

# Term Project Final Report

**Elif Akgün**

1801042251

## 1. Web Server

### 1.1 Assignment Definition

In this assignment, a simple Web server is developed in Python that is capable of processing only one request. Specifically, the Web server creates a connection socket when contacted by a client (browser), receives the HTTP request from this connection, parse the request to determine the specific file being requested, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in the server, the server returns a "404 Not Found" error message.

### 1.2 Program Code

```
1 # Import socket module
2 from socket import *
3
4 serverSocket = socket(AF_INET, SOCK_STREAM) #TCP server socket
5
6 # Prepare a server socket
7 serverPort = 2525
8 serverSocket.bind(('', serverPort))
9 serverSocket.listen(1) #listen 1 connction
10
11 while True:
12     #Establish the connection
13     print ("Ready to serve...")
14
15     # Set up a new connection from the client
16     connectionSocket, addr = serverSocket.accept()
17
18     try:
19         # Receives the request message from the client
20         message = connectionSocket.recv(1500)
21         filename = message.split()[1] #path of request
22         f = open(filename[1:], 'rb')
23         outputdata = f.read()
```

```

24     #print(outputdata)
25
26     #Send one HTTP header line into socket
27     connectionSocket.send(str("HTTP/1.1 200 OK\r\n\r\n").encode())
28
29     #Send the content of the requested file to the client
30     for i in range(0, len(outputdata)):
31         connectionSocket.send(chr(outputdata[i]).encode())
32
33     connectionSocket.close()
34
35 except IOError:
36     #Send response message for file not found
37     connectionSocket.send(str("HTTP/1.1 404 Not Found\r\n\r\n").encode()
38 )
39     connectionSocket.send(str("<html><head></head><body><h1>404 Not
40 Found</h1></body></html>\r\n").encode())
41
42     #Close client socket
43     connectionSocket.close()
44
45 serverSocket.close()

```

### 1.3 Tests

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example: <http://128.238.251.26:6789/HelloWorld.html> 'HelloWorld.html' is the name of the file you placed in the server directory. The browser should then display the contents of HelloWorld.html. If the file is not present at the server, you should get a "404 Not Found" message.

Figure 1: Server runs with HelloWorld.html that is in the same directory

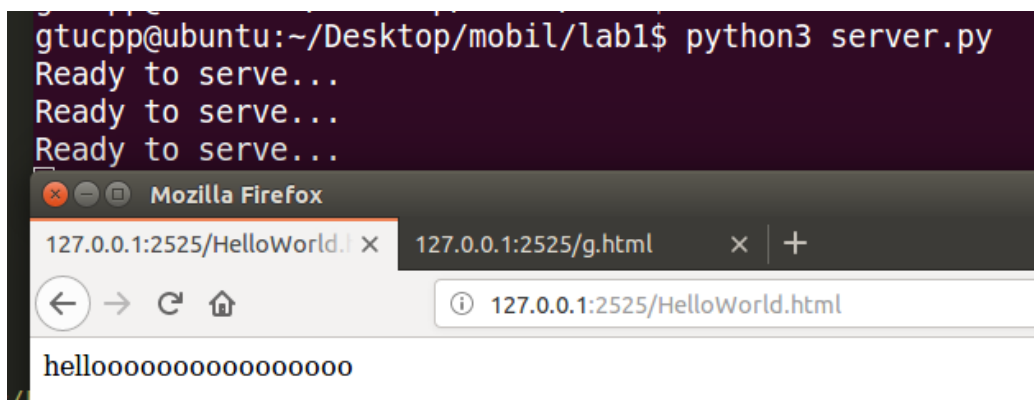
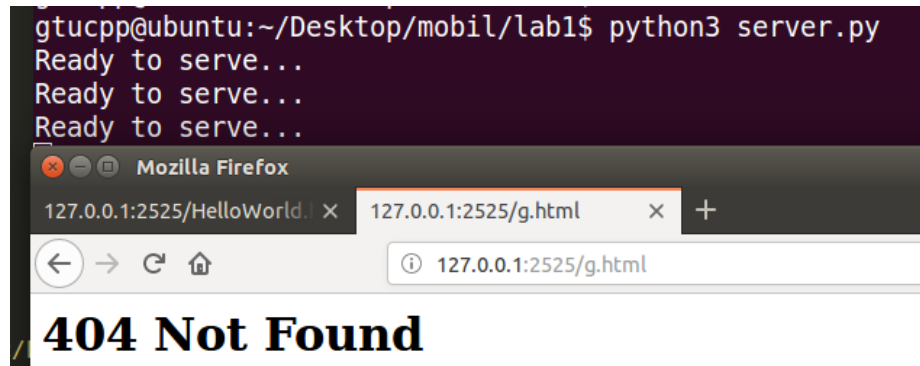


Figure 2: Server runs with g.html that is not present



## 2. UDP Pinger

### 2.1 Assignment Definition

In this assignment, the client sends 10 ping messages to the target server over UDP. For each message, determine and print the delay between when the client sent the ping message and received the pong message. This delay is called the Round Trip Time (RTT). Because UDP is an unreliable protocol, a packet sent by the client or server may be lost. 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 from the server; if no reply is received, it assumes that the packet was lost and print a message accordingly.

### 2.2 Program Code

We have the complete code for the server, so I write only client code. I took IP address and port number as command line argument. Then I create socket. In a for loop, I sent Ping message 10 times. I calculate RTT and if response does not come in one second, it means packet was lost.

```

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
10 # Assign IP address and port number to socket
11 serverSocket.bind(('', 2525))
12
13 while True:
14     # Generate random number in the range of 0 to 10
15     rand = random.randint(0, 10)

```

```

16
17 # Receive the client packet along with the address it is coming from
18 message, address = serverSocket.recvfrom(1024)
19
20 # Capitalize the message from the client
21 message = message.upper()
22
23 # If rand is less is than 4, we consider the packet lost and do not
    respond
24 if rand < 4:
25     continue
26 # Otherwise, the server responds
27 serverSocket.sendto(message, address)

```

```

1 #UDPPingerClient.py
2 from socket import *
3 from time import time, ctime
4 import sys
5
6 #there sohuld be 3 command line arguments
7 if (len(sys.argv) != 3):
8     print(" Usage: UDPPingerClient.py <server_host> <server_portNum>")
9     sys.exit()
10
11 ip_addr = sys.argv[1]
12 portNum = sys.argv[2]
13
14 #create client socket
15 clientSocket = socket(AF_INET, SOCK_DGRAM)
16 #wait one second for reply
17 clientSocket.settimeout(1)
18
19 #send and receive 10 ping messages
20 for i in range(0, 10):
21     t0 = time() #start time
22     message = "Ping " + str(i+1) + " " + ctime(t0)[11:19]
23
24     try:
25         #sends and receives the message
26         clientSocket.sendto(message.encode(),(ip_addr, int(portNum)))
27         response, server_addr = clientSocket.recvfrom(1024)
28
29         t1 = time() #end time
30
31         #response message from server
32         print(" Response: " + response.decode())
33
34         print("Round Trip Time: %.3fs \n" % (t1 - t0))
35     except:
36         print(" Ping " + str(i+1) + " request timed out\n")
37 clientSocket.close()

```

## 2.3 Tests

- 1) Run the UDPPingerServer.py
- 2) Run the UDPPingerClient.py as "UDPPingerClient.py < server\_host > < server\_portNum >"

Figure 3: Sample Output-1

```
gtucpp@ubuntu:~/Desktop/mobil/lab2$ python3 UDPPingerClient.py 127.0.0.1 2525
Response: PING 1 23:18:24
Round Trip Time: 0.003s

Response: PING 2 23:18:24
Round Trip Time: 0.000s

Response: PING 3 23:18:24
Round Trip Time: 0.000s

Ping 4 request timed out

Response: PING 5 23:18:25
Round Trip Time: 0.001s

Response: PING 6 23:18:25
Round Trip Time: 0.001s

Response: PING 7 23:18:25
Round Trip Time: 0.001s

Response: PING 8 23:18:25
Round Trip Time: 0.000s

Response: PING 9 23:18:25
Round Trip Time: 0.001s

Response: PING 10 23:18:25
Round Trip Time: 0.000s

gtucpp@ubuntu:~/Desktop/mobil/lab2$
```

Figure 4: Sample Output-2

```
gtucpp@ubuntu:~/Desktop/mobil/lab2$ python3 UDPPingerClient.py 127.0.0.1 2525
Response: PING 1 23:19:27
Round Trip Time: 0.003s

Response: PING 2 23:19:27
Round Trip Time: 0.000s

Ping 3 request timed out

Ping 4 request timed out

Ping 5 request timed out

Ping 6 request timed out

Response: PING 7 23:19:31
Round Trip Time: 0.001s

Ping 8 request timed out

Response: PING 9 23:19:32
Round Trip Time: 0.001s

Response: PING 10 23:19:32
Round Trip Time: 0.001s

gtucpp@ubuntu:~/Desktop/mobil/lab2$
```

### 3. Mail Client

#### 3.1 Assignment Definition

In this assignment, the task is to develop a simple mail client that sends email to any recipient. The client needs to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server. Python provides a module, called `smtplib`, which has built in methods to send mail using SMTP protocol. However, this module is not used in this assignment, because it hides the details of SMTP and socket programming.

#### 3.2 Program Code

First I set mail server and port. I chose Google mail server and port 587. Other port numbers may occurs error. We should send some commands to send mail. I send following commands respectively.

- HELO command
- STARTTLS command for secure connection
- AUTH LOGIN command
- Mail address and password
- MAIL FROM command with mail address
- RCPT TO command with target mail address
- DATA command
- Subject and message
- QUIT command to quit connection

```

1 from socket import *
2 import ssl
3 import base64
4
5 mailAddress = "mail@gmail.com"
6 password = "password"
7 targetMail = "targetmail@mail.com"
8 subject = "Enter mail's subject"
9 message = "Enter message"
10
11 #set mail server
12 mailServer = ("smtp.gmail.com", 587)
13 #create socket for TCP
14 clientSocket = socket(AF_INET, SOCK_STREAM)
15 clientSocket.connect(mailServer)
16 #get response and print
17 response = clientSocket.recv(1024).decode()
18 print("response: ", response)
19 if(response[:3] != "220"):
20     print("220 reply not received from server.")
21

```

```

22 #send HELO command
23 command = "HELO Alice\r\n"
24 clientSocket.send(command.encode())
25 #get response and print
26 response1 = clientSocket.recv(1024).decode()
27 print("response1: ", response1)
28 if(response1[:3] != "250"):
29     print("250 reply not received from server.")
30
31 #send STARTTLS command
32 command = "STARTTLS\r\n"
33 clientSocket.send(command.encode())
34 #get response and print
35 response2 = clientSocket.recv(1024).decode()
36 print("response2: ", response2)
37 if(response2[:3] != "220"):
38     print("220 reply not received from server.")
39
40 #wrap socket for security
41 tlsSocket = ssl.wrap_socket(clientSocket)
42
43 #send AUTH LOGIN command
44 command = "AUTH LOGIN\r\n"
45 tlsSocket.send(command.encode())
46 #get response and print
47 response3 = tlsSocket.recv(1024).decode()
48 print("response3: ", response3)
49 if(response3[:3] != "334"):
50     print("334 reply not received from server.")
51
52 #send mail address
53 tlsSocket.send(base64.b64encode(mailAddress.encode()))
54 tlsSocket.send((" \r\n").encode())
55 #get response and print
56 response4 = tlsSocket.recv(1024).decode()
57 print("response4: ", response4)
58 if(response4[:3] != "334"):
59     print("334 reply not received from server.")
60
61 #send password
62 tlsSocket.send(base64.b64encode(password.encode()))
63 tlsSocket.send((" \r\n").encode())
64 #get response and print
65 response5 = tlsSocket.recv(1024).decode()
66 print("response5: ", response5)
67 if(response5[:3] != "235"):
68     print("235 reply not received from server.")
69
70 #send MAIL FROM command
71 command = "MAIL FROM:<" + mailAddress + ">\r\n"
72 tlsSocket.send(command.encode())

```

```

73 #get response and print
74 response6 = tlsSocket.recv(1024).decode()
75 print("response6: ", response6)
76 if(response6[:3] != "250"):
77     print("250 reply not received from server.")
78
79 #send RCPT TO command
80 command = "RCPT TO:<" + targetMail + ">\r\n"
81 tlsSocket.send(command.encode())
82 #get response and print
83 response7 = tlsSocket.recv(1024).decode()
84 print("response7: ", response7)
85 if(response7[:3] != "250"):
86     print("250 reply not received from server.")
87
88 #send DATA command
89 command = "DATA\r\n"
90 tlsSocket.send(command.encode())
91 #get response and print
92 response8 = tlsSocket.recv(1024).decode()
93 print("response8: ", response8)
94 if(response8[:3] != "354"):
95     print("354 reply not received from server.")
96
97 #send subject and message
98 subject = "Subject: " + subject + "\r\n\r\n"
99 message = "\r\n" + message + "\r\n.\r\n"
100 tlsSocket.send(subject.encode())
101 tlsSocket.send(message.encode())
102 #get response and print
103 response9 = tlsSocket.recv(1024).decode()
104 print("response9: ", response9)
105 if(response9[:3] != "250"):
106     print("250 reply not received from server.")
107
108 #send QUIT command
109 command = "QUIT\r\n"
110 tlsSocket.send(command.encode())
111 #get response and print
112 response10 = tlsSocket.recv(1024).decode()
113 print("response10: ", response10)
114 if(response10[:3] != "221"):
115     print("221 reply not received from server.")

```



### 3.3 Test

Figure 5: Sample Output

```
mailClient.py
from socket import *
import ssl
import base64

mailAddress = "eliffakgunn@gmail.com"
password = " "
targetMail = "eliffakgunn@gmail.com"
subject = "mail client test"
message = "hello"

#set mail server
mailServer = ("smtp.gmail.com", 587)
#create socket for TCP
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect(mailServer)
#get response and print
response = clientSocket.recv(1024).decode()
print("response: ", response)
if(response[:3] != "220"):
    print("220 reply not received from server.")

#send HELO command
command = "HELO Alice\r\n"
clientSocket.send(command.encode())
#get response and print
response1 = clientSocket.recv(1024).decode()
print("response1: ", response1)
if(response1[:3] != "250"):
    print("250 reply not received from server.")

#send STARTTLS command
command = "STARTTLS\r\n"
```

```
gtucpp@ubuntu: ~/Desktop/mobil/lab3
gtucpp@ubuntu:~/Desktop/mobil/lab3$ python3 mailClient.py
response:  220 smtp.gmail.com ESMTP q2sm31719044edv.93 - gsmt
response1: 250 smtp.gmail.com at your service
response2: 220 2.0.0 Ready to start TLS
response3: 334 VXNlcm5hbWU6
response4: 334 UGFzc3dvcmQ6
response5: 235 2.7.0 Accepted
response6: 250 2.1.0 OK q2sm31719044edv.93 - gsmt
response7: 250 2.1.5 OK q2sm31719044edv.93 - gsmt
response8: 354 Go ahead q2sm31719044edv.93 - gsmt
response9: 250 2.0.0 OK 1609317892 q2sm31719044edv.93 - gsmt
response10: 221 2.0.0 closing connection q2sm31719044edv.93 - gsmt
gtucpp@ubuntu:~/Desktop/mobil/lab3$
```

Figure 6: Inbox

