Section C

Commands:

sudo tcpdump -i en0 -n port 1080 -w http_1080.pcap
http://www.sbunetsyslabs.com:1080/

sudo tcpdump -i en0 -n port 1081 -w tcp_1081.pcap
https://www.sbunetsyslabs.com:1081/

sudo tcpdump -i en0 -n port 1082 -w tcp_1082.pcap
https://www.sbunetsyslabs.com:1082/

Overview:

The task here is to analyze the congestion control in TCP packets.

Code:

1. Class **Packet** is used to read the packet bytes and ascertain the packet attributes. Function **parse** is used to initialize the packet attributes.
2. Class **initializePorts** is used to assign ports and track the flow of each packet.
3. Function **reassemble** is used to reassemble unique HTTP requests/responses and the unique TCP tuple for all TCP segments
4. Function **identifyHTTP** is used to identify HTTP versions in each case.
5. Function **results** finds the fastest and slowest HTTP version, based on load times. It also finds the max and min number of packets and raw bytes sent.

Output:

1. **Reassemble:**

```
Flow 1:
Request: b'GET / HTTP/1.1\r\nHost: www.sbunetsyslabs.com:1080\r\nConnection:
keep-alive\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0
(Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/106.0.0.0 Safari/537.36\r\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
p,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\nAccept-En
coding: gzip, deflate\r\nAccept-Language: en-US,en;q=0.9\r\n\r\n'
Source Port, Destination Port, Sequence Number, Ack Number
65356        1080        146711926   3797638355
```

Source Port, Destination Port, Sequence Number, Ack Number
1080 65356 3797638355 146712373

Flow 2:
Request: b'GET /img/apple.jpeg HTTP/1.1\r\nHost: www.sbunetsyslabs.com:1080\r\nConnection: keep-alive\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36\r\nAccept: image/avif,image/webp,image/apng,image/svg+xml,image/,*/*;q=0.8\r\nReferer: http://www.sbunetsyslabs.com:1080/\r\nAccept-Encoding: gzip, deflate\r\nAccept-Language: en-US,en;q=0.9\r\n\r\n'*
Source Port, Destination Port, Sequence Number, Ack Number
65357 1080 2335708790 2395272436

Source Port, Destination Port, Sequence Number, Ack Number
1080 65357 2395272436 2335709195
1080 65357 2395273806 2335709195
1080 65357 2395275176 2335709195
1080 65357 2395276546 2335709195
---

2. **Identify HTTP:**

At Port 8080:

Flow 1 -> From: 65356 to: 1080
Flow 2 -> From: 65357 to: 1080
Flow 3 -> From: 65359 to: 1080
Flow 4 -> From: 65360 to: 1080
Flow 5 -> From: 65361 to: 1080
Flow 6 -> From: 65362 to: 1080
Flow 7 -> From: 65363 to: 1080
Flow 8 -> From: 65364 to: 1080
Flow 9 -> From: 65365 to: 1080
Flow 10 -> From: 65366 to: 1080
Flow 11 -> From: 65367 to: 1080
Flow 12 -> From: 65368 to: 1080
Flow 13 -> From: 65369 to: 1080
Flow 14 -> From: 65370 to: 1080
Flow 15 -> From: 65371 to: 1080
Flow 16 -> From: 65372 to: 1080
Flow 17 -> From: 65373 to: 1080
Flow 18 -> From: 65374 to: 1080

There are 18 flows in 1080, which points to HTTP 1.0.
It is a non-persistent connection, where every object is fetched via a new connection.

At Port 8081:

Flow 1 -> From: 65376 to: 1081
Flow 2 -> From: 65377 to: 1081
Flow 3 -> From: 65378 to: 1081
Flow 4 -> From: 65379 to: 1081
Flow 5 -> From: 65380 to: 1081
Flow 6 -> From: 65381 to: 1081

There are 6 flows in 1081, which points to an establishment of a parallel connection.
Thus, it is follows HTTP 1.1.

At Port 8082:

Flow 1 -> From: 65382 to: 1082
Flow 2 -> From: 65383 to: 1082

There are only 2 flows in 1082 through which all objects are fetched.
This indicates a persistent TCP connection, which might have been disrupted in between due to page refresh.
This is clearly an HTTP 2.0 connection.

3. Results:

HTTP 1.0
Load Time: 0.3918120861053467
Packets Sent: 1633
Number of raw bytes: 2246790

HTTP 1.1
Load Time: 0.22647905349731445
Packets Sent: 1605
Number of raw bytes: 2260179

HTTP 2.0
Load Time: 0.23560595512390137
Packets Sent: 1611
Number of raw bytes: 2286037

Fastest Load Time: HTTP 1.1 -> 0.22647905349731445 seconds
Slowest Load Time: HTTP 1.0 -> 0.3918120861053467 seconds

Max Packets: HTTP 1.0 -> 1633
Min Packets: HTTP 1.1 -> 1605

Max raw bytes: HTTP 2.0 -> 2286037
Min raw bytes: HTTP 1.0 -> 2246790

Explanation:

1. **Reassemble:**

I have entered a snapshot of the entire output. The entire output is stored in *output.txt.*

We take the GET request from the sender to the receiver and print its TCP Tuple. Then, we print all the HTTP responses of this GET request and all its corresponding tuples whenever payload > 0. There might be multiple responses in the case where data is divided over multiple HTTP responses.

2. **Identify HTTP :**

For 1st case, we see 18 flows (as in *output.txt*). Every flow has singular GET requests and responses. This symbolizes that it is a non-persistent connection and we can declare this as HTTP 1.0.

For the second case, we see over 6 flows. Except for the first flow, we see that the remaining flows have approximately equal timestamps. This indicates that 6 parallel connections have been established. So we declare this as HTTP 1.1.

For the third case, we see only 2 flows. Here, I had to refresh once while creating the pcap file due to network error. So I assume that in the ideal case, this will only have one flow. This indicates that a persistent connection was established, followed by pipelining, and we can declare it as HTTP 2.0.

3. **Results:**

a. **Load Time**

We observe that HTTP 1.1 has the lowest load time (fastest) as it has established parallel connections.

We observe that HTTP 1.0 is the slowest as it is non-persistent.

b. **Number of Bytes**

We observe that HTTP 1.0 sends max number of bytes as it is non-persistent and adds extra headers to maintain state for every request.

We observe that HTTP 1.1 sends min number of bytes as it is establishing parallel connections.

c. Number of Raw Bytes

We observe that HTTP 1.0 has min raw bytes as the extra state packets are connection packets, which are smaller.

We observe that HTTP 2.0 has max raw bytes, which is probably due to slicing and mutliplexing, whose state tracking adds more raw bytes.