

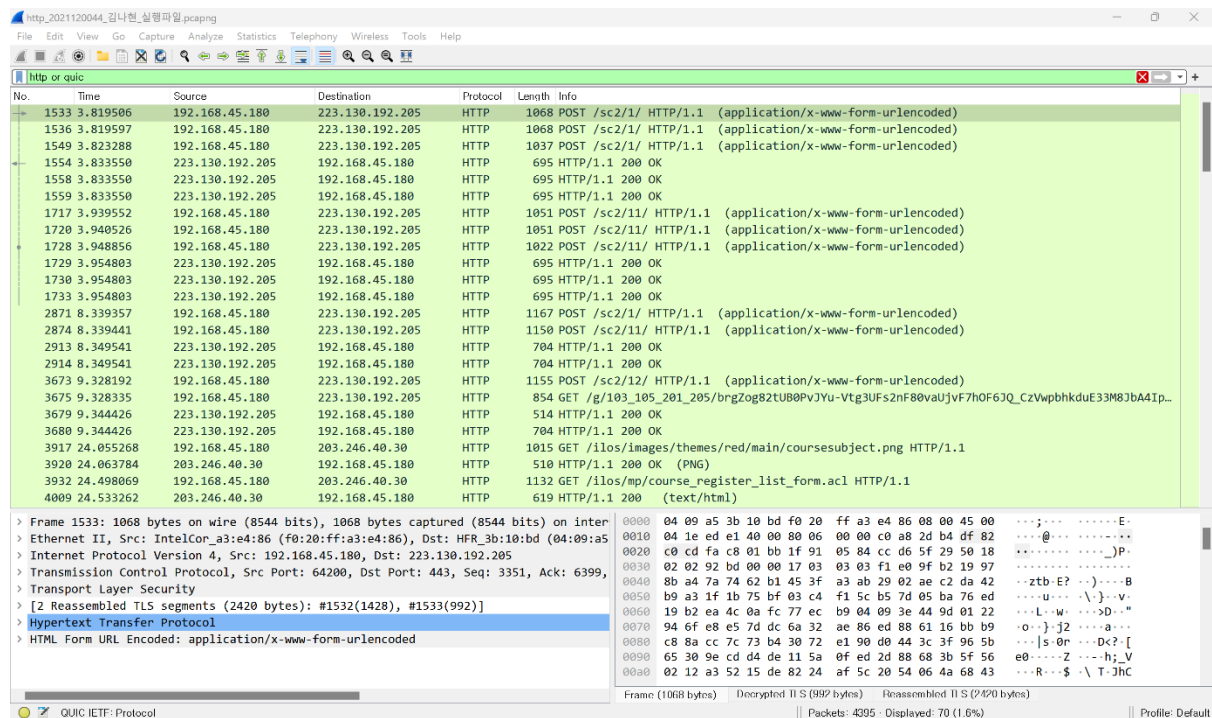
Wireshark 를 통한 트래픽 분석

2021120044

행정학과 김나현

Task 1. 자신의 네트워크에서 웹 트래픽 측정

[A] Display filter 를 활용하여 HTTP or QUIC 패킷만 필터링해보세요. Ethernet 또는 Wi-Fi 랜카드를 대상으로 패킷 캡처를 수행합니다. 자신의 책임 하에 있는 액세스 네트워크에서만 수행하기를 권장합니다. (공공장소에서 캡처하지 마세요)



[B] 웹 브라우저가 어떠한 HTTP (or QUIC) 버전을 사용하고 있는지 확인하세요. 어디서 정보를 찾을 수 있는지 스크린샷과 함께 제시하세요.

[Hypertext Transfer Protocol → "Request Version" or "Response Version" 필드에서 찾을 수 있다.]

HTTP/1.1 버전을 사용하고 있습니다.

```

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK

```

*[C] Display filter 를 활용하여 HTTP status code 를 필터링해보세요. 필터링을 위한 명령어와 캡처된 패킷에서 등장한 각 status code 가 어떠한 의미를 갖는지 표의 형태로 정리해보세요. (hint: http.response.code) 모든 http status code 를 웹에서 복사하라는 이야기가 아닙니다. 본인의 파일에서 얻은 code 만 살펴보면 됩니다.

→ http.response.code

200 OK	요청이 성공적으로 되었습니다. 정보는 요청에 따른 응답으로 반환됩니다.
304 Not Modified	캐시를 목적으로 사용됩니다. 클라이언트에게 응답이 수정되지 않았음을 알려주므로, 클라이언트는 계속해서 응답의 캐시된 버전을 사용할 수 있습니다.
101 Switching Protocol	클라이언트에 의해 보낸 업그레이드 요청 헤더에 대한 응답으로, 서버에서 프로토콜을 변경할 것임을 알려줍니다. 해당 코드는 WebSocket 프로토콜 전환 시에 사용됩니다.

```

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  HTTP/1.1 304 Not Modified\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 304 Not Modified\r\n]
    Response Version: HTTP/1.1
    Status Code: 304
    [Status Code Description: Not Modified]
    Response Phrase: Not Modified
  HTTP/1.1 101 Switching Protocols\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 101 Switching Protocols\r\n]
    Response Version: HTTP/1.1
    Status Code: 101
    [Status Code Description: Switching Protocols]
    Response Phrase: Switching Protocols

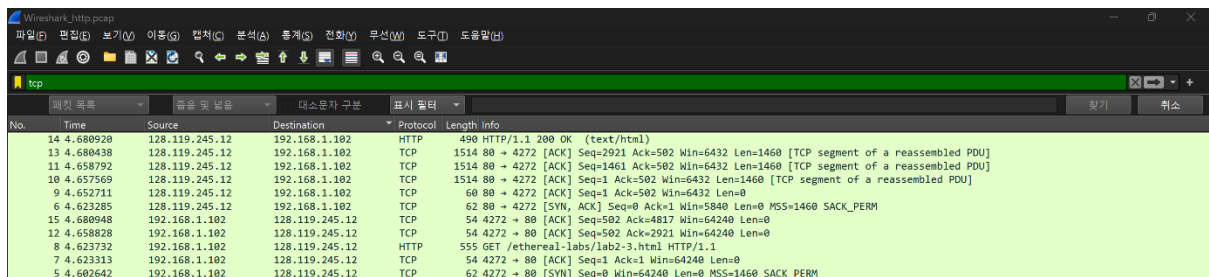
```

Task 2. 주어진 데이터에서 패킷의 의미 분석

[A] 주어진 파일의 display filter 에 "tcp"로 필터링한 후에 HTTP GET method 를 갖는 패킷을 찾고 해당 패킷을 찾는 명령어를 함께 제시하세요. (해당 패킷을 A 패킷으로 칭함)

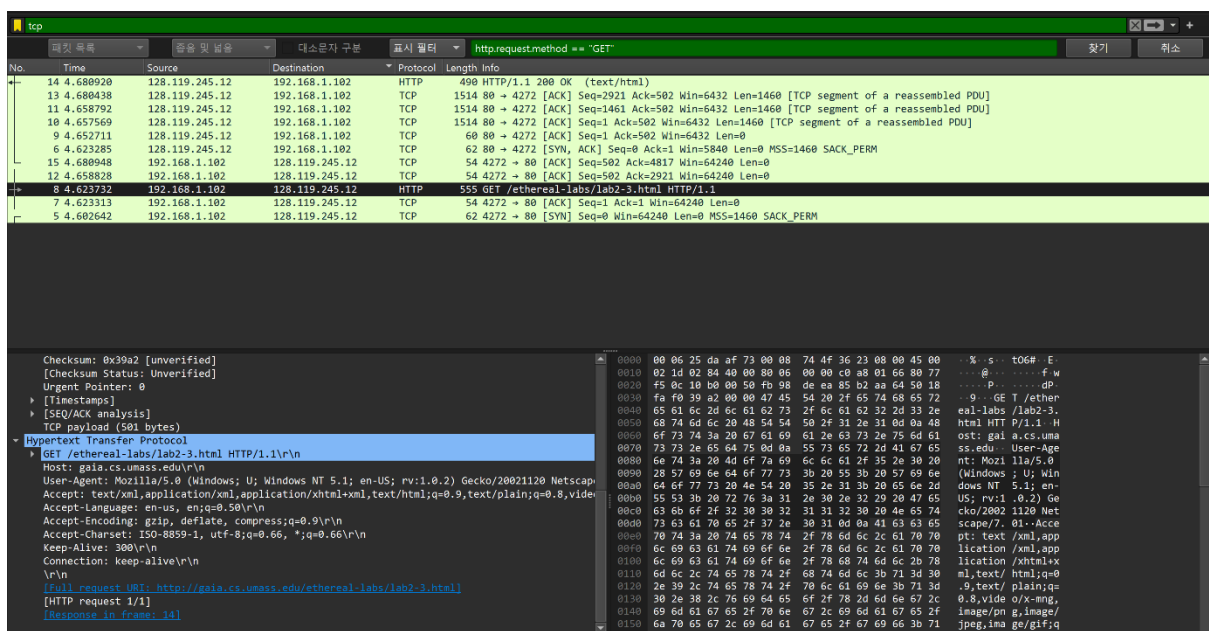
→ `http.request.method == "GET"`

(1) 디스플레이 필터에 tcp 를 입력하여 필터링한다.



No.	Time	Source	Destination	Protocol	Length	Info
14	4.680920	128.119.245.12	192.168.1.102	HTTP	490	HTTP/1.1 200 OK (text/html)
13	4.680438	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
11	4.658792	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
10	4.657569	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
9	4.652711	128.119.245.12	192.168.1.102	TCP	60	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
6	4.623285	128.119.245.12	192.168.1.102	TCP	62	80 → 4272 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
15	4.680948	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=4817 Win=64240 Len=0
12	4.658828	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
8	4.623732	192.168.1.102	128.119.245.12	HTTP	555	GET /etherreal-labs/lab2-3.html HTTP/1.1
7	4.623313	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
5	4.602642	192.168.1.102	128.119.245.12	TCP	62	4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM

(2) 표시필터에 `**http.request.method == "GET"**`를 입력하여 해당 패킷을 찾는다.



No.	Time	Source	Destination	Protocol	Length	Info
14	4.680920	128.119.245.12	192.168.1.102	HTTP	490	HTTP/1.1 200 OK (text/html)
13	4.680438	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
11	4.658792	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
10	4.657569	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
9	4.652711	128.119.245.12	192.168.1.102	TCP	60	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
6	4.623285	128.119.245.12	192.168.1.102	TCP	62	80 → 4272 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
15	4.680948	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=4817 Win=64240 Len=0
12	4.658828	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
8	4.623732	192.168.1.102	128.119.245.12	HTTP	555	GET /etherreal-labs/lab2-3.html HTTP/1.1
7	4.623313	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
5	4.602642	192.168.1.102	128.119.245.12	TCP	62	4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM

Packet Details	Packet Bytes	Packet Info
Checksum: 0x39a2 [unverified] [Checksum Status: Unverified] Urgent Pointer: 0 [Timestamps] [SEQ/ACK analysis] TCP payload (581 bytes) Hypertext Transfer Protocol GET /etherreal-labs/lab2-3.html HTTP/1.1 Host: gaia.cs.umass.edu User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20021120 Netscape Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video;q=0.8;q=0.50 Accept-Language: en-us,en;q=0.50 Accept-Encoding: gzip, deflate, compress;q=0.9 Accept-Charset: ISO-8859-1, utf-8;q=0.66,*;q=0.66 Keep-Alive: 300 Connection: keep-alive [Full request URI: http://gaia.cs.umass.edu/etherreal-labs/lab2-3.html] [HTTP request 1/1] [Response in frame: 14]	0000 00 06 25 da af 73 00 08 74 4f 36 23 08 00 45 00 0010 02 1d 02 84 40 00 80 06 00 00 c0 a8 01 66 80 77 0020 f5 0c 10 d0 00 50 fb 98 de ea 85 b2 aa 64 50 18 0030 fa f0 39 22 00 00 47 45 54 20 2f 65 74 68 65 72 0040 65 61 6c 2d 6c 61 62 73 2f 6c 61 62 32 2d 33 2e 0050 68 74 6d 6c 20 48 54 54 50 2f 31 2e 31 0d 0a 48 0060 6f 73 74 3a 20 67 61 69 61 2e 63 73 2e 75 6d 61 0070 73 73 2e 65 64 75 0d 8a 55 73 65 72 2d 41 67 65 0080 6e 74 3a 20 4d 6f 70 69 6c 6c 61 2f 35 20 30 20 0090 28 57 69 6e 64 6f 77 73 3b 20 55 3b 20 57 69 6e 00a0 64 6f 77 73 20 4e 54 20 35 2e 31 3b 20 65 6e 2d 00b0 55 53 3b 20 72 76 3a 31 2e 30 2e 32 29 20 47 65 00c0 63 65 6f 2f 32 30 30 32 31 31 32 30 20 4e 65 74 00d0 73 63 61 70 65 2f 37 2e 30 31 0d 0a 41 63 63 65 00e0 70 74 3a 20 74 65 78 74 2f 78 6d 6c 2c 61 70 70 00f0 6c 69 63 61 74 69 6f 6e 2f 78 6d 6c 2c 61 70 70 0100 6c 69 63 61 74 69 6f 6e 2f 78 68 74 6d 6c 2b 78 0110 6d 6c 2c 74 65 78 74 2f 68 74 6d 6c 3b 71 2d 30 0120 2e 39 2c 2c 76 69 64 65 6f 2f 78 2d 6d 6e 67 2c 0130 30 2e 38 2c 76 69 64 65 6f 2f 78 2d 6d 6e 67 2c 0140 69 6d 61 67 65 2f 70 6e 67 2c 69 6d 61 67 65 2f 0150 6a 70 65 67 2c 69 6d 61 67 65 2f 67 69 66 3b 71	%s...t0G# E...@.....f.w...P....dP...9-GE T/ether...eal-labs /lab2-3...html HT P/1.1 H...ost: gai a.cs.uma...ss.edu User-Age...nt: Mozil 11a/5.0... (Windows ; U: Win...dows NT 5.1; en-...US; rv:1.0.2) Ge...cko/2002 1120 Net...scape/7.01..Acce...pt: text /xml,app...lication /xml,app...lication /xhtml;x...ml;text/ html;q=0...9;text/ plain;q=...0.8,vide o/x-mng...image/pn g,image/...jpeg,ima ge/gif;q

[B] 주어진 파일에서 client 와 server 의 IP, port 번호를 각각 제시하세요.

- client - IP : 192.168.1.102
- client - port : 4272
- server IP : 128.119.245.12
- server port : 80

```
▶ Internet Protocol Version 4, Src: 192.168.1.102, Dst: 128.119.245.12
▼ Transmission Control Protocol, Src Port: 4272, Dst Port: 80, Seq: 0, Len: 0
  Source Port: 4272
  Destination Port: 80
```

[C] A 패킷의 앞의 3 개의 패킷은 무엇을 의미하는지 설명하세요. 각각의 패킷을 해석하는 것이 아니라 HTTP 에서 어떠한 의미를 갖는지 서술하세요. (hint: handshaking)

HTTP 는 TCP 를 Transport 프로토콜로 사용하기 때문에, 클라이언트는 먼저 서버에 TCP 연결을 해야 한다. TCP 연결을 위해 클라이언트가 서버와 3-way handshake 를 하는 과정을 아래 3 개의 패킷에서 살펴볼 수 있다.

1 단계. 먼저, 클라이언트에서 서버에게 SYN 이라는 플래그 비트를 0 으로 설정하여 연결을 요청하는 패킷을 보낸다.

2 단계. 서버는 도착한 패킷으로부터 SYN 을 추출하고 잘 받았다는 확인의 의미로 ACK 를 1 로 설정하여 SYN-ACK 패킷을 클라이언트에게 보낸다.

3 단계. 마지막으로, 클라이언트는 서버로부터 SYN-ACK 패킷을 수신하면, 서버의 연결 승인 패킷을 잘 받았다는 의미로 ACK 패킷을 보낸다.

위 세 단계를 통해 TCP 연결이 설정되었음을 확인할 수 있는 바이다.

5	4.602642	192.168.1.102	128.119.245.12	TCP	62 4272 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
6	4.623285	128.119.245.12	192.168.1.102	TCP	62 80 → 4272 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
7	4.623313	192.168.1.102	128.119.245.12	TCP	54 4272 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
8	4.623732	192.168.1.102	128.119.245.12	HTTP	555 GET /ethereal-labs/lab2-3.html HTTP/1.1

[D] A 패킷의 뒤의 패킷 HTTP 200 OK 까지 각각의 패킷의 의미 설명

8	4.623732	192.168.1.102	128.119.245.12	HTTP	555 GET /ethereal-labs/lab2-3.html HTTP/1.1
---	----------	---------------	----------------	------	---

[클라이언트 → 서버]

클라이언트에서 GET 메서드를 통해 /ethereal-labs/lab2-3.html 페이지를 요청하는 패킷을 서버로 전송하고 있다. GET 메서드는 서버에게 지정된 자원을 요청할 때 사용되며, 이 경우 클라이언트는 /ethereal-labs/lab2-3.html 이라는 특정 웹 페이지를 요청하고 있다.

9	4.652711	128.119.245.12	192.168.1.102	TCP	60 80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
---	----------	----------------	---------------	-----	---

[서버 → 클라이언트]

서버가 클라이언트의 GET 요청을 받았음을 확인하는 ACK 패킷을 클라이언트에게 전송하고 있다. 이 ACK 패킷을 통해 클라이언트에게 GET 요청의 수신을 확인함과 동시에, 데이터 전송 준비가 되었음을 의미한다. 이는 서버가 다음 단계인 실제 데이터 전송을 시작하기 전의 절차이다.

패킷에 포함된 윈도우 크기 'Win=6432'는 서버가 현재 한 번에 받을 수 있는 데이터의 최대 양을 나타내며, 네트워크의 흐름 제어와 혼잡 제어 기능을 한다.

```
10 4.657569 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
```

[서버 → 클라이언트]

HTTP 응답의 첫 부분을 클라이언트에게 전송하는 것으로, 클라이언트가 요청한 웹 페이지 또는 데이터의 일부를 담고 있다. 시퀀스 번호 1로 시작하여, 이는 연결 내에서 이 데이터 블록이 최초로 전송되는 세그먼트임을 나타낸다.

ACK 플래그와 함께 전송되어, 서버가 이전에 클라이언트로부터 받은 데이터 패킷을 확인했음을 표시한다.

```
11 4.658792 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
```

[서버 → 클라이언트]

서버로부터 클라이언트로 전송된 연속적인 데이터 세그먼트로, 전체 HTTP 응답의 두 번째 부분을 포함하고 있다.

시퀀스 번호는 1461부터 시작하는데, 이는 이전 세그먼트의 끝나는 지점인 1460 바이트 다음 데이터임을 나타낸다. 이 패킷 역시 ACK 플래그를 포함하여 클라이언트의 이전 데이터 수신을 서버가 확인했음을 표시한다.

```
12 4.658828 192.168.1.102 128.119.245.12 TCP 54 4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
```

[클라이언트 → 서버]

클라이언트가 서버로부터 받은 TCP 패킷을 확인하며, 추가 데이터를 수신할 준비가 되었음을 알리는 ACK 패킷을 서버에게 전송하고 있다.

(Ack=2921 은 클라이언트가 2921 번 시퀀스까지의 데이터를 정상적으로 수신했다는 것을 의미한다.)

```
13 4.680438 128.119.245.12 192.168.1.102 TCP 1514 80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
```

[서버 → 클라이언트]

이 패킷은 클라이언트가 요청한 웹 페이지 또는 자료의 일부를 포함하는 TCP 패킷이다.

전체 응답을 TCP 세그먼트 여러 개로 나누어 전송할 수 있기에 서버는 세그먼트 단위로 나누어 클라이언트에게 전송한다. 각 세그먼트는 연속적인 시퀀스 번호를 갖고 있어 데이터의 순서와 완전성을 보장한다.

(시퀀스 번호가 2921로 시작하여 이전 패킷들과 연속성을 유지한다.)

14	4.680920	128.119.245.12	192.168.1.102	HTTP	490	HTTP/1.1 200 OK (text/html)
----	----------	----------------	---------------	------	-----	-----------------------------

[서버 → 클라이언트]

서버가 클라이언트의 HTTP GET 요청을 성공적으로 처리하고, 요청된 /ethereal-labs/lab2-3.html에 대한 내용을 포함하는 HTTP 200 OK 응답을 보내는 패킷이다. 이 응답은 클라이언트에게 요청된 페이지가 성공적으로 찾아졌고 이에 대한 내용이 반환된 것을 의미한다.

8	4.623732	192.168.1.102	128.119.245.12	HTTP	555	GET /ethereal-labs/lab2-3.html HTTP/1.1
9	4.652711	128.119.245.12	192.168.1.102	TCP	60	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=0
10	4.657569	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
11	4.658792	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=1461 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
12	4.658828	192.168.1.102	128.119.245.12	TCP	54	4272 → 80 [ACK] Seq=502 Ack=2921 Win=64240 Len=0
13	4.680438	128.119.245.12	192.168.1.102	TCP	1514	80 → 4272 [ACK] Seq=2921 Ack=502 Win=6432 Len=1460 [TCP segment of a reassembled PDU]
14	4.680920	128.119.245.12	192.168.1.102	HTTP	490	HTTP/1.1 200 OK (text/html)