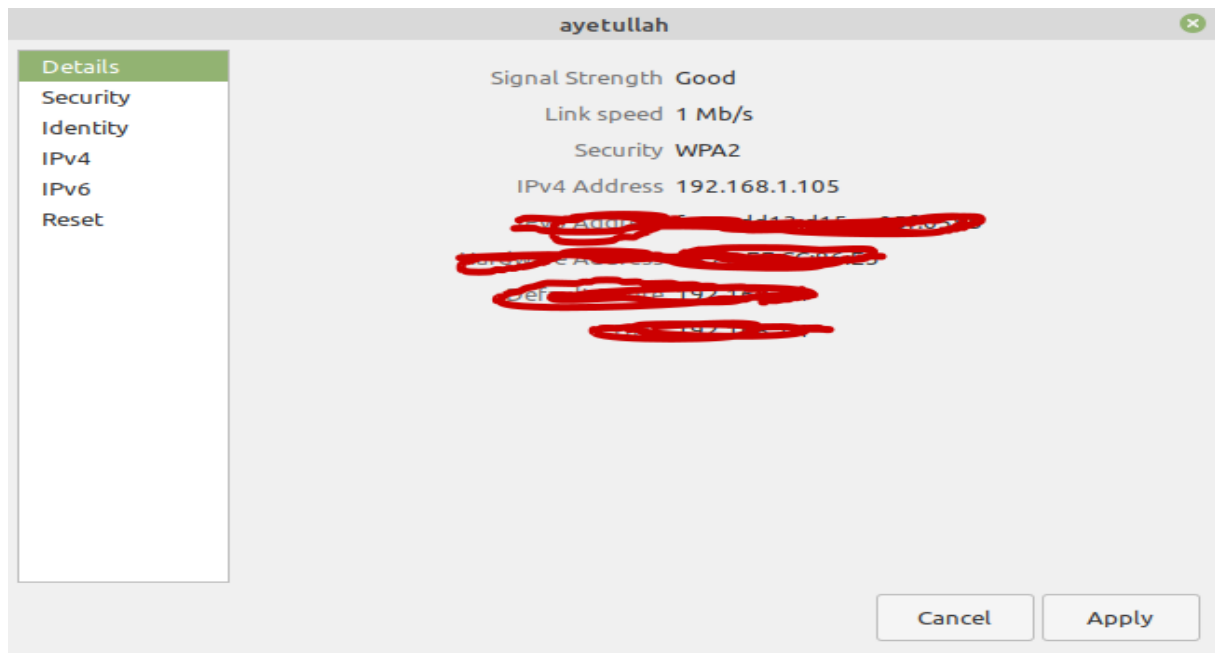
```
bulut@legion: ~/Desktop/Lab assignments/lab1
bulut@legion:~/Desktop/Lab assignments/lab1$ python3 web-server.py
Ready to serve...
HTTP/1.1 404 Not Found
Ready to serve...
```



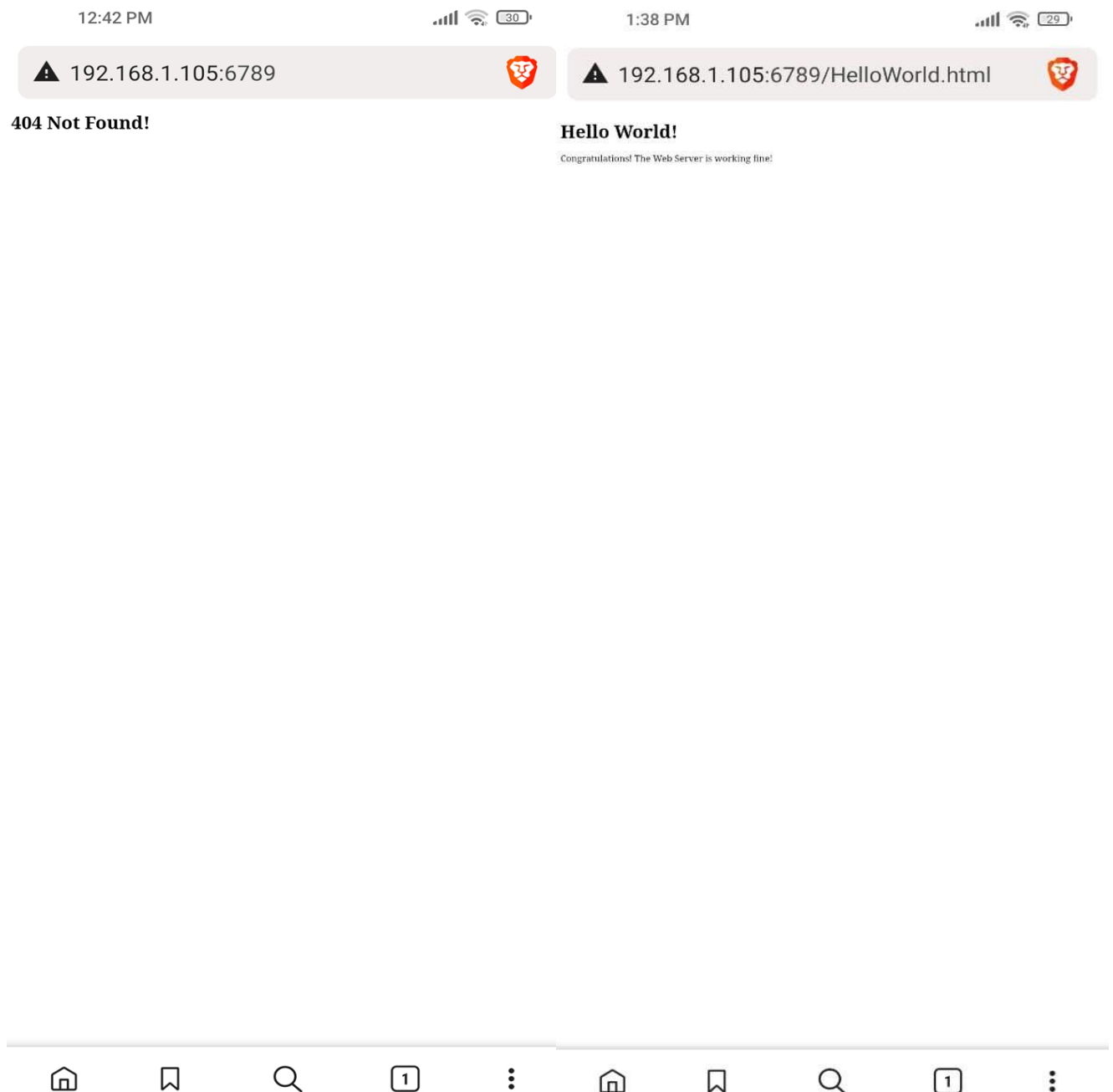
If file is found, it will show the file content and print “OK” message.



My server ip is 192.168.1.105



I tested with my mobile device. If the file is not found, it will print 404 error message. If it finds the file, it prints content of file.

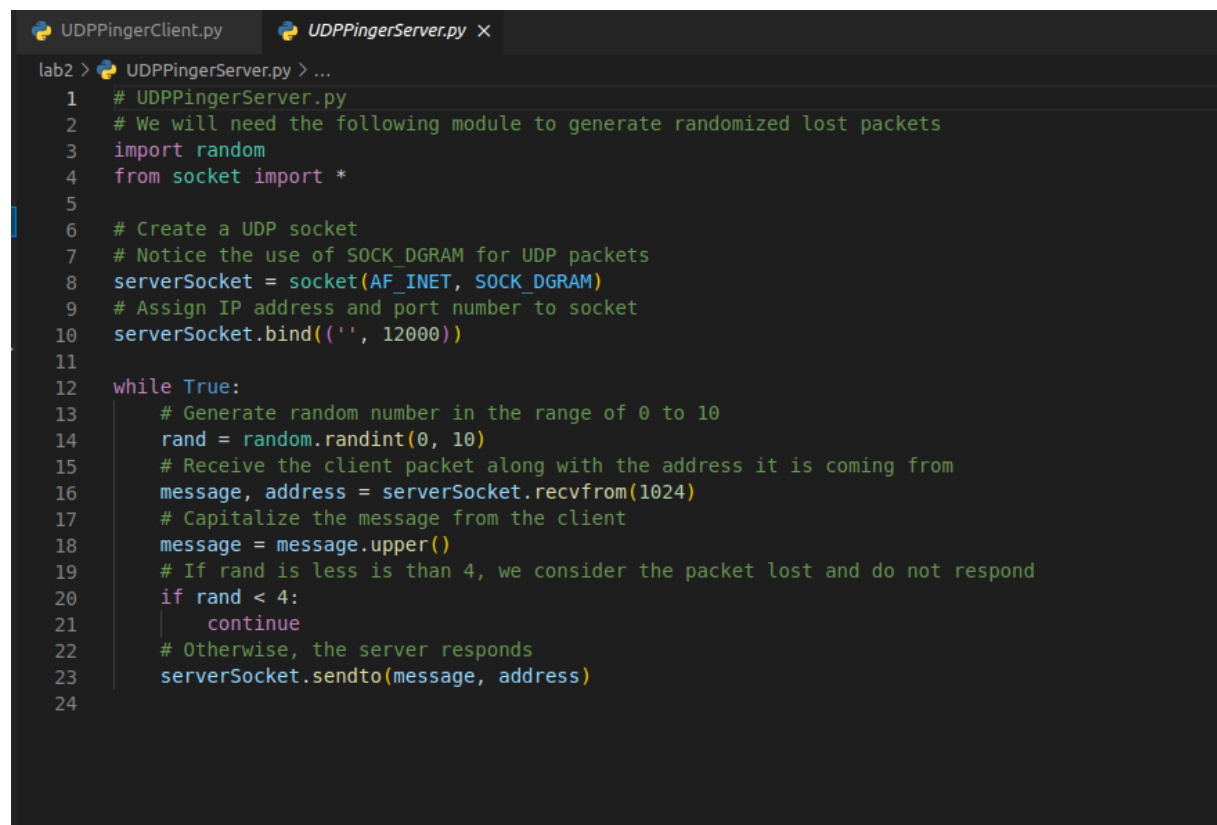


Assignment 2 : UDP Pinger

In this part, a client program is implemented. The client sends 10 pings to the server which is given. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. The client waits up to one second for a reply; if no reply is received within one second, the client program assumes that the packet was lost during transmission across the network. This client program sends the ping message using UDP (connectionless) and it prints the response message from server, if any. It calculates and prints the round trip time (RTT), in seconds, of each packet, if server responds. Otherwise, it prints "Request timed out".

Code Explanation

Server Code | Already given



```
lab2 > UDPPingerServer.py > ...
1  # UDPPingerServer.py
2  # We will need the following module to generate randomized lost packets
3  import random
4  from socket import *
5
6  # Create a UDP socket
7  # Notice the use of SOCK_DGRAM for UDP packets
8  serverSocket = socket(AF_INET, SOCK_DGRAM)
9  # Assign IP address and port number to socket
10 serverSocket.bind('', 12000)
11
12 while True:
13     # Generate random number in the range of 0 to 10
14     rand = random.randint(0, 10)
15     # Receive the client packet along with the address it is coming from
16     message, address = serverSocket.recvfrom(1024)
17     # Capitalize the message from the client
18     message = message.upper()
19     # If rand is less is than 4, we consider the packet lost and do not respond
20     if rand < 4:
21         continue
22     # Otherwise, the server responds
23     serverSocket.sendto(message, address)
24
```

Client code:

Time module will give current time. Socket module is for communication between client and server. I will create an UDP socket and set timeout. By looking at the timeout, the client will wait up one second for reply. Sequence_number is for count client ping number. It will be in a loop for 10 times.

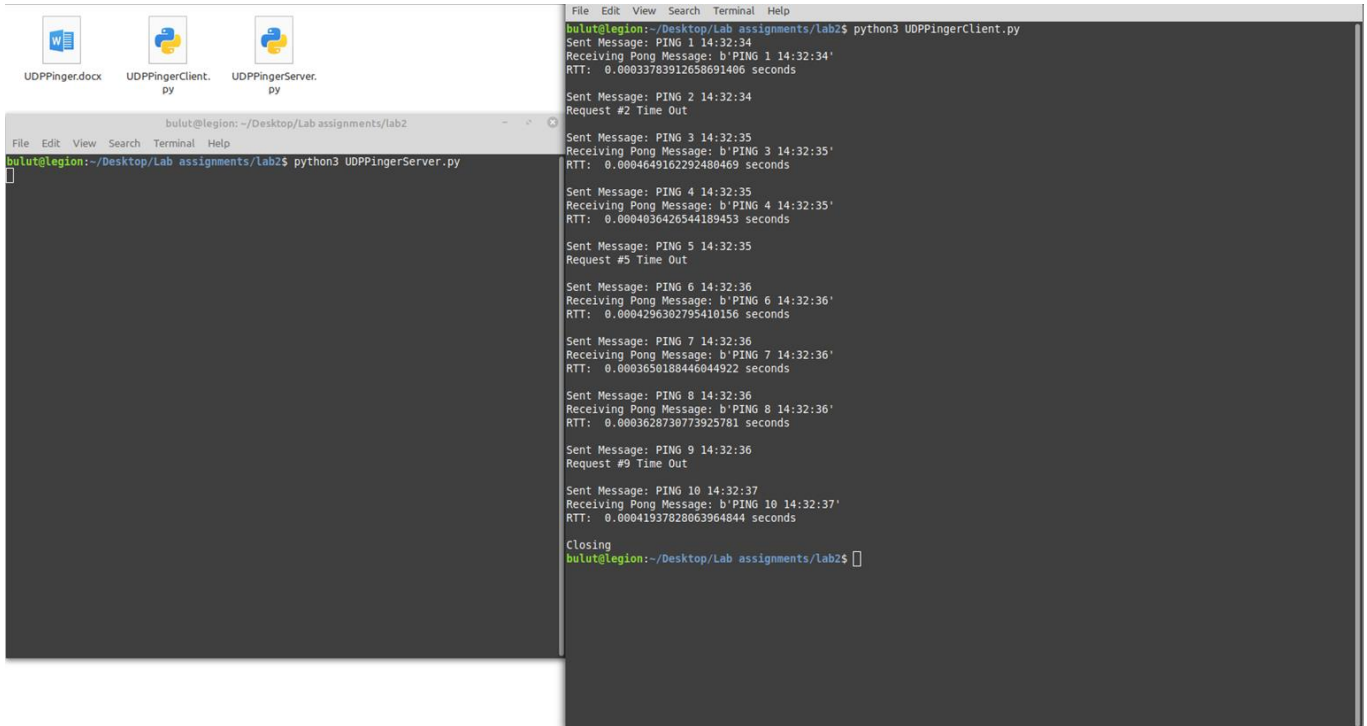
```
lab2 > UDPPingerClient.py > ...
1  from socket import *
2  from time import *
3
4  clientSocket = socket(AF_INET,SOCK_DGRAM) #create the socket.
5  clientSocket.settimeout(1) #set the timeout at 1 second.
6  sequence_number = 1 #ping counter.
7
```

Time() func gives us the current time. I send the ping message via localhost and also print to the terminal. After that, the program gets message from server and calculate finished time. It calculates RTT and prints. On timeout situation, it prints timeout information and goes loop start point. In the end , it closes the socket.

```
UDPPingerClient.py x UDPPingerServer.py
lab2 > UDPPingerClient.py > ...
1  from socket import *
2  from time import *
3
4  clientSocket = socket(AF_INET,SOCK_DGRAM) #create the socket.
5  clientSocket.settimeout(1) #set the timeout at 1 second.
6  sequence_number = 1 #ping counter.
7
8  while sequence_number<=10:
9
10     startTime = time() # keep current time.
11
12     try:
13         message = "PING " + str(sequence_number) + " " + str(strftime("%H:%M:%S"))
14         clientSocket.sendto(message.encode(), ('localhost', 12000)) #send ping message via localhost
15         print("Sent Message: " + message)
16
17         data, address = clientSocket.recvfrom(1024) #recieving message from server.
18         responseTime = time() #finish time.
19         rtt=responseTime-startTime # the round trip time (RTT)
20
21         print("Receiving Pong Message:",data)
22         print("RTT: ", str(rtt) + " seconds\n")
23
24     except timeout:
25         print("Request #" + str(sequence_number) + " Time Out\n")
26
27     sequence_number += 1
28
29 print("Closing")
30 clientSocket.close()
31
```


Tests

test-1: (3 time out – packet loss)



```
bulut@legion: ~/Desktop/Lab assignments/lab2
File Edit View Search Terminal Help
bulut@legion:~/Desktop/Lab assignments/lab2$ python3 UDPPingerClient.py
Sent Message: PING 1 14:32:34
Receiving Pong Message: b'PING 1 14:32:34'
RTT: 0.00033783912658691406 seconds

Sent Message: PING 2 14:32:34
Request #2 Time Out

Sent Message: PING 3 14:32:35
Receiving Pong Message: b'PING 3 14:32:35'
RTT: 0.0004649162292480469 seconds

Sent Message: PING 4 14:32:35
Receiving Pong Message: b'PING 4 14:32:35'
RTT: 0.0004036426544189453 seconds

Sent Message: PING 5 14:32:35
Request #5 Time Out

Sent Message: PING 6 14:32:36
Receiving Pong Message: b'PING 6 14:32:36'
RTT: 0.0004296302795410156 seconds

Sent Message: PING 7 14:32:36
Receiving Pong Message: b'PING 7 14:32:36'
RTT: 0.0003650188446044922 seconds

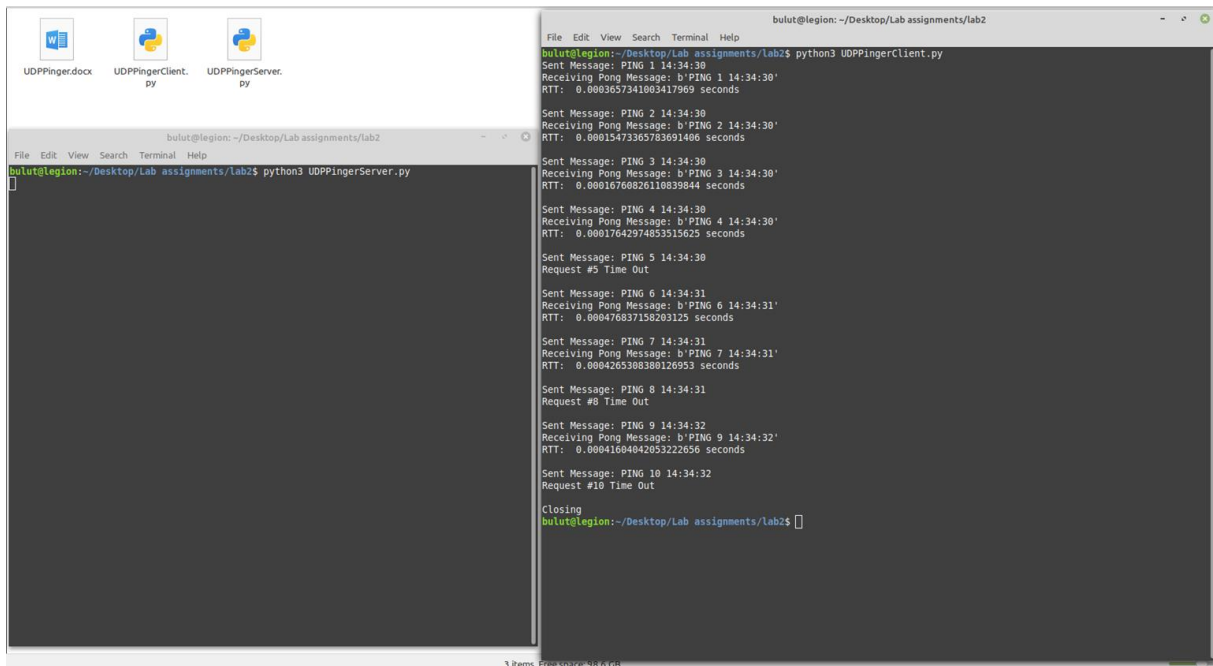
Sent Message: PING 8 14:32:36
Receiving Pong Message: b'PING 8 14:32:36'
RTT: 0.0003628730773925781 seconds

Sent Message: PING 9 14:32:36
Request #9 Time Out

Sent Message: PING 10 14:32:37
Receiving Pong Message: b'PING 10 14:32:37'
RTT: 0.00041937828063964844 seconds

Closing
bulut@legion:~/Desktop/Lab assignments/lab2$
```

test-2: (3 time out – packet loss)



```
bulut@legion: ~/Desktop/Lab assignments/lab2
File Edit View Search Terminal Help
bulut@legion:~/Desktop/Lab assignments/lab2$ python3 UDPPingerClient.py
Sent Message: PING 1 14:34:30
Receiving Pong Message: b'PING 1 14:34:30'
RTT: 0.0003657341003417969 seconds

Sent Message: PING 2 14:34:30
Receiving Pong Message: b'PING 2 14:34:30'
RTT: 0.00015473365783691406 seconds

Sent Message: PING 3 14:34:30
Receiving Pong Message: b'PING 3 14:34:30'
RTT: 0.00016760826110839844 seconds

Sent Message: PING 4 14:34:30
Receiving Pong Message: b'PING 4 14:34:30'
RTT: 0.00017642974853515625 seconds

Sent Message: PING 5 14:34:30
Request #5 Time Out

Sent Message: PING 6 14:34:31
Receiving Pong Message: b'PING 6 14:34:31'
RTT: 0.000476837158203125 seconds

Sent Message: PING 7 14:34:31
Receiving Pong Message: b'PING 7 14:34:31'
RTT: 0.0004265308380126953 seconds

Sent Message: PING 8 14:34:31
Request #8 Time Out

Sent Message: PING 9 14:34:32
Receiving Pong Message: b'PING 9 14:34:32'
RTT: 0.0004160404203222656 seconds

Sent Message: PING 10 14:34:32
Request #10 Time Out

Closing
bulut@legion:~/Desktop/Lab assignments/lab2$
```

Assignment 3 : Mail Client

A mail client is developed which sends email to any recipient. This client establishes a TCP connection with Google mail server, dialogues with the mail server using the SMTP protocol, sends an email message to a recipient via the mail server, and finally closes the TCP connection with the mail server.

I chose Google server for mail server.

Google Gmail SMTP server: smtp.gmail.com

Gmail SMTP port: 587(TLS)

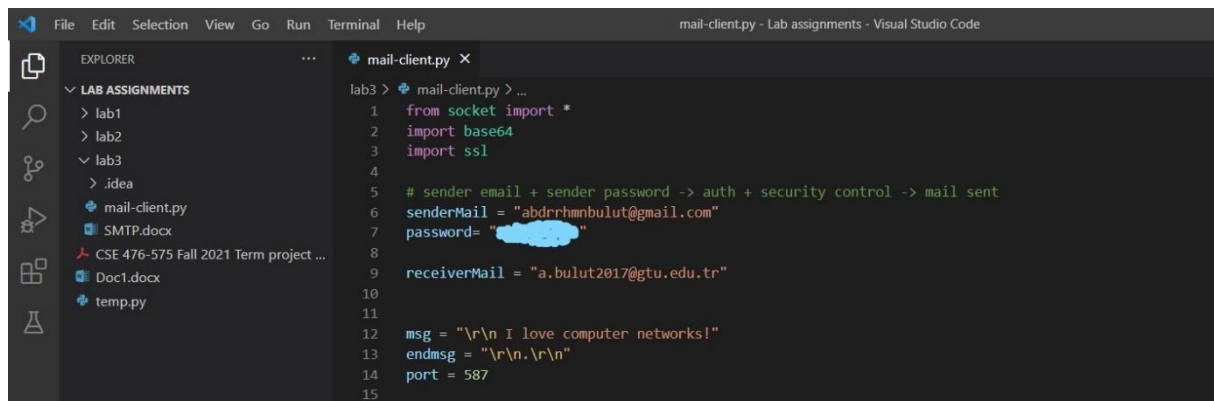
Transport Layer Security(TLS) and Secure Sockets Layer(SSL) is added for authentication and security reasons.

TESTS

Socket module is needed for server and client communication. Base64 module is used for encoding mail address and password. Ssl module is for connecting google gmail server. Gmail SMTP port: 587(TLS). Msg is a message that will be sent to receiver mail address.

Gmail can create some problems for security when sending mail. There may be 2 solutions. One of them is to enable IMAP access. To do this, go Gmail all settings page. Click "Forwarding and POP/IMAP" tab and enable IMAP. The Other one is to let less secure app access. To do this, go Google account settings and security part. Enable Less secure app access. I used my Google account as sender email and my school email as receiver email and I used Python 2.7.18 version.

The image contains two screenshots. The top screenshot shows the Google Account 'Security' page. It features a sidebar with links like Home, Personal info, Data & privacy, Security (highlighted), People & sharing, Payments & subscriptions, and About. The main content area has a 'Less secure app access' warning, stating the account is vulnerable and suggesting to turn off access. A toggle switch is currently 'On', and a link 'Turn off access (recommended)' is provided. The bottom screenshot shows the Gmail 'Settings' page, specifically the 'Forwarding and POP/IMAP' tab. It includes sections for 'Forwarding' (with an 'Add a forwarding address' button), 'POP download' (with options for POP status and message handling), 'IMAP access' (with 'Enable IMAP' selected), and 'When I mark a message in IMAP as deleted' (with 'Auto-Expunge on' selected). There are also links for 'Configuration instructions' and 'Folder size limits'.



```
mail-client.py - Lab assignments - Visual Studio Code

EXPLORER
└─ LAB ASSIGNMENTS
  └─ lab3
    ├── mail-client.py
    ├── SMTP.docx
    ├── CSE 476-575 Fall 2021 Term project ...
    ├── Doc1.docx
    └── temp.py

mail-client.py
1  from socket import *
2  import base64
3  import ssl
4
5  # sender email + sender password -> auth + security control -> mail sent
6  senderMail = "abdrhmmbulut@gmail.com"
7  password= " "
8
9  receiverMail = "a.bulut2017@gtu.edu.tr"
10
11
12 msg = "\r\n I love computer networks!"
13 endmsg = "\r\n.\r\n"
14 port = 587
15
```

Google Gmail SMTP server: smtp.gmail.com with port 587. I created a socket called clientSocket and established a TCP connection with mailserver between mail server and client. Line 33 send HELO command and then prints server response message. If no response is received, the error message will be printed.

```
16
17 # Choose a mail server (I choose Google mail server) and call mailserver
18 mailserver = ("smtp.gmail.com", port)
19
20 # Create socket called clientSocket and establish a TCP connection with mailserver
21 #Fill in start
22 clientSocket = socket(AF_INET, SOCK_STREAM)
23 clientSocket.connect(mailserver)
24 #Fill in end
25
26 recv = clientSocket.recv(1024)
27 print (recv)
28 if recv[:3] != '220':
29     print ('220 reply not received from server.')
30
31 # Send HELO command and print server response.
32 heloCommand = 'HELO Alice\r\n'
33 clientSocket.send(heloCommand)
34 recv1 = clientSocket.recv(1024)
35 print (recv1)
36 if recv1[:3] != '250':
37     print ('250 reply not received from server.' )
38
```

In this part I open a TLS for Gmail mail server. It is needed for security reasons. It encrypes TLS message which is "starttls" and sends to the mail server via send() function. At line 49, It make a SSL socket for security reasons.

```
38
39
40 #TLS for Google mail server
41 starttls = 'starttls\r\n'
42 clientSocket.send((starttls).encode())
43 recv2=clientSocket.recv(1024)
44 print (recv2)
45 if recv2[:3] != '220':
46     print ('220 reply not received from server.')
47
48
49 clientSocket = ssl.wrap_socket(clientSocket)
50
```

Email address and password are encrypted with encode() function. After that, they are encrypted with b64encode() function as one piece. Finally, auth message is created and sent to the mail server.

```
50
51 # AUTH with base64 encoded senderMail and password
52 base64_str = ("\x00"+senderMail+"\x00"+password).encode()
53 base64_str = base64.b64encode(base64_str)
54 authMsg = "AUTH PLAIN ".encode()+base64_str+"\r\n".encode()
55 clientSocket.send(authMsg)
56 recv_auth = clientSocket.recv(1024)
57 print(recv_auth.decode())
58 if recv1[:3] != '250':
59     print('250 reply not received from server.')
60
```

This part has sender and receiver informations. This part encryptes terms and sends to the mail server .

```
61
62 # Send MAIL FROM command and print server response.
63 # Fill in start
64 mailFrom = "MAIL FROM: <"+senderMail+"> \r\n"
65 clientSocket.send(mailFrom.encode())
66 recv6 =clientSocket.recv(1024)
67 print (recv6)
68 if recv6[:3] != '250':
69     print ('250 reply not received from server.')
70 # Fill in end
71
72 # Fill in start
73 # Send RCPT TO command and print server response.
74 rcptTo = "RCPT TO: <"+receiverMail+"> \r\n"
75 clientSocket.send(rcptTo.encode())
76 recv7 = clientSocket.recv(1024)
77 print (recv7)
78 if recv7[:3] != '250':
79     print ('250 reply not received from server.')
80 # Fill in end
81
82 # Fill in start
```

DATA part declares mail content. In message part, Subject can be defined like in picture. If we put newline in Subject message, the part after newline will be the content.

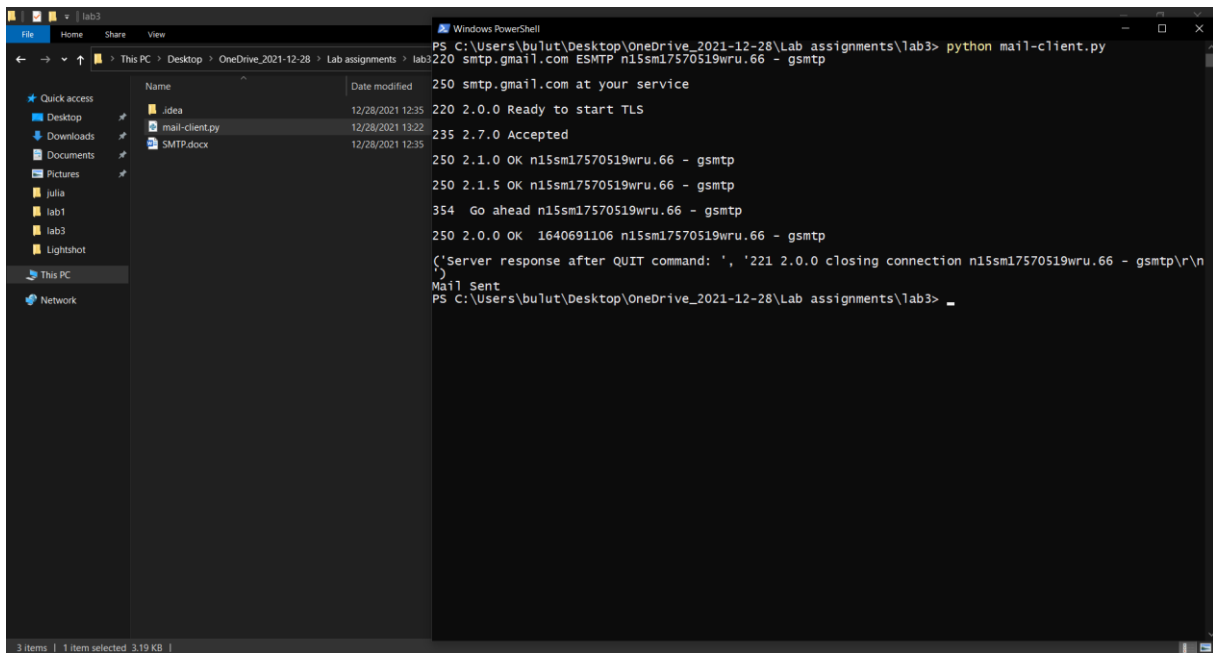
```
82 # Fill in start
83 # Send DATA command and print server response.
84 data = "DATA\r\n"
85 clientSocket.send(data.encode())
86 recv8 = clientSocket.recv(1024).decode()
87 print (recv8)
88 if recv8[:3] != '354':
89     print ('354 reply not received from server.')
90 # Fill in end
91
92
93 # Fill in start
94 # Send message data.
95 message = 'SUBJECT: SMTP Mail Client Testing \nContent \r\n'
96 clientSocket.send(message.encode())
97 clientSocket.send(msg.encode())
98 # Fill in end
99
```

In this part, the message ends with single period. Quit command is sent and server response is received. At the end of code, socket is closed.

```
100 # Fill in start
101 #message ends with single period.
102 clientSocket.send(msg.encode())
103 recv9 = clientSocket.recv(1024)
104 print (recv9.decode())
105 if recv9[:3] != '250':
106     print ('250 reply not received from server.')
107 # Fill in end
108
109
110 # Fill in start
111 # Send QUIT command and get server response.
112 clientSocket.send("QUIT\r\n".encode())
113 recv10=clientSocket.recv(1024)
114 print ('Server response after QUIT command: ', recv10)
115 if recv10[:3] != '221':
116     print ('221 reply not received from server.')
117
118 print("Mail Sent")
119 clientSocket.close() #close socket.
120 # Fill in end
121
122
```

TESTS

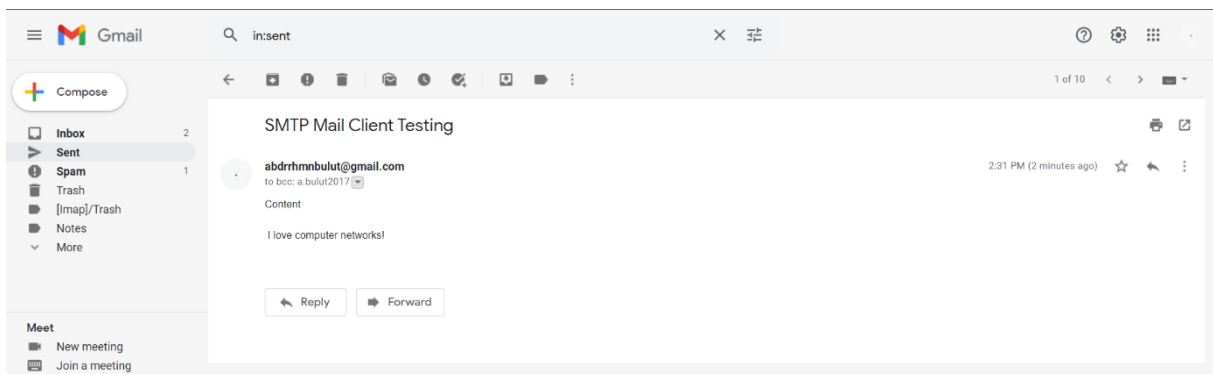
Code work-1 | Powershell | gmail -> gtu mail



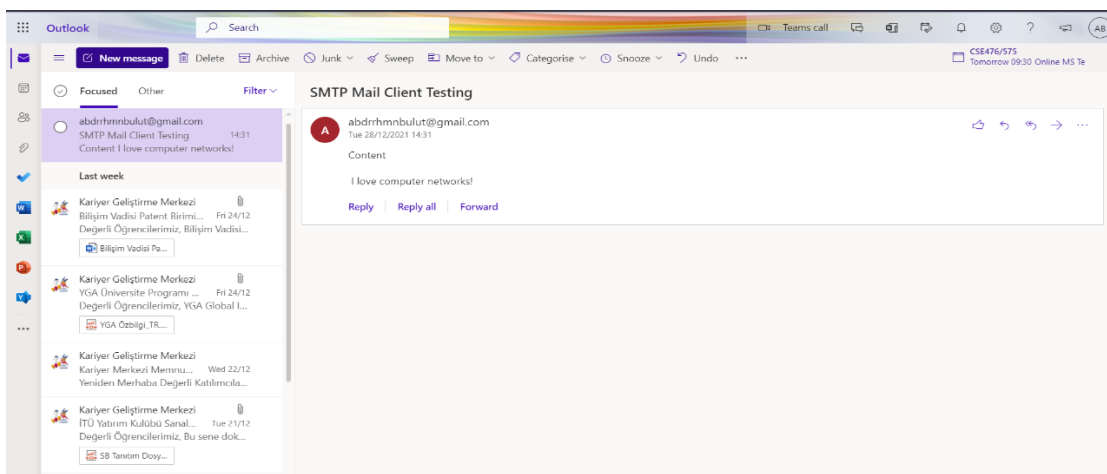
The screenshot shows a Windows File Explorer window on the left, displaying the contents of a folder named 'lab3'. The files listed are 'idea', 'mail-client.py', and 'SMTP.docx'. The right pane shows the 'Date modified' column with dates '12/28/2021 12:35' and '12/28/2021 13:22'. The main window is a PowerShell terminal running the command `python mail-client.py`. The output shows the SMTP connection process to smtp.gmail.com, including TLS negotiation and the successful sending of an email. The final output is 'Mail Sent'.

```
PS C:\Users\bulut\Desktop\OneDrive_2021-12-28\Lab assignments\lab3> python mail-client.py
220 smtp.gmail.com ESMTP n15sm17570519wru.66 - gsmt
250 smtp.gmail.com at your service
220 2.0.0 Ready to start TLS
235 2.7.0 Accepted
250 2.1.0 OK n15sm17570519wru.66 - gsmt
250 2.1.5 OK n15sm17570519wru.66 - gsmt
354 Go ahead n15sm17570519wru.66 - gsmt
250 2.0.0 OK 1640691106 n15sm17570519wru.66 - gsmt
('server response after QUIT command: ', '221 2.0.0 closing connection n15sm17570519wru.66 - gsmt\r\n')
Mail Sent
PS C:\Users\bulut\Desktop\OneDrive_2021-12-28\Lab assignments\lab3>
```

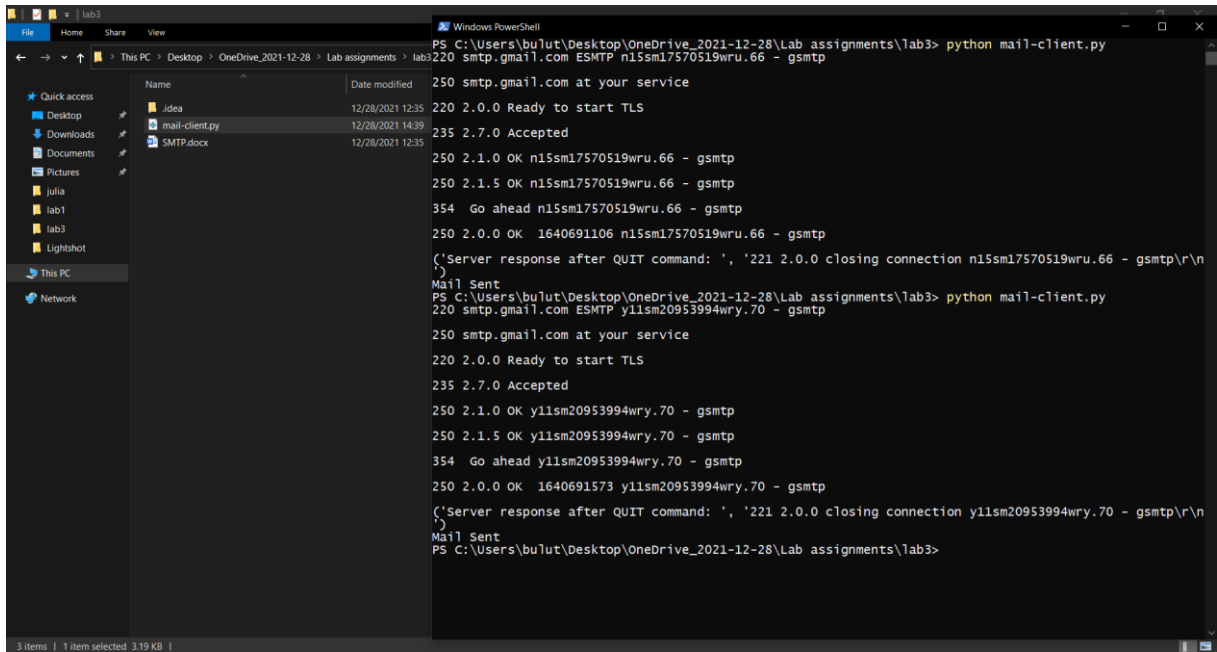
Sender : abdrhmnbulut@gmail.com



Receiver : a.bulut2017@gtu.edu.tr



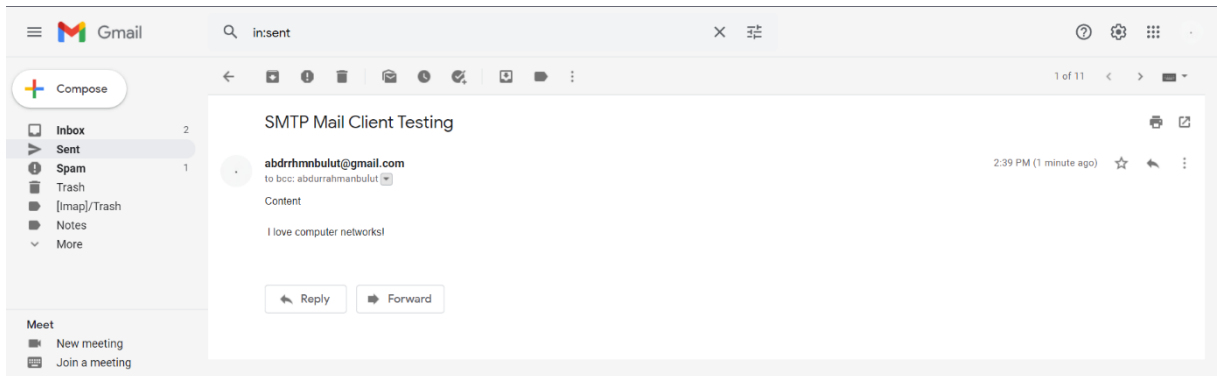
Code work-2 | Powershell | gmail -> yahoo mail



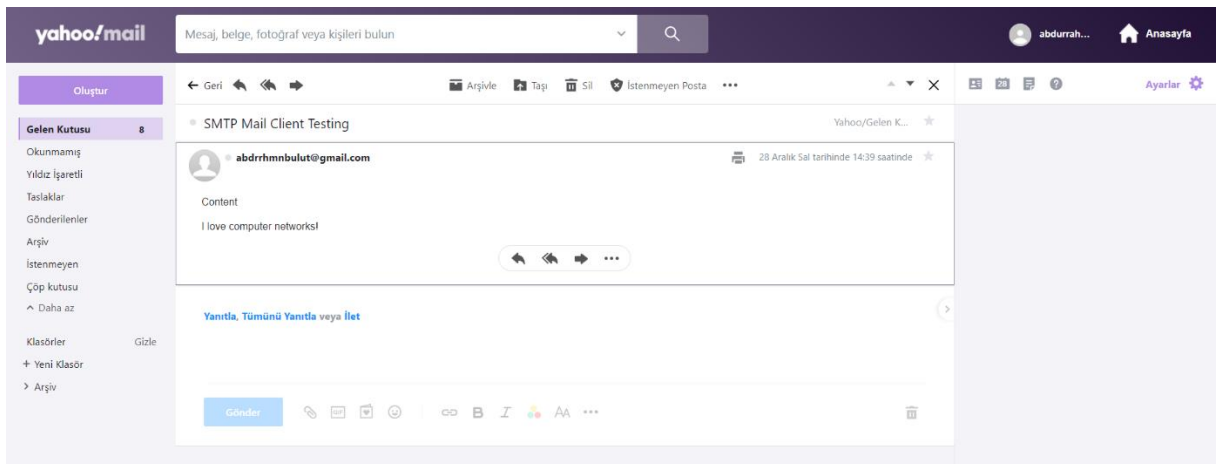
The screenshot shows a Windows File Explorer window on the left, displaying the contents of a folder named 'lab3' on the Desktop. The files listed are 'idea', 'mail-client.py', and 'SMTP.docx'. On the right, a PowerShell window is open, showing the execution of a Python script named 'mail-client.py'. The script connects to the Gmail SMTP server (smtp.gmail.com) and sends an email. The output of the script is visible in the PowerShell window, showing the SMTP session details and the successful completion of the email sending process.

```
PS C:\Users\bulut\Desktop\OneDrive_2021-12-28\Lab assignments\lab3> python mail-client.py
220 smtp.gmail.com ESMT n15sm17570519wru.66 - gsmt
250 smtp.gmail.com at your service
220 2.0.0 Ready to start TLS
235 2.7.0 Accepted
250 2.1.0 OK n15sm17570519wru.66 - gsmt
250 2.1.5 OK n15sm17570519wru.66 - gsmt
354 Go ahead n15sm17570519wru.66 - gsmt
250 2.0.0 OK 1640691106 n15sm17570519wru.66 - gsmt
('Server response after QUIT command: ', '221 2.0.0 closing connection n15sm17570519wru.66 - gsmt\r\n
Mail Sent
PS C:\Users\bulut\Desktop\OneDrive_2021-12-28\Lab assignments\lab3> python mail-client.py
220 smtp.gmail.com ESMT y11sm20953994wry.70 - gsmt
250 smtp.gmail.com at your service
220 2.0.0 Ready to start TLS
235 2.7.0 Accepted
250 2.1.0 OK y11sm20953994wry.70 - gsmt
250 2.1.5 OK y11sm20953994wry.70 - gsmt
354 Go ahead y11sm20953994wry.70 - gsmt
250 2.0.0 OK 1640691573 y11sm20953994wry.70 - gsmt
('Server response after QUIT command: ', '221 2.0.0 closing connection y11sm20953994wry.70 - gsmt\r\n
Mail Sent
PS C:\Users\bulut\Desktop\OneDrive_2021-12-28\Lab assignments\lab3>
```

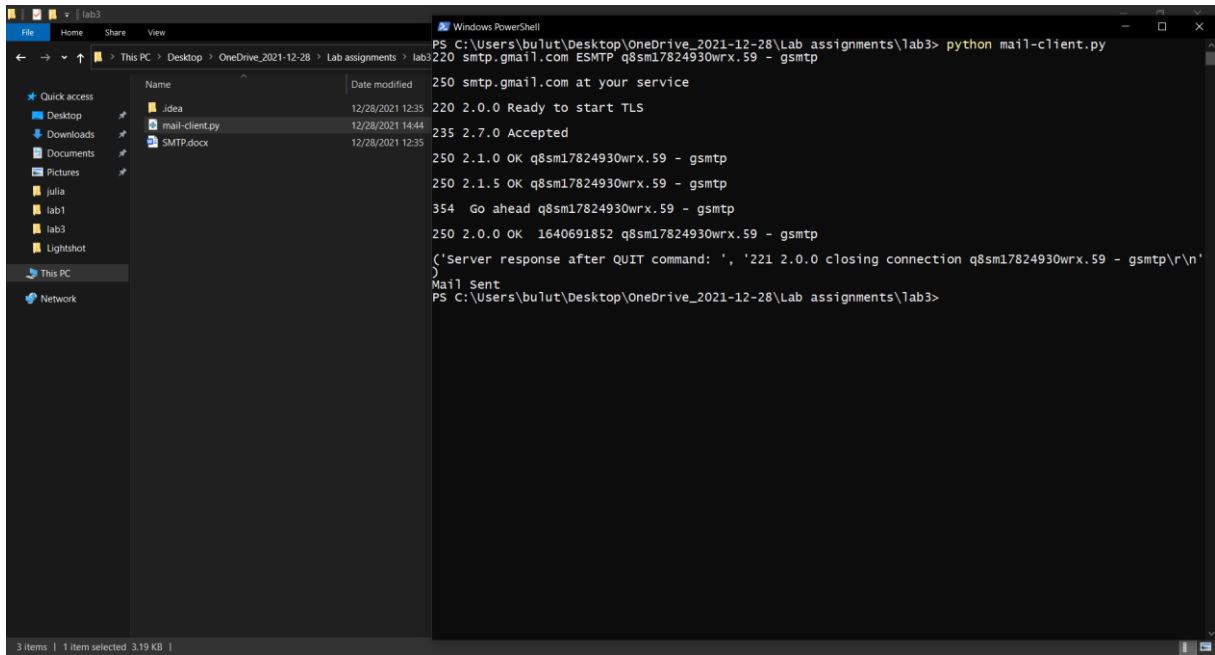
Sender : abdrhmnbulut@gmail.com



Receiver : abdurrahmanbulut@yahoo.com

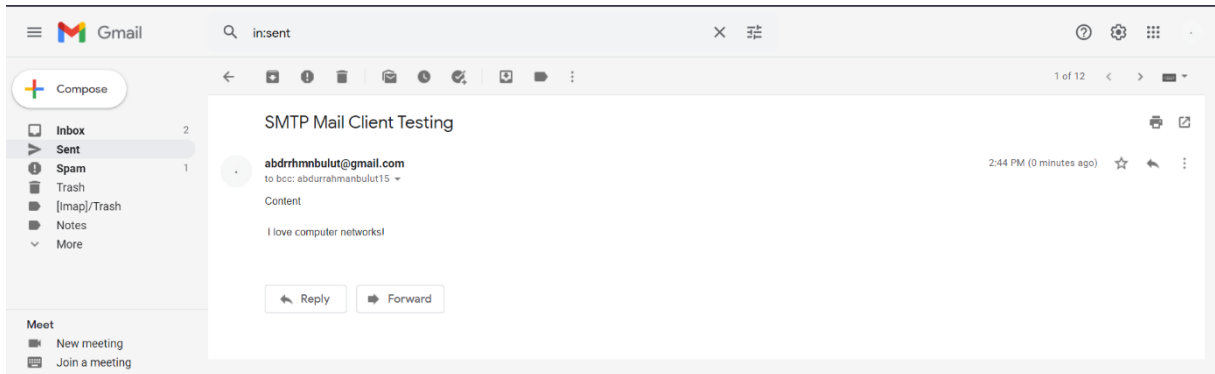


Code work-3 | Powershell | gmail -> gmail



The screenshot shows a Windows File Explorer window on the left, displaying the contents of a folder named 'lab3' located at 'C:\Users\bulut\Desktop\OneDrive_2021-12-28\Lab assignments\lab3'. The files listed are 'idea', 'mail-client.py', and 'SMTP.docx'. On the right, a Windows PowerShell window is open, showing the execution of the 'mail-client.py' script. The command entered is 'python mail-client.py'. The output shows the script connecting to 'smtp.gmail.com' on port 587, authenticating with the username 'abdurrahmanbulut@gmail.com' and password 'q8sm17824930wrX.59', and successfully sending an email. The email content is 'I love computer networks!'. The PowerShell window also shows the SMTP conversation details, including '250 smtp.gmail.com at your service', '220 2.0.0 Ready to start TLS', '235 2.7.0 Accepted', and '250 2.1.0 OK q8sm17824930wrX.59 - gsmtpp'.

Sender : abdrhmnbulut@gmail.com



Receiver : abdurrahmanbulut15@gmail.com

