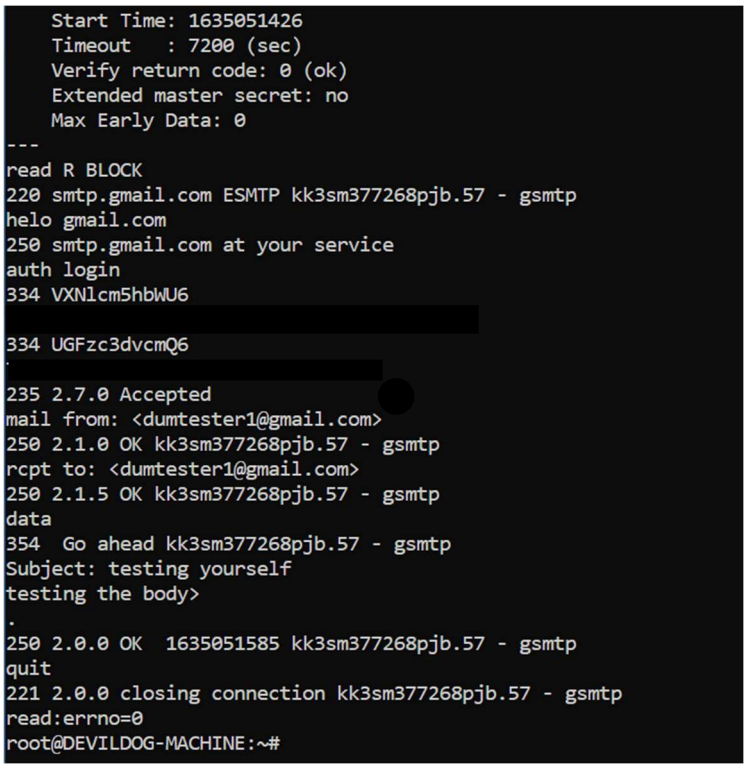
SER-321: Upper Layers
Terry Grant Simpson
10.23.2021

1. Sending emails and SMTP



What filter did you use to catch the traffic and explain why you chose that filter?
I used port 465 as the filter to catch the data because we explicit use this port in the command and this is google's smtp server port.

What is the standard port for SMTP and why do we use port 465 in the example?
The Internet Assigned Numbers Authority (IANA) considers the stand port for SMTP 29 even though many SMTP clients don't use this port due to residential ISP's blocking. Port 465 has been reassigned and the IANA; however, SMTP clients that have used this port prior to the reallocation still utilize the port for SMTP.

Explain each line used in the command line and what it does and why it is needed?

a. *echo –ne username | base64* & *echo –ne password | base64*
What it does: converts username and password to base64 without a new line and interprets backslash escapes
Use: used to copy and paste into the the command *auth login.*
b. *openssl s_client -crlf -ign _eof –connect smtp.gmail.com: 465*
What it does: Opens, test, and prints ssl connect to a specified client and port number. The *crlf* command translates a line feed from the terminal. The *ign_eof* command instructs the connect to to shutdown once the end of the file has been reached.
Use: To estblish a connection to google's smtp port 465.
c. *helo gmail.com*
What it does: *helo* is a command for SMTP servers to identify itself.
Use: Confirm connection to the google's SMTP server.
d. *auth login*
What it does: Allows user to input username and password in base64 to login to their account.
Use: Access gmail account.

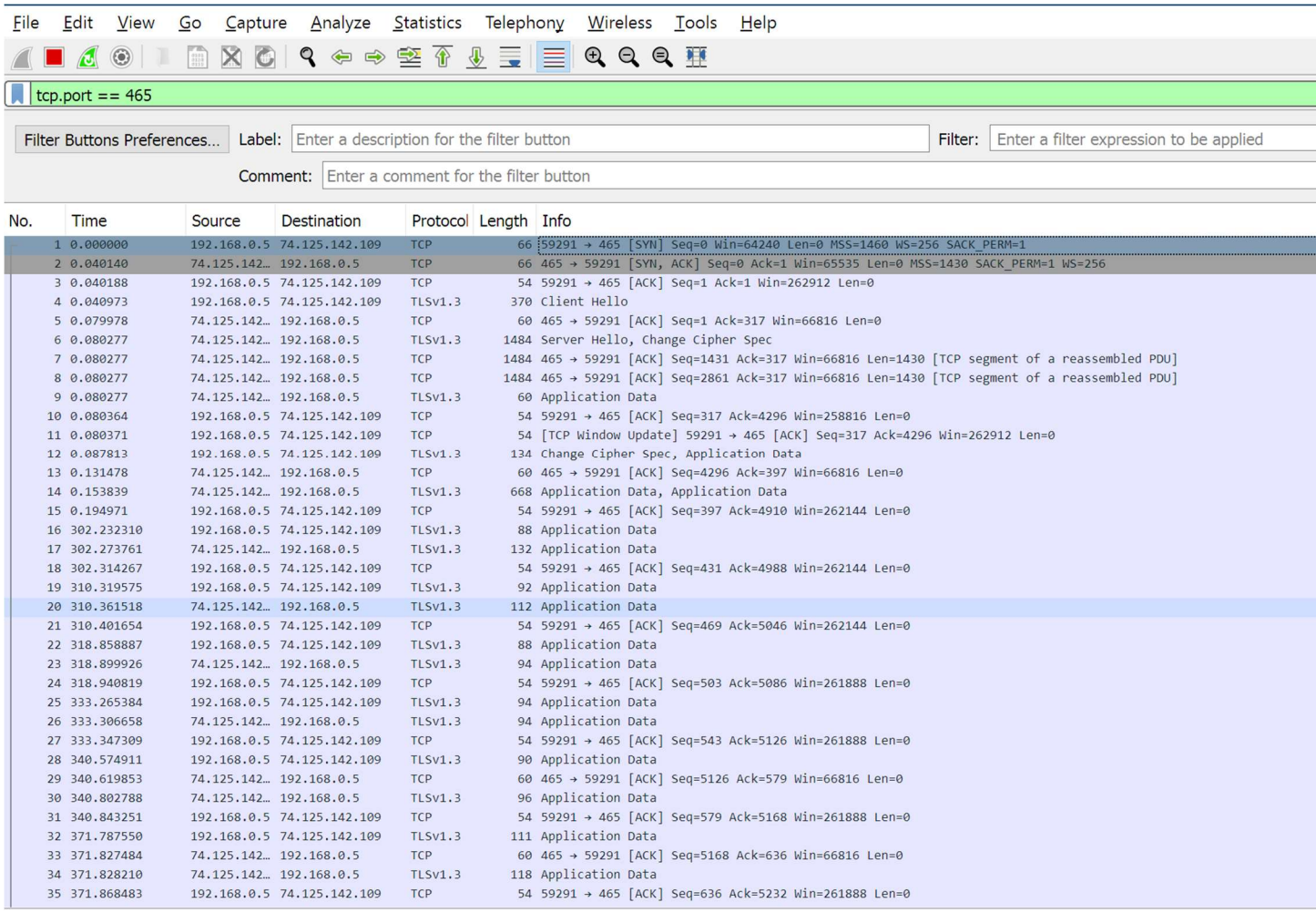
e. mail from:<email.xxx>
What it does: Allows user to input the "from" email address.
Use: Enter "from" email address.
f. rcpt to: <email.xxx>
What it does: Allows user to input the receipt email address.
Use: Enter receipents email address.
g. data
What it does: Allows user to input email content.
Use: Enter email content.
h. Subject:
What it does: Allows user to input the Subject text.
Use: Enter the subject's text.
i. (Period)
What it does: Identifier for the end of file (eof) and send.
Use: Send email.
j. quit
What it does: Shuts down the ssl connection.
Use: Shut down connection.

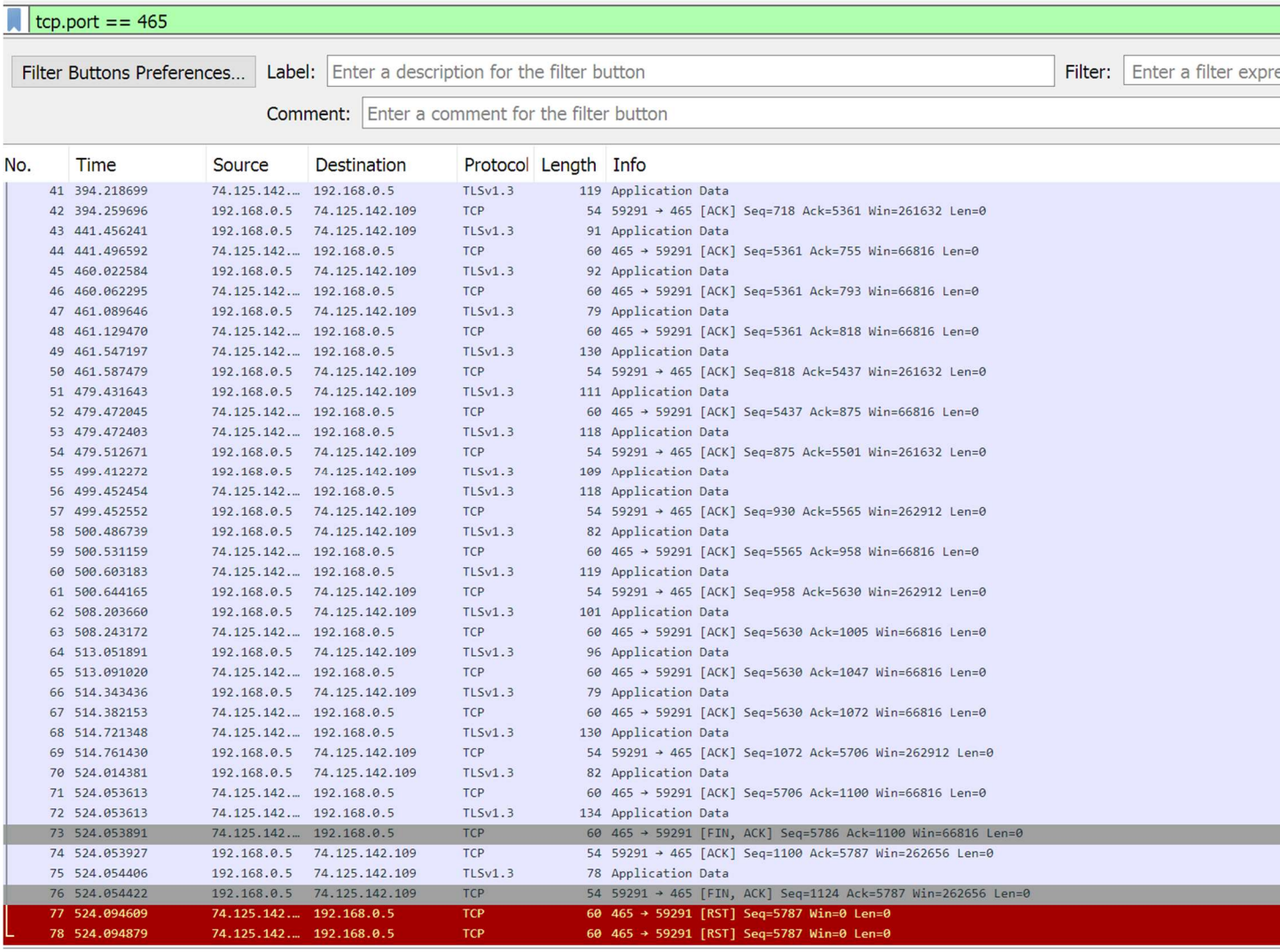
How much back and forth communication do you see for establishing the connection?
Total of 15 packets were used to establish connection exluding the login.

What is the port your local machine is using between sending the two emails when communicating with the SMTP server?
Port number 59291.

Explain who sends the first FIN flag and how the quitting process works.
Port 465 (gmail.com) sends the first FIN flag with an acknlowedge (ACK) bit. The receiver then acknowledges the that bit with an ACK and follows with the second finish (FIN) and acknowledge (ACK) bit. Once that is return then gmail.com resets and terminates the connect two times.

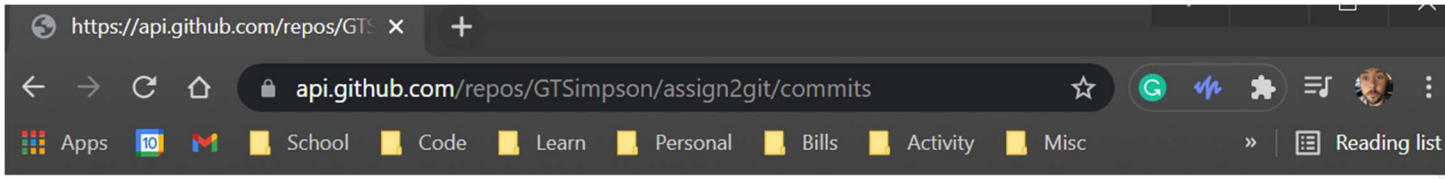
Add a screenshot of your Wireshark output and add it to your document.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 465
Filter Buttons Preferences...
Label: Enter a description for the filter button
Filter: Enter a filter expression to be applied
Comment: Enter a comment for the filter button
No. Time Source Destination Protocol Length Info
1 0.000000 192.168.0.5 74.125.142.109 TCP 66 59291 → 465 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2 0.040140 74.125.142… 192.168.0.5 TCP 66 465 → 59291 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1430 SACK_PERM=1 WS=256
3 0.040188 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=1 Ack=1 Win=262912 Len=0
4 0.040973 192.168.0.5 74.125.142.109 TLSv1.3 370 Client Hello
5 0.079978 74.125.142… 192.168.0.5 TCP 60 465 → 59291 [ACK] Seq=1 Ack=317 Win=66816 Len=0
6 0.080277 74.125.142… 192.168.0.5 TLSv1.3 1484 Server Hello, Change Cipher Spec
7 0.080277 74.125.142… 192.168.0.5 TCP 1484 465 → 59291 [ACK] Seq=1431 Ack=317 Win=66816 Len=1430 [TCP segment of a reassembled PDU]
8 0.080277 74.125.142… 192.168.0.5 TCP 1484 465 → 59291 [ACK] Seq=2861 Ack=317 Win=66816 Len=1430 [TCP segment of a reassembled PDU]
9 0.080277 74.125.142… 192.168.0.5 TLSv1.3 60 Application Data
10 0.080364 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=317 Ack=4296 Win=258816 Len=0
11 0.080371 192.168.0.5 74.125.142.109 TCP 54 [TCP Window Update] 59291 → 465 [ACK] Seq=317 Ack=4296 Win=262912 Len=0
12 0.087813 192.168.0.5 74.125.142.109 TLSv1.3 134 Change Cipher Spec, Application Data
13 0.131478 74.125.142… 192.168.0.5 TCP 60 465 → 59291 [ACK] Seq=4296 Ack=397 Win=66816 Len=0
14 0.153839 74.125.142… 192.168.0.5 TLSv1.3 668 Application Data, Application Data
15 0.194971 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=397 Ack=4910 Win=262144 Len=0
16 302.232310 192.168.0.5 74.125.142.109 TLSv1.3 88 Application Data
17 302.273761 74.125.142… 192.168.0.5 TLSv1.3 132 Application Data
18 302.314267 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=431 Ack=4988 Win=262144 Len=0
19 310.319575 192.168.0.5 74.125.142.109 TLSv1.3 92 Application Data
20 310.361518 74.125.142… 192.168.0.5 TLSv1.3 112 Application Data
21 310.401654 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=469 Ack=5046 Win=262144 Len=0
22 318.858887 192.168.0.5 74.125.142.109 TLSv1.3 88 Application Data
23 318.899926 74.125.142… 192.168.0.5 TLSv1.3 94 Application Data
24 318.940819 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=503 Ack=5086 Win=261888 Len=0
25 333.265384 192.168.0.5 74.125.142.109 TLSv1.3 94 Application Data
26 333.306658 74.125.142… 192.168.0.5 TLSv1.3 94 Application Data
27 333.347309 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=543 Ack=5126 Win=261888 Len=0
28 340.574911 192.168.0.5 74.125.142.109 TLSv1.3 90 Application Data
29 340.619853 74.125.142… 192.168.0.5 TCP 60 465 → 59291 [ACK] Seq=5126 Ack=579 Win=66816 Len=0
30 340.802788 74.125.142… 192.168.0.5 TLSv1.3 96 Application Data
31 340.843251 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=579 Ack=5168 Win=261888 Len=0
32 371.787550 192.168.0.5 74.125.142.109 TLSv1.3 111 Application Data
33 371.827484 74.125.142… 192.168.0.5 TCP 60 465 → 59291 [ACK] Seq=5168 Ack=636 Win=66816 Len=0
34 371.828210 74.125.142… 192.168.0.5 TLSv1.3 118 Application Data
35 371.868483 192.168.0.5 74.125.142.109 TCP 54 59291 → 465 [ACK] Seq=636 Ack=5232 Win=261888 Len=0

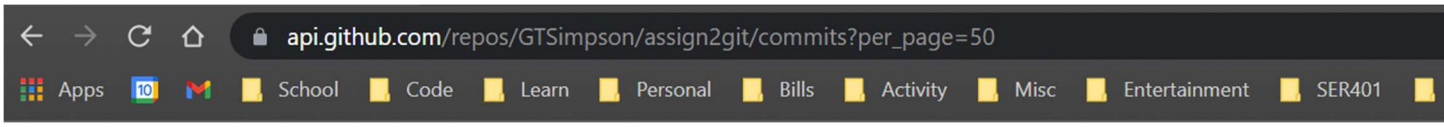


No.	Time	Source	Destination	Protocol	Length	Info
41	394.218699	74.125.142.…	192.168.0.5	TLSv1.3	119	Application Data
42	394.259696	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=718 Ack=5361 Win=261632 Len=0
43	441.456241	192.168.0.5	74.125.142.109	TLSv1.3	91	Application Data
44	441.496592	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5361 Ack=755 Win=66816 Len=0
45	460.022584	192.168.0.5	74.125.142.109	TLSv1.3	92	Application Data
46	460.062295	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5361 Ack=793 Win=66816 Len=0
47	461.089646	192.168.0.5	74.125.142.109	TLSv1.3	79	Application Data
48	461.129470	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5361 Ack=818 Win=66816 Len=0
49	461.547197	74.125.142.…	192.168.0.5	TLSv1.3	130	Application Data
50	461.587479	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=818 Ack=5437 Win=261632 Len=0
51	479.431643	192.168.0.5	74.125.142.109	TLSv1.3	111	Application Data
52	479.472045	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5437 Ack=875 Win=66816 Len=0
53	479.472403	74.125.142.…	192.168.0.5	TLSv1.3	118	Application Data
54	479.512671	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=875 Ack=5501 Win=261632 Len=0
55	499.412272	192.168.0.5	74.125.142.109	TLSv1.3	109	Application Data
56	499.452454	74.125.142.…	192.168.0.5	TLSv1.3	118	Application Data
57	499.452552	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=930 Ack=5565 Win=262912 Len=0
58	500.486739	192.168.0.5	74.125.142.109	TLSv1.3	82	Application Data
59	500.531159	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5565 Ack=958 Win=66816 Len=0
60	500.603183	74.125.142.…	192.168.0.5	TLSv1.3	119	Application Data
61	500.644165	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=958 Ack=5630 Win=262912 Len=0
62	508.203660	192.168.0.5	74.125.142.109	TLSv1.3	101	Application Data
63	508.243172	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5630 Ack=1005 Win=66816 Len=0
64	513.051891	192.168.0.5	74.125.142.109	TLSv1.3	96	Application Data
65	513.091020	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5630 Ack=1047 Win=66816 Len=0
66	514.343436	192.168.0.5	74.125.142.109	TLSv1.3	79	Application Data
67	514.382153	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5630 Ack=1072 Win=66816 Len=0
68	514.721348	74.125.142.…	192.168.0.5	TLSv1.3	130	Application Data
69	514.761430	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=1072 Ack=5706 Win=262912 Len=0
70	524.014381	192.168.0.5	74.125.142.109	TLSv1.3	82	Application Data
71	524.053613	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [ACK] Seq=5706 Ack=1100 Win=66816 Len=0
72	524.053613	74.125.142.…	192.168.0.5	TLSv1.3	134	Application Data
73	524.053891	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [FIN, ACK] Seq=5786 Ack=1100 Win=66816 Len=0
74	524.053927	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [ACK] Seq=1100 Ack=5787 Win=262656 Len=0
75	524.054406	192.168.0.5	74.125.142.109	TLSv1.3	78	Application Data
76	524.054422	192.168.0.5	74.125.142.109	TCP	54	59291 → 465 [FIN, ACK] Seq=1124 Ack=5787 Win=262656 Len=0
77	524.094609	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [RST] Seq=5787 Win=0 Len=0
78	524.094879	74.125.142.…	192.168.0.5	TCP	60	465 → 59291 [RST] Seq=5787 Win=0 Len=0

2. Understanding HTTP




```
[
  {
    "sha": "f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
    "node_id": "MDY6Q29tbWl0MzY5NzExMDQyOmY2MzQ3MTBhNzgyMmE2ZDhmZjgwZTNjMTViNGUzOWUyMmUzOTdlOGQ=",
    "commit": {
      "author": {
        "name": "unknown",
        "email": "gtsimpso@asu.edu",
        "date": "2021-05-22T04:24:45Z"
      },
      "committer": {
        "name": "unknown",
        "email": "gtsimpso@asu.edu",
        "date": "2021-05-22T04:24:45Z"
      },
      "message": "Add testbranch comment",
      "tree": {
        "sha": "39fe1ad10fa7fd6eb18bbcf28ab7144ad6dd9c47",
        "url": "https://api.github.com/repos/GTSimpson/assign2git/git/trees/39fe1ad10fa7fd6eb18bbcf28ab7144ad6dd9c47"
      },
      "url": "https://api.github.com/repos/GTSimpson/assign2git/git/commits/f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
      "comment_count": 0,
      "verification": {
        "verified": false,
        "reason": "unsigned",
        "signature": null,
        "payload": null
      }
    },
    "url": "https://api.github.com/repos/GTSimpson/assign2git/commits/f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
    "html_url": "https://github.com/GTSimpson/assign2git/commit/f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
    "comments_url": "https://api.github.com/repos/GTSimpson/assign2git/commits/f634710a7822a6d8ff80e3c15b4e39e22e397e8d/comment
    "author": {
      "login": "GTSimpson",
      "id": 47611276,
      "node_id": "MDQ6VXNlcjQ3NjExMjc2",
      "avatar_url": "https://avatars.githubusercontent.com/u/47611276?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/GTSimpson",
      "html_url": "https://github.com/GTSimpson",
      "followers_url": "https://api.github.com/users/GTSimpson/followers",
      "following_url": "https://api.github.com/users/GTSimpson/following{/other_user}",
      "gists_url": "https://api.github.com/users/GTSimpson/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/GTSimpson/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/GTSimpson/subscriptions",
      "organizations_url": "https://api.github.com/users/GTSimpson/orgs",
      "repos_url": "https://api.github.com/users/GTSimpson/repos",
      "events_url": "https://api.github.com/users/GTSimpson/events{/privacy}",
      "received_events_url": "https://api.github.com/users/GTSimpson/received_events",
      "type": "User",
      "site_admin": false
    },
    "committer": {
      "login": "GTSimpson",
      "id": 47611276,
```




```
[
  {
    "sha": "f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
    "node_id": "MDY6Q29tbWl0MzY5NzExMDQyOmY2MzQ3MTBhNzgyMmE2ZDhmZjgwZTNjMTViNGUzOWUyMmUzOTdlOGQ=",
    "commit": {
      "author": {
        "name": "unknown",
        "email": "gtsimpso@asu.edu",
        "date": "2021-05-22T04:24:45Z"
      },
      "committer": {
        "name": "unknown",
        "email": "gtsimpso@asu.edu",
        "date": "2021-05-22T04:24:45Z"
      },
      "message": "Add testbranch comment",
      "tree": {
        "sha": "39fe1ad10fa7fd6eb18bbcf28ab7144ad6dd9c47",
        "url": "https://api.github.com/repos/GTSimpson/assign2git/git/trees/39fe1ad10fa7fd6eb18bbcf28ab7144ad6dd9c47"
      },
      "url": "https://api.github.com/repos/GTSimpson/assign2git/git/commits/f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
      "comment_count": 0,
      "verification": {
        "verified": false,
        "reason": "unsigned",
        "signature": null,
        "payload": null
      }
    },
    "url": "https://api.github.com/repos/GTSimpson/assign2git/commits/f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
    "html_url": "https://github.com/GTSimpson/assign2git/commit/f634710a7822a6d8ff80e3c15b4e39e22e397e8d",
    "comments_url": "https://api.github.com/repos/GTSimpson/assign2git/commits/f634710a7822a6d8ff80e3c15b4e39e22e397e8d/comments",
    "author": {
      "login": "GTSimpson",
      "id": 47611276,
      "node_id": "MDQ6VXNlcjQ3NjExMjc2",
      "avatar_url": "https://avatars.githubusercontent.com/u/47611276?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/GTSimpson",
      "html_url": "https://github.com/GTSimpson",
      "followers_url": "https://api.github.com/users/GTSimpson/followers",
      "following_url": "https://api.github.com/users/GTSimpson/following{/other_user}",
      "gists_url": "https://api.github.com/users/GTSimpson/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/GTSimpson/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/GTSimpson/subscriptions",
      "organizations_url": "https://api.github.com/users/GTSimpson/orgs",
      "repos_url": "https://api.github.com/users/GTSimpson/repos",
      "events_url": "https://api.github.com/users/GTSimpson/events{/privacy}",
      "received_events_url": "https://api.github.com/users/GTSimpson/received_events",
      "type": "User",
      "site_admin": false
    },
    "committer": {
      "login": "GTSimpson",
      "id": 47611276,
```


1. Explain the specific API calls you used.

API call #1: *https://api.github.com/repos/GTSimpson/assign2git/commits*
This call was a request over the github api to return all the commits on the default branch. By default, this api call uses the default branch commits if not specified within the calls options.

API call #2: *https://api.github.com/repos/GTSimpson/assign2git/commits?per_page=50*
This call is similar to the call mentioned above; however, the option parameter *per_page* was changed from the default value of 30 to 50.

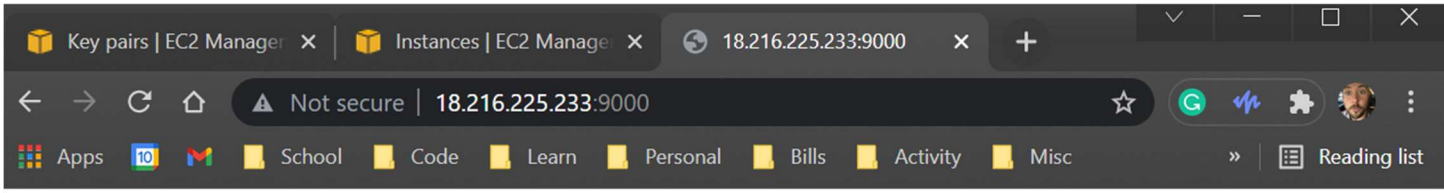
2. Explain the difference between stateless and a state-full communication

A stateless protocol does not require the server to save any state when a request is made by the client; therefore, each request is independent. A state-full protocol requires the server to retain session data.

3. Setup your second system and run Server on it

3.1 Getting the sample code onto your systems

3.2 Running a simple Java WebServer



You can make the following GET requests

- /file/sample.html -- returns the content of the file sample.html
- /json -- returns a json of the /random request
- /random -- returns index.html

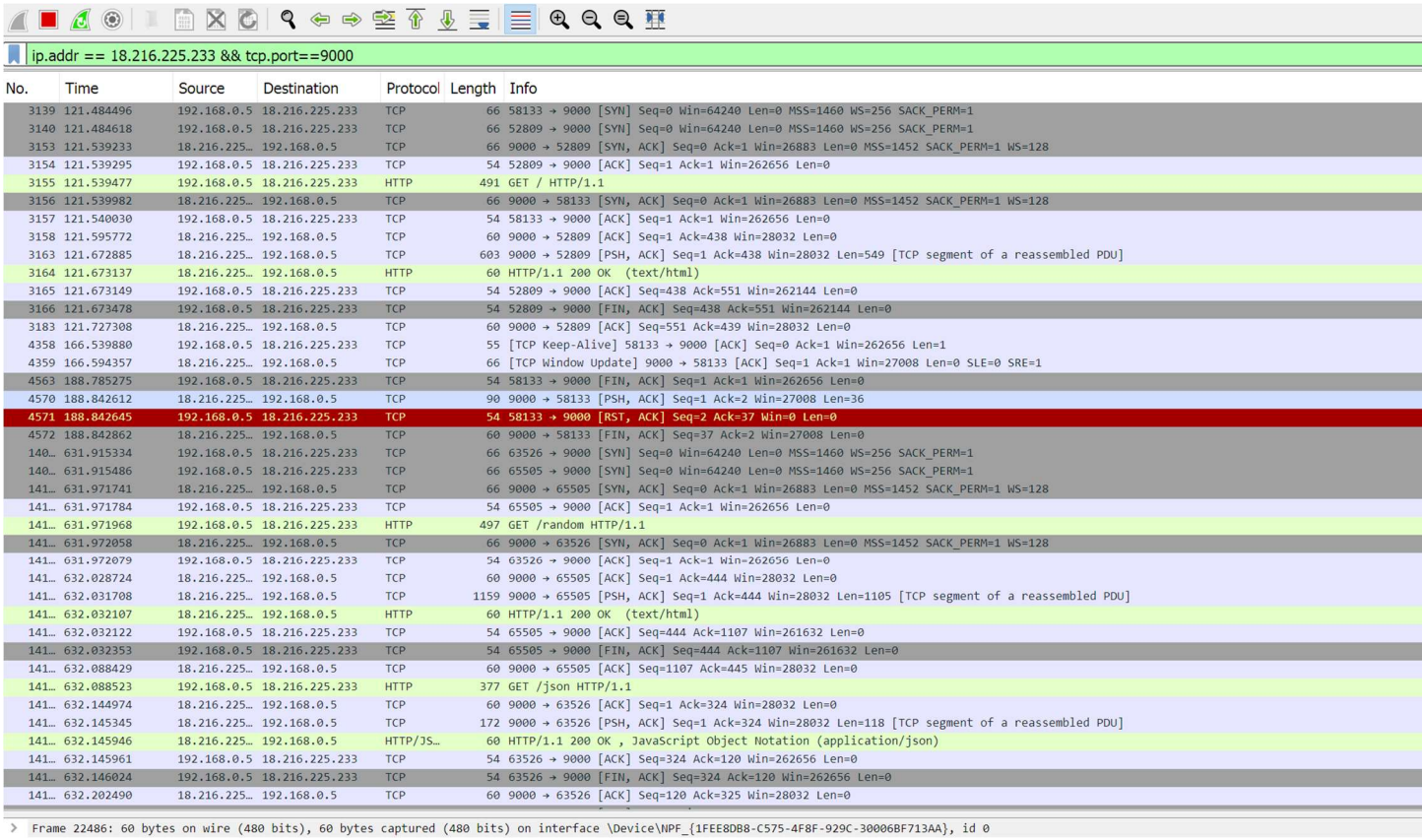
File Structure in www (you can use /file/www/FILENAME):

- index.html
- root.html

```
gradle FunWebServer


The FunWebServer does a little more than the SimpleWebServer. Check out what it does :-)[ec2-user@ip-172-31-46-14
 WebServer]$ gradle FunWebServer
Received: null
FINISHED PARSING HEADER
> Task :FunWebServer
<==========----> 75% EXECUTING [2m 13s]
> :FunWebServer
Received: Connection: keep-alive
Received: Upgrade-Insecure-Requests: 1
Received: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.
0.4606.81 Safari/537.36
Received: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3;q=0.9
Received: Accept-Encoding: gzip, deflate
Received: Accept-Language: en-US,en;q=0.9
Received:
FINISHED PARSING HEADER

<==========----> 75% EXECUTING [1m 47s]
> :FunWebServer
```



3.3 Analyze what happens

1. What filter did you use? Explain why you chose that filter.

I chose to filter the data with ip.add and tcp.port. By filtering with the ip address I can get all inbound and outbound traffic through that address. Also, I chose to filter the ip address with the given port value of 9000. Even though I could have gotten away with filtering by the port address, other ip addresses could have been using port 9000.

2. What happens when you are on /random and click the "Random" button compared to the browser refresh (you can also use the command line output that the WebServer generates to answer this)?

When the “Random” button is selected it gets and return a json of the /random request compared to selecting the “Refresh” button it gets and returns the index.html then gets and returns a json of the /random request.

```
GET /random HTTP/1.1
9000 → 58305 [ACK] Seq=1 Ack=444 Win=28032 Len=0
9000 → 58305 [PSH, ACK] Seq=1 Ack=444 Win=28032 Len=1105 [TCP segment of a
HTTP/1.1 200 OK  (text/html)
58305 → 9000 [ACK] Seq=444 Ack=1107 Win=261632 Len=0
58305 → 9000 [FIN, ACK] Seq=444 Ack=1107 Win=261632 Len=0
9000 → 58305 [ACK] Seq=1107 Ack=445 Win=28032 Len=0
GET /json HTTP/1.1
9000 → 61915 [ACK] Seq=1 Ack=324 Win=28032 Len=0
9000 → 61915 [PSH, ACK] Seq=1 Ack=324 Win=28032 Len=120 [TCP segment of a
HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
```

Figure 1: Refresh Button

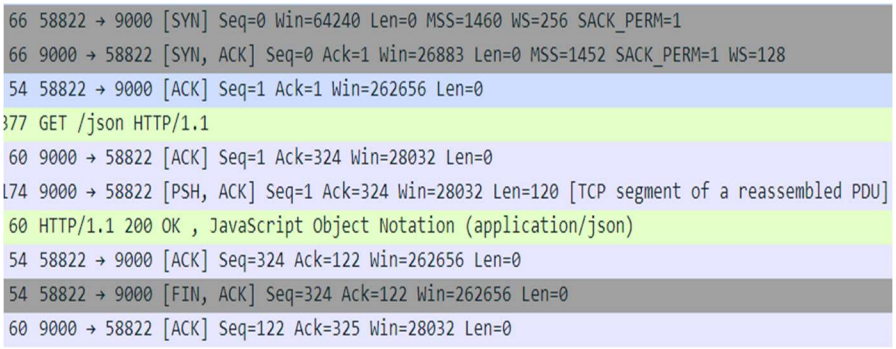


Figure 2: Random Button

3. What kinds of response codes are you able to get through different requests to your server?
200 OK (text/html)
200 OK , JavaScript Object Notation (application/json)
400 Bad Request (text/html)
404 Not Found (text/html)

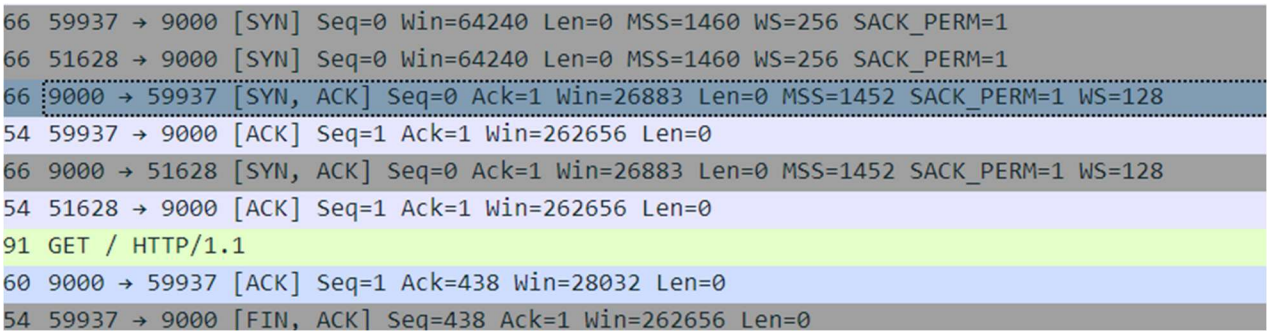
4. Explain the response codes you get and why you get them?
200 OK (text/html)
What it is: Request is valid and successful.
Why: Request made was valid and successful when entering a valid command

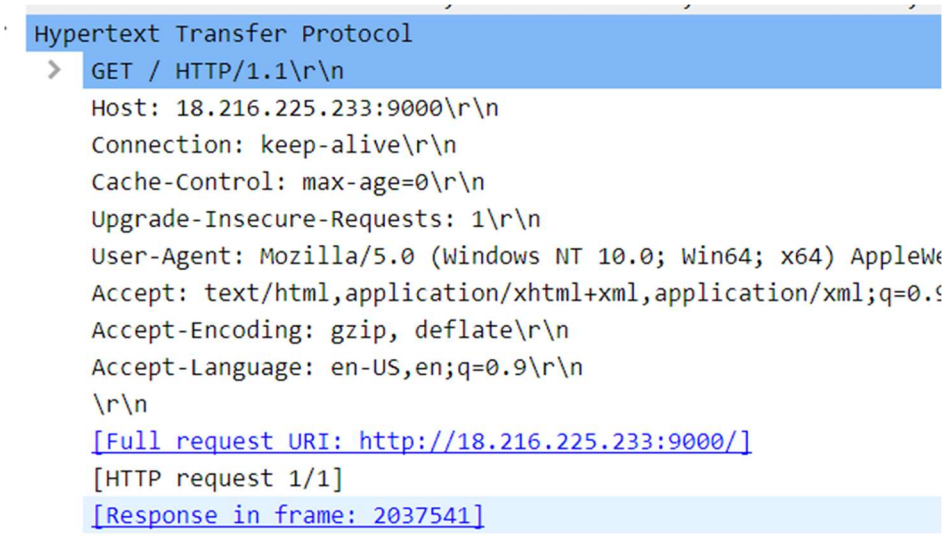
200 OK , JavaScript Object Notation (application/json)
What it is: Request is valid and successful while use json notation
Why: Request made was valid and successful when via json

400 Bad Request (text/html)
What it is: Invalid request due to invalid syntax
Why: Request was not syntatically valid therefore the server response was bad.

404 Not Found (text/html)
What it is: Server could not find the resouce.
Why: Request with proper syntax was made; however, the resource could not be found.

5. When you do an ipOfSecondMachine:9000 take a look what Wireshark generates as a server response. Are you able to find the data that the server sends back to you?





6. Based on the above question explain why HTTPs is now more common than HTTP.
HTTPs is more common because it uses encrytion while sending data over the line.

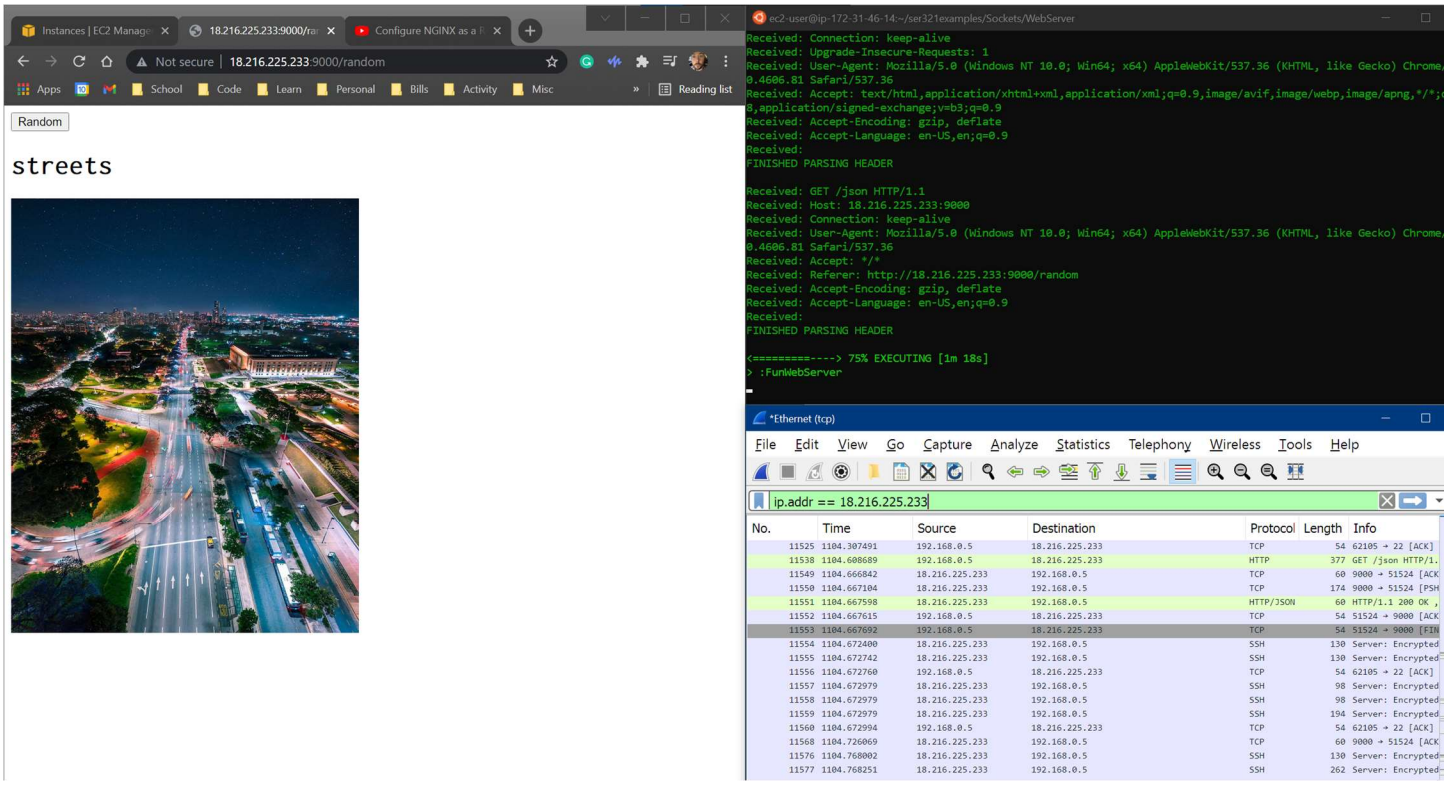
7. What port does the server listen to for HTTP requests in our case and is that the most common port for HTTP?
In our case we are using port 9000 for http requests, which is not the default or common port (80) for HTTP.

8. What local port is used when sending different requests to the WebServer? How does it differ to the traffic to your SMTP server from part 1?
In part 1 the local port being used was port number 59291, compare the local port 65068 that's being used to make the requests to the WebServer.

3.4 Setting up a "real" Web server

1. Check your traffic to your Webserver now. What port is the traffic going to now? Is it the same as previously used or is it and should it be different?
The port number has not from 9000, this is due to the nginx directive "proxy_pass". This directive is reverse proxy that protects the server side by using port 9000 as its proxy.

2. Is it still HTTP or is it now HTTPs? Why?
HTTPS requires an SSL or TLS certificate which we do not have. If we did have one, we would still need to upload the certificate. With that said, the server still implements a SSH protocol which is some protection compare to none.



3.5 Setting up HTTPs Extra Credit

3.6 Some programming on your WebServer

3.6.1 Multiply

The error handling I included was valid integer values, valid arguments names, and the correct number of arguments. I chose to use error code 400 Bad Request because all of the error handling is based on proper syntax. The error message display states the requirements for multiply and provides an example.

3.6.2 GitHub

3.6.3 Make your own requests

3.6.4 Webserver for everyone