Part C HTTP Analysis task

- tcpdump command:
  sudo tcpdump -w http_1080.pcap port 1080 -i en0
  sudo tcpdump -w tcp_1081.pcap port 1081 -i en0
  sudo tcpdump -w tcp_1082.pcap port 1082 -i en0
- How to run:
  Place all the pcap files under the same folder, then run analysis_pcap_http.py directly.

1. **Reassemble each unique HTTP Request/Response for http_1080.pcap (the other two are encrypted, so you will not be able to reassemble easily). The output of this part should be the Packet type (request or response) and the unique TCP tuple for all the TCP segments that contain data for that request.**

   We use ports to distinguish each TCP flow.

   For finding HTTP flows, firstly, record all the acknowledge-numbers from the server packets whose tcp_payload > 0, then we use these ACKs to find the related SEQs from the client packets whose tcp_payload > 0. Then we can confirm the HTTP Request/Response.

   There are 17 distinct HTTP flows and 17 distinct TCP flows:

```
========== port: 1080 ==========
Request: (60737, 1080, 3240488997, 1882215550), Response: (1080, 60737, 1882215550, 3240489447)
Request: (60738, 1080, 203828184, 2444267609), Response: (1080, 60738, 2444267609, 203828528)
Request: (60739, 1080, 3838271571, 736447236), Response: (1080, 60739, 736447236, 3838271979)
Request: (60742, 1080, 2463360416, 3522816713), Response: (1080, 60742, 3522816713, 2463360825)
Request: (60740, 1080, 3845791368, 1445650331), Response: (1080, 60740, 1445650331, 3845791777)
Request: (60741, 1080, 1202297577, 2930079685), Response: (1080, 60741, 2930079685, 1202297990)
Request: (60743, 1080, 1073740354, 2241958270), Response: (1080, 60743, 2241958270, 1073740773)
Request: (60745, 1080, 3201096243, 3401999065), Response: (1080, 60745, 3401999065, 3201096660)
Request: (60744, 1080, 3018207384, 3367619894), Response: (1080, 60744, 3367619894, 3018207798)
Request: (60747, 1080, 2289159591, 3413082523), Response: (1080, 60747, 3413082523, 2289160003)
Request: (60746, 1080, 97307021, 4075950170), Response: (1080, 60746, 4075950170, 97307435)
Request: (60748, 1080, 803547122, 2538562470), Response: (1080, 60748, 2538562470, 803547534)
Request: (60749, 1080, 3427253014, 2457400714), Response: (1080, 60749, 2457400714, 3427253427)
Request: (60751, 1080, 291222055, 2044625750), Response: (1080, 60751, 2044625750, 291222469)
Request: (60750, 1080, 2754317854, 1031631084), Response: (1080, 60750, 1031631084, 2754318266)
Request: (60752, 1080, 3384702523, 3197354076), Response: (1080, 60752, 3197354076, 3384702932)
Request: (60753, 1080, 4272686340, 4131960704), Response: (1080, 60753, 4131960704, 4272686749)
data flows: 17
tcp connections: 17
```

2. **Identify which HTTP protocol is being used for each PCAP file. Note that two of the sites are encrypted so you must use your knowledge of HTTP and TCP to programmatically solve this question. Include the logic behind your code in the write-up.**

1. Port 1080 is on HTTP 1.0.

Because each HTTP connection possesses a different TCP connection, which means the TCP connections are not to be resued, and it will close every time the HTTP request completes.

2. Port 1081 is on HTTP 1.1.

Calculate the TCP connections and number of data flows using the similar method as 1, we can see there are 23 data flows and 6 TCP connections, which means it allows multiple HTTP requests to be sent over a single TCP.
The data flows can be described as follows:

```
========== port: 1081 ==========
For TCP connection on (54087, 1081)
Request: (54087, 1081, 1904956759, 1068247562), Response: (1081, 54087, 1068247562, 1904957276)
Request: (54087, 1081, 1904957402, 1068252048), Response: (1081, 54087, 1068252048, 1904957402)
Request: (54087, 1081, 1904958107, 1068253213), Response: (1081, 54087, 1068253213, 1904958754)
Request: (54087, 1081, 1904958754, 1068257611), Response: (1081, 54087, 1068257611, 1904959338)
Request: (54087, 1081, 1904959338, 1068258133), Response: (1081, 54087, 1068258133, 1904959986)
Request: (54087, 1081, 1904959986, 1068267343), Response: (1081, 54087, 1068267343, 1904960637)
Request: (54087, 1081, 1904960637, 1068273870), Response: (1081, 54087, 1068273870, 1904961289)
For TCP connection on (54088, 1081)
Request: (54088, 1081, 3247006256, 3550291474), Response: (1081, 54088, 3550291474, 3247006787)
Request: (54088, 1081, 3247006787, 3550291626), Response: (1081, 54088, 3550291626, 3247007486)
Request: (54088, 1081, 3247007486, 3550295869), Response: (1081, 54088, 3550295869, 3247008137)
Request: (54088, 1081, 3247008137, 3550330911), Response: (1081, 54088, 3550330911, 3247008785)
For TCP connection on (54089, 1081)
Request: (54089, 1081, 1064196814, 3918126706), Response: (1081, 54089, 3918126706, 1064197345)
Request: (54089, 1081, 1064197345, 3918126858), Response: (1081, 54089, 3918126858, 1064198048)
Request: (54089, 1081, 1064198048, 3918178345), Response: (1081, 54089, 3918178345, 1064198696)
For TCP connection on (54090, 1081)
Request: (54090, 1081, 3643118585, 3521694500), Response: (1081, 54090, 3521694500, 3643119116)
Request: (54090, 1081, 3643119167, 3521694652), Response: (1081, 54090, 3521694652, 3643119823)
Request: (54090, 1081, 3643119823, 3521721520), Response: (1081, 54090, 3521721520, 3643120474)
For TCP connection on (54091, 1081)
Request: (54091, 1081, 2861193064, 1572339842), Response: (1081, 54091, 1572339842, 2861193595)
Request: (54091, 1081, 2861193595, 1572339994), Response: (1081, 54091, 1572339994, 2861194304)
Request: (54091, 1081, 2861194304, 1572347880), Response: (1081, 54091, 1572347880, 2861194957)
Request: (54091, 1081, 2861194957, 1572357565), Response: (1081, 54091, 1572357565, 2861195610)
For TCP connection on (54092, 1081)
Request: (54092, 1081, 3506267150, 1316681900), Response: (1081, 54092, 1316681900, 3506267681)
Request: (54092, 1081, 3506267681, 1316682052), Response: (1081, 54092, 1316682052, 3506268385)
data flows: 23
tcp connections: 6
```

3. Port 1082 is on HTTP 2.0

There are 5 data flows and 1 TCP connection, which indicates it enables full request and response multiplexing, all communication is performed over a single TCP connection that can carry any number of streams.

```
========== port: 1082 ==========
For TCP connection on (54396, 1082)
Request: (54396, 1082, 940548458, 2532748960), Response: (1082, 54396, 2532748960, 940548975)
Request: (54396, 1082, 940549200, 2532753440), Response: (1082, 54396, 2532753440, 940549101)
Request: (54396, 1082, 940549688, 2532753771), Response: (1082, 54396, 2532753771, 940549688)
Request: (54396, 1082, 940551097, 2532754587), Response: (1082, 54396, 2532754587, 940549848)
Request: (54396, 1082, 940551210, 2532767871), Response: (1082, 54396, 2532767871, 940551167)
data flows: 5
tcp connections: 1
```

3. **Finally, after you've labeled the PCAPs with their appropriate versions of HTTP, answer the following: Which version of the protocol did the site load the fastest under? The Slowest? Which sent the most number of packets and raw bytes? Which protocol sent the least? Report your results and write a brief explanation for your observations.**

```
========== summary ==========
port: 1080
interval: 0.944 s
packet_total: 2240
payload_total: 2295044 bytes

port: 1081
interval: 0.602 s
packet_total: 2128
payload_total: 2306895 bytes

port: 1082
interval: 0.548 s
packet_total: 2010
payload_total: 2310272 bytes
```

HTTP 2.0 loads faster because it reduces latency by enabling full request and response multiplexing, minimizes protocol overhead via efficient compression of HTTP header fields, and adds support for request prioritization and server push.
HTTP 1.0 is the slowest. It incurs frequent round-trip delays due to connection establishment, performs slow start in both directions for short duration connections, and incurs heavy latency penalties due to the mismatch of the typical access profiles with the single request per transaction model. HTTP/1.0 also requires busy servers to dedicate resources to maintain TIME_WAIT information for large numbers of closed connections.

HTTP 2.0 sent the least number of packets. Because of the header compression, it effectively limits the number of packets that can be sent for the first few round trips.
HTTP 1.0 sent the most number of packets. Because it requires TCP connections establishment for every request, also it does not support compression.

However, HTTP 2.0 sent the most number of bytes while HTTP 1.0 sent the least number of bytes. My guess is that HTTP 1.0 has a simpler header and it even does not officially require a Host header. As for HTTP 2.0, the guess of its large raw bytes could be due to its packet loss and retransmissions.