

CS 421 – COMPUTER NETWORKS

Programming Assignment 1

Utku Gökçen: CS 21703746

Hasan Kaan Çakmak: EEE 21703598



1. Introduction

In this assignment, a TCP socket programming is expected by programming the client socket. A server socket programmed is provided by the assignment itself. The server-client socket interaction can be summarized as socket creating, connection, object exchange and terminating the connection. For the sake of simplicity, object exchange part is divided into 3 parts which should be done sequentially in this assignment. General structure is given in Figure 1.

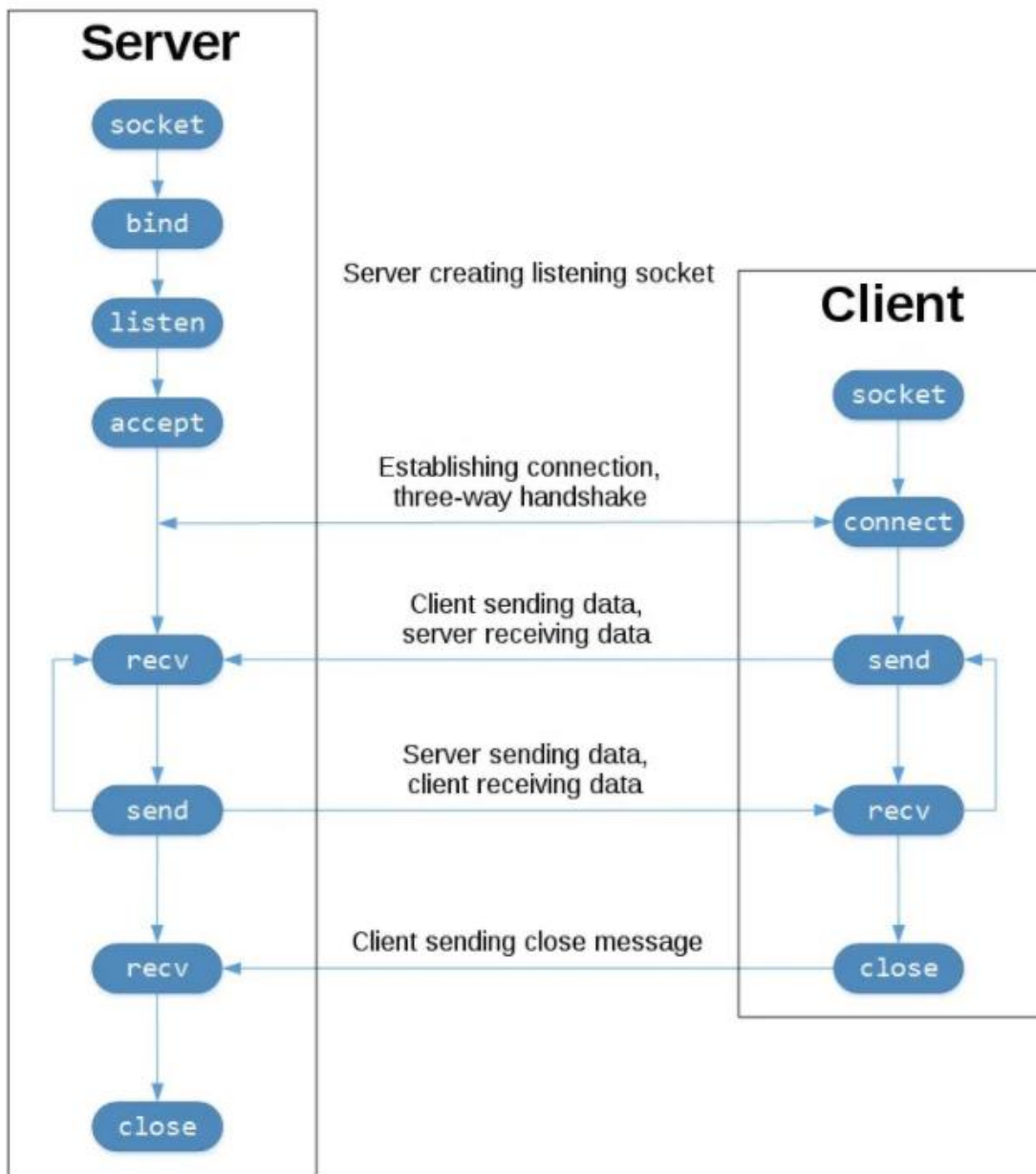


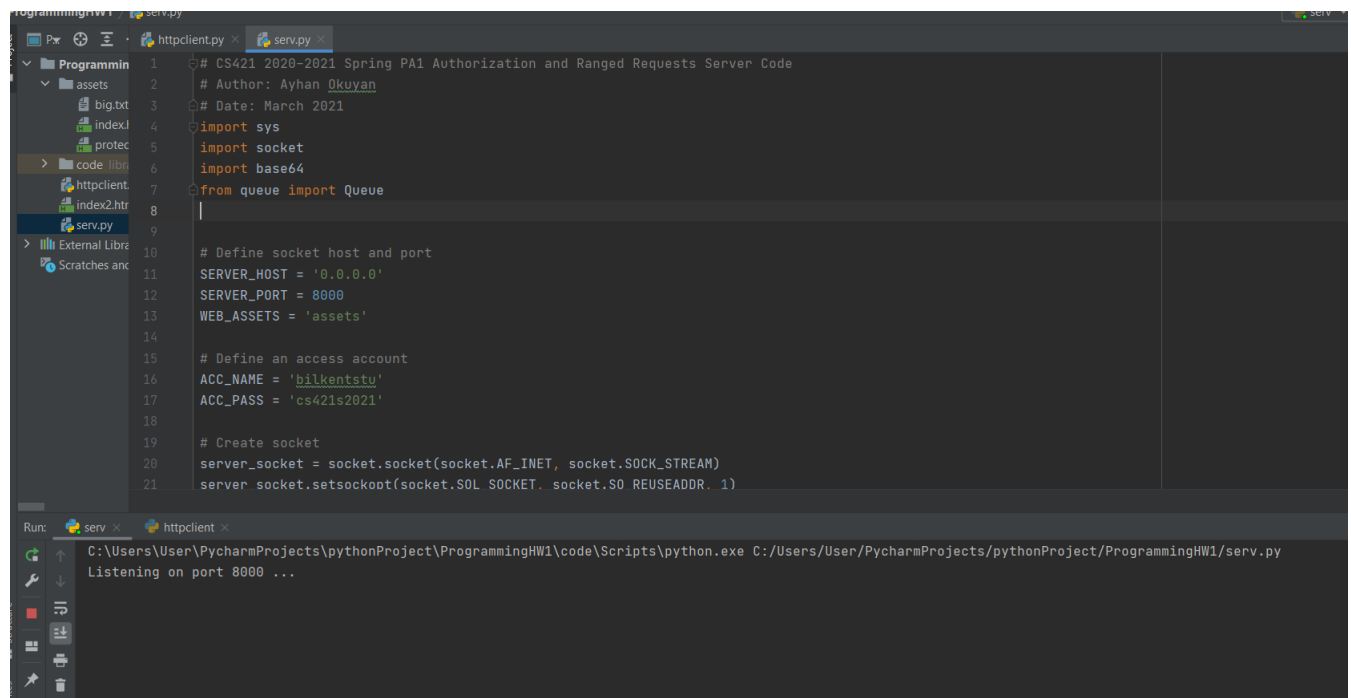
Figure 1: General Structure for Socket Programming

2. First HTTP Connection and TCP Socket Programming

In order to make first connection with the server, both the server socket have to listen and the client socket have to make correct requests to the correct places.

Activating Server Socket

As mentioned in the introduction, server socket program already provided by the assignment. In order to make connection with the client, it should be active in the time which client tries to reach to server socket. In order to do that server.py is activated before the client socket run.



The screenshot displays the PyCharm IDE interface. The left sidebar shows a project structure with folders like 'Programmin', 'assets', and 'code', and files like 'big.txt', 'index1', 'index2.htm', and 'serv.py'. The main editor window shows the code for 'serv.py'. The code includes comments about the course (CS421 2020-2021 Spring PA1) and the author (Ayhan Okuyan), followed by imports for 'sys', 'socket', 'base64', and 'Queue'. It then defines server parameters: 'SERVER_HOST' as '0.0.0.0', 'SERVER_PORT' as 8000, and 'WEB_ASSETS' as 'assets'. It also defines an access account with 'ACC_NAME' as 'bilkentstu' and 'ACC_PASS' as 'cs421s2021'. Finally, it creates a socket object and sets options for reuse and address reuse. The bottom panel shows the 'Run' output, indicating that the server is listening on port 8000.

```
1  # CS421 2020-2021 Spring PA1 Authorization and Ranged Requests Server Code
2  # Author: Ayhan Okuyan
3  # Date: March 2021
4  import sys
5  import socket
6  import base64
7  from queue import Queue
8
9
10 # Define socket host and port
11 SERVER_HOST = '0.0.0.0'
12 SERVER_PORT = 8000
13 WEB_ASSETS = 'assets'
14
15 # Define an access account
16 ACC_NAME = 'bilkentstu'
17 ACC_PASS = 'cs421s2021'
18
19 # Create socket
20 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21 server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Run: serv x httpclient x
C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\code\Scripts\python.exe C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\serv.py
Listening on port 8000 ...

Figure 2: Server Socket is Activated

As the figure 2 indicates, server socket is listening from port 8000 for any request at client side.

Creating Client Socket

To create socket at the client side, host name and the port is required. In order to find host name, `socket.gethostname` method is used and the port has been given as 8000 already in assignment. Then the socket is created and parameters are implemented as shown in Figure 3.

```
1 import socket
2
3
4 host = socket.gethostname()
5 port = 8000 # socket server port number
6
7 client_socket = socket.socket() # instantiate
8 client_socket.connect((host, port)) # connect to the server
9
```

Figure 3: Creating Client Socket

Get Request

In order to receive the webpage, the HTTP GET is required. HTTP GET has 2 parameters that should be defined. First parameter is the object asked from the server. Second parameter is the server name and port. In the first connection to the server socket, object is not defined and written as “/” while the host parameters used same with the parameters when creating client socket. Lastly, the HTTP GET message is encoded and send by the client socket as shown in the Figure 4.

```
host = socket.gethostname()
port = 8000 # socket server port number

client_socket = socket.socket() # instantiate
client_socket.connect((host, port)) # connect to the server

request = 'GET / HTTP/1.1\r\n|'
host = 'Host:'+host+':8000/ \r\n\r\n'

client_socket.sendall(request.encode() + host.encode())
```

Figure 4: HTTP GET request

Server's Response and Retrieved HTML File

After the HTTP Get request is initiated from the client's side server responses with the status code and the base HTML file.

```
C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\code\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/ProgrammingHW1/serv.py
Listening on port 8000 ...
['GET / HTTP/1.1', 'Host:DESKTOP-U3Q87L4:8000/ ']
```

Figure 5: Server Receive the HTTP GET request

As the Figure 5 indicates, the server receives the message send by the client and prints in the terminal. Then the server send the html file asked by the client with its socket.

```
C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\code\Scripts\python.exe C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\httpClient.py
HTTP/1.1 200 OK

<html>
<head>
  <title>Welcome Page</title>
  <link rel="shortcut icon" href="#" />
</head>
<body>
  <h1>Hello CS421 Students!</h1>
  <p>Welcome to the web page for your 1st Programming Assignment!</p>
  <p>Here's a link to the <a href="protected.html">webpage that you should access</a>
    using the Basic Authorization procedure.</p>
</body>
</html>
```

Figure 6: Retrieved HTML File

The client received the HTML File from the server as shown in the Figure 6. In order to receive and decode the HTML file send by the server, the code is updated as shown in the Figure 7. The received object assigned to a parameter called req. As a side note, the terminal does not print the received HTML file but it is externally printed to demonstrate received object from the server.

```
1  import socket
2
3
4  host = socket.gethostname()
5  port = 8000 # socket server port number
6
7  client_socket = socket.socket() # instantiate
8  client_socket.connect((host, port)) # connect to the server
9
10 request = 'GET / HTTP/1.1\r\n'
11 host = 'Host:' + host + ':8000/ \r\n\r\n'
12
13 client_socket.sendall(request.encode() + host.encode())
14
15 req = client_socket.recv(1024).decode()
16 print(req)
17
```

Figure 7: Client Socket Program Update to Receive HTML File

Saving and Parsing HTML File

After the HTML file is received at the client side, it should be saved as index2.html to the root directory. Firstly, an index2.html file is opened inside the client socket program. Then the assigned variable written into the index2.html and lastly it is saved.







	.idea	15.03.2021 03:25	Dosya klasörü	
	assets	15.03.2021 01:06	Dosya klasörü	
	code	15.03.2021 01:05	Dosya klasörü	
	httpclient	15.03.2021 02:01	Python File	1 KB
	index2	15.03.2021 02:29	Chrome HTML Do...	1 KB
	serv	15.03.2021 01:15	Python File	7 KB

Figure 8: Saved Index2.html File in the Root Directory

```
1  import socket
2
3
4  host = socket.gethostname()
5  port = 8000 # socket server port number
6
7  client_socket = socket.socket() # instantiate
8  client_socket.connect((host, port)) # connect to the server
9
10 request = 'GET / HTTP/1.1\r\n'
11 host = 'Host:'+host+':8000/ \r\n\r\n'
12
13 client_socket.sendall(request.encode() + host.encode())
14
15 req = client_socket.recv(1024).decode()
16 print(req)
17
18 # Writing the retrieved data to index2.html file
19 f = open('index2.html', 'w')
20
21 f.write(req)
22 f.close()
23
```

Figure 9: Client Socket Program to Save Index2.html

After the index2.html is saved in the root directory, the parsing came as the next process that should be done. In the assignment, the next HTML code which will be used as a new entity for the next part is hidden inside the received HTML file. To extract the entity, an algorithm is used to find an unknown parameter plus the “.html” string. First the location of the “.html” string is founded by the .find property of the python. Then the string is looked for “=” which starts from the founded

location of “.html” and goes backward. After the first “=” before the “.html” is founded, the HTML code is extracted and saved as a variable.

```
24 # The next entity is scraped from the index.html
25 html_string = ".html"
26
27 result = req.find(html_string)
28 start_string = result
29
30 for i in req[result:0:-1]:
31     if i == "=":
32         index = start_string
33         break
34     else:
35         start_string = start_string - 1
36
37 print(req[start_string+2:result+5])
38
```

Figure 10: Code to Extract the Next Entity

After the code shown in the Figure 10 is used, then the next entity is obtained and printed to demonstrate

```
C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\code\Scripts\python.exe C:\Users\User\PycharmProjects\pythonProject\ProgrammingHW1\httpClient.py
HTTP/1.1 200 OK

<html>
<head>
  <title>Welcome Page</title>
  <link rel="shortcut icon" href="#" />
</head>
<body>
  <h1>Hello CS421 Students!</h1>
  <p>Welcome to the web page for your 1st Programming Assignment!</p>
  <p>Here's a link to the <a href="protected.html">webpage that you should access</a>
    using the Basic Authorization procedure.</p>
</body>
</html>
protected.html

Process finished with exit code 0
|
```

Figure 11: Next Entity

3. HTTP Authentication

Get Request without Authentication

After the protected.html entity is extracted from the html obtained in the previous part, the protected.html is tried to be reached without Authentication. The server responded with 401 unauthorized message as shown in the Figure 12.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to Protected Site"
```

Figure 12: 401 Unauthorized Message

Basic Authorization Key

The attempt to access is failed due to not having authorization. In order to obtain the Basic Authorization, first the given username and password which “bilkentstu” and “cs421s2021” are assigned to a variable called credentials. Then it first encoded in ascii format and then base64 format. Lastly, base64 credentials are obtained and written into a string called “last” with the string “Authorization: Basic” which demonstrates the type of Authorization to the server. The code responsible for the process is demonstrated in the Figure 13.

```
# Obtaining base64 encoding
credentials = 'bilkentstu:cs421s2021'
credentials_bytes = credentials.encode('ascii')
credentials_encoded = base64.b64encode(credentials_bytes)
base64_credentials = credentials_encoded.decode('ascii')

authorization = 'Authorization: Basic '

last = authorization + base64_credentials + "\r\n\r\n"
```

Figure 13: Basic Authorization Code

Get Request with Basic Authentication

After the Basic Authorization key is prepared, the authorization key, get and host parts should be bind to each other. Firstly get part is created with the entity observed in the previous part and named as a “new_request”. Host and the port number is remained same. At the end the get, host and authorization key are bind, encoded and send by the socket at the client side as the Figure 14.

```
46 new_request = 'GET ' + entity + ' HTTP/1.1\r\n'
47
48 # Obtaining base64 encoding
49 credentials = 'bilkentstu:cs421s2021'
50 credentials_bytes = credentials.encode('ascii')
51 credentials_encoded = base64.b64encode(credentials_bytes)
52 base64_credentials = credentials_encoded.decode('ascii')
53
54 authorization = 'Authorization: Basic '
55
56 last = authorization + base64_credentials + "\r\n\r\n"
57
58
59
60
61 client_socket.sendall(new_request.encode() + host.encode() + last.encode())
```

Figure 14: Get Request with Basic Authorization

```
Listening on port 8000 ...
['GET / HTTP/1.1', 'Host: DESKTOP-U3Q87L4:8000/ ']
['GET /protected.html HTTP/1.1', 'Host: DESKTOP-U3Q87L4:8000/ ', '', 'Authorization: Basic Ymlsa2VudHN0dTpjczQyMlYMDIx']
```

Figure 15: Get Request at the Server Side

As the figure 15 indicates, the server receives the get message with the encoded authorization key. After the server receives the get message, it response with the “200 OK” message and the html file as shown in the figure 16.

```
HTTP/1.1 200 OK

<html>
<head>
  <title>Auth Page!</title>
</head>
<body>
  <h1>Authorization Protected Webpage!</h1>
  <p>This is an authorization protected webpage.
    By reaching this page, you have provided that you are knowledgeable
    about the Basic Authorization procedure that works under HTTPS protocol.
    The final part of the homework requires you to download
    <a href="big.txt">this text file</a> using range requests.
  </p>
</body>
</html>
```

Figure 16: Server's Response to Get Message with Basic Authorization

After the html file is obtained from server, two things should be done. The html file should be saved as “protected2.html” to the root directory and the new entity should be extracted from the html file.

```
60
61 client_socket.sendall(new_request.encode() + host.encode() + last.encode())
62
63 req1 = client_socket.recv(1024).decode()
64 print(req1)
65
66 # Writing the retrieved data to protected2.html file
67 f = open('protected2.html', 'w')
68
69 f.write(req1)
70 f.close()
71
72 # Extracting the next entity
73 href_string = "href="
74
75 result1 = req1.find(href_string)
76
77 next_entity = "/" + req1[result1+6:result1+13]
```

Figure 17: Code for Saving Html File and Parsing the New Entity

As the figure 17 indicates, the response obtained from the server assigned to the variable called “req1”. Then the “protected2.html” is opened and “req1” is saved inside the file as the Figure 18 indicates. After saving the html file, the next entity is founded by a similar approach with the previous parsing algorithm. Next entity founded as “big.txt”.








	.idea	17.03.2021 22:22	Dosya klasörü	
	assets	15.03.2021 01:06	Dosya klasörü	
	code	15.03.2021 01:05	Dosya klasörü	
	httpclient	17.03.2021 21:26	Python File	3 KB
	index2	17.03.2021 21:39	Chrome HTML Do...	1 KB
	protected2	17.03.2021 21:39	Chrome HTML Do...	1 KB
	serv	15.03.2021 01:15	Python File	7 KB

Figure 18: Protected2.html Saved in the Root Directory

4. HTTP Range Request

Head Request

The head requests are used for to obtain the length of the content that specified by the entity and also they indicates that whether the content supports range requests or not.

```

77 next_entity = "/" + req1[result1+6:result1+13]
78
79 last_request_for_txt = 'HEAD ' + next_entity + ' HTTP/1.1\r\n'
80 last_request_for_html = 'HEAD /index.html HTTP/1.1\r\n'
81
82
83 client_socket.sendall(last_request_for_txt.encode() + host.encode())
84
85 req2 = client_socket.recv(1024).decode()
86 print(req2)
87
88 client_socket.sendall(last_request_for_html.encode() + host.encode())
89
90 req3 = client_socket.recv(1024).decode()
91 print(req3)

```

Figure 19: Head Requests for “big.txt” and Index.html

As the Figure 19 indicates, the entity obtained from the previous part and the index.html are used to create head messages. After the head messages created, they are encoded and send by the socket. The responses printed in order to demonstrate the result as Figure 20 indicates.

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 6488394

HTTP/1.1 200 OK
Accept-Ranges: none
Content-Length: 362
```

Figure 20: The Length and Accept-Range status for big.txt and index.html

Range Request

Range requests are used for downloading the contents from web part by part. In the example given, the contents were downloaded from the text file named "big.txt" using the range values 10, 100, 1000, 10000, 15000.

```
req13 = ""
a = 15000
range1 = 0
for i in range(0, 6488394, a):
    range1 = i + a
    if range1 > 6488394:
        range1 = 6488394
    range_request = "Range: bytes=" + str(i) + "-" + str(range1) + "\r\n\r\n"

    client_socket.sendall(get_request.encode() + host.encode() + range_request.encode())

    req13 = req13 + client_socket.recv(20000).decode()
print(req13)
```

Figure 21: Algorithm for range request

```
ists/P6/ /HTTP/1.1 206 Partial Content
Content-Range: bytes 6488390-6488394/6488394
Content-Length: 4

/ "
Total runtime of the program is 28188.48496890068 secs
```

Figure 22: Execution time for downloading the content in range 10

```
h calves
Retrieved from "http //en wiktionary org/wiki/Wiktionary Frequency_lists/P6/ / / "
Total runtime of the program is 3561.376897573471 secs
```

Figure 23: Execution time for downloading the content in range 100

```
eanliness collective angela filth philippines timely herein ignoble canton
Retrieved from "http //en wiktionary org/wiki/Wiktionary Frequency_lists/P6/ / / "
Total runtime of the program is 355.25186705589294 secs
```

Figure 24: Execution time for downloading the content in range 1000

```
bitten aux toleration lucia scar bohemian vested affinity carlo sous peni
Retrieved from "http //en wiktionary org/wiki/Wiktionary Frequency_lists/P6/ / / "
Total runtime of the program is 37.2384238243103 secs
```

Figure 25: Execution time for downloading the content in range 10000

```
bitten aux toleration lucia scar bohemian vested affinity carlo sous peni
Retrieved from "http //en wiktionary org/wiki/Wiktionary Frequency_lists/P6/ / / "
Total runtime of the program is 24.520553588867188 secs
```

Figure 26: Execution time for downloading the content in range 15000

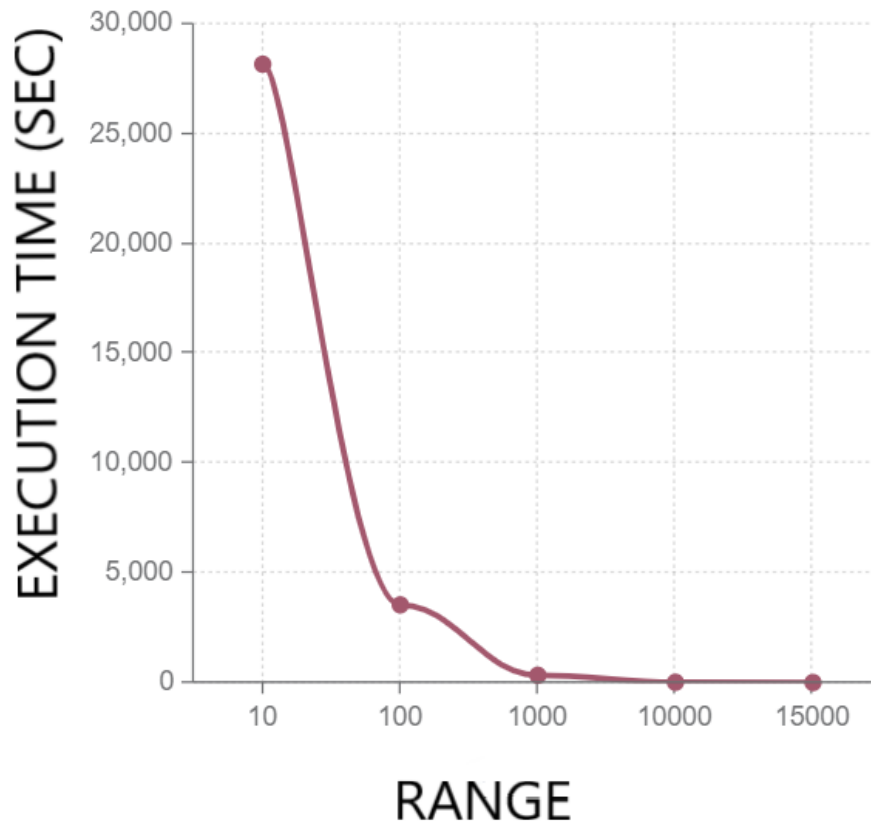


Figure 27: Execution time vs. range graph

As seen in the Figure 27, while the range increases the time of execution decreases. The reason for it is as the range decreases the number of requests increases which prolongs the processing time. When the single GET request is used, the range determined is a fixed number which is 1024 byte. As seen in the graph, single GET request is not the most efficient way to obtain data from web.

When implementing the algorithm for obtaining data from “big.txt”, we could not get the idea that the buffer size should get bigger when the range increases at the beginning. However, we could fix the problem by rearranging the buffer size.