HTTP协议分析大作业

一、从截获报文中选择TCP建立连接和释放连接的报文，分析各个字段的值并概括HTTP协议的工作过程

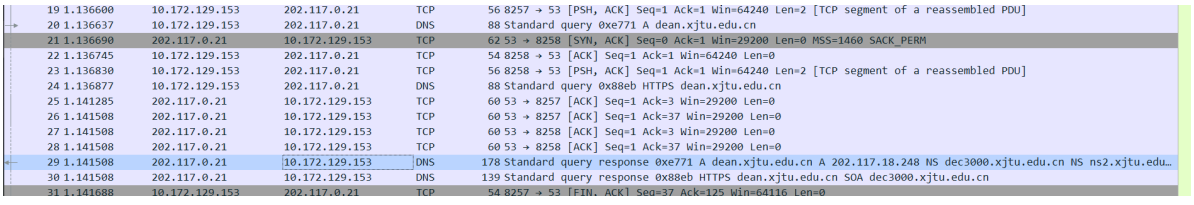
实验要求：在报文截图中找出TCP连接建立和连接释放的报文，并且说明TCP连接建立和连接释放的过程

```
C:\Users\lenovo>ping dean.xjtu.edu.cn

正在 Ping dean.xjtu.edu.cn [202.117.18.248] 具有 32 字节的数据:
来自 202.117.18.248 的回复: 字节=32 时间=3ms TTL=59
来自 202.117.18.248 的回复: 字节=32 时间=4ms TTL=59
来自 202.117.18.248 的回复: 字节=32 时间=4ms TTL=59
来自 202.117.18.248 的回复: 字节=32 时间=5ms TTL=59
```

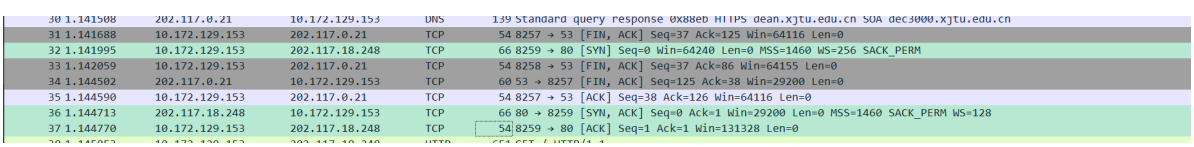
本校教务处IP地址为：202.117.18.248

DNS解析：



本地主机(10.172.129.153)先向DNS服务器(202.117.0.21)发送 dean.xjtu.edu.cn域名的IP地址请求，随后，DNS服务器向本机发送dean.xjtu.edu.cn的IP为202.117.18.248　(上图中第20和第29条报文)

TCP三次握手：

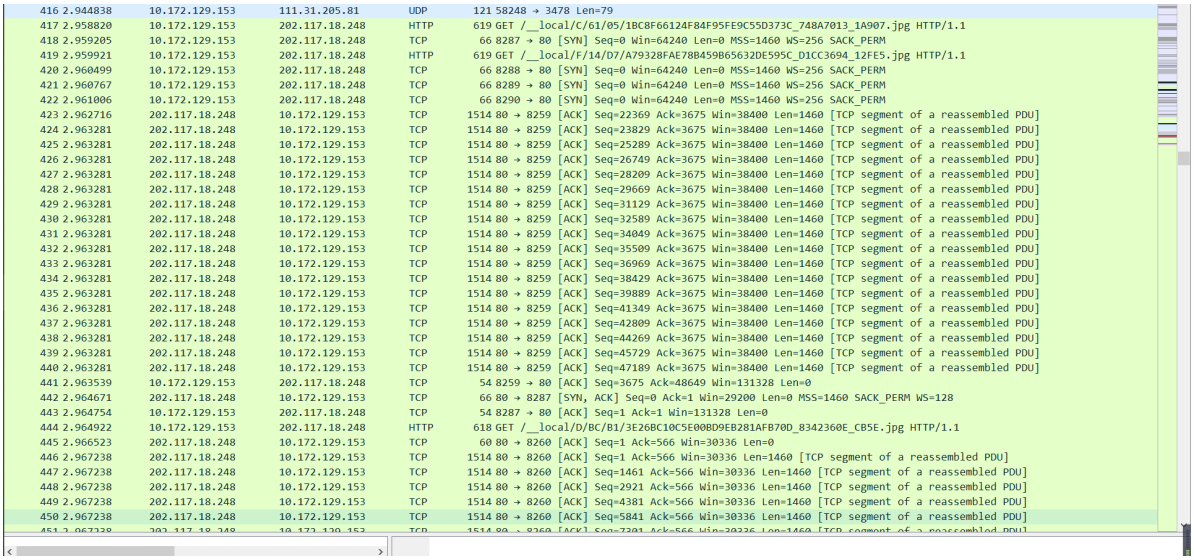


1.主机10.172.129.153向服务器202.117.18.248发起连接请求，可以看到在这时，SYN被置为1，序列号Seq=0

2.服务器202.117.18.248应答主机10.172.129.153，可以看到这个时候的应答包含了SYN，ACK，ACK = Seq + 1 = 1， 这里的Seq是第一次握手发起请求的Seq值，并不是下图报文中黑框中第二行表示的Seq值

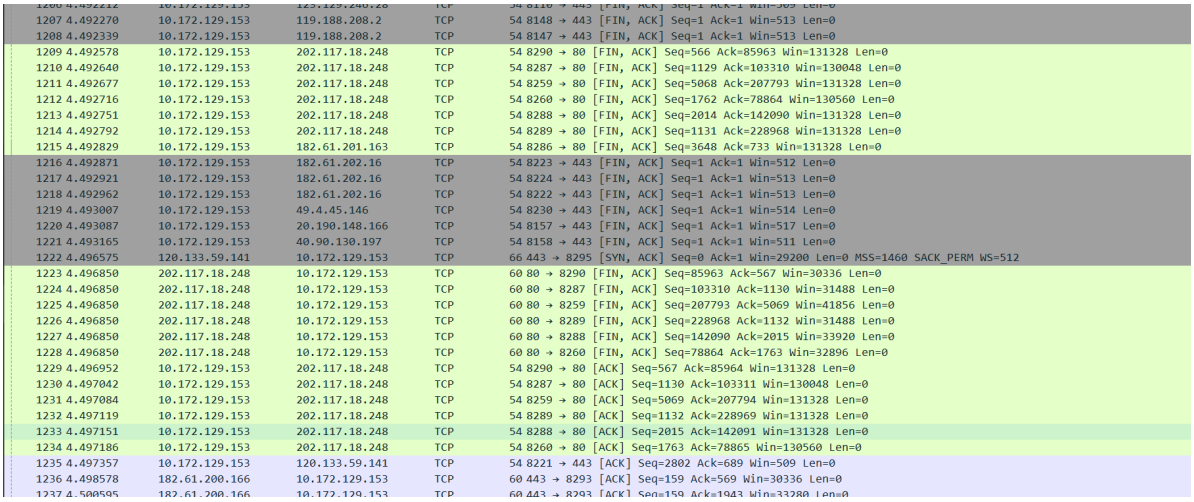
3.主机应答服务器，可以看到这个时候的应答包是ACK， ACK = Seq + 1 = 1，这里的Seq是第二次握手服务器产生的序列值，即报文中黑框第二行表示的Seq值　　（见上图第32、36、37条报文）

建立连接后开始传输网页等信息



416 2.944838	10.172.129.153	111.31.205.81	UDP	121	58248 → 3478 Len=79
417 2.958820	10.172.129.153	202.117.18.248	HTTP	619	GET /__local/C/61/05/1BC8F66124F84F95FE9C55D373C_748A7013_1A907.jpg HTTP/1.1
418 2.959205	10.172.129.153	202.117.18.248	TCP	66	8287 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
419 2.959921	10.172.129.153	202.117.18.248	HTTP	619	GET /__local/F/14/D7/A79328FAE78B459B65632DE595C_D1CC3694_12FE5.jpg HTTP/1.1
420 2.960499	10.172.129.153	202.117.18.248	TCP	66	8288 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
421 2.960767	10.172.129.153	202.117.18.248	TCP	66	8289 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
422 2.961006	10.172.129.153	202.117.18.248	TCP	66	8290 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
423 2.962716	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=22369 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
424 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=23829 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
425 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=25289 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
426 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=26749 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
427 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=28209 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
428 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=29669 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
429 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=31129 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
430 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=32589 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
431 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=34049 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
432 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=35509 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
433 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=36969 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
434 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=38429 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
435 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=39889 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
436 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=41349 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
437 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=42809 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
438 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=44269 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
439 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=45729 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
440 2.963281	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=47189 Ack=3675 Win=38400 Len=1460 [TCP segment of a reassembled PDU]
441 2.963539	10.172.129.153	202.117.18.248	TCP	54	8259 → 80 [ACK] Seq=3675 Ack=48649 Win=131328 Len=0
442 2.964671	202.117.18.248	10.172.129.153	TCP	66	80 → 8287 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
443 2.964754	10.172.129.153	202.117.18.248	TCP	54	8287 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
444 2.964922	10.172.129.153	202.117.18.248	HTTP	618	GET /__local/D/BC/B1/3E26BC10C5E00BD9EB281AFB70D_8342360E_CB5E.jpg HTTP/1.1
445 2.966523	202.117.18.248	10.172.129.153	TCP	60	80 → 8260 [ACK] Seq=1 Ack=566 Win=30336 Len=0
446 2.967238	202.117.18.248	10.172.129.153	TCP	1514	80 → 8260 [ACK] Seq=1 Ack=566 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
447 2.967238	202.117.18.248	10.172.129.153	TCP	1514	80 → 8260 [ACK] Seq=1461 Ack=566 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
448 2.967238	202.117.18.248	10.172.129.153	TCP	1514	80 → 8260 [ACK] Seq=2921 Ack=566 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
449 2.967238	202.117.18.248	10.172.129.153	TCP	1514	80 → 8260 [ACK] Seq=4381 Ack=566 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
450 2.967238	202.117.18.248	10.172.129.153	TCP	1514	80 → 8260 [ACK] Seq=5841 Ack=566 Win=30336 Len=1460 [TCP segment of a reassembled PDU]

TCP四次挥手：



1207 4.492270	10.172.129.153	119.188.208.2	TCP	54	8148 → 443 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
1208 4.492339	10.172.129.153	119.188.208.2	TCP	54	8147 → 443 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
1209 4.492578	10.172.129.153	202.117.18.248	TCP	54	8290 → 80 [FIN, ACK] Seq=566 Ack=85963 Win=131328 Len=0
1210 4.492640	10.172.129.153	202.117.18.248	TCP	54	8287 → 80 [FIN, ACK] Seq=1129 Ack=103310 Win=130048 Len=0
1211 4.492677	10.172.129.153	202.117.18.248	TCP	54	8259 → 80 [FIN, ACK] Seq=5068 Ack=207793 Win=131328 Len=0
1212 4.492716	10.172.129.153	202.117.18.248	TCP	54	8260 → 80 [FIN, ACK] Seq=1762 Ack=78864 Win=130560 Len=0
1213 4.492751	10.172.129.153	202.117.18.248	TCP	54	8288 → 80 [FIN, ACK] Seq=2014 Ack=142090 Win=131328 Len=0
1214 4.492792	10.172.129.153	202.117.18.248	TCP	54	8289 → 80 [FIN, ACK] Seq=1131 Ack=228968 Win=131328 Len=0
1215 4.492829	10.172.129.153	182.61.201.163	TCP	54	8286 → 80 [FIN, ACK] Seq=3648 Ack=733 Win=131328 Len=0
1216 4.492871	10.172.129.153	182.61.202.16	TCP	54	8223 → 443 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0
1217 4.492921	10.172.129.153	182.61.202.16	TCP	54	8224 → 443 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
1218 4.492962	10.172.129.153	182.61.202.16	TCP	54	8222 → 443 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
1219 4.493007	10.172.129.153	49.4.45.146	TCP	54	8230 → 443 [FIN, ACK] Seq=1 Ack=1 Win=514 Len=0
1220 4.493087	10.172.129.153	20.190.148.166	TCP	54	8157 → 443 [FIN, ACK] Seq=1 Ack=1 Win=517 Len=0
1221 4.493165	10.172.129.153	40.90.130.197	TCP	54	8158 → 443 [FIN, ACK] Seq=1 Ack=1 Win=511 Len=0
1222 4.496575	120.133.59.141	10.172.129.153	TCP	66	443 → 8295 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=512
1223 4.496850	202.117.18.248	10.172.129.153	TCP	60	80 → 8290 [FIN, ACK] Seq=85963 Ack=567 Win=30336 Len=0
1224 4.496850	202.117.18.248	10.172.129.153	TCP	60	80 → 8287 [FIN, ACK] Seq=103310 Ack=1130 Win=31488 Len=0
1225 4.496850	202.117.18.248	10.172.129.153	TCP	60	80 → 8259 [FIN, ACK] Seq=207793 Ack=5069 Win=41856 Len=0
1226 4.496850	202.117.18.248	10.172.129.153	TCP	60	80 → 8289 [FIN, ACK] Seq=228968 Ack=1132 Win=31488 Len=0
1227 4.496850	202.117.18.248	10.172.129.153	TCP	60	80 → 8288 [FIN, ACK] Seq=142090 Ack=2015 Win=33920 Len=0
1228 4.496850	202.117.18.248	10.172.129.153	TCP	60	80 → 8260 [FIN, ACK] Seq=78864 Ack=1763 Win=32896 Len=0
1229 4.496952	10.172.129.153	202.117.18.248	TCP	54	8290 → 80 [ACK] Seq=567 Ack=85964 Win=131328 Len=0
1230 4.497042	10.172.129.153	202.117.18.248	TCP	54	8287 → 80 [ACK] Seq=1130 Ack=103311 Win=130048 Len=0
1231 4.497084	10.172.129.153	202.117.18.248	TCP	54	8259 → 80 [ACK] Seq=5069 Ack=207794 Win=131328 Len=0
1232 4.497119	10.172.129.153	202.117.18.248	TCP	54	8289 → 80 [ACK] Seq=1132 Ack=228969 Win=131328 Len=0
1233 4.497151	10.172.129.153	202.117.18.248	TCP	54	8288 → 80 [ACK] Seq=2015 Ack=142091 Win=131328 Len=0
1234 4.497186	10.172.129.153	202.117.18.248	TCP	54	8260 → 80 [ACK] Seq=1763 Ack=78865 Win=130560 Len=0
1235 4.497357	10.172.129.153	120.133.59.141	TCP	54	8221 → 443 [ACK] Seq=2802 Ack=689 Win=509 Len=0
1236 4.498578	182.61.200.166	10.172.129.153	TCP	60	443 → 8293 [ACK] Seq=159 Ack=569 Win=30336 Len=0

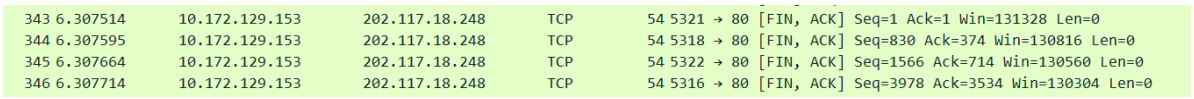
1209~1214　第一次挥手

1223~1228　第二、三次挥手(由于服务器还有数据没有发完)

1229~1234　第四次挥手

为展示四次挥手全过程，下面又重新捕获了一次报文。包括更详细的说明

TCP四次挥手补充：



343 6.307514	10.172.129.153	202.117.18.248	TCP	54	5321 → 80 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
344 6.307595	10.172.129.153	202.117.18.248	TCP	54	5318 → 80 [FIN, ACK] Seq=830 Ack=374 Win=130816 Len=0
345 6.307664	10.172.129.153	202.117.18.248	TCP	54	5322 → 80 [FIN, ACK] Seq=1566 Ack=714 Win=130560 Len=0
346 6.307714	10.172.129.153	202.117.18.248	TCP	54	5316 → 80 [FIN, ACK] Seq=3978 Ack=3534 Win=130304 Len=0

第一次挥手:

- 主机向服务器发送一个 FIN 数据包（FIN = 1，seq = u）主动断开连接，报文中会指定一个序列号（Seq）。主机就是在告诉服务器：我要跟你断开连接了，不会再给你发数据了；
- 主机此时还是可以接收数据的，如果一直没有收到被动连接方（服务器）的确认包，则可以重新发送这个包。此时主机处于 **FIN_WAIT1** 状态。

```
374 6.387986    202.117.18.248    10.172.129.153    TCP    60 80 → 5321 [ACK] Seq=1 Ack=2 Win=29312 Len=0
```

第二次挥手：

- 服务器收到 FIN 数据包之后，向主机发送确认包（ACK = 1,Ack = u + 1），把主机的序列号值 + 1 作为报文的Ack值，表明已经收到主机的报文了。这是服务器在告诉主机：我知道你要断开了，但是我还有数据没有发送完，等发送完了所有的数据就进行第三次挥手
- 此时服务端处于 **CLOSE_WAIT** 状态，主机处于 FIN_WAIT2 状态

第三次挥手：

```
375 6.397505    202.117.18.248    10.172.129.153    TCP    60 80 → 5322 [FIN, ACK] Seq=714 Ack=1567 Win=32768 Len=0
376 6.397505    202.117.18.248    10.172.129.153    TCP    60 80 → 5318 [FIN, ACK] Seq=374 Ack=831 Win=30976 Len=0
377 6.397505    202.117.18.248    10.172.129.153    TCP    60 80 → 5321 [FIN, ACK] Seq=1 Ack=2 Win=29312 Len=0
378 6.397505    202.117.18.248    10.172.129.153    TCP    60 80 → 5316 [FIN, ACK] Seq=3534 Ack=3979 Win=38912 Len=0
```

- 服务器向主机发送FIN 数据包（FIN=1，seq = w），且指定一个序列号，以及确认包（ACK = 1， Ack = u + 1），用来停止向主机发送数据这个动作是告诉主机：我的数据也发送完了，不再给你发数据了
- 此时服务端处于**LAST_ACK**状态，客户端处于TIME_WAIT状态

说明：

在我的实验过程中，发现四个报文中进行第二次挥手的只有第一个报文，查阅资料以后发现：第二和第三次挥手是**有可能**合并传输的。

我们知道，TCP四次挥手里，第二次和第三次挥手之间，是有可能有数据传输的。第三次挥手的目的是为了告诉主动方，"被动方没有数据要发了"。所以，在第一次挥手之后，如果被动方没有数据要发给主动方。第二和第三次挥手是**有可能**合并传输的。这样就出现了三次挥手。

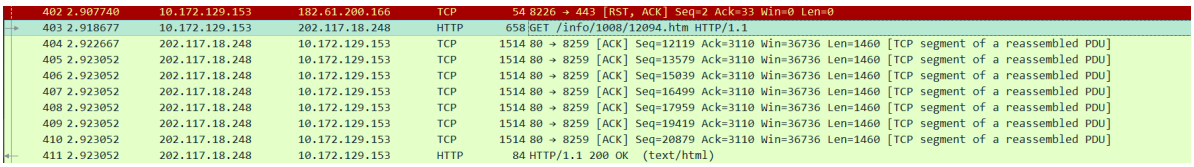
第四次挥手：

```
391 6.397589    10.172.129.153    202.117.18.248    TCP    54 5322 → 80 [ACK] Seq=1567 Ack=715 Win=130560 Len=0
392 6.397653    10.172.129.153    202.117.18.248    TCP    54 5318 → 80 [ACK] Seq=831 Ack=375 Win=130816 Len=0
393 6.397679    10.172.129.153    202.117.18.248    TCP    54 5321 → 80 [ACK] Seq=2 Ack=2 Win=131328 Len=0
394 6.397705    10.172.129.153    202.117.18.248    TCP    54 5316 → 80 [ACK] Seq=3979 Ack=3535 Win=130304 Len=0
```

- 主机收到 FIN数据包 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 + 1 作为自己 ACK 报文的Ack值
- 此时客户端处于 **TIME_WAIT** 状态。
- 需要过一了一定时间（2MSL）之后，主机发送确认包（ACK = 1， Ack = w + 1），此时主机才会进入 **CLOSED** 状态，以确保发送方的ACK可以到达接收方，防止已失效连接请求报文段出现在此连接中。
- 至此，完成四次挥手。

二、从截获的报文中选取HTTP请求报文（即get报文）和HTTP应答报文，并分析各字段的值。 综合分析截获的报文，概括HTTP协议的工作过程

实验要求：主要报文列表截图，标出HTTP报文、TCP报文。并以报文截图为线索说明HTTP请求和应答报文的格式，并说明HTTP协议如何使用TCP协议

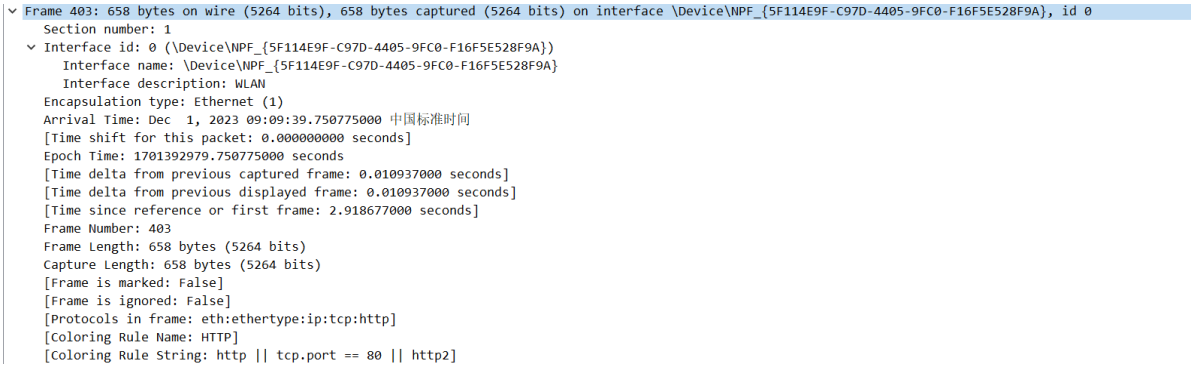


402	2.907740	10.172.129.153	182.61.200.166	TCP	54	8226 → 443 [RST, ACK] Seq=2 Ack=33 Win=0 Len=0
403	2.918677	10.172.129.153	202.117.18.248	HTTP	658	GET /info/1008/12094.htm HTTP/1.1
404	2.922667	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=12119 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
405	2.923052	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=13579 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
406	2.923052	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=15039 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
407	2.923052	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=16499 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
408	2.923052	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=17959 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
409	2.923052	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=19419 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
410	2.923052	202.117.18.248	10.172.129.153	TCP	1514	80 → 8259 [ACK] Seq=20879 Ack=3110 Win=36736 Len=1460 [TCP segment of a reassembled PDU]
411	2.923052	202.117.18.248	10.172.129.153	HTTP	84	HTTP/1.1 200 OK (text/html)

分析上图中的403号HTTP请求报文和411号HTTP应答报文。

HTTP请求报文：

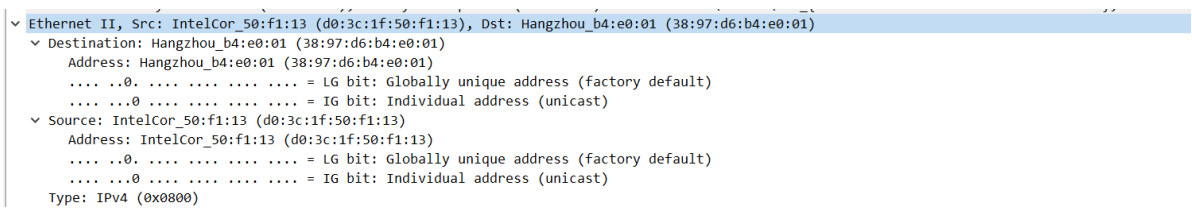
帧的信息



```
Frame 403: 658 bytes on wire (5264 bits), 658 bytes captured (5264 bits) on interface \Device\NPF_{5F114E9F-C97D-4405-9FC0-F16F5E528F9A}, id 0
  Section number: 1
  Interface id: 0 (\Device\NPF_{5F114E9F-C97D-4405-9FC0-F16F5E528F9A})
    Interface name: \Device\NPF_{5F114E9F-C97D-4405-9FC0-F16F5E528F9A}
    Interface description: WLAN
  Encapsulation type: Ethernet (1)
  Arrival Time: Dec  1, 2023 09:09:39.750775000 中国标准时间
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1701392979.750775000 seconds
  [Time delta from previous captured frame: 0.010937000 seconds]
  [Time delta from previous displayed frame: 0.010937000 seconds]
  [Time since reference or first frame: 2.918677000 seconds]
  Frame Number: 403
  Frame Length: 658 bytes (5264 bits)
  Capture Length: 658 bytes (5264 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:ip:tcp:http]
  [Coloring Rule Name: HTTP]
  [Coloring Rule String: http || tcp.port == 80 || http2]
```

1. 帧信息：
    - Frame 403: 帧的编号。
    - Frame Length: 帧的长度为658字节。
    - Capture Length: 捕获的长度为658字节。
    - Protocols in frame: 帧中包含的协议有以太网（eth）、以太网类型（ethertype）、IP协议（ip）、TCP协议（tcp）、HTTP协议（http）。
2. 时间信息：
    - Arrival Time: 捕获到该帧的时间。
    - Epoch Time: 以秒为单位的Epoch时间，表示自1970年1月1日以来的秒数。
    - Time delta from previous captured frame: 与前一个捕获帧之间的时间差。
    - Time delta from previous displayed frame: 与前一个显示帧之间的时间差。
    - Time since reference or first frame: 自参考点或第一帧以来的时间。
3. 网络接口信息：
    - Interface id: 网络接口的ID。
    - Interface name: 网络接口的名称，这里是WLAN。
    - Interface description: 网络接口的描述。
4. 协议信息：
    - Encapsulation type: 封装类型为Ethernet（以太网）。
    - Protocols in frame: 帧中包含的协议，这里涉及以太网、IP、TCP和HTTP。

Ethernet II帧头部分



```
Ethernet II, Src: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13), Dst: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
  Destination: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
      Address: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)
      Address: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

1. 源和目标MAC地址：
    - Source (源):

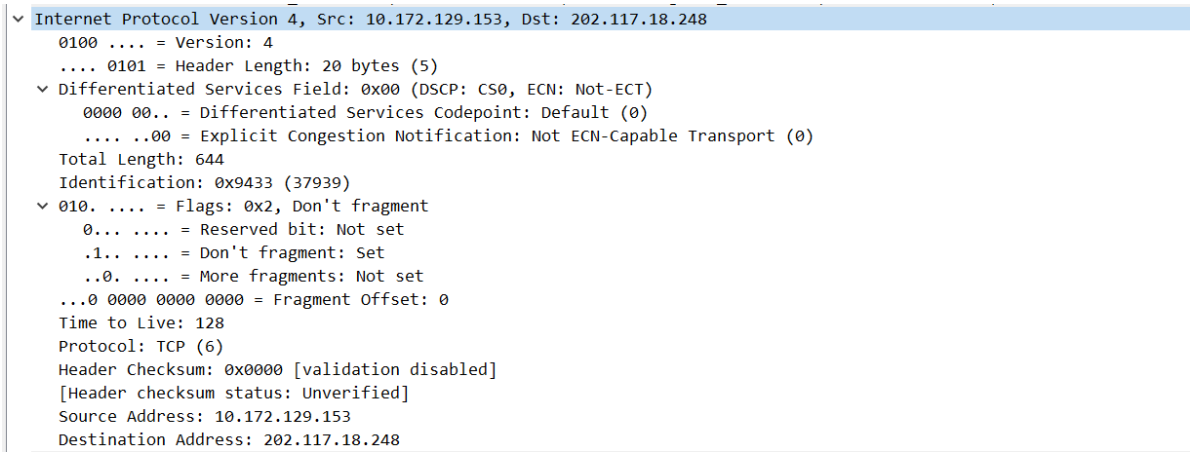
      IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)

        - Address: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)
        - LG bit: Globally unique address (factory default) - 表示这是一个全球唯一的地址（出厂默认值）。
        - IG bit: Individual address (unicast) - 表示这是一个单播地址。
    - Destination (目标): Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
        - Address: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
        - LG bit: Globally unique address (factory default) - 表示这是一个全球唯一的地址（出厂默认值）。
        - IG bit: Individual address (unicast) - 表示这是一个单播地址。

2. 帧类型(Type):
    - Type: IPv4 (0x0800) - 表示上层协议是IPv4。

总体而言，这部分内容描述了Ethernet II帧的源MAC地址、目标MAC地址以及帧中所携带的上层协议类型（IPv4）。每个网络接口都有唯一的MAC地址，用于在局域网中识别设备。

IPv4协议头部分



1. 版本和首部长度：
    - Version: 4 - 表示IPv4协议的版本。
    - Header Length: 20 bytes (5) - 表示IPv4首部的长度，单位是4字节，因此20字节。
2. 区分服务字段（Differentiated Services Field）：
    - DSCP (Differentiated Services Codepoint): CS0 - 表示Differentiated Services Codepoint为默认值。
    - ECN (Explicit Congestion Notification): Not ECN-Capable Transport - 表示不支持显式拥塞通知。
3. 总长度（Total Length）：
    - Total Length: 644 bytes - 表示整个IPv4数据包的长度，包括IPv4首部和数据部分。
4. 标识、标志和片段偏移：
    - Identification: 0x9433 (37939) - 用于将片段组装成原始报文的标识。
    - Flags: 0x2 (Don't fragment) - 表示不分片。
    - Fragment Offset: 0 - 表示片段的偏移，这里为0。
5. 生存时间（Time to Live）：
    - Time to Live: 128 - 表示生存时间，每经过一个路由器，该值减少1，防止数据包在网络中无限循环。
6. 协议类型（Protocol）：
    - Protocol: TCP (6) - 表示上层协议是TCP。
7. 首部校验和（Header Checksum）：

- Header Checksum: 0x0000 - 首部校验和，用于检测首部的完整性。这里显示校验和验证被禁用。

8. 源和目标IP地址：
    - Source Address: 10.172.129.153 - 源IP地址。
    - Destination Address: 202.117.18.248 - 目标IP地址。

总体而言，这部分内容提供了有关IPv4协议头的详细信息，包括版本、首部长度、区分服务字段、总长度、标识、标志、片段偏移、生存时间、协议类型、首部校验和以及源和目标IP地址。

TCP协议头部分

```
Transmission Control Protocol, Src Port: 8259, Dst Port: 80, Seq: 2506, Ack: 12119, Len: 604
    Source Port: 8259
    Destination Port: 80
    [Stream index: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 604]
    Sequence Number: 2506    (relative sequence number)
    Sequence Number (raw): 1757060779
    [Next Sequence Number: 3110    (relative sequence number)]
    Acknowledgment Number: 12119    (relative ack number)
    Acknowledgment number (raw): 1263671536
    0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······AP···]
    Window: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
    Checksum: 0x6c29 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  [Timestamps]
      [Time since first frame in this TCP stream: 1.776682000 seconds]
      [Time since previous frame in this TCP stream: 1.474821000 seconds]
  [SEQ/ACK analysis]
      [iRTT: 0.002775000 seconds]
      [Bytes in flight: 604]
      [Bytes sent since last PSH flag: 604]
    TCP payload (604 bytes)
```

1. 源和目标端口：
    - Source Port: 8259 - 表示源端口号。
    - Destination Port: 80 - 表示目标端口号，这是HTTP服务的默认端口。

2. 序列号和确认号：
    - Sequence Number: 2506 (relative sequence number) - 表示TCP数据流中的相对序列号。
    - Acknowledgment Number: 12119 (relative ack number) - 表示已收到的数据的确认序列号。

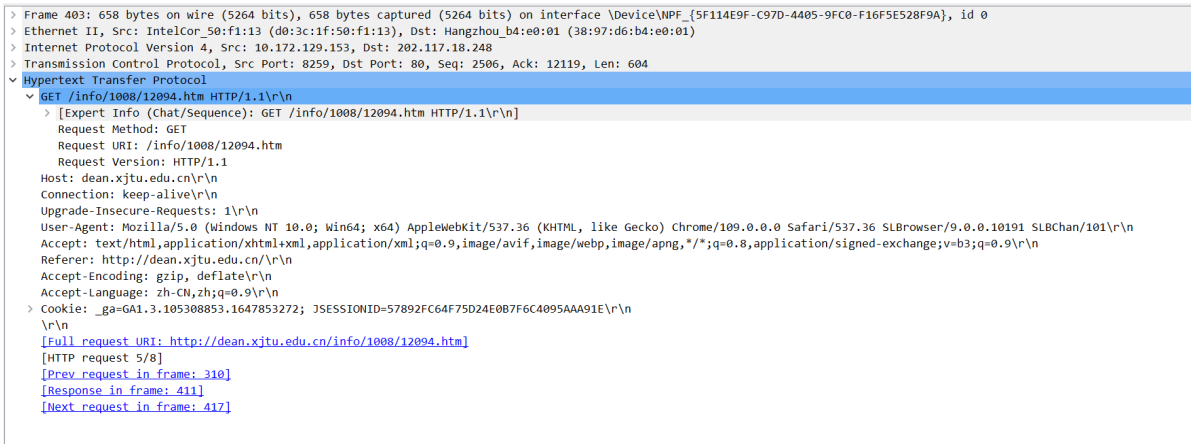
3. 标志：
    - Flags:

      0x018 (PSH, ACK) - 表示标志字段中包含了PSH（Push）和ACK（Acknowledgment）标志。

        - PSH标志指示接收方应该尽快将数据交给应用层，而不是等待缓冲区满再交付。
        - ACK标志表示该段包含有效的确认序列号。
    4. 窗口大小：
        - Window: 513 - 表示接收方的窗口大小，即发送方可以发送的字节数，而不需要等待确认。
    5. 校验和和紧急指针：
        - Checksum: 0x6c29 - TCP头部和数据的校验和。
        - Urgent Pointer: 0 - 紧急指针，用于指示紧急数据的末尾。
    6. 时间戳：
        - 包含了时间戳信息，用于计算往返时间（RTT）等。
    7. TCP负载（Payload）：
        - TCP payload (604 bytes): 指示TCP段中携带的实际数据，长度为604字节。
    8. TCP标志解析：
        - Flags解析：
            - PSH标志：Push标志，表示接收方应该立即将数据推送给应用层。
            - ACK标志：Acknowledgment标志，表示确认号字段包含有效的确认序列号。
    9. 序列/确认号分析：
        - 提供了一些有关序列号、确认号的分析信息，包括往返时间（RTT）等。

总体而言，这部分内容提供了有关TCP连接的详细信息，包括端口号、序列号、确认号、标志、窗口大小、校验和等。

HTTP协议头部分



- GET请求只有请求头，没有请求体(与POST请求不同)
- 第一行是请求行，分别是**请求方法**（GET）、**请求路径**（/info/1008/12094.htm）、**协议版本**(HTTP/2)，以"\r\n"来结尾
- 其余所有行均以XXX: XXXX的格式表示；
- 最后需要一个"\r\n"的空行作为请求头结束的标志。

下面对请求体进行分析：

- Host: 指定了服务器的域名，即 `dean.xjtu.edu.cn`。
- Connection: 表示客户端与服务器之间的连接方式，这里是 `keep-alive`，表示持久连接。
- Upgrade-Insecure-Requests: 用于提示服务器，客户端愿意升级连接以支持安全连接。
- User-Agent: 包含了客户端的相关信息，如操作系统和浏览器类型。
- Accept: 表示客户端可以接受的媒体类型。

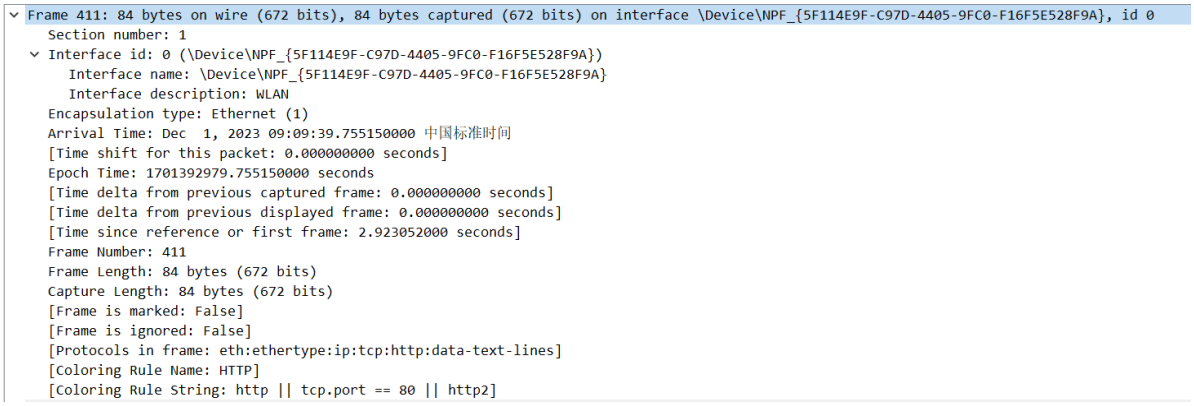
- Referer: 表示请求的来源页面。
- Accept-Encoding: 表示客户端支持的压缩算法，这里支持 gzip 和 deflate。
- Accept-Language: 表示客户端首选的自然语言。这里是中文。
- Cookie: 包含了客户端的一些状态信息，这里有两个 cookie。

在最后的"\r\n"之后的信息属于其他信息：

- Full request URI: 完整的请求 URI，包括协议、域名和路径。
- HTTP request 5/8: 表示这是整个会话中的第5个请求。
- Prev request in frame: 前一个请求的信息在帧中的位置。
- Response in frame: 相应的信息在帧中的位置。这里的值是411。
- Next request in frame: 下一个请求的信息在帧中的位置。

HTTP应答报文：

帧的信息



1. 帧信息：
    - Frame 411： 这是Wireshark对捕获的帧分配的唯一标识号。
    - Frame Length： 84 bytes - 表示整个帧的长度，包括所有的帧头和数据。
    - Capture Length： 84 bytes - 表示实际捕获到的帧的长度。
2. 接口信息：
    - Interface id： 0 (\Device\NPF_{5F114E9F-C97D-4405-9FC0-F16F5E528F9A}) - 表示捕获帧的网络接口ID。
    - Interface name： \Device\NPF_{5F114E9F-C97D-4405-9FC0-F16F5E528F9A} - 表示网络接口的名称。
    - Interface description： WLAN - 表示网络接口的描述。
3. 封装信息：
    - Encapsulation type： Ethernet (1) - 表示帧的封装类型为Ethernet。
4. 时间信息：
    - Arrival Time： 表示帧抓取的时间。
    - Epoch Time：以秒为单位的Epoch时间，表示自1970年1月1日以来的秒数。
5. 时间间隔信息：
    - 提供了与先前捕获的帧之间的时间间隔的相关信息。
6. 协议信息：
    - **Protocols in frame：** eth:ethertype:ip:tcp:http:data-text-lines - 表示帧中包含的协议层次结构，以太网（eth）、IP、TCP、HTTP以及数据文本行。
7. 着色规则信息：
    - **Coloring Rule Name：** HTTP - 表示Wireshark使用的颜色规则的名称。

- **Coloring Rule String:** http || tcp.port == 80 || http2 - 表示何时应用颜色规则的条件。

总体而言，这些信息提供了有关捕获帧的多个方面的详细信息，包括时间、长度、协议等。这对于网络协议分析和故障排除非常有用。

Ethernet II帧头部分

```
Ethernet II, Src: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01), Dst: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)
  Destination: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)
      Address: IntelCor_50:f1:13 (d0:3c:1f:50:f1:13)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
      Address: Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

1. 目标地址（Destination Address）：
    - Destination:

      IntelCor_50:f1:13 (d0:3c:1f:50:f1:13) - 表示帧传输的目标物理地址。

        - **Address:** IntelCor_50:f1:13 (d0:3c:1f:50:f1:13) - 目标地址的具体值。
        - **LG bit:** 0 - 表示该地址是全球唯一地址（factory default）。
        - **IG bit:** 0 - 表示该地址是个体地址（unicast）。
2. 源地址**（Source Address）**：
    - Source:

      Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01) - 表示帧传输的源物理地址。

        - **Address:** Hangzhou_b4:e0:01 (38:97:d6:b4:e0:01) - 源地址的具体值。
        - **LG bit:** 0 - 表示该地址是全球唯一地址（factory default）。
        - **IG bit:** 0 - 表示该地址是个体地址（unicast）。
3. 类型字段（Type）：
    - **Type:** IPv4 (0x0800) - 表示上层协议的类型，这里是IPv4协议。

这部分解析提供了有关Ethernet II帧头的详细信息，包括源地址、目标地址和上层协议类型。Ethernet II是一种常见的以太网帧封装协议，用于在以太网网络中传输数据。

IPv4协议头部分

```
Internet Protocol Version 4, Src: 202.117.18.248, Dst: 10.172.129.153
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 70
    Identification: 0xa66b (42603)
  010. .... = Flags: 0x2, Don't fragment
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 59
    Protocol: TCP (6)
    Header Checksum: 0x2f94 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 202.117.18.248
    Destination Address: 10.172.129.153
```

1. 版本和头部长度（Version and Header Length）：
    - **Version:** 4 - 表示IPv4协议的版本。

- **Header Length:** 20 bytes (5) - 表示IP头部的长度，以32位字为单位，因此20字节等于5个32位字。

2. 区分服务字段（Differentiated Services Field）：
   - **DSCP（Differentiated Services Codepoint）：**Default (0) - 表示区分服务代码点，指示对数据包的处理方式。
   - **ECN（Explicit Congestion Notification）：**Not ECN-Capable Transport (0) - 表示显式拥塞通知，这里是不支持的。
3. 总长度字段（Total Length）：
   - **Total Length:** 70 - 表示整个IPv4数据报的长度，包括头部和数据。
4. 标识字段（Identification）：
   - **Identification:** 0xa66b (42603) - 表示数据报的唯一标识符。
5. 标志和片偏移字段（Flags and Fragment Offset）：
   - **Flags:** 0x2 - 表示标志字段，其中最高位为1（Don't fragment），不允许分片。
   - **Fragment Offset:** 0 - 表示片偏移字段，指示数据报的片偏移量。
6. 生存时间字段（Time to Live）：
   - **Time to Live:** 59 - 表示生存时间，以跳数为单位，每经过一个路由器，该值减1。
7. 协议字段（Protocol）：
   - **Protocol:** TCP (6) - 表示上层协议，这里是TCP。
8. 头部校验和字段（Header Checksum）：
   - **Header Checksum:** 0x2f94 - 表示头部校验和，用于验证IP头部的完整性。这里显示校验和验证被禁用。
9. 源地址和目标地址（Source Address and Destination Address）：
   - **Source Address:** 202.117.18.248 - 表示源IP地址。
   - **Destination Address:** 10.172.129.153 - 表示目标IP地址。

这些字段组成了IPv4头部，提供了有关IP数据报的关键信息，包括版本、服务质量、总长度、标志、生存时间、协议和地址等。

TCP协议头部分

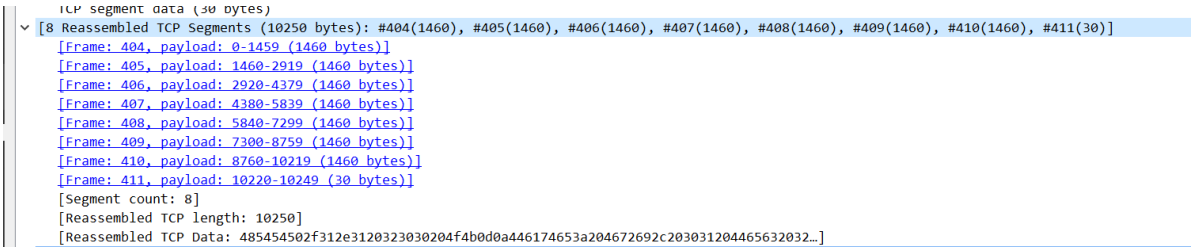
```
Transmission Control Protocol, Src Port: 80, Dst Port: 8259, Seq: 22339, Ack: 3110, Len: 30
    Source Port: 80
    Destination Port: 8259
    [Stream index: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 30]
    Sequence Number: 22339    (relative sequence number)
    Sequence Number (raw): 1263681756
    [Next Sequence Number: 22369    (relative sequence number)]
    Acknowledgment Number: 3110    (relative ack number)
    Acknowledgment number (raw): 1757061383
    0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····AP···]
    Window: 287
    [Calculated window size: 36736]
    [Window size scaling factor: 128]
    Checksum: 0x214f [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  [Timestamps]
      [Time since first frame in this TCP stream: 1.781057000 seconds]
      [Time since previous frame in this TCP stream: 0.000000000 seconds]
  [SEQ/ACK analysis]
```

```
[Time since previous frame in this TCP stream: 0.000000000 seconds]
[SEQ/ACK analysis]
  [iRTT: 0.002775000 seconds]
  [Bytes in flight: 10250]
  [Bytes sent since last PSH flag: 10250]
TCP payload (30 bytes)
TCP segment data (30 bytes)
```

1. 源端口和目标端口（Source Port and Destination Port）：
    - **Source Port:** 80 - 表示源端口号，这里是HTTP协议的默认端口。
    - **Destination Port:** 8259 - 表示目标端口号。
2. 流索引和会话完整性（Stream index and Conversation completeness）：
    - **Stream index:** 4 - 表示TCP流的索引。
    - **Conversation completeness:** Complete, WITH_DATA (31) - 表示TCP会话已经完成，有数据传输。
3. TCP段长度和序列号（TCP Segment Len and Sequence Number）：
    - **TCP Segment Len:** 30 - 表示TCP段的长度，即数据部分的长度。
    - **Sequence Number:** 22339 - 表示相对序列号，即数据部分在整个数据流中的相对位置。
    - **Sequence Number (raw):** 1263681756 - 表示原始的序列号值。
    - **Next Sequence Number:** 22369 - 表示下一个期望的序列号。
4. 确认号（Acknowledgment Number）：
    - **Acknowledgment Number:** 3110 - 表示相对确认号，即期望收到的下一个序列号。
    - **Acknowledgment number (raw):** 1757061383 - 表示原始的确认号值。
5. 标志字段（Flags）：
    - **Flags:** 0x018 (PSH, ACK) - 表示TCP头部中的标志，这里包括 PUSH（数据推送）和 ACK（确认）。
6. 窗口大小（Window）：
    - **Window:** 287 - 表示接收方的窗口大小，即允许发送方发送的未确认数据的大小。
7. 校验和字段（Checksum）：
    - **Checksum:** 0x214f - 表示TCP头部的校验和。
8. 紧急指针和时间戳（Urgent Pointer and Timestamps）：
    - **Urgent Pointer:** 0 - 表示紧急指针的值。
    - **Timestamps:** 提供了时间戳信息，包括与流的第一帧的时间差和与前一帧的时间差。
9. TCP负载和数据（TCP payload and TCP segment data）：
    - **TCP payload:** 30 bytes - 表示TCP段的实际数据长度。
    - **TCP segment data:** 30 bytes - 表示TCP段的具体数据内容。

这些字段提供了有关TCP通信的详细信息，包括端口号、序列号、确认号、标志、窗口大小等。这对于分析网络通信和故障排除非常有帮助。

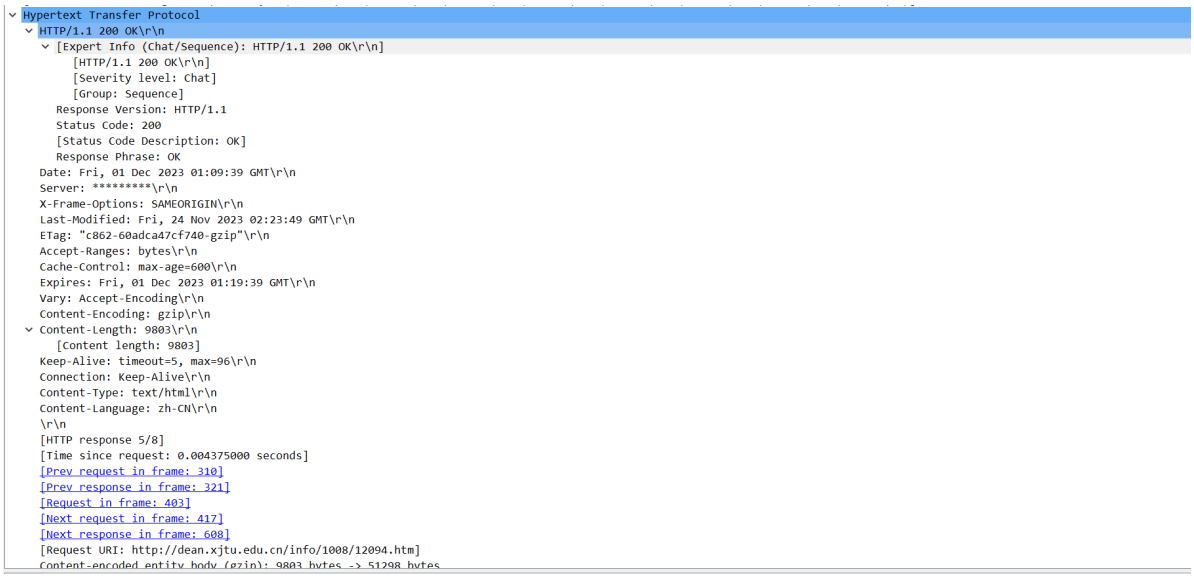
传回的内容



1. Reassembled TCP Segments（重新组装的TCP段）：
    - 提示了在重新组装过程中使用了多少个TCP段，以及每个TCP段的大小。
2. 各个TCP段的详细信息：
    - 每个TCP段都显示了帧号（Frame）和该段的有效载荷（payload）范围。
3. Segment count（段数）：
    - 表示已重新组装的TCP段的总数。
4. Reassembled TCP length（重新组装的TCP长度）：
    - 表示整个重新组装的TCP数据的长度。
5. Reassembled TCP Data（重新组装的TCP数据）：
    - 表示以十六进制形式呈现的整个重新组装的TCP数据流。

在这个特定的情景下，TCP数据流是一个HTTP响应，其内容以十六进制字符串形式显示。这种重新组装使得可以更容易地查看和分析整个TCP数据流，而不是只查看单个TCP段。

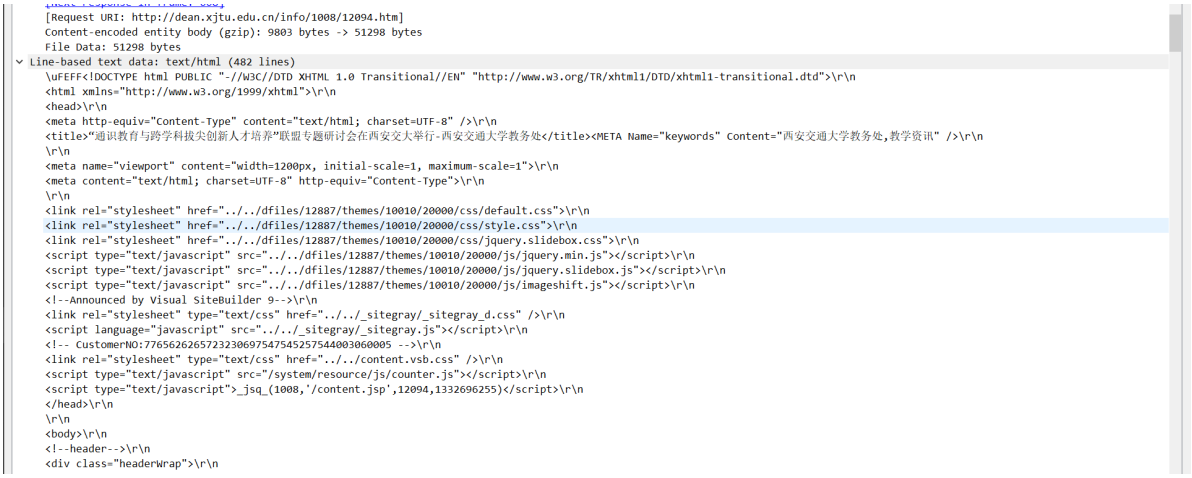
HTTP协议头部分



- 起始行包括三项：协议类型及版本号(HTTp/1.1)、状态码(200)、状态码的文字描述(OK)，以"\r\n"来结尾，状态码200表示请求成功
- **Date:** 指示响应的日期和时间。
- **Server:** 服务器的信息，被屏蔽了。应该是出于安全性的考虑。
- **X-Frame-Options:** 定义了页面的嵌套权限，这里是SAMEORIGIN，表示只允许相同域的页面嵌套。
- **Last-Modified:** 指示资源最后一次修改的时间。
- **ETag:** 实体标签，用于标识资源的版本。
- **Accept-Ranges:** 表示服务器支持按字节范围请求。
- **Cache-Control:** 控制缓存的行为，这里设置了最大缓存时间为600秒。
- **Expires:** 指示响应过期的日期和时间。
- **Vary:** 指示代理服务器缓存的策略。
- **Content-Encoding:** 表示实体内容使用了gzip压缩。

- **Content-Length:** 实体内容的长度。
- **Keep-Alive:** 保持连接的参数。
- **Connection:** 连接的管理，这里是Keep-Alive表示保持连接。
- **Content-Type:** 实体内容的类型，这里是text/html。
- **Content-Language:** 实体内容的语言，这里是zh-CN。
- **HTTP response 5/8:** 表示这是整个会话中的第5个响应。
- **Time since request:** 自上一个请求以来的时间。
- **Request URI:** 请求的URL，这里是 `http://dean.xjtu.edu.cn/info/1008/12094.htm`。
- **Content-encoded entity body (gzip):** 实体内容经过gzip压缩，原始长度9803字节，压缩后长度51298字节。

文本信息部分



'Line-based text data: text/html (482 lines)'表示实体内容是纯文本，并且使用换行符来分隔。类型是HTML文档，一共有482行。

三、TCP协议的工作过程

TCP分三个阶段
第一阶段：连接建立（三次握手）
第二阶段：数据传输
第三阶段：连接拆除（四次握手）

1第一阶段 连接建立

第一次握手：TCP客户进程先创建传输控制块TCB，然后向服务器发出连接请求报文，此次报文首部中的同部位SYN=1，同时选择一个初始序列号 seq=x ，此时，TCP客户端进程进入了 **SYN-SENT 同步已发送状态**

第二次握手：TCP服务器收到请求报文后，如果同意连接，则会向客户端发出确认报文。确认报文中应该 ACK=1，SYN=1，确认号是ack=x+1，同时也要为自己初始化一个序列号 seq=y，此时，TCP服务器进程进入了 **SYN-RCVD 同步收到状态**

第三次握手：TCP客户端收到确认后，还要向服务器给出确认。确认报文的ACK=1，ack=y+1，自己的序列号seq=x+1，此时，TCP连接建立，客户端进入**ESTABLISHED已建立连接状态**

我的理解是：

- 第一次握手： 客户端向服务器端发送报文
  证明客户端的发送能力正常

- 第二次握手：服务器端接收到报文并向客户端发送报文
  证明服务器端的接收能力、发送能力正常
- 第三次握手：客户端向服务器发送报文
  证明客户端的接收能力正常

2第二阶段

服务器发送的报文中会标注报文的第一个字节的序号，客户机的报文中会标注自己希望下一段收到的报文的开头第一个字节的序号。每一个报文都会标记自身的长度。

3第三阶段 连接拆除

第一次挥手： 客户端发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入**FIN-WAIT-1（终止等待1）状态**

第二次挥手： 服务器端接收到连接释放报文后，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了**CLOSE-WAIT 关闭等待状态**

第三次挥手 客户端接收到服务器端的确认请求后，客户端就会进入**FIN-WAIT-2（终止等待2）状态**，等待服务器发送连接释放报文，服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，服务器就进入了**LAST-ACK（最后确认）状态**，等待客户端的确认。

第四次挥手 客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了**TIME-WAIT（时间等待）状态**，但此时TCP连接还未终止，必须要经过2MSL后（最长报文寿命），当客户端撤销相应的TCB后，客户端才会进入CLOSED关闭状态，服务器端接收到确认报文后，会立即进入CLOSED关闭状态，到这里TCP连接就断开了，四次挥手完成

等待2MSL的原因是：为了保证客户端发送那个的第一个ACK报文能到到服务器，因为这个ACK报文可能丢失，并且2MSL是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃，这样新的连接中不会出现旧连接的请求报文。