



哈尔滨工业大学
Harbin Institute of Technology

计算机网络 课程实验报告

实验名称	利用 Wireshark 进行协议分析					
姓名		院系	计算学部			
班级		学号				
任课教师		指导教师				
实验地点	格物 207		实验时间			
实验课表现	出勤、表现得分(10)		实验报告 得分(40)		实验总分	
	操作结果得分(50)					
教师评语						



实验目的：

熟悉并掌握 Wireshark 的基本操作，了解网络协议实体间进行交互以及报文交换的情况。

实验内容：

- 1) 学习 Wireshark 的使用
- 2) 利用 Wireshark 分析 HTTP 协议
- 3) 利用 Wireshark 分析 TCP 协议
- 4) 利用 Wireshark 分析 IP 协议
- 5) 利用 Wireshark 分析 Ethernet 数据帧

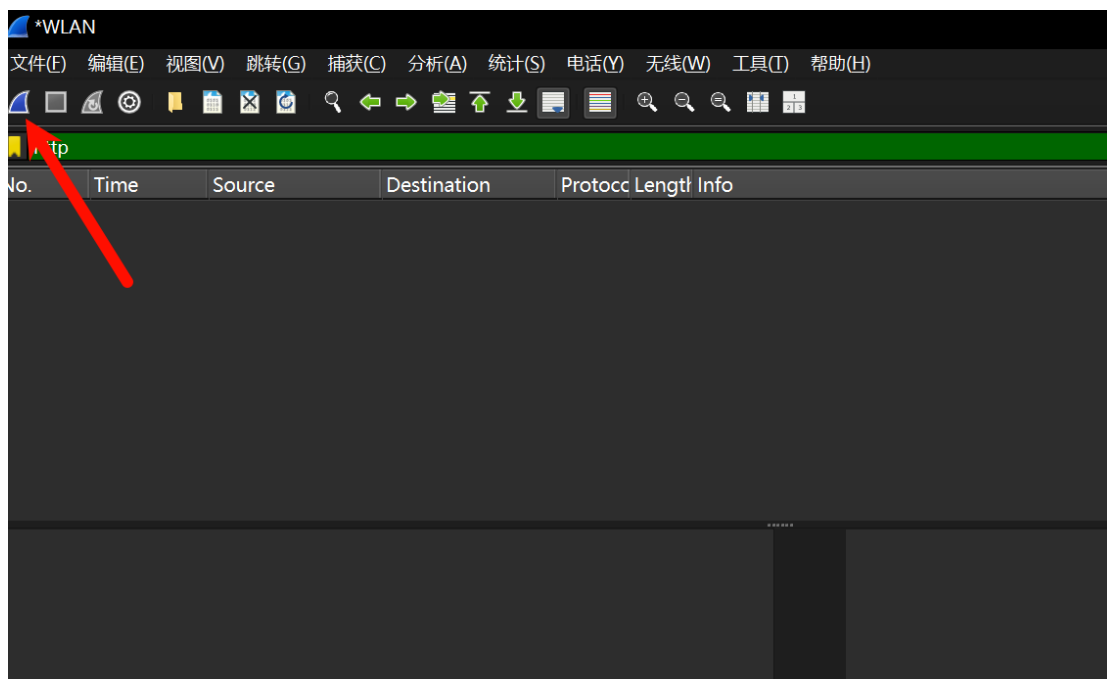
选做内容：

- a) 利用 Wireshark 分析 DNS 协议
- b) 利用 Wireshark 分析 UDP 协议
- c) 利用 Wireshark 分析 ARP 协议

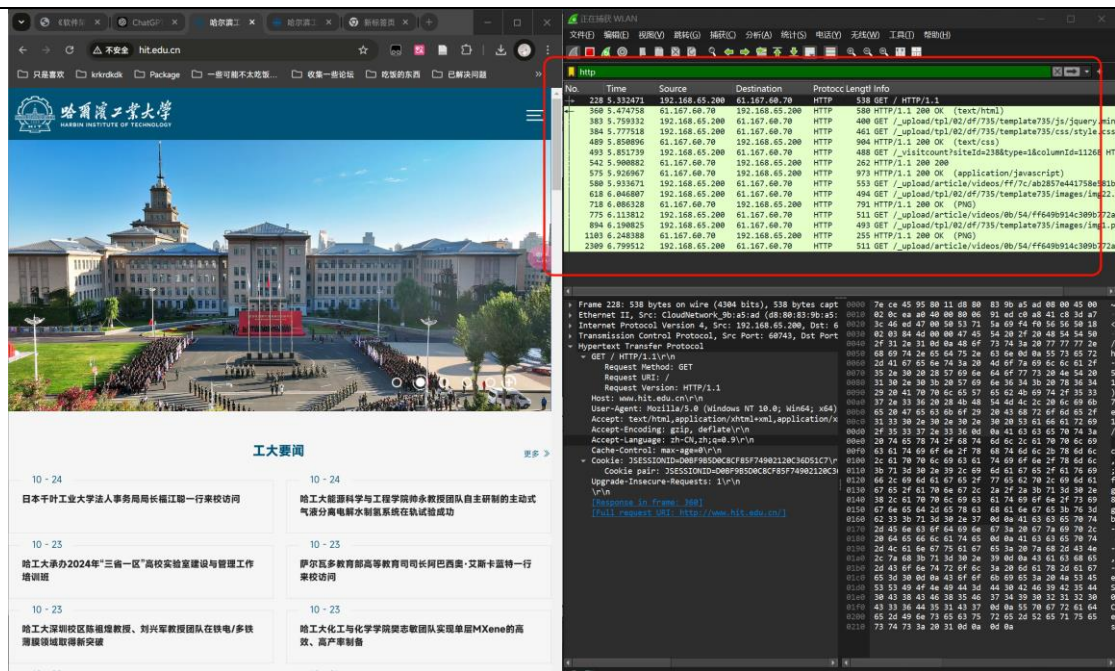
实验过程及实验结果：

1. WireShark的使用

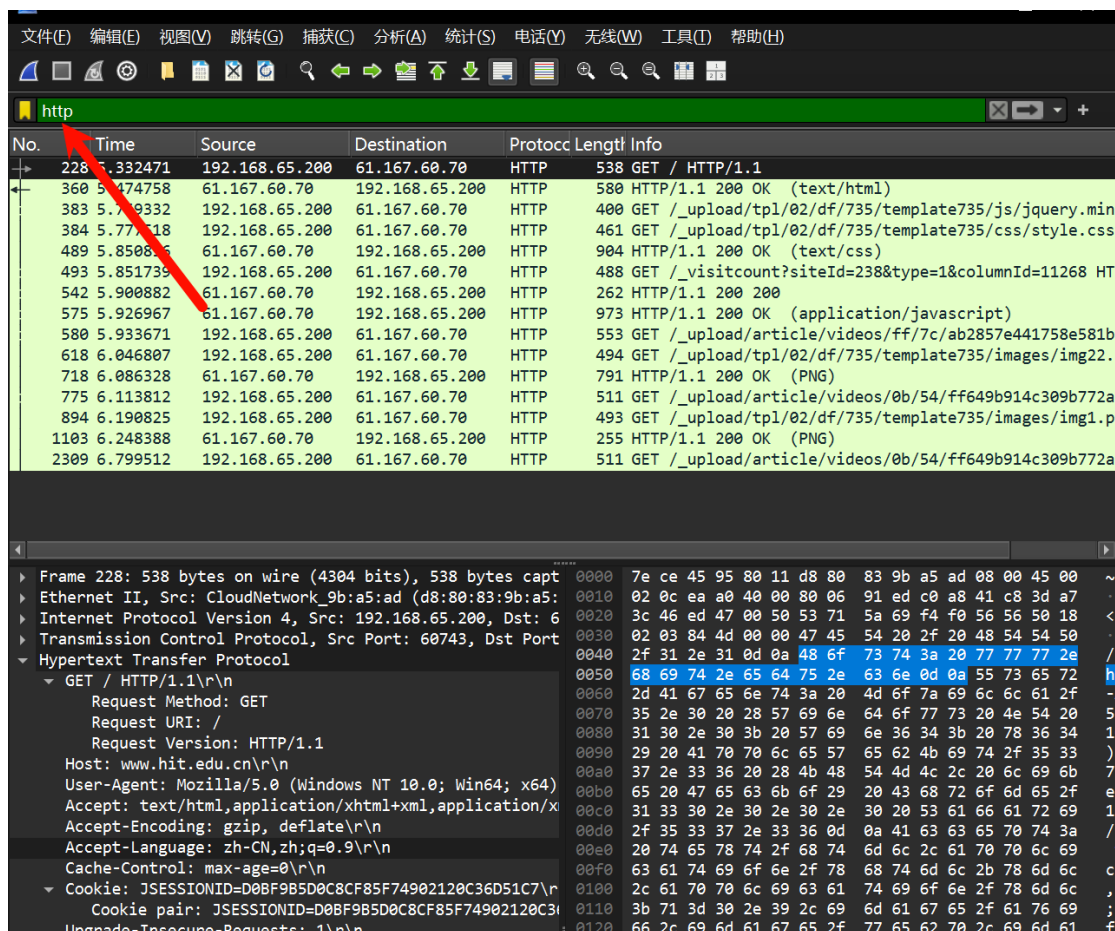
首先打开WireShark并且开始抓包，



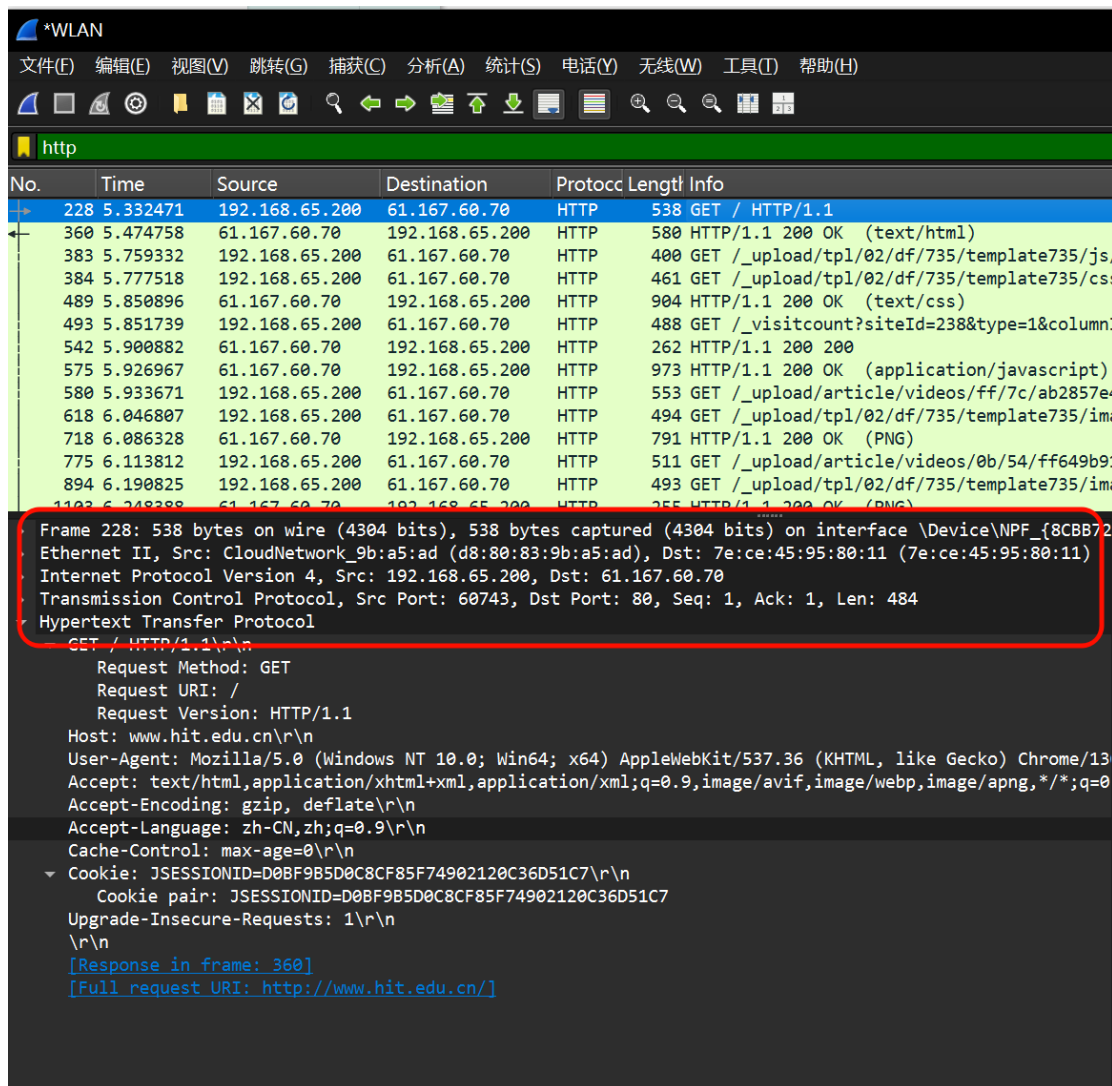
浏览器访问www.hit.edu.cn, 在WireShark中可以查看到俘获到的所有分组。



在搜索框中输入http从而筛选http报文，



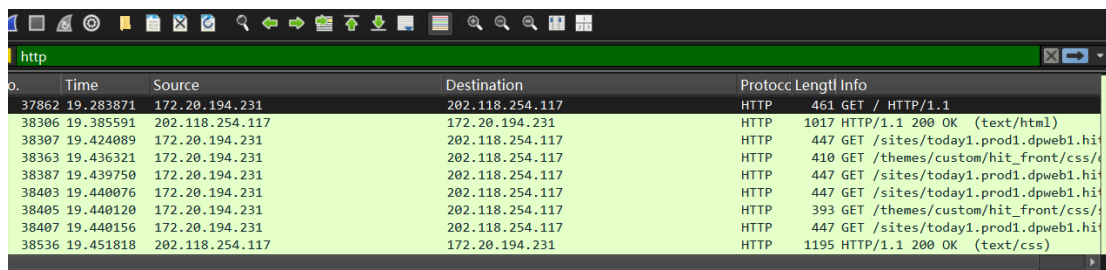
选择第一条 http 报文，以太网帧、IP 数据报、TCP 报文段、以及 HTTP 报文首部信息都将显示在分组首部子窗口中。并且点击三角箭头可以查看每个具体报文的详细信息。



2. HTTP分析

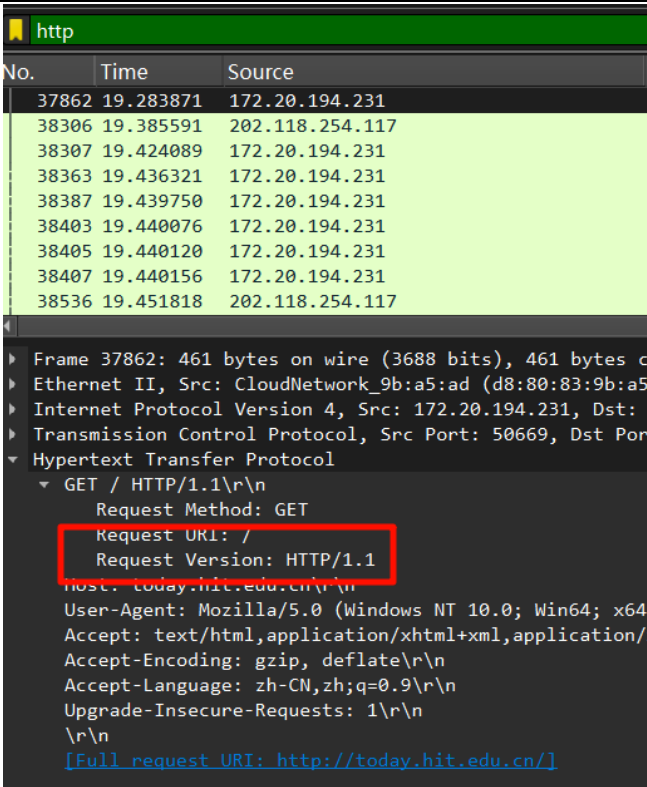
1) HTTP GET/response 交互

Wireshark开始抓包，并且在浏览器中访问<http://today.hit.edu.cn>，今日哈工大网站可以正确返回If-MODIFIED-SINCE的Http header，所以使用此网站能有良好的实验效果，在Wireshark中过滤出http分组，并进行查看。

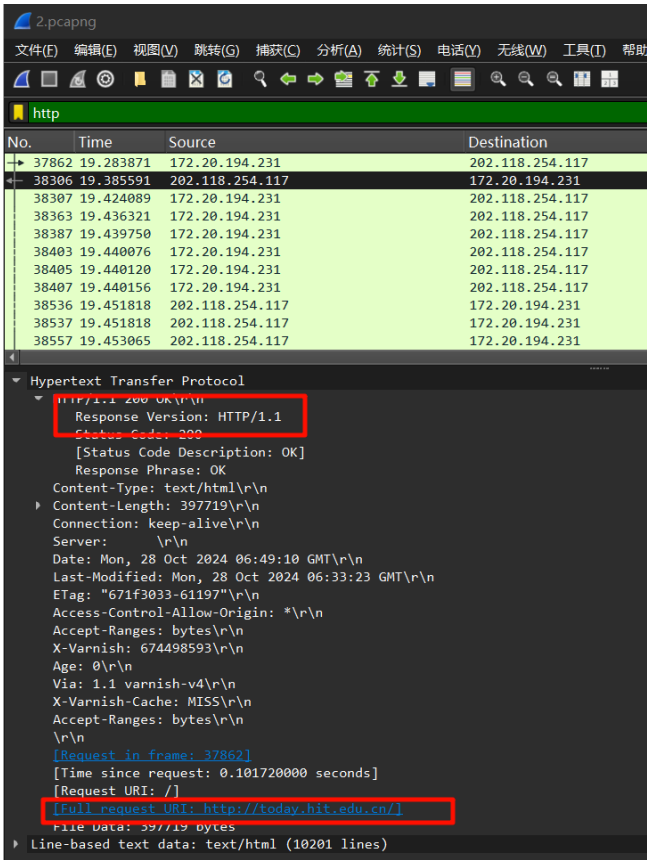


查看其中的第一条http报文，首先是由本机发送给目标服务器的报文，

(1) 可以看见我的浏览器运行的是HTTP1.1协议：



(2) 查看浏览器返回的报文：服务器运行的HTTP协议的版本号也是持久性连接的HTTP1.1，并且返回的报文中的状态码为200。



(3) 从下图可以看出浏览器向服务器指定的接受的语言版本的对象是zh-CN以及zh。

```
Request Method: GET
Request URI: /
Request Version: HTTP/1.1
Host: today.hit.edu.cn\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML like Gecko) Chrome/80.0.4016.101 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: zh-CN,zh;q=0.9\r\n
Upgrade-Insecure-Requests: 1\r\n
\r\n
[Response in frame: 38306]
[Full request URI: http://today.hit.edu.cn/]
```

- (4) 从下图的IP数据报中可以看出本机的IP地址和服务器的IP地址，其中本机的IP地址是172.20.194.231，服务器的IP地址为202.118.254.117。

```
Internet Protocol Version 4, Src: 172.20.194.231, Dst: 202.118.254.117
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 447
Identification: 0x8189 (33161)
  010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: TCP (6)
Header Checksum: 0x3fc7 [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.20.194.231
Destination Address: 202.118.254.117
[Stream Index: 5]
Transmission Control Protocol, Seq: 50660, Port: 80, Len: 1
```

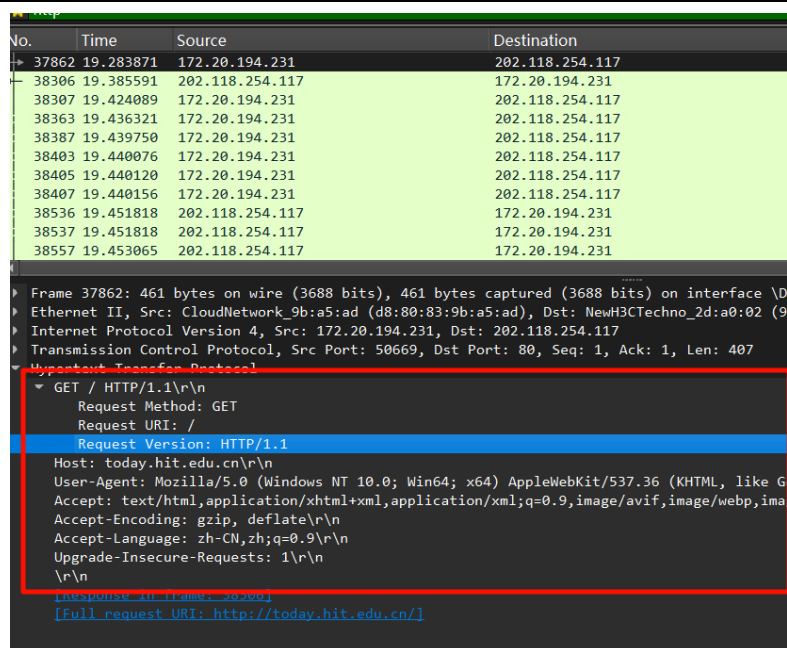
- (5) 浏览器返回的状态码为200

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
    Content-Type: text/html; charset=UTF-8\r\n
```

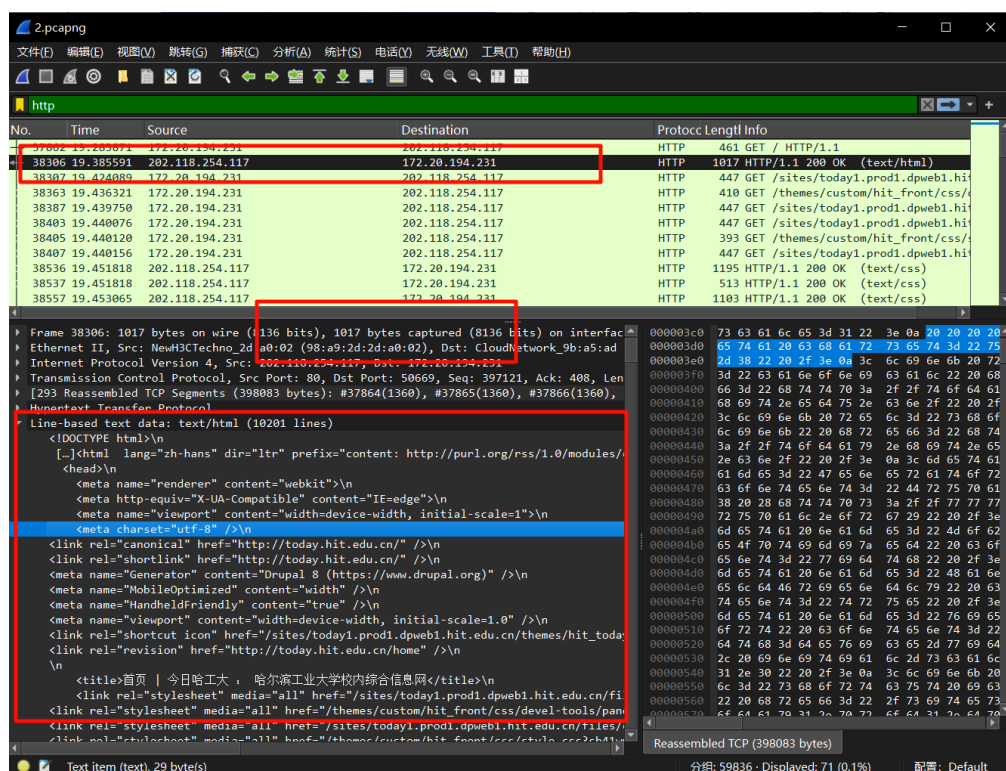
2) HTTP条件GET/response交互

清理浏览器缓存后，开启wireshark，刷新当前页面，查看wireshark俘获的http报文中第一条由客户端发送给服务器的get请求，

- (1) 清理缓存后的首次访问今日哈工大主页是不会有IF-MODIFIED-SINCE请求头：



- (2) 下图可以看到，服务器是否明确的返回文件可以在wireshark中看见是否有data内容，下图中存在名为Line-based text data: text/html 一栏，并且链路层数据包大小为1017bytes，所以服务器明确返回了文件的内容：



- (3) 而刷新后重新访问的请求头中可以查看到当前存在IF-MODIFIED-SINCE请求。该首部行后面跟着的信息应当是上一次服务器返回该资源的时间，也就是上一次的返回的response中的Last-Modified的http header中的时间数据，主机询问在该时间后服务器中的资源是否被修改。

No.	Time	Source	Destination	Protocol
12905	1.423391	172.20.194.231	202.118.254.117	HTTP
12907	1.428447	202.118.254.117	172.20.194.231	HTTP
12910	1.571147	172.20.194.231	202.118.254.117	HTTP
12916	1.577337	202.118.254.117	172.20.194.231	HTTP
12925	1.755858	172.20.194.231	202.118.254.117	HTTP
12927	1.755968	172.20.194.231	202.118.254.117	HTTP
12929	1.756142	172.20.194.231	202.118.254.117	HTTP
12936	1.772693	172.20.194.231	202.118.254.117	HTTP
12939	1.806543	202.118.254.117	172.20.194.231	HTTP
12940	1.838801	202.118.254.117	172.20.194.231	HTTP
12944	1.888954	202.118.254.117	172.20.194.231	HTTP

Frame 12905: 570 bytes on wire (4560 bits), 570 bytes captured (4560 bits) on interface \Device\NPF{...}	00
Ethernet II, Src: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad), Dst: NewH3CTechno_2d:a0:02 (98:40:2d:a0:02)	00
Internet Protocol Version 4, Src: 172.20.194.231, Dst: 202.118.254.117	00
Transmission Control Protocol, Src Port: 51006, Dst Port: 80, Seq: 1, Ack: 1, Len: 516	00
Hypertext Transfer Protocol	00
GET / HTTP/1.1\r\n	00
Request Method: GET	00
Request URI: /	00
Request Version: HTTP/1.1	00
Host: today.hit.edu.cn\r\n	00
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36\r\n	00
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/svg+xml\r\n	00
Accept-Encoding: gzip, deflate\r\n	00
Accept-Language: zh-CN,zh;q=0.9\r\n	00
Cache-Control: max-age=0\r\n	00
If-Modified-Since: Mon, 28 Oct 2024 06:33:23 GMT\r\n	00
If-None-Match: "671f3033-61197"\r\n	00
Upgrade-Insecure-Requests: 1\r\n	00
\r\n	00
[Request in frame: 12907]	00
[Full request URI: http://today.hit.edu.cn/]	00

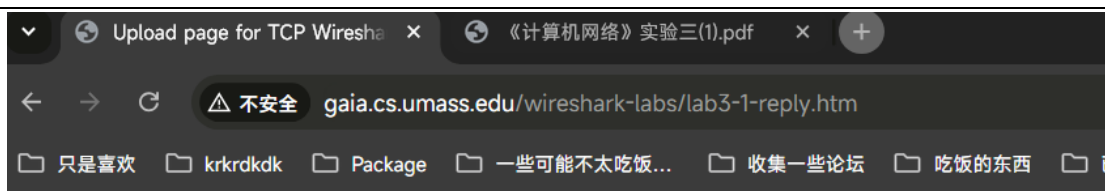
- (4) 第二次http get请求的时候, 服务器返回的http状态码为304, 返回的数据中不存在Line-based text data: text/html栏, 并且链路层的数据包大小也仅有334bytes, 服务器没有明确返回文件内容:

No.	Time	Source	Destination	Protocol
12905	1.423391	172.20.194.231	202.118.254.117	HTTP
12907	1.428447	202.118.254.117	172.20.194.231	HTTP
12910	1.571147	172.20.194.231	202.118.254.117	HTTP
12916	1.577337	202.118.254.117	172.20.194.231	HTTP
12925	1.755858	172.20.194.231	202.118.254.117	HTTP
12927	1.755968	172.20.194.231	202.118.254.117	HTTP
12929	1.756142	172.20.194.231	202.118.254.117	HTTP
12936	1.772693	172.20.194.231	202.118.254.117	HTTP
12939	1.806543	202.118.254.117	172.20.194.231	HTTP
12940	1.838801	202.118.254.117	172.20.194.231	HTTP
12944	1.888954	202.118.254.117	172.20.194.231	HTTP

Frame 12907: 335 bytes on wire (2680 bits), 335 bytes captured (2680 bits) on interface \Device\NPF{...}	0000
Ethernet II, Src: NewH3CTechno_2d:a0:02 (98:40:2d:a0:02), Dst: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad)	0010
Internet Protocol Version 4, Src: 202.118.254.117, Dst: 172.20.194.231	0020
Transmission Control Protocol, Src Port: 80, Dst Port: 51006, Seq: 1, Ack: 517, Len: 281	0030
Hypertext Transfer Protocol	0040
HTTP/1.1 304 Not Modified\r\n	0050
Response Version: HTTP/1.1	0060
Status Code: 304	0070
[Status Code Description: Not Modified]	0080
Response Phrase: Not Modified	0090
Connection: keep-alive\r\n	00a0
Server: \r\n	00b0
Date: Mon, 28 Oct 2024 06:58:12 GMT\r\n	00c0
Last-Modified: Mon, 28 Oct 2024 06:33:23 GMT\r\n	00d0
Etag: "671f3033-61197"\r\n	00e0
Access-Control-Allow-Origin: *\r\n	00f0
X-Varnish: 675551253\r\n	0100
Age: 0\r\n	0110
Via: 1.1 varnish-v4\r\n	0120
X-Varnish-Cache: MISS\r\n	0130
\r\n	0140
[Request in frame: 12905]	
[Time since request: 0.005056000 seconds]	
[Request URI: /]	
[Full request URI: http://today.hit.edu.cn/]	

3. TCP分析

在启动完wireshark并且上传完文件后, 当上传页面变为下图时, 停止wireshark的俘获:



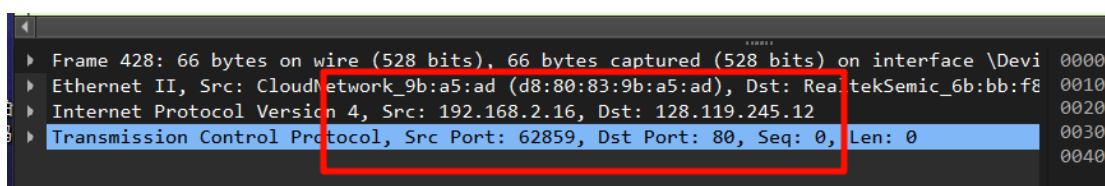
Congratulations!

You've now transferred a copy of alice.txt from your computer to gaia.cs.umass.edu. You should have captured Wireshark packets!

在wireshark中限定tcp以及ip.addr == 128.119.245.12后，查看到过滤的所有和服务器进行交互的tcp数据包：

No.	Time	Source	Destination	Protocol	Length	Info
428	19.126975	192.168.2.16	128.119.245.12	TCP	66	62859 → 80 [SYN] Seq=0 Win=642
430	19.469434	128.119.245.12	192.168.2.16	TCP	66	80 → 62859 [SYN, ACK] Seq=0 Ac
431	19.469564	192.168.2.16	128.119.245.12	TCP	54	62859 → 80 [ACK] Seq=1 Ack=1
463	20.118393	192.168.2.16	128.119.245.12	TCP	666	62859 → 80 [PSH, ACK] Seq=1 A
464	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=613 Ack=
465	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=2013 Ack=
466	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=3413 Ack=
467	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=4813 Ack=
468	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=6213 Ack=

选取第二个数据查看：



- (1) 客户端主机的IP地址为192.168.2.16，因为我在寝室的路由器下，所以网段和校园网有所差异，端口为62859，可以看见图中同时出现了62858和62859两个端口，但62858端口在往后的传输中未被使用，62859在后续的传输中占据了主要地位。
- (2) 服务器主机的IP地址是128.119.245.12，并且端口号是80。

No.	Time	Source	Destination	Protocol	Length	Info
428	19.126975	192.168.2.16	128.119.245.12	TCP	66	62859 → 80 [SYN] Seq=0 Win=642 Len=0 MSS=1460 WS=256 SACK_PERM
430	19.469434	128.119.245.12	192.168.2.16	TCP	66	80 → 62859 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1400 SACK_PERF
431	19.469564	192.168.2.16	128.119.245.12	TCP	54	62859 → 80 [ACK] Seq=1 Ack=1 Win=131584 Len=0
463	20.118393	192.168.2.16	128.119.245.12	TCP	666	62859 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131584 Len=612 [TCP PDU reass
464	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=613 Ack=1 Win=131584 Len=1400 [TCP PDU reass
465	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=2013 Ack=1 Win=131584 Len=1400 [TCP PDU reass
466	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=3413 Ack=1 Win=131584 Len=1400 [TCP PDU reass
467	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=4813 Ack=1 Win=131584 Len=1400 [TCP PDU reass

- (3) 客户服务器之间用于初始化 TCP 连接的 TCP SYN 报文段的序号 (sequence number) 是0，在该报文段中，将SYN标志位置1，ACK标志位置0，表示这是请求建立TCP的第一次握手，标示该报文段是 SYN报文段。

```

[Conversation completeness: Incomplete, DATA (15)]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 3713225941
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1000 .... = Header Length: 32 bytes (8)
Flags: 0x002 (SYN)
 000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
...0 .... = Congestion Window Reduced: Not set
...0 .... = ECN-Echo: Not set
...0 .... = Urgent: Not set
...0 .... = Acknowledgment: Not set
...0 .... = Push: Not set
...0 .... = Reset: Not set
...1 .... = Syn: Set
...0 .... = Fin: Not set
    
```

- (4) 服务器向客户端发送的 SYNACK 报文段序号是0，该报文段中，Acknowledgement 字段的值是1，客户端发送的第一次握手中并没有承载字节信息，但是仍要占据一个序号，因此客户端下一个要发送的数据字节序号从1开始，将ack_seq置为1，表示seq=0及之前的数据都收到，可以发送后面的数据。将ACK标志位与SYN标志位都置1，表示这是TCP连接建立过程中的第二次握手，标示该报文段是SYNACK报文段。

```

[Stream Packet Number: 2]
[Conversation completeness: Incomplete, DATA (15)]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 405721622
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 3713225942
1000 .... = Header Length: 32 bytes (8)
Flags: 0x012 (SYN, 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
...0 .... = Push: Not set
...0 .... = Reset: Not set
...1 .... = Syn: Set
...0 .... = Fin: Not set
[TCP Flags: .....A..S.]
Window: 29200
    
```

(5) 分析出 TCP 三次握手过程

a) TCP的三次握手过程如下图，首先是客户端首先将TCP头部的SYN标志位置为1，主机选取0为发送的第一个字节数据的序号，将相对序号seq位赋为0，实际的seq序号（raw）是下图中的3713225941，将报文发送给服务器端用于第一次握手，等待连接的建立。

No.	Time	Source	Destination	Proto
428	19.126975	192.168.2.16	128.119.245.12	TCP
430	19.469434	128.119.245.12	192.168.2.16	TCP
431	19.469564	192.168.2.16	128.119.245.12	TCP
463	20.118393	192.168.2.16	128.119.245.12	TCP
464	20.118511	192.168.2.16	128.119.245.12	TCP
465	20.118511	192.168.2.16	128.119.245.12	TCP
466	20.118511	192.168.2.16	128.119.245.12	TCP
467	20.118511	192.168.2.16	128.119.245.12	TCP

```

tcp && ip.addr == 128.119.245.12 && tcp.port == 62859
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 3713225941
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1000 .... = Header Length: 32 bytes (8)
Flags: 0x002 (SYN)
 000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
...0 .... = Congestion Window Reduced: Not set
...0 .... = ECN-Echo: Not set
...0 .... = Urgent: Not set
...0 .... = Acknowledgment: Not set
...0 .... = Push: Not set
...0 .... = Reset: Not set
...1 .... = Syn: Set
...0 .... = Fin: Not set
    
```

b) 其次过程如下图，服务器端接受到客户端发送的请求连接报文后，准备返回的tcp报文，将TCP头部的SYN标志位和ACK标志位设置为1，设置next_seq_number为1，实际的raw_seq_number为405721622，并且返回ack_num为1，raw_ack_num和上一过程中客户端发送给服务器的raw_seq_num一致，都为3713225942。

No.	Time	Source	Destination
428	19.126975	192.168.2.16	128.119.245.12
430	19.469434	128.119.245.12	192.168.2.16
431	19.469564	192.168.2.16	128.119.245.12
463	20.118393	192.168.2.16	128.119.245.12
464	20.118511	192.168.2.16	128.119.245.12
465	20.118511	192.168.2.16	128.119.245.12
466	20.118511	192.168.2.16	128.119.245.12
467	20.118511	192.168.2.16	128.119.245.12

[TCP Segment Len: 0]		0000
Sequence Number: 0	(relative sequence number)	0010
Sequence Number (raw): 405721622		0020
[Next Sequence Number: 1	(relative sequence number)]	0030
Acknowledgment Number: 1	(relative ack number)	0040
Acknowledgment number (raw): 3713225942		
1000 = Header Length: 32 bytes (8)		
▼ Flags: 0x012 (SYN, 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	
....0..	Push: Not set	
.... ..0..	Reset: Not set	
....1.	Syn: Set	
....0	Fin: Not set	

c) 客户端收到确认报文后，建立TCP连接，为了确认收到了服务器端的报文，将ACK标志位置1，SYN标志设置为0，第一个报文没有载荷数据但是也占据了一个序号，因此序号seq为1，和上一个步骤服务器返回的next_seq_num保持一致，同时将相对确认序号ack_seq赋为1，raw_ack_num和上一步骤服务器发送的raw_seq_num保持一致，将报文发送给服务器端用于第三次握手，该报文中可以承载信息。服务器端收到该报文后建立连接。

No.	Time	Source	Destination	Protocol
428	19.126975	192.168.2.16	128.119.245.12	TCP
430	19.469434	128.119.245.12	192.168.2.16	TCP
431	19.469564	192.168.2.16	128.119.245.12	TCP
463	20.118393	192.168.2.16	128.119.245.12	TCP
464	20.118511	192.168.2.16	128.119.245.12	TCP
465	20.118511	192.168.2.16	128.119.245.12	TCP
466	20.118511	192.168.2.16	128.119.245.12	TCP
467	20.118511	192.168.2.16	128.119.245.12	TCP

[TCP Segment Len: 0]		0000
Sequence Number: 1	(relative sequence number)	0010
Sequence Number (raw): 3713225942		0020
[Next Sequence Number: 1	(relative sequence number)]	0030
Acknowledgment Number: 1	(relative ack number)	
Acknowledgment number (raw): 405721623		
0101 = Header Length: 20 bytes (5)		
▼ Flags: 0x010 (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	
....0..	Push: Not set	
.... ..0..	Reset: Not set	
....0.	Syn: Not set	
....0	Fin: Not set	

(6) 包含HTTP POST命令的TCP报文段应当是PSH标识被设置为1的TCP段，如下图所示，因为这是在建立TCP连接后，客户端向服务器段发送的第一个TCP数据段，这个段的相对序号是1，而raw_seq_num被设置为3713225942，似乎是复用了TCP连接的最后一步中的seq序号。

No.	Time	Source	Destination	Protocol	Length	Info
428	19.126975	192.168.2.16	128.119.245.12	TCP	66	62859 → 80 [SYN] Seq=0 Win=0 Len=0
430	19.469434	128.119.245.12	192.168.2.16	TCP	66	80 → 62859 [SYN, ACK] Seq=613 Win=0 Len=0
431	19.469564	192.168.2.16	128.119.245.12	TCP	54	62859 → 80 [ACK] Seq=613 Win=0 Len=0
463	20.118393	192.168.2.16	128.119.245.12	TCP	666	62859 → 80 [PSH, ACK] Seq=613 Win=131584 Len=612
464	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=613 Win=131584 Len=1400
465	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=2013 Win=131584 Len=1400
466	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=3413 Win=131584 Len=1400
467	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=4813 Win=131584 Len=1400

Sequence Number: 1 (relative sequence number)	0000	00 e0 4c 6b bb f8 d8 80	83 9b a5 ad 08 00 45 00	...
Sequence Number (raw): 3713225942	0010	02 8c fd ae 40 00 80 06	c2 80 c0 a8 02 10 80 77	...
[Next Sequence Number: 613 (relative sequence number)]	0020	f5 0c f5 8b 00 50 dd 53	54 d6 18 2e d2 17 50 18	...
Acknowledgment Number: 1 (relative acknowledgment number)	0030	02 02 77 3d 00 00 50 4f	53 54 20 2f 77 69 72 65	...
Acknowledgment number (raw): 405721623	0040	73 68 61 72 6b 2d 6c 61	62 73 2f 6c 61 62 33 2d	...
0101 = Header Length: 20 bytes (5)	0050	31 2d 72 65 70 6c 79 2e	68 74 6d 20 48 54 54 50	...
Flags: 0x018 (PSH, ACK)	0060	2f 31 2e 31 0d 0a 48 6f	73 74 3a 20 67 61 69 61	...
0000 = Reserved: Not set	0070	2e 63 73 2e 75 6d 61 73	73 2e 65 64 75 0d 0a 43	...
0000 = Accurate ECN: Not set	0080	6f 6e 6e 65 63 74 69 6f	6e 3a 20 6b 65 65 70 2d	...
0000 = Congestion Window Reduction: Not set	0090	61 6c 69 76 65 0d 0a 43	6f 6e 74 65 6e 74 2d 4c	...
0000 = ECN-Echo: Not set	00a0	65 6e 67 74 68 3a 20 31	35 32 33 32 31 0d 0a 43	...
0000 = Urgent: Not set	00b0	61 63 68 65 2d 43 6f 6e	74 72 6f 6c 3a 20 6d 61	...
0000 = Acknowledgment: Set	00c0	78 2d 61 67 65 3d 30 0d	0a 55 70 67 72 61 64 65	...
0000 = Reset: Not set	00d0	2d 49 6e 73 65 63 75 72	65 2d 52 65 71 75 65 73	...
0000 = Syn: Not set	00e0	74 73 3a 20 31 0d 0a 55	73 65 72 2d 41 67 65 6e	...
0000 = Fin: Not set	00f0	74 3a 20 4d 6f 7a 69 6c	4e 54 2f 35 2e 30 20 28	...
	0100	57 69 6e 64 6f 77 73 20	4e 54 2f 31 30 2e 30 3b	...
	0110	20 57 69 6e 36 34 3b 20	78 36 34 29 20 41 70 70	...
	0120	6c 65 57 65 62 4b 69 74	2f 35 33 37 2e 33 36 20	...

- (7) 第六个报文段的相对序号为6213, raw_seq_num为3713232154, 发送的时间是TCP建立连接之后, 并且在FIN之前, wireshark中时间为20.118511时刻。

No.	Time	Source	Destination	Protocol	Length	Info
428	19.126975	192.168.2.16	128.119.245.12	TCP	66	62859 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM...
430	19.469434	128.119.245.12	192.168.2.16	TCP	66	80 → 62859 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1400 SACK_PERM...
431	19.469564	192.168.2.16	128.119.245.12	TCP	54	62859 → 80 [ACK] Seq=1 Ack=1 Win=131584 Len=0
463	20.118393	192.168.2.16	128.119.245.12	TCP	666	62859 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131584 Len=612 [TCP PDU reassemb...
464	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=613 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
465	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=2013 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
466	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=3413 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
467	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=4813 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
468	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=6213 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...

Source Port: 62859	0020	f5 0c f5 8b 00 50 dd 53	6d 1a 18 2e d2 17 50 10	...
Destination Port: 80	0030	02 02 19 8d 00 00 3a 20	62 73 68 65 20 6c 6f 6f	...
[Stream index: 30]	0040	6b 65 64 20 75 70 2c 20	62 75 74 20 69 74 20 77	...
[Conversation completeness: Incomplete, DATA]	0050	61 73 20 61 6c 6c 20 64	61 72 6b 20 6f 76 65 72	...
[TCP Segment Len: 1400]	0060	68 65 61 64 3b 20 62 65	66 6f 72 65 20 68 65 72	...
Sequence Number: 6213 (relative sequence number)	0070	0d 0a 77 61 73 20 61 6e	6f 74 68 65 72 20 6c 6f	...
Sequence Number (raw): 3713232154	0080	6e 67 20 70 61 73 73 61	6f 65 2c 20 61 6e 64 20	...
[Next Sequence Number: 7613 (relative sequence number)]	0090	74 68 65 20 57 68 69 74	65 20 52 61 62 62 69 74	...
Acknowledgment Number: 1 (relative acknowledgment number)	00a0	20 77 61 73 20 73 74 69	6c 6c 20 69 6e 0d 0a 73	...
Acknowledgment number (raw): 405721623	00b0	69 67 68 74 2c 20 68 75	72 72 79 69 6e 67 20 64	...
0101 = Header Length: 20 bytes (5)	00c0	6f 77 6e 20 69 74 2e 20	20 54 68 65 72 65 20 77	...
Flags: 0x010 (ACK)	00d0	61 73 20 6e 6f 74 20 61	20 6d 6f 6d 65 6e 74 20	...
	00e0	74 6f 20 62 65 20 6c 6f	73 74 3a 0d 0a 61 77 61	...
	00f0	79 20 77 65 6e 74 20 41	6c 69 63 65 70 6c 69 6b	...

而这个包的确认被包含在 Ack=13213 (相对 Ack) 中, 这是由于 tcp 的积累确认机制, 所以这个包的接受的时间应当在 Ack=13213 发送之前就已经确认接受。

430	19.469434	128.119.245.12	192.168.2.16	TCP	66	80 → 62859 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1400 SACK_PERM W...
431	19.469564	192.168.2.16	128.119.245.12	TCP	54	62859 → 80 [ACK] Seq=1 Ack=1 Win=131584 Len=0
463	20.118393	192.168.2.16	128.119.245.12	TCP	666	62859 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131584 Len=612 [TCP PDU reassemb...
464	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=613 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
465	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=2013 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
466	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=3413 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
467	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=4813 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
468	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=6213 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
469	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=7613 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
470	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=9013 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
471	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=10413 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
472	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=11813 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
501	20.437267	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=13213 Win=55680 Len=0
502	20.437267	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=13213 Win=55680 Len=0
503	20.437464	128.119.245.12	192.168.2.16	TCP	1454	62859 → 80 [ACK] Seq=13213 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...

- (8) 前六个包的长度为612、1400、1400、1400、1400、1400。

463	20.118393	192.168.2.16	128.119.245.12	TCP	666	62859 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131584 Len=612 [TCP PDU reassemb...
464	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=613 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
465	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=2013 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
466	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=3413 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
467	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=4813 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
468	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=6213 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
469	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=7613 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
470	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=9013 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...
471	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=10413 Ack=1 Win=131584 Len=1400 [TCP PDU reassemb...

- (9) 在整个跟踪过程中, 接收端公示的最小的可用缓存空间是win=30646, 在限制发送端的传输后, 接收端的缓存是足够使用的, 可以看见缓存在逐步增大, 最终稳定在10k+。

472	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=11813 Ack=1 Win=131584 Len=1400 [TCP PD..
501	20.437267	128.119.245.12	192.168.2.16	TCP	60	80 → 62859	[ACK] Seq=1 Ack=13213 Win=55680 Len=0
502	20.437267	128.119.245.12	192.168.2.16	TCP	60	80 → 62859	[ACK] Seq=1 Ack=6 Win=30464 Len=0
503	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=13213 Ack=1 Win=131584 Len=1400 [TCP PD..
504	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=14613 Ack=1 Win=131584 Len=1400 [TCP PD..
505	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[PSH, ACK] Seq=16013 Ack=1 Win=131584 Len=1400 [T..
506	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=17413 Ack=1 Win=131584 Len=1400 [TCP PD..
507	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=18813 Ack=1 Win=131584 Len=1400 [TCP PD..
508	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=20213 Ack=1 Win=131584 Len=1400 [TCP PD..
509	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=21613 Ack=1 Win=131584 Len=1400 [TCP PD..
510	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=23013 Ack=1 Win=131584 Len=1400 [TCP PD..
511	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=24413 Ack=1 Win=131584 Len=1400 [TCP PD..
512	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=25813 Ack=1 Win=131584 Len=1400 [TCP PD..
513	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=27213 Ack=1 Win=131584 Len=1400 [TCP PD..
514	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=28613 Ack=1 Win=131584 Len=1400 [TCP PD..
515	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=30013 Ack=1 Win=131584 Len=1400 [TCP PD..
516	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=31413 Ack=1 Win=131584 Len=1400 [TCP PD..
517	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[PSH, ACK] Seq=32813 Ack=1 Win=131584 Len=1400 [T..
518	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=34213 Ack=1 Win=131584 Len=1400 [TCP PD..
519	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=35613 Ack=1 Win=131584 Len=1400 [TCP PD..
520	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=37013 Ack=1 Win=131584 Len=1400 [TCP PD..
534	20.792062	128.119.245.12	192.168.2.16	TCP	60	80 → 62859	[ACK] Seq=1 Ack=17413 Win=64128 Len=0
535	20.792062	128.119.245.12	192.168.2.16	TCP	60	80 → 62859	[ACK] Seq=1 Ack=34213 Win=97664 Len=0
536	20.792062	128.119.245.12	192.168.2.16	TCP	60	80 → 62859	[ACK] Seq=1 Ack=38413 Win=106112 Len=0
537	20.792197	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80	[ACK] Seq=38413 Ack=1 Win=131584 Len=1400 [TCP PD..

- (10) 文件中没有重传的数据段，通过俘获页面的tcp数据段的序号可以看出，序号总体来说是上升的，并且没有重复和减少的情况出现，说明传输过程中没有发生报文的重传

No.	Time	Source	Destination	Protocol	Length	Info
469	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=76413 Ack=1 Win=131584 Len=1400 [T..
470	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=9013 Ack=1 Win=131584 Len=1400 [T..
471	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=10413 Ack=1 Win=131584 Len=1400 [T..
472	20.118511	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=11813 Ack=1 Win=131584 Len=1400 [T..
501	20.437267	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=13213 Win=55680 Len=0
502	20.437267	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=6 Win=30464 Len=0
503	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=13213 Ack=1 Win=131584 Len=1400 [T..
504	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=14613 Ack=1 Win=131584 Len=1400 [T..
505	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [PSH, ACK] Seq=16013 Ack=1 Win=131584 Len=1400 [T..
506	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=17413 Ack=1 Win=131584 Len=1400 [T..
507	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=18813 Ack=1 Win=131584 Len=1400 [T..
508	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=20213 Ack=1 Win=131584 Len=1400 [T..
509	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=21613 Ack=1 Win=131584 Len=1400 [T..
510	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=23013 Ack=1 Win=131584 Len=1400 [T..
511	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=24413 Ack=1 Win=131584 Len=1400 [T..
512	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=25813 Ack=1 Win=131584 Len=1400 [T..
513	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=27213 Ack=1 Win=131584 Len=1400 [T..
514	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=28613 Ack=1 Win=131584 Len=1400 [T..
515	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=30013 Ack=1 Win=131584 Len=1400 [T..
516	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=31413 Ack=1 Win=131584 Len=1400 [T..
517	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [PSH, ACK] Seq=32813 Ack=1 Win=131584 Len=1400 [T..
518	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=34213 Ack=1 Win=131584 Len=1400 [T..
519	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=35613 Ack=1 Win=131584 Len=1400 [T..
520	20.437464	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=37013 Ack=1 Win=131584 Len=1400 [T..
534	20.792062	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=17413 Win=64128 Len=0
535	20.792062	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=34213 Win=97664 Len=0
536	20.792062	128.119.245.12	192.168.2.16	TCP	60	80 → 62859 [ACK] Seq=1 Ack=38413 Win=106112 Len=0
537	20.792197	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=38413 Ack=1 Win=131584 Len=1400 [T..
538	20.792197	192.168.2.16	128.119.245.12	TCP	1454	62859 → 80 [ACK] Seq=39813 Ack=1 Win=131584 Len=1400 [T..

- (11) 文件开始传输的部分是

389	9.420436	192.168.2.16	128.119.245.12	TCP	666	60685 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131584 Len=612 [T..
390	9.420611	192.168.2.16	128.119.245.12	TCP	1454	60685 → 80 [ACK] Seq=613 Ack=1 Win=131584 Len=1400 [T..
391	9.420611	192.168.2.16	128.119.245.12	TCP	1454	60685 → 80 [ACK] Seq=2013 Ack=1 Win=131584 Len=1400 [T..

文件结束部分是一个post请求结束，没有显式的FIN标志设置。最后一个字节是152933，因为下一个期待的next_seq_num是152934。所以计算：

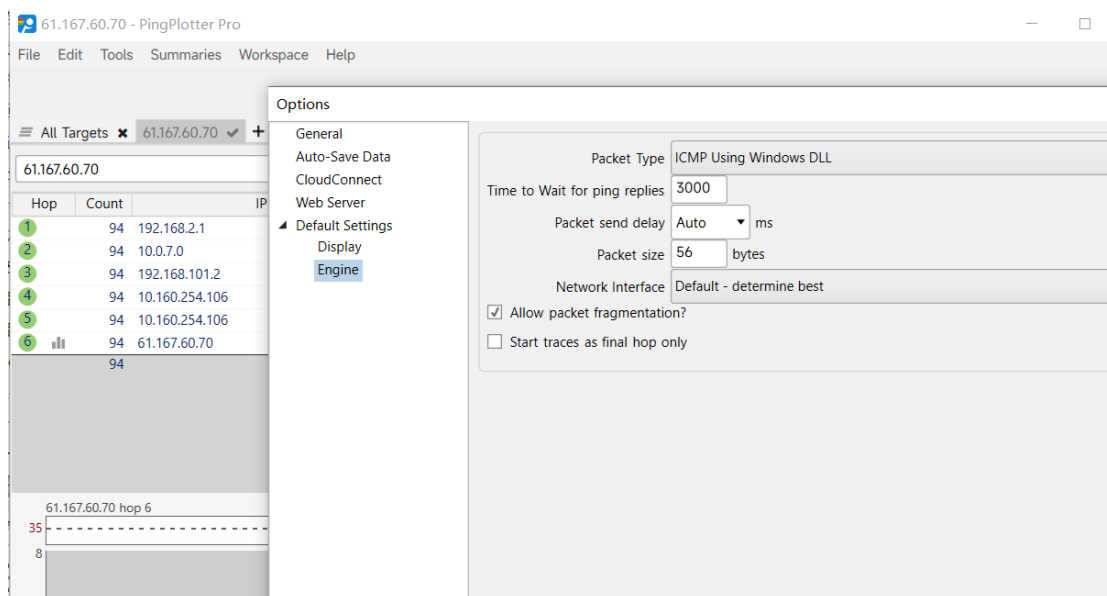
$$throughput = \frac{152933B}{(10.494647 - 9.420426)s} = 142,366.42Bps$$

536	10.494697	192.168.2.16	128.119.245.12	HTTP	1175	POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1 (text/plain)
539	10.498486	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=88813 Win=179560 Len=0
544	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=95813 Win=179584 Len=0
545	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=100013 Win=176640 Len=0
546	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=102813 Win=174592 Len=0
547	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=109813 Win=169728 Len=0
548	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=116813 Win=182144 Len=0
549	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=123813 Win=182144 Len=0
550	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=130813 Win=182144 Len=0
551	10.878103	128.119.2.16	192.168.2.16	TCP	60	80 → 60685 [ACK] Seq=1 Ack=132213 Win=186112 Len=0
						Frame 536: 1175 bytes on wire (9400 bits), 1175 bytes captured (9400 bits) on interface 0
						Ethernet II, Src: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad), Dst: 192.168.2.16 (02:00:0c:00:00:00)
						Internet Protocol Version 4, Src: 192.168.2.16, Dst: 128.119.245.12
						Transmission Control Protocol, Src Port: 60685, Dst Port: 80
						Source Port: 60685
						Destination Port: 80
						[Stream index: 21]
						[Stream Packet Number: 123]
						[Conversation completeness: Incomplete, DATA (1175 bytes)]
						[TCP Segment Len: 1121]
						Sequence Number: 151813 (relative sequence number 151813)
						Sequence Number (raw): 1070120424
						[Next Sequence Number: 152934 (relative sequence number 152934)]
						Acknowledgment Number: 1 (relative ack number 1)
						Acknowledgment number (raw): 1703746529

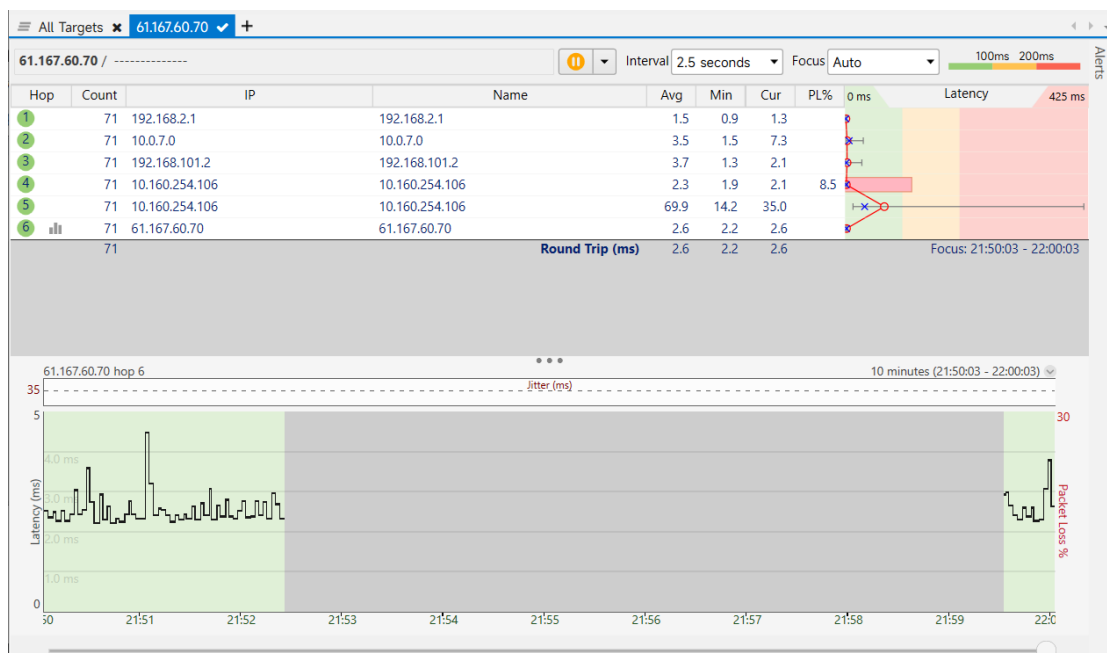
4. IP分析

首先安装Ping Plotter，由于默认的向www.hit.edu.cn发送包的时候使用的是ipv6，遂通过ping www.hit.edu.cn -4命令获得源ipv4地址为61.167.60.70，于是改向此ipv4地址发送包。

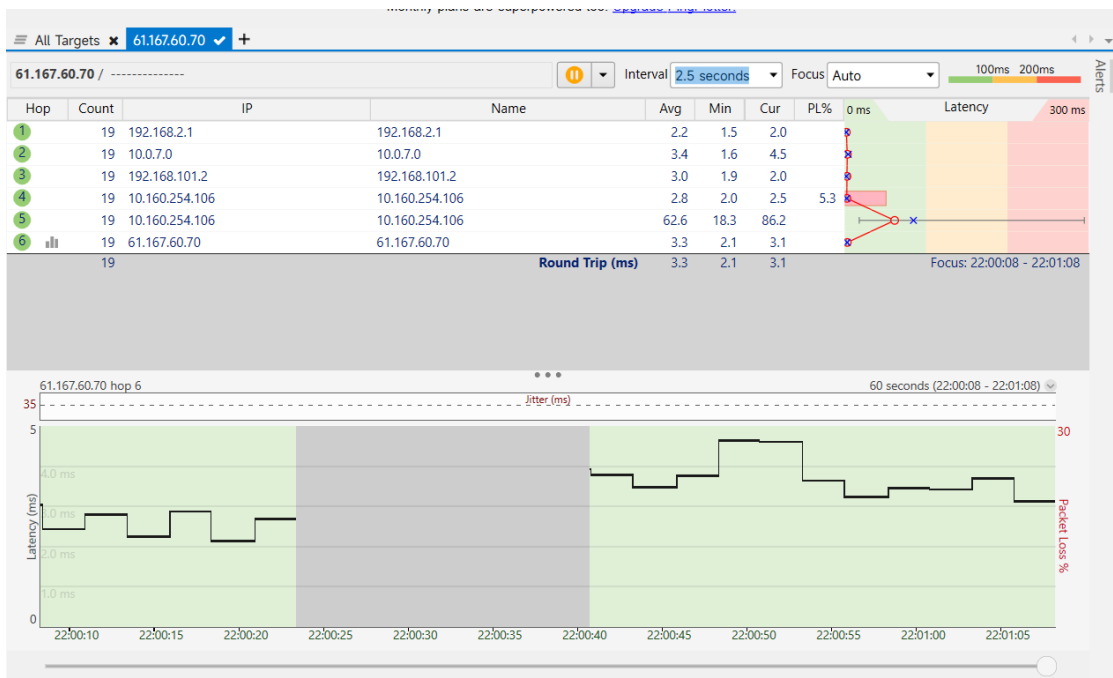
在edit -> options -> default settings -> engine中可以修改包的大小，但是并没有# of times to Trace选项。



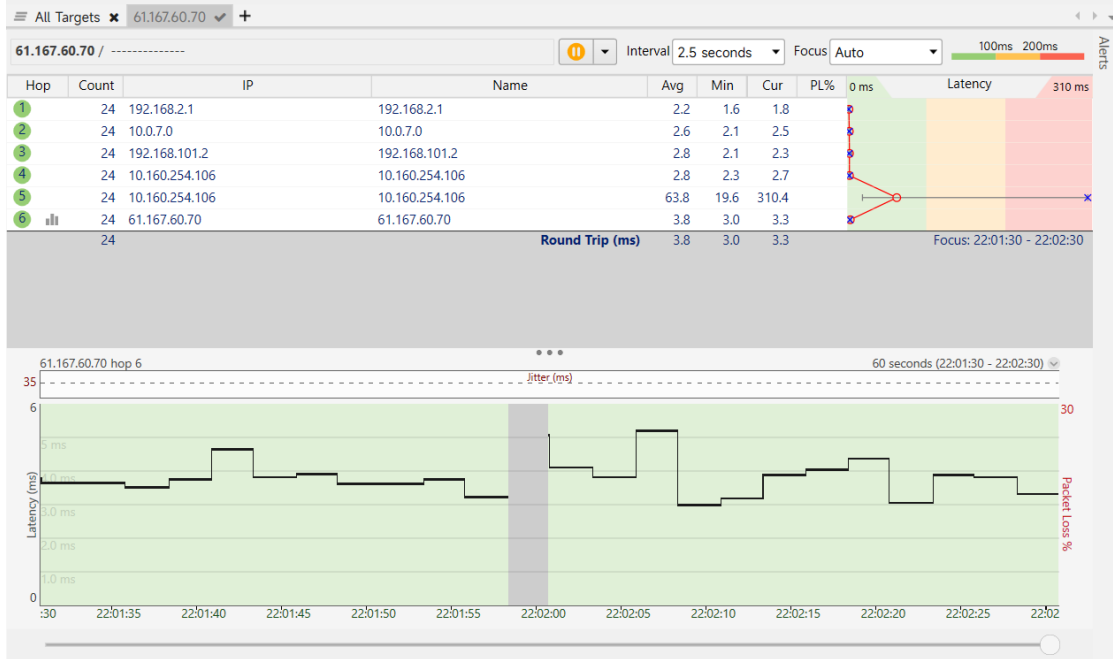
首先设置包的大小为56B。点击start按钮开始向服务器发送ICMP包。



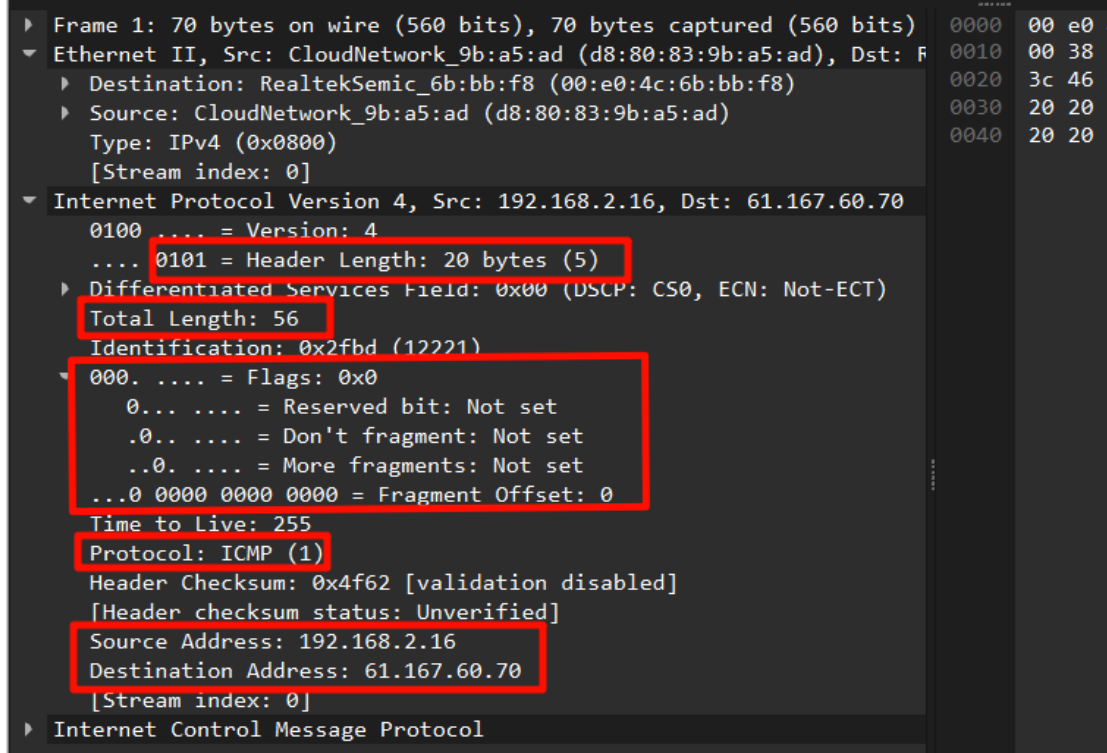
重新设置包的大小为2000B，再次发送。



重新设置包的大小为3500B，再次发送。



选择第一个我的主机发出的ICMP Echo Request消息，在packet details窗口展开数据包的Internet Protocol部分。



(一) 思考一，从上图可以看出

- 1) 本地主机的IP地址是192.168.2.16，服务器的IP地址是61.167.60.70。
- 2) IP数据包头中，上层协议的字段值是 1，此处表示的是ICMP协议。
- 3) IP头的总长度（Total Length字段）是56B，其中IP数据包的头部(Header Length字段)是20B，所以真正的净载字节的大小为56B-20B=36B。
- 4) IP数据包的净载大小为46B。
- 5) 该数据包没有分片，可以查看Flags标志中的DF（don't fragment）标志以及MF（more fragments）标志为0，标志该数据包可以被分片，并且没有被分片，或者该数据包为最后一篇，且Fragment Offset片位移为0，所以分组是没有被分片的

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1270/62980, ttl=255 (reply in 2)
5	0.038682	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1271/63235, ttl=1 (no response found!)
8	0.077715	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1272/63492, ttl=2 (no response found!)
11	0.115738	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1273/63748, ttl=3 (no response found!)
14	0.154925	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1274/64004, ttl=4 (no response found!)
16	0.193948	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1275/64260, ttl=5 (no response found!)
18	0.231962	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1276/64515, ttl=6 (reply in 29)
22	2.500812	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1277/64772, ttl=255 (reply in 23)
24	2.550874	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1278/65028, ttl=1 (no response found!)
26	2.601351	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1279/65284, ttl=2 (no response found!)
28	2.651934	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1280/5, ttl=3 (no response found!)
30	2.701868	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1281/261, ttl=4 (no response found!)
32	2.752558	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1282/517, ttl=5 (no response found!)
34	2.802591	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1283/773, ttl=6 (reply in 35)
38	5.001440	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1284/1029, ttl=255 (reply in 39)
40	5.051417	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1285/1285, ttl=1 (no response found!)
42	5.102475	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1286/1541, ttl=2 (no response found!)
44	5.152222	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1287/1797, ttl=3 (no response found!)
46	5.202266	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1288/2053, ttl=4 (no response found!)
48	5.252671	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1289/2309, ttl=5 (no response found!)
50	5.303131	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1290/2565, ttl=6 (reply in 53)
82	7.501743	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1291/2821, ttl=255 (reply in 83)
84	7.551951	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1292/3077, ttl=1 (no response found!)
86	7.602731	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1293/3333, ttl=2 (no response found!)
88	7.652844	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1294/3589, ttl=3 (no response found!)
90	7.703165	192.168.2.16	61.167.60.70	ICMP	70	Echo (ping) request id=0x0001, seq=1295/3845, ttl=4 (no response found!)

Wireshark packet capture analysis showing ICMP Echo (ping) requests. The top section shows packets 5, 8, 11, and 14, all with 'no response found!'. The bottom section shows packets 14, 16, 18, and 20, with packet 20 showing a successful 'reply in 19'. Detailed packet 8 and packet 11 are expanded to show IP and ICMP headers. In packet 8, the Identification field is 0x2fbf (12223) and TTL is 2. In packet 11, the Identification field is 0x2fc0 (12224) and TTL is 3. The ICMP header shows 'Echo (ping) request'.

(二) 思考二

对俘获的数据包按照Source IP进行排序，根据对应的数据包进行分析

- 1) 主机发出的ICMP中的IP数据包的identification、TTL、Header Checksum字段总是在改变。
- 2) IP数据报中的版本、首部长、标志位、协议、源IP地址和目的IP地址等字段总保持不变，是为了确保数据包能被正确解析、路由，并确保通信的完整性和可靠性。如果这些关键字段在传输过程中发生变化，可能会导致数据包无法正常送达或被错误处理。因此这些字段都应当保持一致。TTL变化的原因是：每经过一个路由器递减 1，防止数据包无限循环。Identification变化的原因是：每次发送新的数据包时递增，用于标识和重组分片。Header Checksum变化的原因是：头部字段（如TTL）变化后，需要重新计算校验和。图中可以看出TTL从1-5都没有正确回复，当TTL为6时才有正确回复以及帧序号。
- 3) IP数据包中的Identification字段是以16进制数的形式出现的，并且在相邻的两个两个IP数据包中，这个字段是逐步的加一的递增的。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.2...	61.167.60...	ICMP	70	Echo (ping) request id=0x0001, seq=1270/62980, ttl=255 (reply in 2)
2	0.002762	61.167.60...	192.168.2...	ICMP	70	Echo (ping) reply id=0x0001, seq=1270/62980, ttl=123 (request in 1)
5	0.008682	192.168.2...	61.167.60...	ICMP	70	Echo (ping) request id=0x0001, seq=1271/63236, ttl=1 (no response found!)
6	0.040698	192.168.2...	192.168.2...	ICMP	98	Time-to-live exceeded (Time to live exceeded in transit)
8	0.077715	192.168.2...	61.167.60...	ICMP	70	Echo (ping) request id=0x0001, seq=1272/63492, ttl=2 (no response found!)
9	0.080369	10.0.7.0	192.168.2...	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
11	0.115738	192.168.2...	61.167.60...	ICMP	70	Echo (ping) request id=0x0001, seq=1273/63748, ttl=3 (no response found!)
12	0.118154	192.168.1...	192.168.2...	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
14	0.154925	192.168.2...	61.167.60...	ICMP	70	Echo (ping) request id=0x0001, seq=1274/64004, ttl=4 (no response found!)
15	0.157893	10.160.25...	192.168.2...	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
16	0.193948	192.168.2...	61.167.60...	ICMP	70	Echo (ping) request id=0x0001, seq=1275/64260, ttl=5 (no response found!)
17	0.213316	10.160.25...	192.168.2...	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

Frame 6: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on	0000	d8 80 83 9b a5 ad 00 e0 4c 6b bb f8 08 00 45 c0Lk....E
Ethernet II, Src: RealtekSemic_6b:bb:f8 (00:e0:4c:6b:bb:f8), Dst: Clc	0010	00 54 56 2f 00 00 40 01 9e 58 c0 a8 02 01 c0 a8	..TV...@..X.....
Internet Protocol Version 4, Src: 192.168.2.1, Dst: 192.168.2.16	0020	02 10 0b 00 f4 ff 00 00 00 00 45 00 38 2f beE 8/...
0100 = Version: 4	0030	00 00 01 01 4d 62 c0 a8 02 10 3d a7 3c 46 08 00Mb.....<F...
.... 0101 = Header Length: 20 bytes (5)	0040	31 46 00 01 04 f7 20 20 20 20 20 20 20 20 20	1F.....
Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)	0050	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
Total Length: 84	0060	20 20	
Identification: 0x562f (22063)			
0000 = Flags: 0x0			
... 0000 0000 0000 = Fragment Offset: 0			
Time to Live: 64			
Protocol: ICMP (1)			
Header Checksum: 0x9e58 [validation disabled]			
[Header checksum status: Unverified]			
Source Address: 192.168.2.1			
Destination Address: 192.168.2.16			
[Stream index: 2]			
Internet Control Message Protocol			

(三) 思考三

上图中是最近的第一条路由器返回的ICMP的TTL exceeded消息。

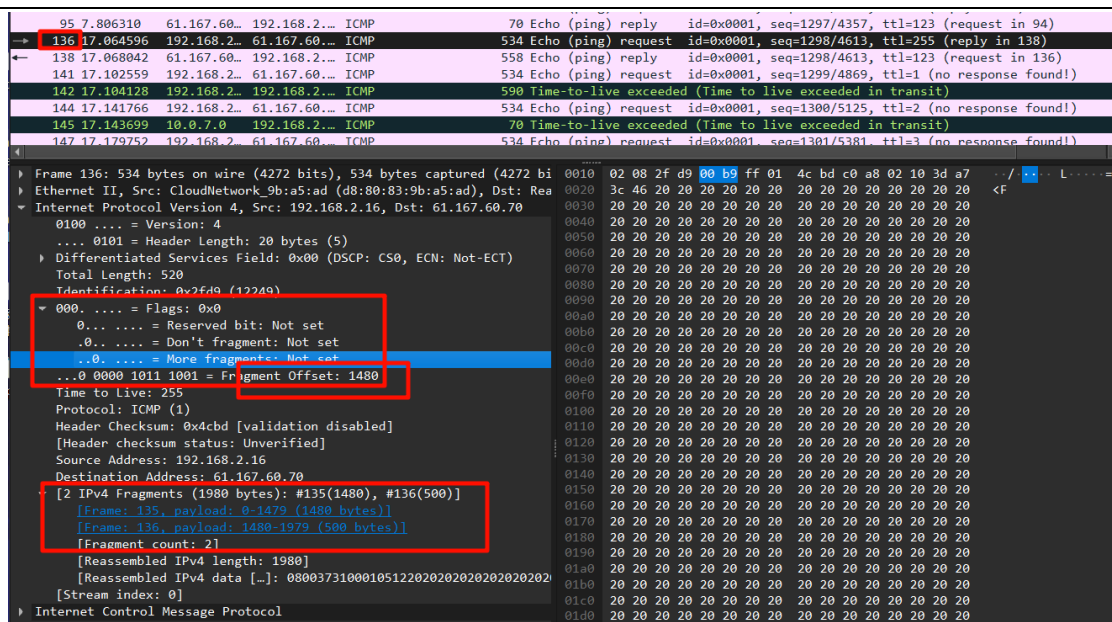
- 1) Identification字段是 0x562f (22063)，TTL字段是64。
- 2) 查看下面的图片，

Identification字段是会改变的，因为Identification（标识符）用于在 IP 分片时标识属于同一数据包的所有片段，确保接收方可以正确重组它们。如果数据报没有分片要求，Identification 仍会分配一个值，作为该数据包的唯一标识。但是生成策略与具体的系统有关。部分嵌入式系统以及windows可能会采用随机数生成，而linux系统可能会采用顺序Identification值。而TTL字段的初始值的选择也与最近的路由的操作系统有关，Linux下此值通常为64,也就是说当前的网络下的路由器可能是Linux系统或者其他嵌入式类型的系统。

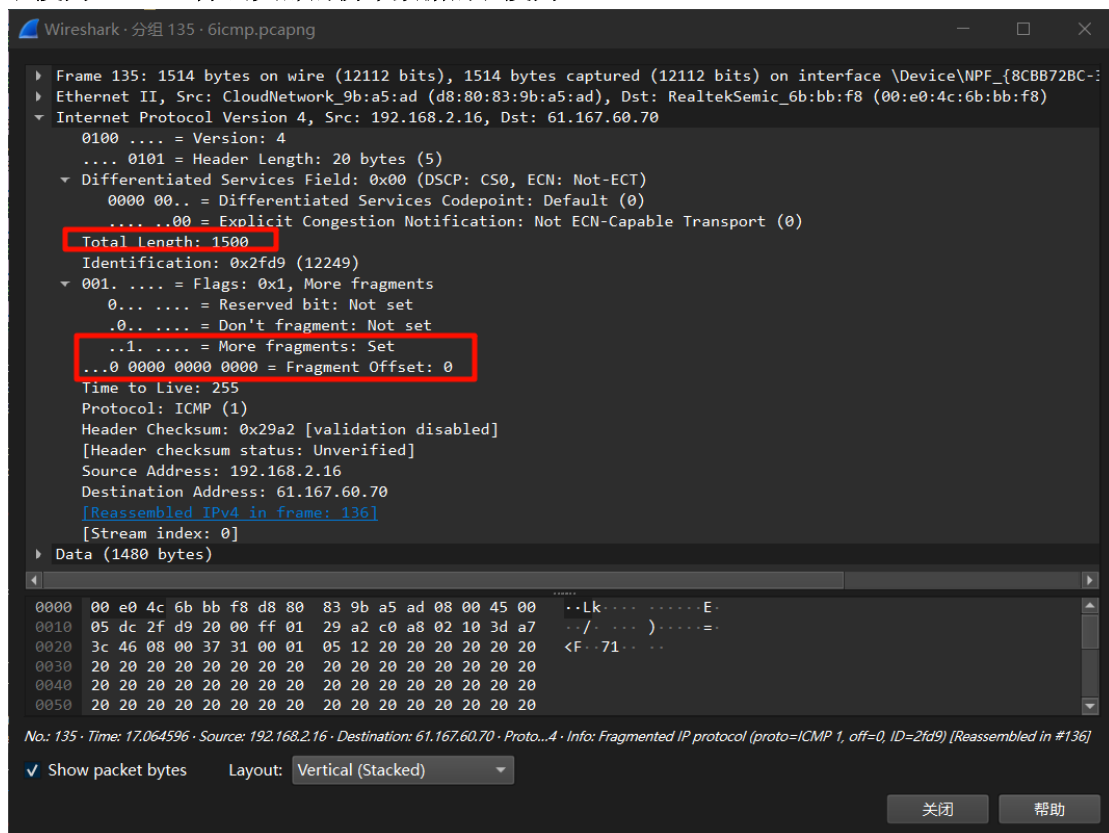
Frame 25: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) o	0000	d8 80 83 9b a5 ad 00 e0 4c 6b bb f8 08 00 45 c0Lk....E
Ethernet II, Src: RealtekSemic_6b:bb:f8 (00:e0:4c:6b:bb:f8), Dst: Clo	0010	00 54 56 2f 00 00 40 01 9e 58 c0 a8 02 01 c0 a8	..TV...@..X.....
Internet Protocol Version 4, Src: 192.168.2.1, Dst: 192.168.2.16	0020	02 10 0b 00 f4 ff 00 00 00 00 45 00 38 2f beE 8/...
0100 = Version: 4	0030	00 00 01 01 4d 62 c0 a8 02 10 3d a7 3c 46 08 00Mb.....<F...
.... 0101 = Header Length: 20 bytes (5)	0040	31 46 00 01 04 f7 20 20 20 20 20 20 20 20 20	1F.....
Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)	0050	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
Total Length: 84	0060	20 20	
Identification: 0x5857 (22615)			
0000 = Flags: 0x0			
... 0000 0000 0000 = Fragment Offset: 0			
Time to Live: 64			
Protocol: ICMP (1)			
Header Checksum: 0x9c30 [validation disabled]			
[Header checksum status: Unverified]			
Source Address: 192.168.2.1			
Destination Address: 192.168.2.16			
[Stream index: 2]			
Internet Control Message Protocol			

(四) 思考四

包的大小被更改为2000字节后的第一个ICMP Echo Request消息如图所示：

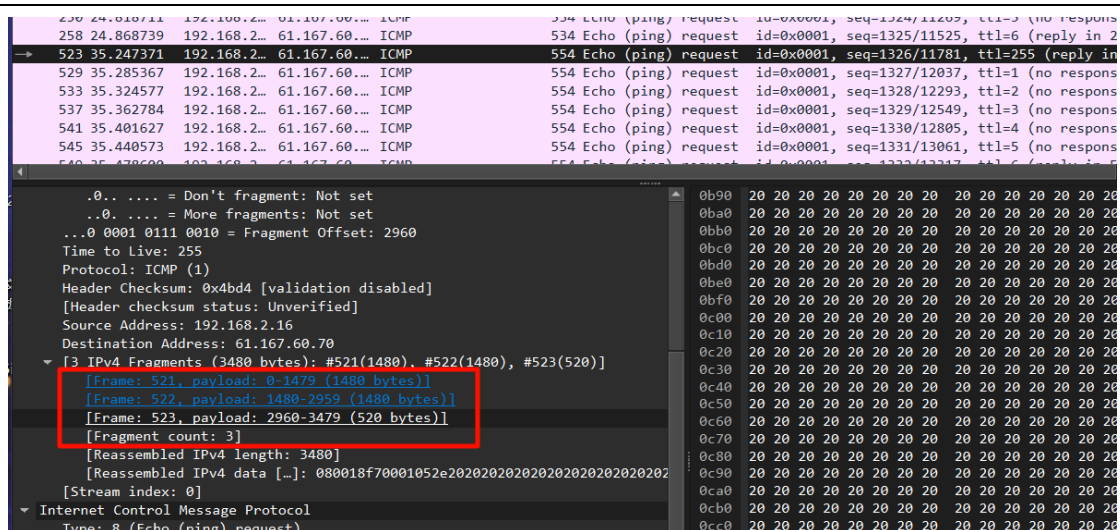


- 1) 此消息被分解为了多个IP数据包
- 2) 可以看见IP头部中的don't fragment和More fragment的值为0（此处为0是因为查看的是合并后的IP数据包，下图中含有分片后单独的数据包。），但是Fragment Offset值为1480，更多的可以在下面的内容中看见两片IPv4的分片，长度分别为1480B和500B，所以该消息被分片了。分别查看分片，对于分片1来说，offset为0，并且more fragments 字段为1。并且分片1的总长度为1500B，除去头部的携带数据的长度为1480B。

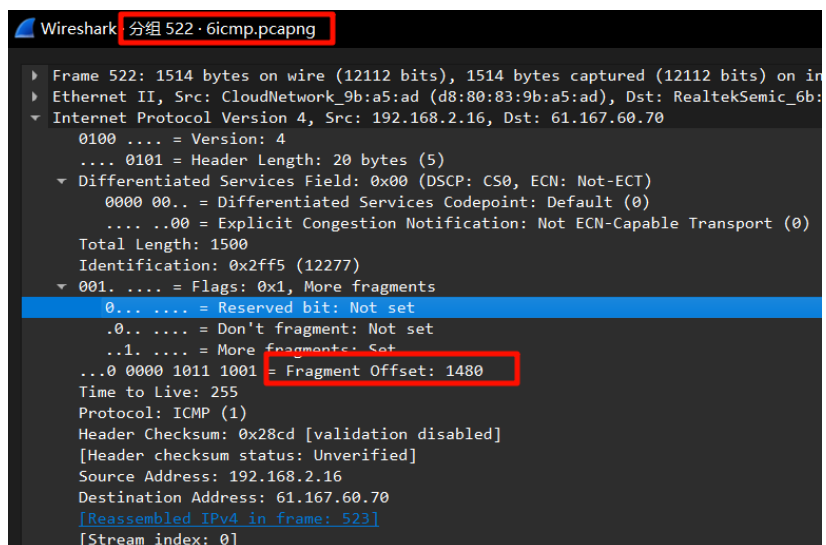
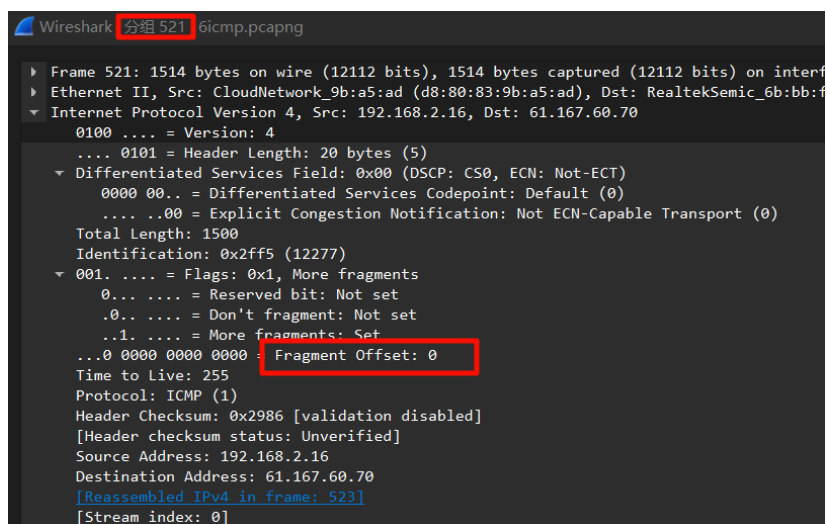


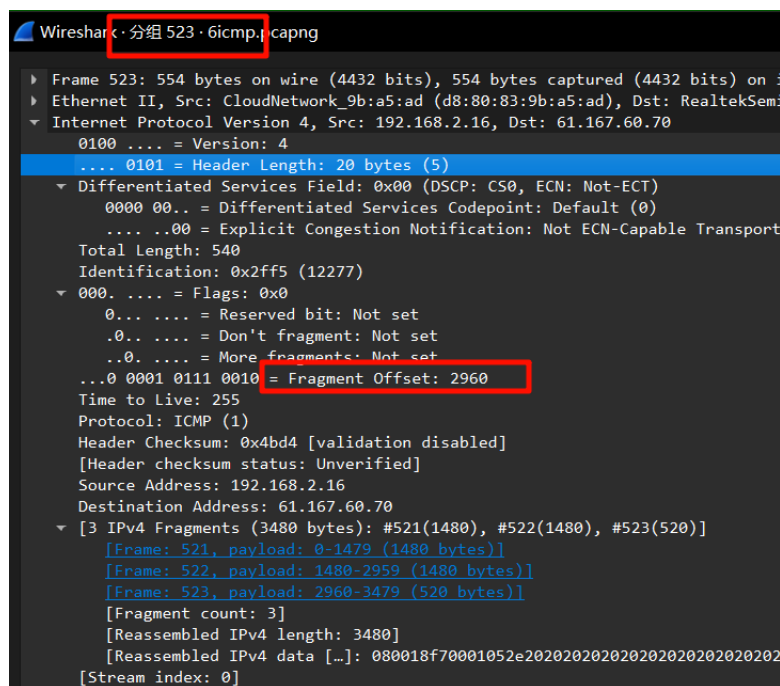
(五) 思考五

搜索到发送包大小为3500B的ICMP数据包：



- 1) 首先从图中可以看出，这些数据包被分成了三片。
- 2) 分别查看三个分片中的IP数据包头部，





从中可以看出，这三个分片的一般字段与identification字段的值都相同，标志位、总长度和片偏移发生了变化。前两个分片的标志位为001（0x1），总长度为1500，第三个分片的标志位是000，总长度是540，三个分片的片偏移分别为0、1480和2960。

5. 抓取ARP数据包

- (1) 输入命令 `arp -a` 查看当前主机的arp缓存，如下面两张图所示。ARP缓存的第一列是当前局域网设备的IP地址，第二列是上述IP地址对应的MAC地址，以16进制表示。第三列表示ARP缓存的类型，dynamic表示缓存项是通过 ARP 请求动态获取的，在一段时间后会过期，Static表示缓存项是通过静态配置添加的，不会自动过期。

接口: 192.168.56.1 --- 0x18		
Internet 地址	物理地址	类型
192.168.56.255	ff-ff-ff-ff-ff-ff	静态
224.0.0.2	01-00-5e-00-00-02	静态
224.0.0.22	01-00-5e-00-00-16	静态
224.0.0.167	01-00-5e-00-00-a7	静态
224.0.0.251	01-00-5e-00-00-fb	静态
224.0.0.252	01-00-5e-00-00-fc	静态
224.32.32.72	01-00-5e-20-20-48	静态
239.255.255.250	01-00-5e-7f-ff-fa	静态
239.255.255.251	01-00-5e-7f-ff-fb	静态
接口: 192.168.224.1 --- 0x29		
Internet 地址	物理地址	类型
192.168.239.255	ff-ff-ff-ff-ff-ff	静态
224.0.0.2	01-00-5e-00-00-02	静态
224.0.0.22	01-00-5e-00-00-16	静态
224.0.0.167	01-00-5e-00-00-a7	静态
224.0.0.251	01-00-5e-00-00-fb	静态
224.32.32.72	01-00-5e-20-20-48	静态
239.255.255.250	01-00-5e-7f-ff-fa	静态
239.255.255.251	01-00-5e-7f-ff-fb	静态
255.255.255.255	ff-ff-ff-ff-ff-ff	静态
接口: 169.254.83.107 --- 0x4a		
Internet 地址	物理地址	类型
224.0.0.22		静态
224.0.0.251		静态
239.255.255.250		静态

接口: 192.168.2.16 --- 0xc		
Internet 地址	物理地址	类型
192.168.2.1	00-e0-4c-6b-bb-f8	动态
192.168.2.8	88-a4-c2-c0-c8-9e	动态
192.168.2.9	24-cf-24-cf-75-91	动态
192.168.2.20	76-4c-f4-ef-59-ff	动态
192.168.2.24	70-32-17-cc-f7-31	动态
192.168.2.25	d4-5d-64-36-bc-5d	动态
192.168.2.255	ff-ff-ff-ff-ff-ff	静态
224.0.0.2	01-00-5e-00-00-02	静态
224.0.0.22	01-00-5e-00-00-16	静态
224.0.0.167	01-00-5e-00-00-a7	静态
224.0.0.251	01-00-5e-00-00-fb	静态
224.0.0.252	01-00-5e-00-00-fc	静态
224.32.32.72	01-00-5e-20-20-48	静态
239.255.255.250	01-00-5e-7f-ff-fa	静态
239.255.255.251	01-00-5e-7f-ff-fb	静态
255.255.255.255	ff-ff-ff-ff-ff-ff	静态

- (2) 通过命令 `arp -d *` 删除本地的全部ARP缓存，通过对本地的192.168.2.14端口进行ping请求，并进行wireshark的抓包。

No.	Time	Source	Destination	Protocol	Length	Info
684	2.533056	CloudNetw...	Broadcast	ARP	42	Who has 192.168.2.1? Tell 192.168.2.16
685	2.534376	RealtekSe...	CloudNetw...	ARP	60	192.168.2.1 is at 00:e0:4c:6b:bb:f8
1718	6.015329	CloudNetw...	Broadcast	ARP	42	Who has 192.168.2.14? Tell 192.168.2.16
1731	6.091620	12:60:cc:...	CloudNetw...	ARP	42	192.168.2.14 is at 12:60:cc:79:ad:cd

Destination: Broadcast (ff:ff:ff:ff:ff:ff)	0000	ff ff ff ff ff d8 80	83 9b a5 ad 08 06 00 01
...1... = IG bit: Locally administered address (true)	0010	08 00 06 04 00 01 d8 80	83 9b a5 ad c0 a8 02 10
...1... = IG bit: Group address (multicast/broadcast)	0020	00 00 00 00 00 00 c0 a8	02 01

Source: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad)	...0... = IG bit: Globally unique address (factor 1)
...0... = IG bit: Individual address (unicast)	Type: ARP (0x0806)
[Stream index: 2]	Address Resolution Protocol (request)
Hardware type: Ethernet (1)	Protocol type: IPv4 (0x0800)
Hardware size: 6	Protocol size: 4
Opcode: request (1)	Sender MAC address: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad)
Sender IP address: 192.168.2.16	Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 192.168.2.1	

当前版本的Wireshark使用Ethernet II解析ARP包，Ethernet II的封装格式如图：

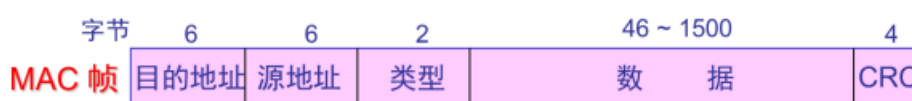


图 6-10 Ethernet II 的封装格式

以太网中的ARP请求和应答的分组格式如图：



- 1) ARP数据包的格式如上图所示，长度共28字节，由九部分组成，其中硬件类型部分占2字节，协议类型部分占2字节，硬件地址长度和协议地址长度分别占 1字节，OP字段占2字节，发送端MAC地址占6字节，发送端IP地址占4字节，目标MAC地址占6字节，目标IP地址占4字节。**和我们在wireshark抓包获取的长度完全一致。**
- 2) 通过OP字段来判断当前ARP数据包是请求包还是应答包，如下图所示，当OP字段为0x0001时表示为请求包，为0x0002时表示为应答包：

Address Resolution Protocol (reply) Hardware type: Ethernet (1) Protocol type: IPv4 (0x0800) Hardware size: 6 Protocol size: 4 Opcode: reply (2) Sender MAC address: RealtekSemic_6b:bb:f8 Sender IP address: 192.168.2.1 Target MAC address: CloudNetwork_9b:a5:ad Target IP address: 192.168.2.16	Address Resolution Protocol (request) Hardware type: Ethernet (1) Protocol type: IPv4 (0x0800) Hardware size: 6 Protocol size: 4 Opcode: request (1) Sender MAC address: CloudNetwork_9b:a5:ad Sender IP address: 192.168.2.16 Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00) Target IP address: 192.168.2.1
---	---

- 3) 因为ARP查询的时候，主机只知道目标的IP地址，不知道目标的MAC地址，只能通过广播MAC地址向所有的设备发送ARP请求，能够确保所有设备都可以接受到请求，当IP匹配的设备才会发出响应。而ARP响应的时候，由于ARP查询过程中包括了查询主机的IP地址和MAC地址，可以之间将应答确切的发送给主机，不需要进行广播过程。

6. 抓取UDP数据包

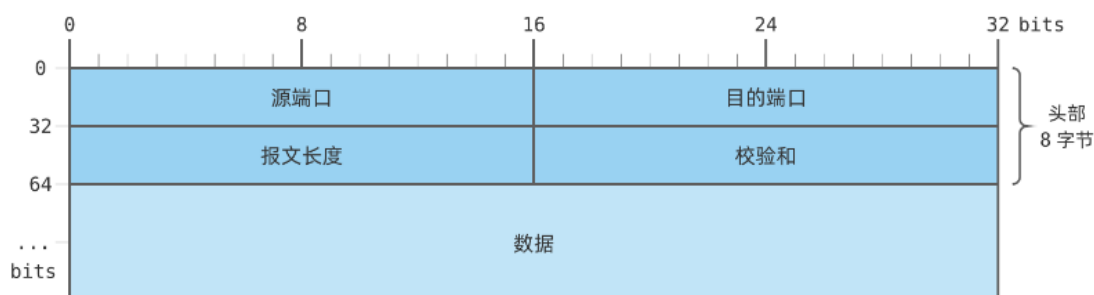
很不幸!!! QQ9的协议更新为新的名为TQ的协议, 这是一个基于TCP的协议, 所以实验无法正常进行, 我们选择其他方式进行实验。

我们借助QUIC协议进行测试, QUIC协议是一种由谷歌开发的新型网络传输协议, 它基于UDP实现, 提供了快速连接建立、多路复用、端到端加密、连接迁移等特性, 旨在降低延迟、提高吞吐量和可靠性, 以取代传统的TCP协议, 目前已被选为HTTP/3的底层传输协议。通过访问<https://http3check.net/about>, 可以通过wireshark抓去quic包。

No.	Time	Source	Destination	Protocol	Length	Info
785	1.290773	192.168.2.16	208.167.245.252	QUIC	1292	Initial, DCID=a4a9
786	1.290853	192.168.2.16	208.167.245.252	QUIC	1292	Initial, DCID=a4a9
822	1.421752	192.168.2.14	224.0.0.251	MDNS	161	Standard query 0x0
823	1.421774	fe80::10cd:7bae:48ea:e4da	ff02::fb	MDNS	181	Standard query 0x0
897	1.560597	208.167.245.252	192.168.2.16	QUIC	125	Retry, SCID=3ea836
898	1.560980	192.168.2.16	208.167.245.252	QUIC	1292	Initial, DCID=3ea8
899	1.561049	192.168.2.16	208.167.245.252	QUIC	1292	Initial, DCID=3ea8
1044	1.829361	208.167.245.252	192.168.2.16	QUIC	82	Initial, SCID=0793
1045	1.830002	208.167.245.252	192.168.2.16	QUIC	1294	Initial, SCID=0793
1046	1.830002	208.167.245.252	192.168.2.16	QUIC	82	Initial, SCID=0793
1047	1.830002	208.167.245.252	192.168.2.16	QUIC	1294	Handshake, SCID=07
1048	1.830002	208.167.245.252	192.168.2.16	QUIC	823	Handshake, SCID=07
1051	1.830609	192.168.2.16	208.167.245.252	QUIC	81	Handshake, DCID=07
1052	1.830660	192.168.2.16	208.167.245.252	QUIC	186	Protected Payload
1228	2.098040	208.167.245.252	192.168.2.16	QUIC	81	Handshake, SCID=07
1229	2.098388	208.167.245.252	192.168.2.16	QUIC	493	Protected Payload
1230	2.098388	208.167.245.252	192.168.2.16	QUIC	204	Protected Payload
1231	2.098565	192.168.2.16	208.167.245.252	QUIC	75	Protected Payload

Frame 899: 1292 bytes on wire (10336 bits), 1292 bytes captured (10336 bits) on interface \Device...
 Ethernet II, Src: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad), Dst: RealtekSemic_6b:bb:f8 (00:e0:4c...)
 Internet Protocol Version 4, Src: 192.168.2.16, Dst: 208.167.245.252
 User Datagram Protocol, Src Port: 56832, Dst Port: 443
QUIC IETF
 QUIC Connection information
 [Connection Number: 0]
 [Packet Length: 1250]
 1... = Header Form: Long Header (1)

- 1) Quic是基于udp协议的。
- 2) 本地主机ip地址为192.168.2.16, 目的主机的ip是208.167.245.252.
- 3) 我的端口号是56832, 目的主机端口号是443
- 4) UDP数据报的格式如下图所示: 其中源端口号大小为2B, 目的端口号大小为2B, UDP段长度大小为2B, 校验和大小为2B。



- 5) 和ICQ协议一致。QUIC协议在“发送一个ICQ数据包后, 服务器又返回给主机一个 ICQ数据包”这种情况下也是一致的, 这是因为在QUIC中, 当客户端向服务器发送数据包时, 服务器可以根据接收到的数据包进行响应。这是因为UDP提供的是不可靠的无连接的传输服务, 客户端无法确认服务器是否接收到信息, 因此需要一个QUIC报文表示收到 (ACK机制)。可以看出UDP是物链接的, 与TCP需要进行三次握手以建立连接不同, UDP不需要建立连接, 数据包可以直接发送。但QUIC提供了连接管理尽管底层使用UDP。在QUIC中, 连接是有状态的, 客户端和服务端之间通

过特定的连接ID保持状态，能够识别和管理不同的流和数据包。这使得QUIC具备了某种程度上的连接性，同时保留了UDP的高效性和低延迟。

7. 利用Wireshark进行DNS协议分析

按照实验指导书的要求，访问baidu.com并利用wireshark进行抓包，与DNS有关的结果如下：

No.	Time	Source	Destination	Protocol	Length	Info
1888	2.654345	114.114.114.114	192.168.2.16	DNS	173	Standard query response 0x58cf HTTPS dss0.bdstatic.com CNAME sslbaidu6.jomdns.com SOA ns1.jomod...
1889	2.654345	114.114.114.114	192.168.2.16	DNS	126	Standard query response 0x142d HTTPS dss1.bdstatic.com CNAME sslbaidu6.jomdns.com SOA ns1.jomod...
1890	2.654345	114.114.114.114	192.168.2.16	DNS	126	Standard query response 0x0341 A dss1.bdstatic.com CNAME sslbaidu6.jomdns.com A 111.40.186.33
1891	2.654943	114.114.114.114	192.168.2.16	DNS	126	Standard query response 0x058f A ssl.bdstatic.com CNAME sslbaidu6.jomdns.com A 111.40.186.32
1892	2.654943	114.114.114.114	192.168.2.16	DNS	173	Standard query response 0x129a HTTPS ssl.bdstatic.com CNAME sslbaidu6.jomdns.com SOA ns1.jomod...
1893	2.657290	114.114.114.114	192.168.2.16	DNS	138	Standard query response 0x1c99 AAAA dss0.bdstatic.com CNAME sslbaidu6.jomdns.com AAAA 2409:8c3c...
1894	2.657290	114.114.114.114	192.168.2.16	DNS	138	Standard query response 0x0063 AAAA dss1.bdstatic.com CNAME sslbaidu6.jomdns.com AAAA 2409:8c3c...
1895	2.658009	192.168.2.16	114.114.114.114	DNS	73	Standard query 0x0301 AAAA sp2.baidu.com
1896	2.658167	192.168.2.16	114.114.114.114	DNS	73	Standard query 0x0fd2 A sp2.baidu.com
1897	2.658301	192.168.2.16	114.114.114.114	DNS	73	Standard query 0xc08c HTTPS sp2.baidu.com
1898	2.658559	192.168.2.16	114.114.114.114	DNS	73	Standard query 0xea00 AAAA sp1.baidu.com
1899	2.658661	192.168.2.16	114.114.114.114	DNS	73	Standard query 0x75ed A sp1.baidu.com
1900	2.658760	192.168.2.16	114.114.114.114	DNS	73	Standard query 0xf4a4 HTTPS sp1.baidu.com
1901	2.659956	114.114.114.114	192.168.2.16	DNS	173	Standard query response 0xf98c AAAA ssl.bdstatic.com CNAME sslbaidu6.jomdns.com SOA ns1.jomod...
1902	2.660446	192.168.2.16	114.114.114.114	DNS	73	Standard query 0xb2fb AAAA sp0.baidu.com
1903	2.660586	192.168.2.16	114.114.114.114	DNS	73	Standard query 0x1bee A sp0.baidu.com
1904	2.660701	192.168.2.16	114.114.114.114	DNS	73	Standard query 0x2ca8 HTTPS sp0.baidu.com
1905	2.660722	114.114.114.114	192.168.2.16	DNS	132	Standard query response 0xfdf2 A sp2.baidu.com CNAME www.a.shifen.com A 39.156.66.14 A 39.156.66...
1906	2.660722	114.114.114.114	192.168.2.16	DNS	156	Standard query response 0x0301 AAAA sp2.baidu.com CNAME www.a.shifen.com AAAA 2409:8c00:6c21:104f...
1907	2.660722	114.114.114.114	192.168.2.16	DNS	157	Standard query response 0xc08c HTTPS sp2.baidu.com CNAME www.a.shifen.com SOA ns1.a.shifen.com
1908	2.660722	114.114.114.114	192.168.2.16	DNS	156	Standard query response 0xea00 AAAA sp1.baidu.com CNAME www.a.shifen.com AAAA 2409:8c00:6c21:1051...
1909	2.660722	114.114.114.114	192.168.2.16	DNS	132	Standard query response 0x75ed A sp1.baidu.com CNAME www.a.shifen.com A 39.156.66.14 A 39.156.66...
1910	2.661022	114.114.114.114	192.168.2.16	DNS	157	Standard query response 0xf4a4 HTTPS sp1.baidu.com CNAME www.a.shifen.com SOA ns1.a.shifen.com
1916	2.663378	114.114.114.114	192.168.2.16	DNS	156	Standard query response 0xb2fb AAAA sp0.baidu.com CNAME www.a.shifen.com AAAA 2409:8c00:6c21:104f...
1917	2.663378	114.114.114.114	192.168.2.16	DNS	157	Standard query response 0x2ca8 HTTPS sp0.baidu.com CNAME www.a.shifen.com SOA ns1.a.shifen.com
1918	2.663378	114.114.114.114	192.168.2.16	DNS	132	Standard query response 0x1bee A sp0.baidu.com CNAME www.a.shifen.com A 39.156.66.18 A 39.156.66...
1928	2.712540	192.168.2.16	114.114.114.114	DNS	77	Standard query 0xb4d1 AAAA dss2.bdstatic.com
1929	2.712740	192.168.2.16	114.114.114.114	DNS	77	Standard query 0x42b1 A dss2.bdstatic.com
1930	2.712802	192.168.2.16	114.114.114.114	DNS	77	Standard query 0xc302 HTTPS dss2.bdstatic.com
1931	2.713523	192.168.2.16	114.114.114.114	DNS	75	Standard query 0x7315 AAAA gim3.baidu.com
1932	2.713661	192.168.2.16	114.114.114.114	DNS	75	Standard query 0xe2a9 A gim3.baidu.com
1933	2.713842	192.168.2.16	114.114.114.114	DNS	75	Standard query 0x1da3 HTTPS gim3.baidu.com
1934	2.715338	114.114.114.114	192.168.2.16	DNS	138	Standard query response 0xb4d1 AAAA dss2.bdstatic.com CNAME sslbaidu6.jomdns.com AAAA 2409:8c3c...
1935	2.715338	114.114.114.114	192.168.2.16	DNS	173	Standard query response 0xc302 HTTPS dss2.bdstatic.com CNAME sslbaidu6.jomdns.com SOA ns1.jomod...
1936	2.715700	114.114.114.114	192.168.2.16	DNS	126	Standard query response 0x42b1 A dss2.bdstatic.com CNAME sslbaidu6.jomdns.com A 111.40.186.33
1937	2.715700	114.114.114.114	192.168.2.16	DNS	180	Standard query response 0x7315 AAAA gim3.baidu.com CNAME gim3.baidu.com a.bdydns.com CNAME open...
1938	2.715700	114.114.114.114	192.168.2.16	DNS	180	Standard query response 0x1da3 HTTPS gim3.baidu.com CNAME gim3.baidu.com a.bdydns.com CNAME open...

主机想要访问域名时，首先向本地域名服务器查询是否存储该域名对应IP的映射，若本地域名服务器中没有缓存，则由本地域名服务器继续查询，分为递归和迭代两种方式，本地域名服务器首先查询根域名服务器，由根域名服务器返回顶级域名服务器的IP地址，再继续查询。

其中，查询baidu.com的DNS报文详细信息如下：

No.	Time	Source	Destination	Protocol	Length	Info
1945	2.718039	192.168.2.16	114.114.114.114	DNS	82	Standard query 0x181d AAAA hectorstatic.baidu.com
1946	2.718803	192.168.2.16	114.114.114.114	DNS	82	Standard query 0x2e2d A hectorstatic.baidu.com
1947	2.718932	192.168.2.16	114.114.114.114	DNS	82	Standard query 0x72d3 HTTPS hectorstatic.baidu.com
1948	2.721595	114.114.114.114	192.168.2.16	DNS	176	Standard query response 0x2e2d A hectorstatic.baidu.com
1949	2.721595	114.114.114.114	192.168.2.16	DNS	188	Standard query response 0x181b AAAA hectorstatic.baidu.com
1950	2.724505	114.114.114.114	192.168.2.16	DNS	217	Standard query response 0x72d3 HTTPS hectorstatic.baidu.com
1976	2.755728	192.168.2.16	114.114.114.114	DNS	81	Standard query 0x3226 AAAA api.live.bilibili.com
1978	2.758152	114.114.114.114	192.168.2.16	DNS	304	Standard query response 0x3226 AAAA api.live.bilibili.com
2135	2.910885	192.168.2.16	114.114.114.114	DNS	73	Standard query 0x8dd9 AAAA hnd.baidu.com

Frame 1946: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_{8CB...} Ethernet II, Src: CloudNetwork_9b:a5:ad (d8:80:83:9b:a5:ad), Dst: RealtekSemic_6b:bb:f8 (00:e0:4c:6...)

Internet Protocol Version 4, Src: 192.168.2.16, Dst: 114.114.114.114

User Datagram Protocol, Src Port: 53402, Dst Port: 53

Domain Name System (query)

Transaction ID: 0x2e2d

Flags: 0x0100, Standard query

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

Queries

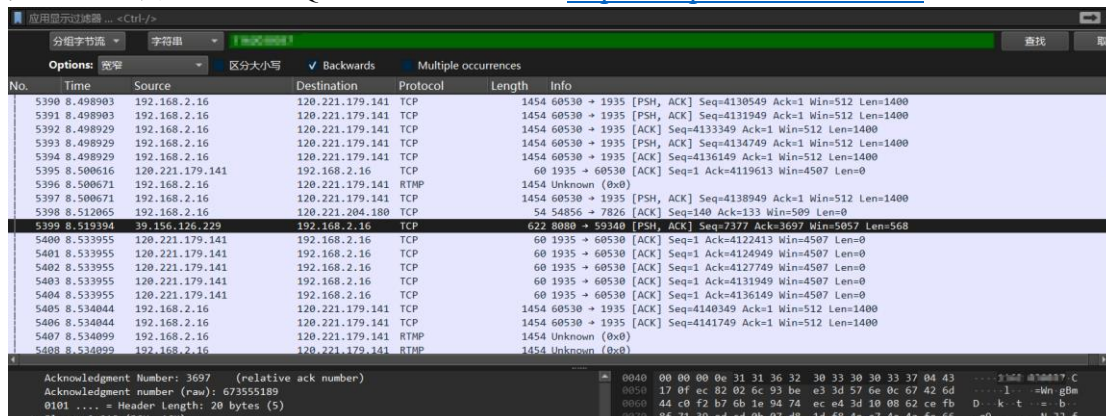
- hectorstatic.baidu.com: type A, class IN
 - Name: hectorstatic.baidu.com
 - [Name Length: 22]
 - [Label Count: 3]
 - Type: A (1) (Host Address)
 - Class: IN (0x0001)

[Response Tx: 1948]

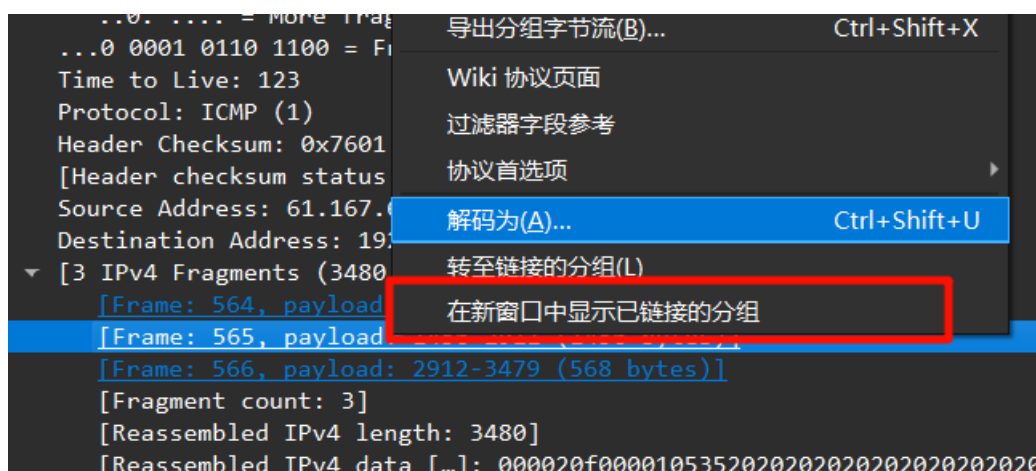
由上可知，本机IP是192.168.2.16，目的IP是114.114.114.114，源端口号是53402，目的端口号是53。并且由信息可知，DNS协议利用的是无连接不可靠的UDP服务，只标识了端口号、校验和等少量信息，不传输更多的控制信息。DNS通过TransactionID来标识查询和响应报文，并且同一查询对应的响应报文ID是相同的序号。对应的响应报文如上图所示，其中还标注了记录类型type为A，class为IN：

问题讨论：

- (1) <http://hitgs.hit.edu.cn/news>网址已经迁移，访问今日哈工大<http://today.hit.edu.cn>
- (2) 新版qq更改了消息传输协议，原有的udp+oicq的过滤方式无法找到，我通过qq号查找字节流获得了数据包，发现为tcp协议，qq协议已经更改为TCP，使用谷歌的QUIC协议完成实验内容，测试QUIC协议的网站是<https://http3check.net/about>。



- (3) 新版pingplotter在实现traceroute程序功能时，具体实现方法与实验指导书上不同，设置packet size的路径是edit>options>engine>packet size，通过修改此处并且重新点击页面中的绿色开始按钮，就可以发送设定长度大小的一系列数据包。
- (4) IP分析时无法查看到more fragment被设置为1的情况。需要单独通过对对应的分片在独立窗口进行打开，才能查看到不同的偏移和MF标志为1。



心得体会：

结合实验过程和结果给出实验的体会和收获。