浙江大学

本科实验报告

课程名称：	计算机网络基础
实验名称：	网络协议分析
姓　　名：	沈子衿
学　　院：	计算机学院
系：	软件工程
专　　业：	软件工程
学　　号：	3160104734
指导教师：	董玮

2018 年 12 月 20 日

浙江大学实验报告

实验名称：网络协议分析　　实验类型：分析实验

同组学生：林宇翔　　实验地点：计算机网络实验室

一、 实验目的

- 进一步学习使用 Wireshark 抓包工具。
- 观察和理解常见网络协议的交互过程
- 理解数据包分层结构和格式。

二、 实验内容

- 熟练掌握网络协议分析软件 Wireshark 的使用
- 观察所在网络出现的各类网络协议，了解其种类和分层结构
- 观察捕获到的数据包格式，理解各字段含义
- 根据要求配置 Wireshark，捕获某一类协议的数据包，并分析解读

三、 主要仪器设备

- 联网的 PC 机
- WireShark 协议分析软件

四、 操作方法与实验步骤

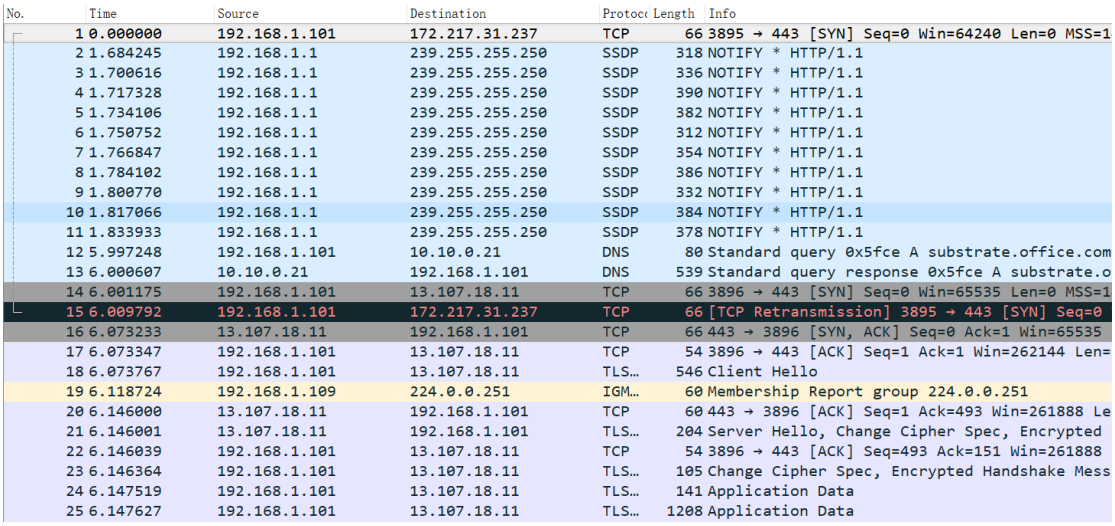
- 配置网络包捕获软件，捕获所有机器的数据包
- 观察捕获到的数据包，并对照解析结果和原始数据包
- 配置网络包捕获软件，只捕获特定 IP 或特定类型的包
- 抓取以下通信协议数据包，观察通信过程和数据包格式
  - ✓ PING：测试一个目标地址是否可达（在实验一基础上）
  - ✓ TRACE ROUTE：跟踪一个目标地址的途经路由（在实验一基础上）
  - ✓ NSLOOKUP：查询一个域名（在实验一基础上）
  - ✓ HTTP：访问一个网页
  - ✓ FTP：上传或下载一个文件
  - ✓ SMTP：发送一封邮件
  - ✓ POP3/IMAP：接收一封邮件
  - ✓ RTP：抓取一段音频流

提醒：为了避免捕获到大量无关数据包，影响实验观察，建议关闭所有无关软件。

五、 实验数据记录和处理

✧ Part One

- **打开 WireShark，开始捕获网络数据包后，你看到了什么？有哪些协议？**

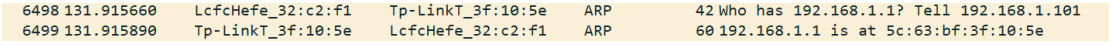


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	172.217.31.237	TCP	66	3895 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1
2	1.684245	192.168.1.1	239.255.255.250	SSDP	318	NOTIFY * HTTP/1.1
3	1.700616	192.168.1.1	239.255.255.250	SSDP	336	NOTIFY * HTTP/1.1
4	1.717328	192.168.1.1	239.255.255.250	SSDP	390	NOTIFY * HTTP/1.1
5	1.734106	192.168.1.1	239.255.255.250	SSDP	382	NOTIFY * HTTP/1.1
6	1.750752	192.168.1.1	239.255.255.250	SSDP	312	NOTIFY * HTTP/1.1
7	1.766847	192.168.1.1	239.255.255.250	SSDP	354	NOTIFY * HTTP/1.1
8	1.784102	192.168.1.1	239.255.255.250	SSDP	386	NOTIFY * HTTP/1.1
9	1.800770	192.168.1.1	239.255.255.250	SSDP	332	NOTIFY * HTTP/1.1
10	1.817066	192.168.1.1	239.255.255.250	SSDP	384	NOTIFY * HTTP/1.1
11	1.833933	192.168.1.1	239.255.255.250	SSDP	378	NOTIFY * HTTP/1.1
12	5.997248	192.168.1.101	10.10.0.21	DNS	80	Standard query 0x5fce A substrate.office.com
13	6.000607	10.10.0.21	192.168.1.101	DNS	539	Standard query response 0x5fce A substrate.o
14	6.001175	192.168.1.101	13.107.18.11	TCP	66	3896 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1
15	6.009792	192.168.1.101	172.217.31.237	TCP	66	[TCP Retransmission] 3895 → 443 [SYN] Seq=0
16	6.073233	13.107.18.11	192.168.1.101	TCP	66	443 → 3896 [SYN, ACK] Seq=0 Ack=1 Win=65535
17	6.073347	192.168.1.101	13.107.18.11	TCP	54	3896 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=
18	6.073767	192.168.1.101	13.107.18.11	TLS…	546	Client Hello
19	6.118724	192.168.1.109	224.0.0.251	IGM…	60	Membership Report group 224.0.0.251
20	6.146000	13.107.18.11	192.168.1.101	TCP	60	443 → 3896 [ACK] Seq=1 Ack=493 Win=261888 Le
21	6.146001	13.107.18.11	192.168.1.101	TLS…	204	Server Hello, Change Cipher Spec, Encrypted
22	6.146039	192.168.1.101	13.107.18.11	TCP	54	3896 → 443 [ACK] Seq=493 Ack=151 Win=261888
23	6.146364	192.168.1.101	13.107.18.11	TLS…	105	Change Cipher Spec, Encrypted Handshake Mess
24	6.147519	192.168.1.101	13.107.18.11	TLS…	141	Application Data
25	6.147627	192.168.1.101	13.107.18.11	TLS…	1208	Application Data

打开 Wireshark，选择网络接口为当前活跃的“以太网”，可以看到如下信息，包括数据包的序号、从开始捕获到收到对应数据包经历的时间、源地址、目的地址、协议类型和包中的信息。在被捕获的数据包中，有以下协议：TCP/SSDP/IGMPv2/TLSv1.2/MDNS/ICMPv6/DNS/ICMPv6/NBNS，也包含少数 OICQ 等协议。

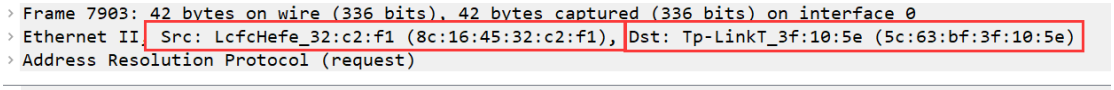
- **找一个包含 Ethernet 的数据包，这是什么协议？标出源和目标 MAC 地址。**

浏览捕获的数据包列表，发现本机和寝室路由器接口交互的 ARP 包：

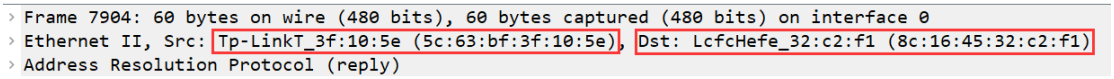


6498	131.915660	LcfcHefe_32:c2:f1	Tp-LinkT_3f:10:5e	ARP	42	Who has 192.168.1.1? Tell 192.168.1.101
6499	131.915890	Tp-LinkT_3f:10:5e	LcfcHefe_32:c2:f1	ARP	60	192.168.1.1 is at 5c:63:bf:3f:10:5e

查看这两个 ARP 包的详细信息，可以清楚地看到这两个 ARP 包涉及的源和目标 MAC 地址：



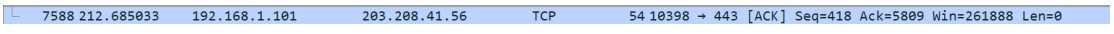
```
› Frame 7903: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
› Ethernet II, Src: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1), Dst: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e)
› Address Resolution Protocol (request)
```



```
› Frame 7904: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
› Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
› Address Resolution Protocol (reply)
```

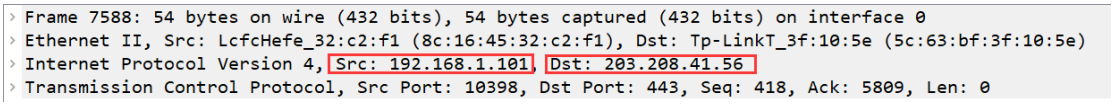
- **找一个包含 IP 的数据包，这是什么协议？标出源 IP 地址、目标 IP 地址。**

以此 TCP 数据包为例：



7588	212.685033	192.168.1.101	203.208.41.56	TCP	54	10398 → 443 [ACK] Seq=418 Ack=5809 Win=261888 Len=0

查看详细信息：



```
› Frame 7588: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
› Ethernet II, Src: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1), Dst: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e)
› Internet Protocol Version 4, Src: 192.168.1.101, Dst: 203.208.41.56
› Transmission Control Protocol, Src Port: 10398, Dst Port: 443, Seq: 418, Ack: 5809, Len: 0
```

可以看出，其源 IP 地址为 192.168.1.101，目的 IP 地址为 203.208.41.56.

- **找一个 ARP 数据包，这是请求还是应答？标注发送者的 MAC 地址。**

以之前的这组 arp 数据包为例：

```
6498 131.915660    LcfcHefe_32:c2:f1    Tp-LinkT_3f:10:5e    ARP          42 Who has 192.168.1.1? Tell 192.168.1.101
6499 131.915890    Tp-LinkT_3f:10:5e    LcfcHefe_32:c2:f1    ARP          60 192.168.1.1 is at 5c:63:bf:3f:10:5e

> Frame 7903: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
> Ethernet II, Src: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1), Dst: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e)
> Address Resolution Protocol (request)

> Frame 7904: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
> Address Resolution Protocol (reply)
```

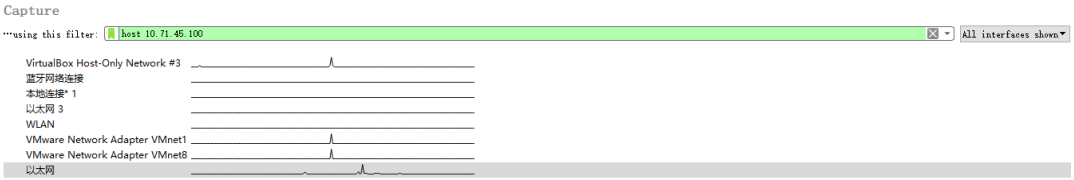
通过内容可以判断，前者为请求，后者为应答。前者中发送者的 MAC 地址为 8C164532C2F1，后者 MAC 地址为 5C63BF3F105E。

请在下面的每次捕获任务完成后，保存 Wireshark 抓包记录（.pcap 格式），随报告一起提交。每一个协议一个单独文件，文件名请取得便于理解。

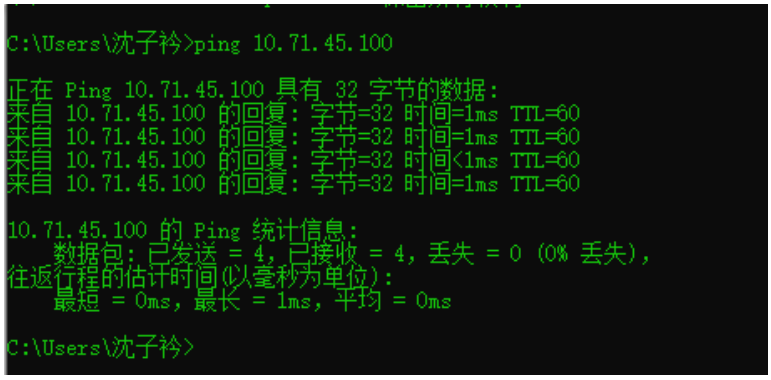
✧ Part Two

- **使用 Ping 命令，测试某个 IP 地址的连通性，并捕获这次的数据包。数据包由几层协议构成？分别是什么协议？选择一个请求包和一个响应包，展开最高层协议的详细内容，标出请求包和应答包、类型、序号。**

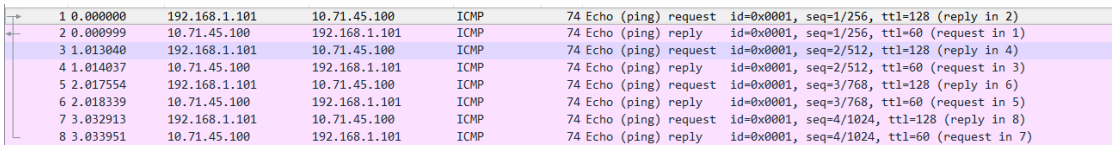
我们选择 ping 计算机学院课程网站 10.71.45.100，首先设置捕获参数如下：



开始捕获后，在 cmd 中输入 ping 命令：



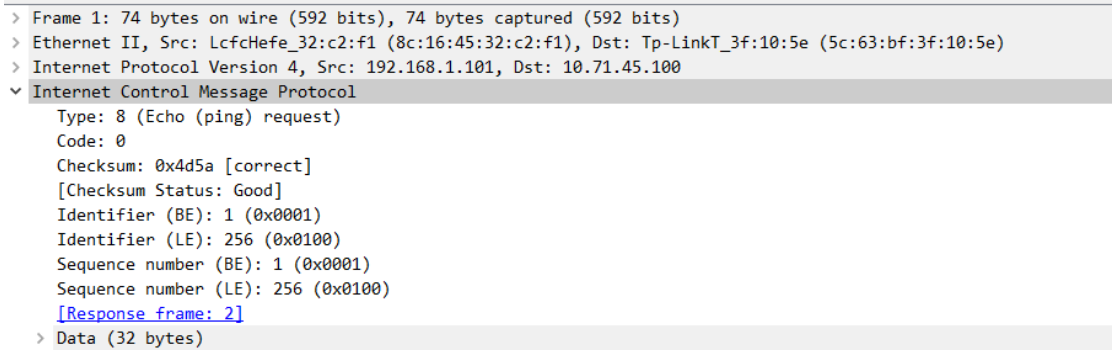
捕获数据包如下：



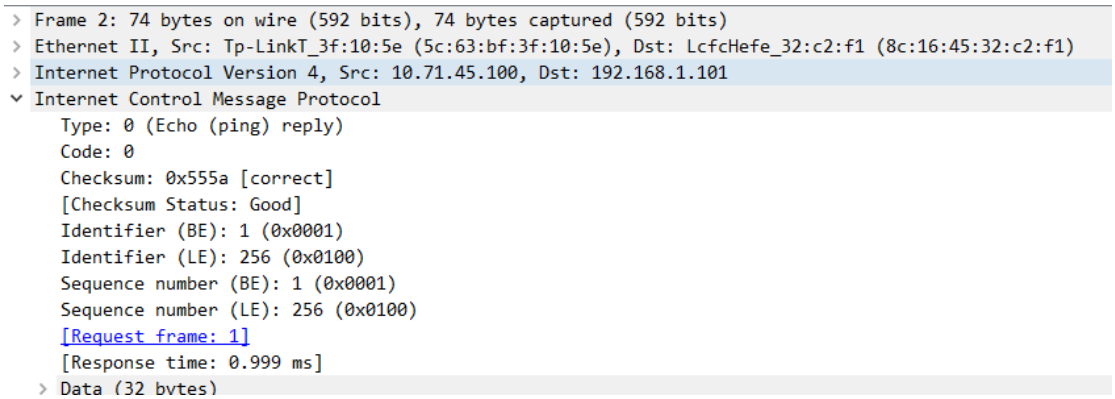
1	0.000000	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping) request id=0x0001, seq=1/256, ttl=128 (reply in 2)
2	0.000999	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=1/256, ttl=60 (request in 1)
3	1.013040	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping) request id=0x0001, seq=2/512, ttl=128 (reply in 4)
4	1.014037	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=2/512, ttl=60 (request in 3)
5	2.017554	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping) request id=0x0001, seq=3/768, ttl=128 (reply in 6)
6	2.018339	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=3/768, ttl=60 (request in 5)
7	3.032913	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping) request id=0x0001, seq=4/1024, ttl=128 (reply in 8)
8	3.033951	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping) reply id=0x0001, seq=4/1024, ttl=60 (request in 7)

可以看到 8 个数据包，分别是 4 次请求的请求、应答包。

以第一对请求、应答包为例：



```
> Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1), Dst: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e)
> Internet Protocol Version 4, Src: 192.168.1.101, Dst: 10.71.45.100
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d5a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
  > Data (32 bytes)
```



```
> Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
> Internet Protocol Version 4, Src: 10.71.45.100, Dst: 192.168.1.101
v Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x555a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Request frame: 1]
    [Response time: 0.999 ms]
  > Data (32 bytes)
```

可以看出，数据包均由以下协议组成：Ethernet II 协议，IPv4 协议和 ICMP 协议，其中最高层为 ICMP 协议。请求包和应答包的 ICMP 协议的详细内容如下：

```
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d5a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
  Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f707172737475767761...
      [Length: 32]
```

```
0000  5c 63 bf 3f 10 5e 8c 16  45 32 c2 f1 08 00 45 00   \c·?·^·· E2····E·
0010  00 3c 85 21 00 00 80 01  00 00 c0 a8 01 65 0a 47   ·<·!···· ·····e·G
0020  2d 64 08 00 4d 5a 00 01  00 01 61 62 63 64 65 66   -d··MZ·· ··abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                    wabcdefg hi
```

```
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x555a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Request frame: 1]
    [Response time: 0.999 ms]
  Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f707172737475767761...
      [Length: 32]
```

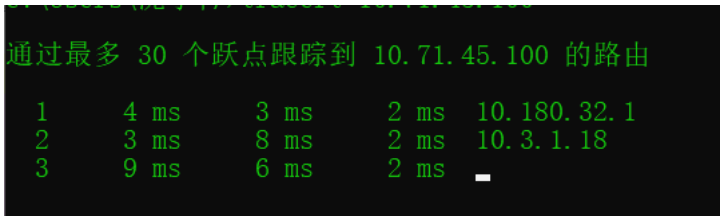
标出请求包和应答包：

No.	Time	Source	Destination	Protocol	Length	Info		请求or响应
1	0.000000	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping)	request	id=0x0001,
2	0.000999	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping)	reply	id=0x0001,
3	1.013040	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping)	request	id=0x0001,
4	1.014037	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping)	reply	id=0x0001,
5	2.017554	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping)	request	id=0x0001,
6	2.018339	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping)	reply	id=0x0001,
7	3.032913	192.168.1.101	10.71.45.100	ICMP	74	Echo (ping)	request	id=0x0001,
8	3.033951	10.71.45.100	192.168.1.101	ICMP	74	Echo (ping)	reply	id=0x0001,

序号　　协议

- **使用 Tracert 命令（Mac 下使用 Traceroute 命令），跟踪某个外部 IP 地址的路由，并捕获这次的数据包。数据包由几层协议构成？分别是什么协议？查看并标记多个请求包的 IP 协议层的 TTL 字段，发现了什么规律？选择一个请求包和一个响应包，展开最高层协议的详细内容，标出类型、序号等关键字段。与 Ping 命令的数据包有什么不同？**

同样选择 10.71.45.100 作为外部 IP 地址：

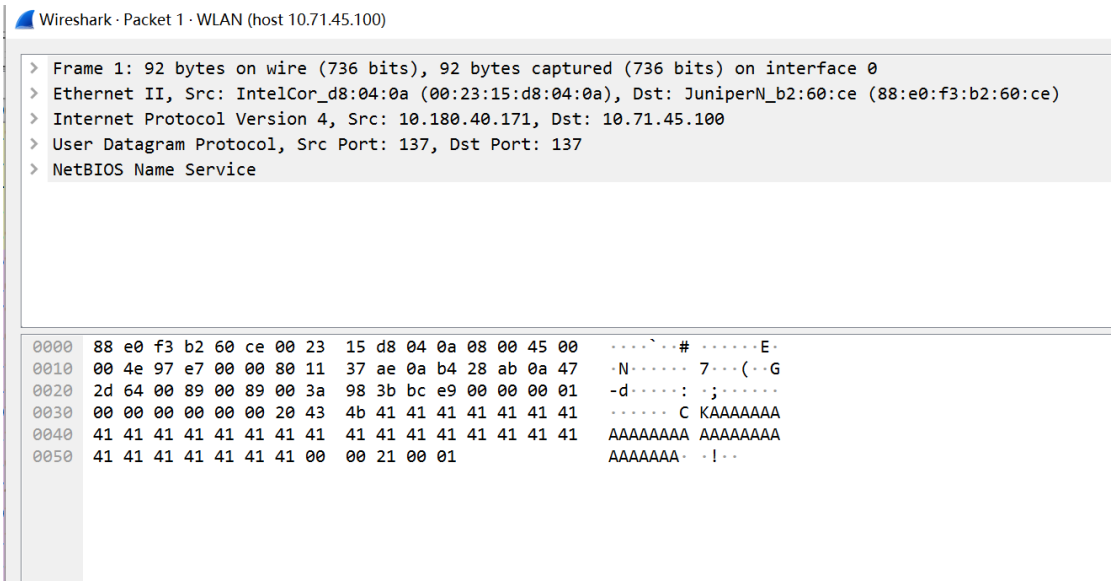


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
2	0.000299	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
3	1.504217	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
4	1.504436	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
5	3.005355	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
6	3.005595	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
7	4.549693	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=5/1280, ttl=1 (no response found!)
8	4.564103	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=6/1536, ttl=1 (no response found!)
9	4.570791	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=7/1792, ttl=1 (no response found!)
10	10.125090	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=8/2048, ttl=2 (no response found!)
11	10.131829	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=9/2304, ttl=2 (no response found!)
12	10.142893	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=10/2560, ttl=2 (no response found!)
13	15.680756	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=11/2816, ttl=3 (no response found!)
14	15.693705	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=12/3072, ttl=3 (no response found!)
15	15.703023	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=13/3328, ttl=3 (no response found!)
16	21.247421	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=14/3584, ttl=4 (reply in 17)
17	21.250794	10.71.45.100	10.180.40.171	ICMP	106	Echo (ping) reply id=0x0001, seq=14/3584, ttl=61 (request in 16)
18	21.251771	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=15/3840, ttl=4 (reply in 19)
19	21.253803	10.71.45.100	10.180.40.171	ICMP	106	Echo (ping) reply id=0x0001, seq=15/3840, ttl=61 (request in 18)
20	21.255155	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=16/4096, ttl=4 (reply in 21)
21	21.257634	10.71.45.100	10.180.40.171	ICMP	106	Echo (ping) reply id=0x0001, seq=16/4096, ttl=61 (request in 20)
22	21.265509	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
23	21.265716	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
24	22.775466	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
25	22.775709	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
26	24.289228	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
27	24.289632	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>

可以看出，整个过程涉及两类数据包：NBNS 和 ICMP。

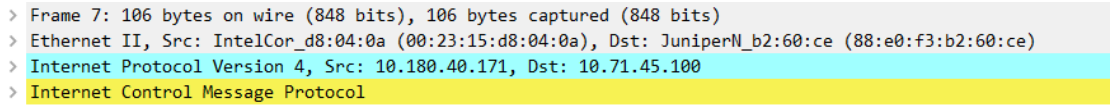
分别分析这两类数据包：

对于 NBNS 数据包，以第一个数据包为例：



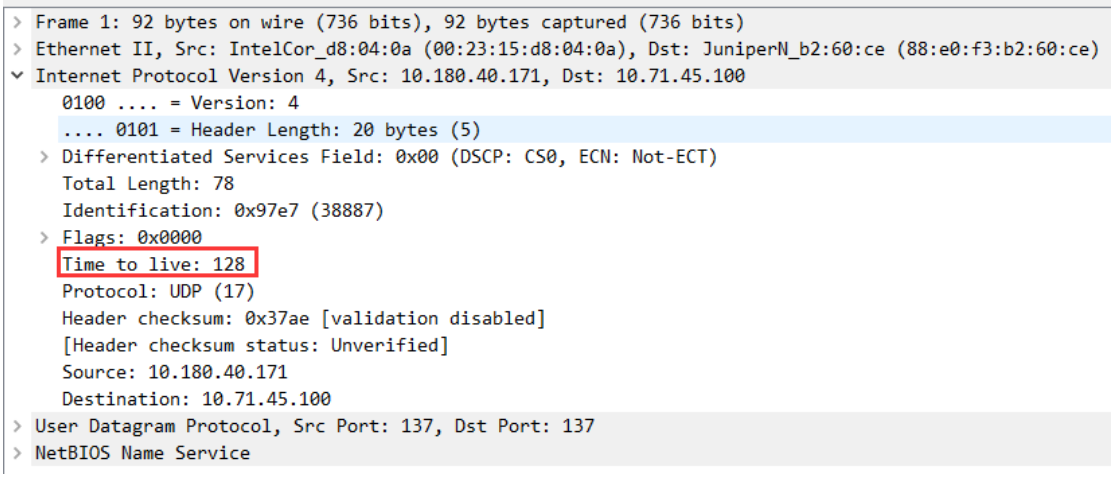
该数据包由四层协议构成，分别是：Ethernet II 协议，IPv4 协议，UDP 协议和 NBNS 协议。

对于 ICMP 数据包，以第 7 个为例：

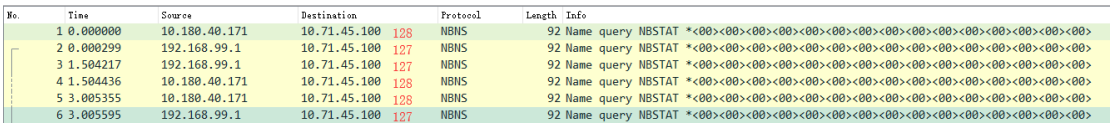


该数据包由三层协议构成，分别是：Ethernet II 协议，IPv4 协议和 ICMP 协议。

TTL 字段可以在这里查看：



依次查看各包 TTL 字段情况如下：



7 4.549693	10.180.40.171	10.71.45.100	1	ICMP	106 Echo (ping) request	id=0x0001, seq=5/1280, ttl=1 (no response found!)
8 4.564103	10.180.40.171	10.71.45.100	1	ICMP	106 Echo (ping) request	id=0x0001, seq=6/1536, ttl=1 (no response found!)
9 4.570791	10.180.40.171	10.71.45.100	1	ICMP	106 Echo (ping) request	id=0x0001, seq=7/1792, ttl=1 (no response found!)
10 10.125090	10.180.40.171	10.71.45.100	2	ICMP	106 Echo (ping) request	id=0x0001, seq=8/2048, ttl=2 (no response found!)
11 10.131829	10.180.40.171	10.71.45.100	2	ICMP	106 Echo (ping) request	id=0x0001, seq=9/2304, ttl=2 (no response found!)
12 10.142893	10.180.40.171	10.71.45.100	2	ICMP	106 Echo (ping) request	id=0x0001, seq=10/2560 ttl=2 (no response found!)
13 15.680756	10.180.40.171	10.71.45.100	3	ICMP	106 Echo (ping) request	id=0x0001, seq=11/2816 ttl=3 (no response found!)
14 15.693705	10.180.40.171	10.71.45.100	3	ICMP	106 Echo (ping) request	id=0x0001, seq=12/3072 ttl=3 (no response found!)
15 15.703023	10.180.40.171	10.71.45.100	3	ICMP	106 Echo (ping) request	id=0x0001, seq=13/3328 ttl=3 (no response found!)
16 21.247421	10.180.40.171	10.71.45.100	4	ICMP	106 Echo (ping) request	id=0x0001, seq=14/3584 ttl=4 (reply in 17)
17 21.250794	10.71.45.100	10.180.40.171	61	ICMP	106 Echo (ping) reply	id=0x0001, seq=14/3584 ttl=61 (request in 16)
18 21.251771	10.180.40.171	10.71.45.100	4	ICMP	106 Echo (ping) request	id=0x0001, seq=15/3840 ttl=4 (reply in 19)
19 21.253803	10.71.45.100	10.180.40.171	61	ICMP	106 Echo (ping) reply	id=0x0001, seq=15/3840 ttl=61 (request in 18)
20 21.255155	10.180.40.171	10.71.45.100	4	ICMP	106 Echo (ping) request	id=0x0001, seq=16/4096 ttl=4 (reply in 21)
21 21.257634	10.71.45.100	10.180.40.171	61	ICMP	106 Echo (ping) reply	id=0x0001, seq=16/4096 ttl=61 (request in 20)
22 21.265509	10.180.40.171	10.71.45.100	128	NBNS	92 Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>	
23 21.265716	192.168.99.1	10.71.45.100	127	NBNS	92 Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>	
24 22.775466	192.168.99.1	10.71.45.100	127	NBNS	92 Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>	
25 22.775709	10.180.40.171	10.71.45.100	128	NBNS	92 Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>	
26 24.289228	10.180.40.171	10.71.45.100	128	NBNS	92 Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>	
27 24.289632	192.168.99.1	10.71.45.100	127	NBNS	92 Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00>	

可以发现如下规律：NBNS 的数据包中，来源为 192.168.99.1 的包 TTL 均为 127，来源为 10.180.40.171 的包 TTL 均为 128。两拨 NBNS 包中，前三个为普通的 name query 包，后四者为广播的 name query 包；ICMP 数据包中，在远程主机未响应阶段（no response found!!），10.180.40.171 请求 10.71.45.100 的数据包的 TTL 不断增加，直到 10.71.45.100 响应，此时请求数据包的 TTL 为 4，而 3 次响应数据包的 TTL 固定为 61。

- **使用 nslookup 命令，查询某个域名，并捕获这次的数据包。数据包由几层协议构成？分别是什么协议？标记 UDP 协议层的端口字段。选择一个请求包和一个响应包，展开最高层协议的详细内容，标出类型、序号、域名信息。**

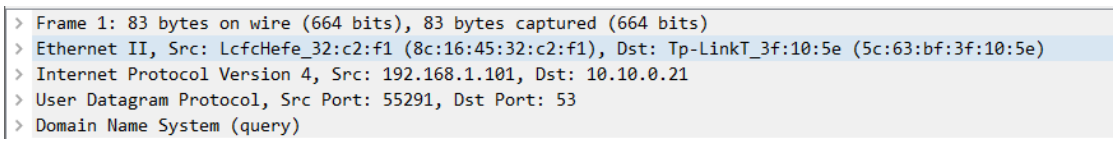
为了防止干扰，我们以计算机学院官网 www.cs.zju.edu.cn 为例：

结合实验一知识，配置捕获筛选器为 **host 10.10.0.21（DNS 服务器地址）**，因为查询域名对应 IP 并不涉及和域名对应主机的交互。

捕获数据包如下：

1 0.000000	192.168.1.101	10.10.0.21	DNS	83	Standard query 0x0001 PTR 21.0.10.10.in-addr.arpa
2 0.001211	10.10.0.21	192.168.1.101	DNS	142	Standard query response 0x0001 PTR 21.0.10.10.in-addr.arpa PTR dns1.zju.edu.cn NS dns1.zju.edu.cn A 10.10.0.8
3 0.003161	192.168.1.101	10.10.0.21	DNS	77	Standard query 0x0002 A www.cs.zju.edu.cn
4 0.004658	10.10.0.21	192.168.1.101	DNS	128	Standard query response 0x0002 A www.cs.zju.edu.cn A 10.202.70.61 NS dns1.zju.edu.cn A 10.10.0.8
5 0.004979	192.168.1.101	10.10.0.21	DNS	77	Standard query 0x0003 AAAA www.cs.zju.edu.cn
6 0.006210	10.10.0.21	192.168.1.101	DNS	123	Standard query response 0x0003 AAAA www.cs.zju.edu.cn SOA dns1.zju.edu.cn

以第一个数据包为例：



可以看到该数据包包含四层协议：Ethernet II 协议，IPv4 协议，UDP 协议和 DNS 协议。

UDP 协议层的端口号如下：

```
> Frame 1: 83 bytes on wire (664 bits), 83 bytes captured (664 bits)
> Ethernet II, Src: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1), Dst: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e)
> Internet Protocol Version 4, Src: 192.168.1.101, Dst: 10.10.0.21
v User Datagram Protocol, Src Port: 55291, Dst Port: 53
    Source Port: 55291
    Destination Port: 53
    Length: 49
    Checksum: 0xcc6e [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
> Domain Name System (query)
```

以第三个请求包和第四个响应包为例：

第三个请求包：

```
v Domain Name System (query)                                  类型
    Transaction ID: 0x0002                                    Transaction ID
  v Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > www.cs.zju.edu.cn: type A, class IN                     查询域名信息
    [Response In: 4]
```

第四个响应包：

```
v Domain Name System (response)                               协议名
    Transaction ID: 0x0002                                    Transaction ID（和请求的一致）
  v Flags: 0x8580 Standard query response, No error
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .1.. .... .... = Authoritative: Server is an authority for domain
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... 1... .... = Recursion available: Server can do recursive queries
      .... .... .0.. .... = Z: reserved (0)
      .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
      .... .... ...0 .... = Non-authenticated data: Unacceptable
      .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 1
    Additional RRs: 1
  v Queries
    > www.cs.zju.edu.cn: type A, class IN                     查询的域名信息
  v Answers
    > www.cs.zju.edu.cn: type A, class IN, addr 10.202.70.61  DNS服务器的答复
  v Authoritative nameservers
    > zju.edu.cn: type NS, class IN, ns dns1.zju.edu.cn
  v Additional records
    > dns1.zju.edu.cn: type A, class IN, addr 10.10.0.8
    [Request In: 3]
    [Time: 0.001497000 seconds]
```

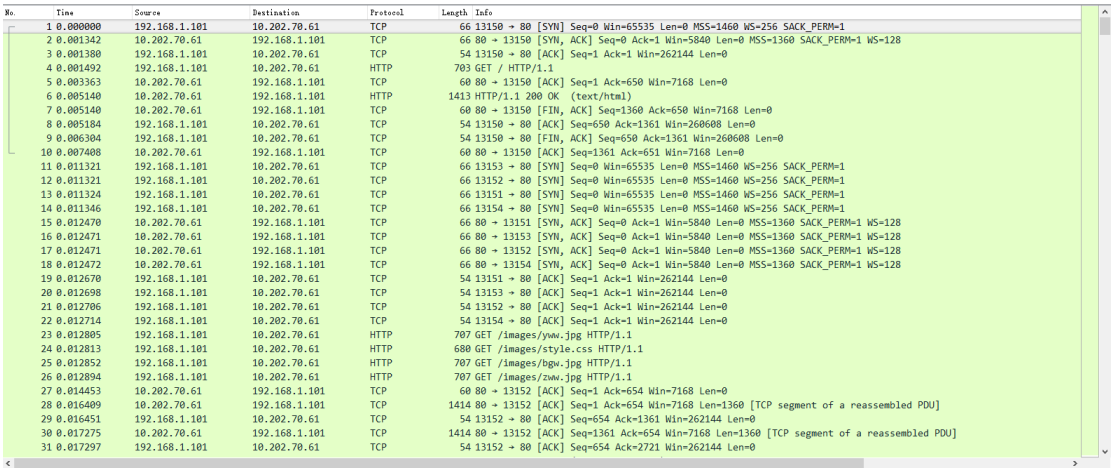
✧ Part Three

- **运行 ipconfig /flushdns 命令清空 DNS 缓存，然后打开浏览器，访问一个网页，并捕获这次的数据包（网页完全打开后，停止捕获）。数据包由几层协议构成？分别是什么协议？标出数据包的源和目标 IP 地址、源和目标端口。**

  首先刷新 DNS 缓存：

  然后，同样以访问 www.cs.zju.edu.cn 为例，配置 host www.cs.zju.edu.cn 捕获此次访问的数据包：

  待网页完全加载好之后，迅速停止捕获，以下是捕获的数据包（部分）



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	10.202.70.61	TCP	66	13150 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.001342	10.202.70.61	192.168.1.101	TCP	66	80 → 13150 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
3	0.001380	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
4	0.001492	192.168.1.101	10.202.70.61	HTTP	703	GET / HTTP/1.1
5	0.003363	10.202.70.61	192.168.1.101	TCP	60	80 → 13150 [ACK] Seq=1 Ack=650 Win=7168 Len=0
6	0.005140	10.202.70.61	192.168.1.101	HTTP	1413	HTTP/1.1 200 OK (text/html)
7	0.005140	10.202.70.61	192.168.1.101	TCP	60	80 → 13150 [FIN, ACK] Seq=1360 Ack=650 Win=7168 Len=0
8	0.005184	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [ACK] Seq=650 Ack=1361 Win=260608 Len=0
9	0.006304	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [FIN, ACK] Seq=650 Ack=1361 Win=260608 Len=0
10	0.007408	10.202.70.61	192.168.1.101	TCP	60	80 → 13150 [ACK] Seq=1361 Ack=651 Win=7168 Len=0
11	0.011321	192.168.1.101	10.202.70.61	TCP	66	13153 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
12	0.011321	192.168.1.101	10.202.70.61	TCP	66	13152 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
13	0.011324	192.168.1.101	10.202.70.61	TCP	66	13151 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
14	0.011346	192.168.1.101	10.202.70.61	TCP	66	13154 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
15	0.012470	10.202.70.61	192.168.1.101	TCP	66	80 → 13151 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
16	0.012471	10.202.70.61	192.168.1.101	TCP	66	80 → 13153 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
17	0.012471	10.202.70.61	192.168.1.101	TCP	66	80 → 13152 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
18	0.012472	10.202.70.61	192.168.1.101	TCP	66	80 → 13154 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
19	0.012670	192.168.1.101	10.202.70.61	TCP	54	13151 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
20	0.012698	192.168.1.101	10.202.70.61	TCP	54	13153 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
21	0.012706	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
22	0.012714	192.168.1.101	10.202.70.61	TCP	54	13154 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
23	0.012805	192.168.1.101	10.202.70.61	HTTP	707	GET /images/yww.jpg HTTP/1.1
24	0.012813	192.168.1.101	10.202.70.61	HTTP	680	GET /images/style.css HTTP/1.1
25	0.012852	192.168.1.101	10.202.70.61	HTTP	707	GET /images/bgw.jpg HTTP/1.1
26	0.012894	192.168.1.101	10.202.70.61	HTTP	707	GET /images/zww.jpg HTTP/1.1
27	0.014453	10.202.70.61	192.168.1.101	TCP	60	80 → 13152 [ACK] Seq=1 Ack=654 Win=7168 Len=0
28	0.016409	10.202.70.61	192.168.1.101	TCP	1414	80 → 13152 [ACK] Seq=1 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
29	0.016451	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=654 Ack=1361 Win=262144 Len=0
30	0.017275	10.202.70.61	192.168.1.101	TCP	1414	80 → 13152 [ACK] Seq=1361 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
31	0.017297	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=654 Ack=2721 Win=262144 Len=0

分析所有捕获的数据包，只涉及两类数据包：TCP 数据包和 HTTP 数据包。

其中，TCP 数据包的结构如下（以数据包 1 为例）：

```
> Frame 15: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:
> Internet Protocol Version 4, Src: 10.202.70.61, Dst: 192.168.1.101
> Transmission Control Protocol, Src Port: 80, Dst Port: 13151, Seq: 0, Ack: 1, Len: 0
```

可以看出，TCP 数据包由三层协议构成：Ethernet II 协议、IPv4 协议和 TCP 协议构成。

HTTP 数据包结构如下（以数据包 4 为例）：

```
> Frame 4: 703 bytes on wire (5624 bits), 703 bytes captured (5624 bits)
> Ethernet II, Src: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1), Dst: Tp-LinkT_3f:10:5e (5c:63:bf:
> Internet Protocol Version 4, Src: 192.168.1.101, Dst: 10.202.70.61
> Transmission Control Protocol, Src Port: 13150, Dst Port: 80, Seq: 1, Ack: 1, Len: 649
> Hypertext Transfer Protocol
```

可以看出，HTTP 数据包由四层协议构成：Ethernet II 协议、IPv4 协议、TCP 协议和 HTTP 协议(Hypertext Transfer Control Protocol)构成。

下面以数据包 4 为例分析源和目的：

```
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 10.202.70.61
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 689
    Identification: 0x093b (2363)
  > Flags: 0x4000, Don't fragment
    Time to live: 128
    Protocol: TCP (6)
    Header checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.101
    Destination: 10.202.70.61
```

源IP和目的IP

```
Transmission Control Protocol, Src Port: 13150, Dst Port: 80, Seq: 1, Ack: 1, Len: 649
    Source Port: 13150
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 649]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 650    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 1024
    [Calculated window size: 262144]
    [Window size scaling factor: 256]
    Checksum: 0x15b8 [unverified]
```

源端口和目的端口

- **找到建立 TCP 连接的三个数据包（称为三次握手），展开 TCP 协议层的 Flags 字段，分别标记三个数据包的 SYN 标志位和 ACK 标志位。**

观察包内容，找到建立 TCP 连接的三个数据包如下：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	10.202.70.61	TCP	66	13150 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.001342	10.202.70.61	192.168.1.101	TCP	66	80 → 13150 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
3	0.001380	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0

第一个数据包：

```
Transmission Control Protocol, Src Port: 13150, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 13150
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set      ACK
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    > .... .... ..1. = Syn: Set
        > [Expert Info (Chat/Sequence): Connection establish request (SYN): server port 80]    SYN
      .... .... ...0 = Fin: Not set
      [TCP Flags: ··········S·]
    Window size value: 65535
    [Calculated window size: 65535]
    Checksum: 0x133b [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation
  > [Timestamps]
```

第二个数据包：

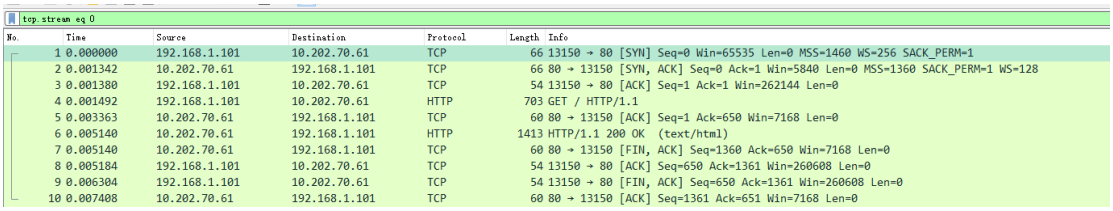
```
Transmission Control Protocol, Src Port: 80, Dst Port: 13150, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 13150
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x012 (SYN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set    ACK
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    > .... .... ..1. = Syn: Set        SYN
        > [Expert Info (Chat/Sequence): Connection establish acknowledge (SYN+ACK): server port 80]
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····A··S·]
    Window size value: 5840
    [Calculated window size: 5840]
    Checksum: 0xea78 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (
  > [SEQ/ACK analysis]
  > [Timestamps]
```

第三个数据包：

```
Destination: 10.202.70.61
Transmission Control Protocol, Src Port: 13150, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 13150
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set   ACK
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set   SYN
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····A····]
    Window size value: 1024
    [Calculated window size: 262144]
    [Window size scaling factor: 256]
    Checksum: 0x132f [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  [SEQ/ACK analysis]
  [Timestamps]
```

- **选择一个包，点击右键，选择跟踪一个 TCP 流，截取完整的 HTTP 请求消息和部分响应消息，标记 HTTP 请求头部的 Method 字段、URI 字段和 Host 字段，标记 HTTP 响应头部的 Status Code 字段、Content-Type 和 Content-Length 字段，以及区分响应头部和体部的标记（单独的回车换行符）。**

选择跟踪数据包 1 的 TCP 流，结果如下：



根据题目要求作出标记：

```
GET / HTTP/1.1  Method
                URI
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-Hans-CN,zh-Hans;q=0.8,ja;q=0.6,en-US;q=0.4,en;q=0.2
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140
Safari/537.36 Edge/18.17763
Accept-Encoding: gzip, deflate
Host: www.cs.zju.edu.cn   HOST
Connection: Keep-Alive
Cookie: 8a762667df5cb9d5_gr_last_sent_cs1=389652; 8a762667df5cb9d5_gr_cs1=389652;
Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1534080271; gr_user_id=c2eb516c-bdc6-4318-a725-e47a6ab9d36b;
grwng_uid=5afacac6-a6ae-4cec-994d-a139217545ce

HTTP/1.1 200 OK Status Code
Date: Fri, 21 Dec 2018 14:15:30 GMT
Server: Apache
Last-Modified: Mon, 12 Nov 2018 07:25:42 GMT
ETag: "4299b3c-45e-57a729bc82980"
Accept-Ranges: bytes
Content-Length: 1118  content-length
Connection: close
Content-Type: text/html  content-type
                         区分标记
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<link href="images/style.css" rel="stylesheet" type="text/css" />
<title>.....</title>
</head>
<body>
<div class="main">
          <div class="link">
          <a style="position:absolute; top:0; left:0" href="http://www.cs.zju.edu.cn/chinese/"><img src="images/
zww.jpg" width="141" height="36" /></a>
        <a style=" position:absolute; top:0; left:185px; display:block" href="http://cspo.zju.edu.cn/"><img
src="images/bgw.jpg" width="140" height="36" /></a>
        <a style=" position:absolute; top:0; right:0;" href="http://www.en.cs.zju.edu.cn/"><img src="images/
yww.jpg" width="140" height="36" /></a>
    </div>
</div>
<div style="clear:both; font-size:0"></div>
<div class="footer">
    <p>Copyright@ 2010 ..... ....?.....  ......+86-571-87953025 ......
+86-571-87951250</p>
</div>
</body>
</html>
```

- **使用过滤器 tcp.stream eq X，让 X 从 0 开始变化，直到没有数据。观察总共捕获到了几个 TCP 连接（一个 TCP 流对应一个 TCP 连接）？存在几个 HTTP 会话（一对 HTTP 请求和响应对应一次 HTTP 会话）？注意：一个 TCP 流上可能存在多个 HTTP 会话。**

Eq 0：

tcp.stream eq 0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	10.202.70.61	TCP	66	13150 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.001342	10.202.70.61	192.168.1.101	TCP	66	80 → 13150 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
3	0.001380	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
4	0.001492	192.168.1.101	10.202.70.61	HTTP	703	GET / HTTP/1.1
5	0.003363	10.202.70.61	192.168.1.101	TCP	60	80 → 13150 [ACK] Seq=1 Ack=650 Win=7168 Len=0
6	0.005140	10.202.70.61	192.168.1.101	HTTP	1413	HTTP/1.1 200 OK (text/html)
7	0.005140	10.202.70.61	192.168.1.101	TCP	60	80 → 13150 [FIN, ACK] Seq=1360 Ack=650 Win=7168 Len=0
8	0.005184	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [ACK] Seq=650 Ack=1361 Win=260608 Len=0
9	0.006304	192.168.1.101	10.202.70.61	TCP	54	13150 → 80 [FIN, ACK] Seq=650 Ack=1361 Win=260608 Len=0
10	0.007408	10.202.70.61	192.168.1.101	TCP	60	80 → 13150 [ACK] Seq=1361 Ack=651 Win=7168 Len=0

Eq 1：

tcp.stream eq 1

No.	Time	Source	Destination	Protocol	Length	Info
11	0.011321	192.168.1.101	10.202.70.61	TCP	66	13153 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
16	0.012471	10.202.70.61	192.168.1.101	TCP	66	80 → 13153 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
20	0.012698	192.168.1.101	10.202.70.61	TCP	54	13153 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
24	0.012813	192.168.1.101	10.202.70.61	HTTP	680	GET /images/style.css HTTP/1.1
34	0.018406	10.202.70.61	192.168.1.101	TCP	60	80 → 13153 [ACK] Seq=1 Ack=627 Win=7168 Len=0
37	0.019408	10.202.70.61	192.168.1.101	HTTP	1094	HTTP/1.1 200 OK (text/css)
38	0.019409	10.202.70.61	192.168.1.101	TCP	60	80 → 13153 [FIN, ACK] Seq=1041 Ack=627 Win=7168 Len=0
40	0.019450	192.168.1.101	10.202.70.61	TCP	54	13153 → 80 [ACK] Seq=627 Ack=1042 Win=260864 Len=0
43	0.021317	192.168.1.101	10.202.70.61	TCP	54	13153 → 80 [FIN, ACK] Seq=627 Ack=1042 Win=260864 Len=0
58	0.026419	10.202.70.61	192.168.1.101	TCP	60	80 → 13153 [ACK] Seq=1042 Ack=628 Win=7168 Len=0

Eq 2：

tcp.stream eq 2

No.	Time	Source	Destination	Protocol	Length	Info
12	0.011321	192.168.1.101	10.202.70.61	TCP	66	13152 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
17	0.012471	10.202.70.61	192.168.1.101	TCP	66	80 → 13152 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
21	0.012706	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
23	0.012805	192.168.1.101	10.202.70.61	HTTP	707	GET /images/yww.jpg HTTP/1.1
27	0.014453	10.202.70.61	192.168.1.101	TCP	60	80 → 13152 [ACK] Seq=1 Ack=654 Win=7168 Len=0
28	0.016409	10.202.70.61	192.168.1.101	TCP	1414	80 → 13152 [ACK] Seq=1 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
29	0.016451	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=654 Ack=1361 Win=262144 Len=0
30	0.017275	10.202.70.61	192.168.1.101	TCP	1414	80 → 13152 [ACK] Seq=1361 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
31	0.017297	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=654 Ack=2721 Win=262144 Len=0
32	0.018404	10.202.70.61	192.168.1.101	HTTP	1126	HTTP/1.1 200 OK (JPEG JFIF image)
33	0.018405	10.202.70.61	192.168.1.101	TCP	60	80 → 13152 [FIN, ACK] Seq=3793 Ack=654 Win=7168 Len=0
35	0.018433	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [ACK] Seq=654 Ack=3794 Win=260864 Len=0
36	0.018495	192.168.1.101	10.202.70.61	TCP	54	13152 → 80 [FIN, ACK] Seq=654 Ack=3794 Win=260864 Len=0
57	0.026418	10.202.70.61	192.168.1.101	TCP	60	80 → 13152 [ACK] Seq=3794 Ack=655 Win=7168 Len=0

Eq 3：

tcp.stream eq 3

No.	Time	Source	Destination	Protocol	Length	Info
13	0.011324	192.168.1.101	10.202.70.61	TCP	66	13151 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
15	0.012470	10.202.70.61	192.168.1.101	TCP	66	80 → 13151 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
19	0.012670	192.168.1.101	10.202.70.61	TCP	54	13151 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
26	0.012894	192.168.1.101	10.202.70.61	HTTP	707	GET /images/zww.jpg HTTP/1.1
48	0.023194	10.202.70.61	192.168.1.101	TCP	60	80 → 13151 [ACK] Seq=1 Ack=654 Win=7168 Len=0
51	0.024401	10.202.70.61	192.168.1.101	TCP	1414	80 → 13151 [ACK] Seq=1 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
52	0.024438	192.168.1.101	10.202.70.61	TCP	54	13151 → 80 [ACK] Seq=654 Ack=1361 Win=262144 Len=0
53	0.025410	10.202.70.61	192.168.1.101	TCP	1414	80 → 13151 [ACK] Seq=1361 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
54	0.025440	192.168.1.101	10.202.70.61	TCP	54	13151 → 80 [ACK] Seq=654 Ack=2721 Win=262144 Len=0
55	0.026418	10.202.70.61	192.168.1.101	TCP	1414	80 → 13151 [ACK] Seq=2721 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
56	0.026418	10.202.70.61	192.168.1.101	HTTP	155	HTTP/1.1 200 OK (JPEG JFIF image)
60	0.026480	192.168.1.101	10.202.70.61	TCP	54	13151 → 80 [ACK] Seq=654 Ack=4183 Win=262144 Len=0
61	0.026710	192.168.1.101	10.202.70.61	TCP	54	13151 → 80 [FIN, ACK] Seq=654 Ack=4183 Win=262144 Len=0
62	0.027440	10.202.70.61	192.168.1.101	TCP	60	80 → 13151 [ACK] Seq=4183 Ack=655 Win=7168 Len=0

Eq 4：

tcp.stream eq 4

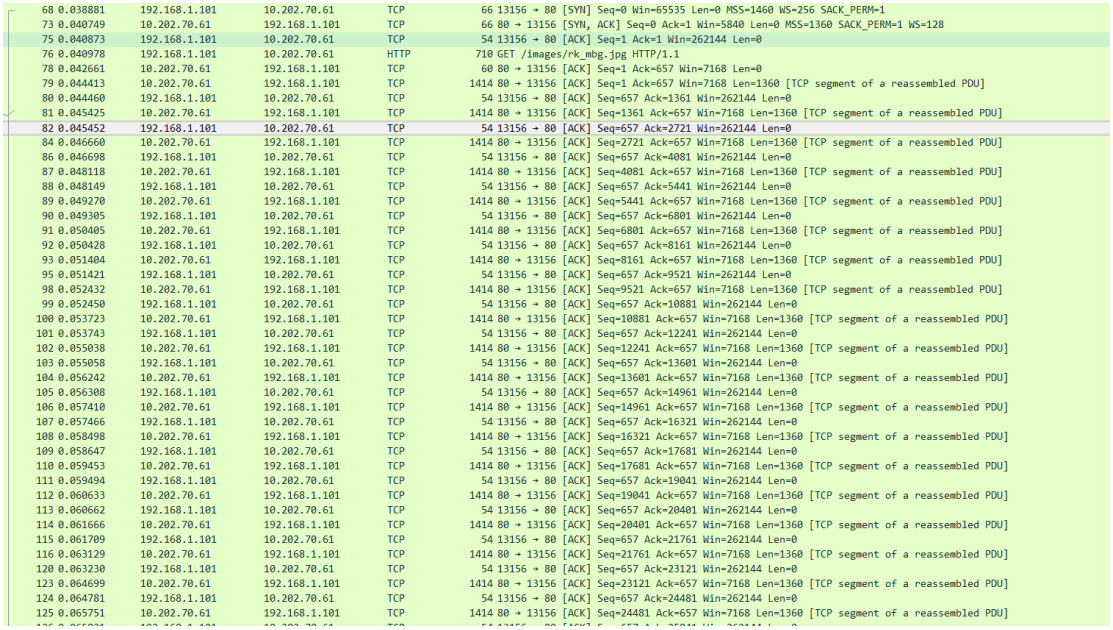
No.	Time	Source	Destination	Protocol	Length	Info
14	0.011346	192.168.1.101	10.202.70.61	TCP	66	13154 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
18	0.012472	10.202.70.61	192.168.1.101	TCP	66	80 → 13154 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
22	0.012714	192.168.1.101	10.202.70.61	TCP	54	13154 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
25	0.012852	192.168.1.101	10.202.70.61	HTTP	707	GET /images/bgw.jpg HTTP/1.1
39	0.019409	10.202.70.61	192.168.1.101	TCP	60	80 → 13154 [ACK] Seq=1 Ack=654 Win=7168 Len=0
41	0.020663	10.202.70.61	192.168.1.101	TCP	1414	80 → 13154 [ACK] Seq=1 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
42	0.020696	192.168.1.101	10.202.70.61	TCP	54	13154 → 80 [ACK] Seq=654 Ack=1361 Win=262144 Len=0
44	0.021706	10.202.70.61	192.168.1.101	TCP	1414	80 → 13154 [ACK] Seq=1361 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
45	0.021734	192.168.1.101	10.202.70.61	TCP	54	13154 → 80 [ACK] Seq=654 Ack=2721 Win=262144 Len=0
46	0.023191	10.202.70.61	192.168.1.101	TCP	1414	80 → 13154 [ACK] Seq=2721 Ack=654 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
47	0.023193	10.202.70.61	192.168.1.101	HTTP	74	HTTP/1.1 200 OK (JPEG JFIF image)
49	0.023245	192.168.1.101	10.202.70.61	TCP	54	13154 → 80 [ACK] Seq=654 Ack=4102 Win=262144 Len=0
50	0.023406	192.168.1.101	10.202.70.61	TCP	54	13154 → 80 [FIN, ACK] Seq=654 Ack=4102 Win=262144 Len=0
59	0.026419	10.202.70.61	192.168.1.101	TCP	60	80 → 13154 [ACK] Seq=4102 Ack=655 Win=7168 Len=0

Eq 5：

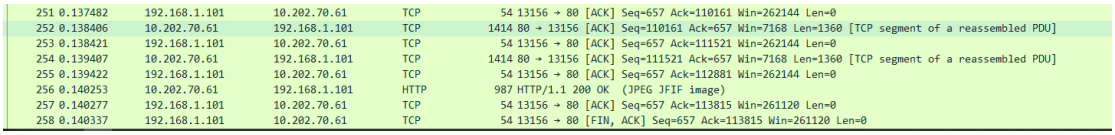
tcp.stream eq 5

No.	Time	Source	Destination	Protocol	Length	Info
63	0.035775	192.168.1.101	10.202.70.61	TCP	66	13155 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
64	0.036668	10.202.70.61	192.168.1.101	TCP	66	80 → 13155 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
65	0.036746	192.168.1.101	10.202.70.61	TCP	54	13155 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
66	0.036871	192.168.1.101	10.202.70.61	HTTP	711	GET /images/body_bg.jpg HTTP/1.1
67	0.038413	10.202.70.61	192.168.1.101	TCP	60	80 → 13155 [ACK] Seq=1 Ack=658 Win=7168 Len=0
69	0.040413	10.202.70.61	192.168.1.101	TCP	1414	80 → 13155 [ACK] Seq=1 Ack=658 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
70	0.040457	192.168.1.101	10.202.70.61	TCP	54	13155 → 80 [ACK] Seq=658 Ack=1361 Win=262144 Len=0
71	0.040747	10.202.70.61	192.168.1.101	HTTP	900	HTTP/1.1 200 OK (JPEG JFIF image)
72	0.040748	10.202.70.61	192.168.1.101	TCP	60	80 → 13155 [FIN, ACK] Seq=2207 Ack=658 Win=7168 Len=0
74	0.040778	192.168.1.101	10.202.70.61	TCP	54	13155 → 80 [ACK] Seq=658 Ack=2208 Win=261120 Len=0
77	0.042593	192.168.1.101	10.202.70.61	TCP	54	13155 → 80 [FIN, ACK] Seq=658 Ack=2208 Win=261120 Len=0
85	0.046660	10.202.70.61	192.168.1.101	TCP	60	80 → 13155 [ACK] Seq=2208 Ack=659 Win=7168 Len=0

Eq 6（部分）：

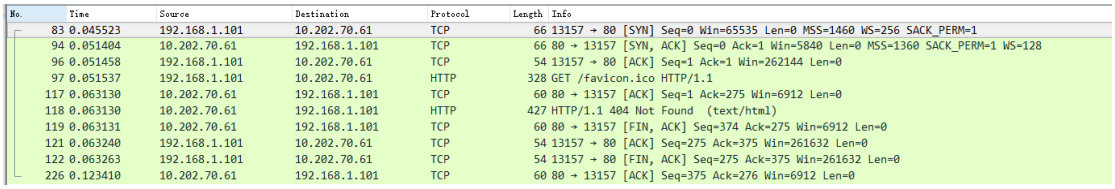


68	0.038881	192.168.1.101	10.202.70.61	TCP	66	13156 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
73	0.040749	10.202.70.61	192.168.1.101	TCP	66	80 → 13156 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
75	0.040873	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
76	0.040978	192.168.1.101	10.202.70.61	HTTP	710	GET /images/rk_mbg.jpg HTTP/1.1
78	0.042661	10.202.70.61	192.168.1.101	TCP	60	80 → 13156 [ACK] Seq=1 Ack=657 Win=7168 Len=0
79	0.044413	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=1 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
80	0.044460	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=1361 Win=262144 Len=0
81	0.045425	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=1361 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
82	0.045452	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=2721 Win=262144 Len=0
84	0.046660	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=2721 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
86	0.046698	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=4081 Win=262144 Len=0
87	0.048118	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=4081 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
88	0.048149	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=5441 Win=262144 Len=0
89	0.049270	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=5441 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
90	0.049305	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=6801 Win=262144 Len=0
91	0.050405	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=6801 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
92	0.050428	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=8161 Win=262144 Len=0
93	0.051404	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=8161 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
95	0.051421	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=9521 Win=262144 Len=0
98	0.052432	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=9521 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
99	0.052450	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=10881 Win=262144 Len=0
100	0.053723	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=10881 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
101	0.053743	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=12241 Win=262144 Len=0
102	0.055038	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=12241 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
103	0.055058	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=13601 Win=262144 Len=0
104	0.056242	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=13601 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
105	0.056308	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=14961 Win=262144 Len=0
106	0.057410	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=14961 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
107	0.057466	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=16321 Win=262144 Len=0
108	0.058498	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=16321 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
109	0.058647	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=17681 Win=262144 Len=0
110	0.059453	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=17681 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
111	0.059494	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=19041 Win=262144 Len=0
112	0.060633	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=19041 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
113	0.060662	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=20401 Win=262144 Len=0
114	0.061666	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=20401 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
115	0.061709	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=21761 Win=262144 Len=0
116	0.063129	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=21761 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
120	0.063230	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=23121 Win=262144 Len=0
123	0.064699	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=23121 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
124	0.064781	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=24481 Win=262144 Len=0
125	0.065751	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=24481 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]



251	0.137482	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=110161 Win=262144 Len=0
252	0.138406	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=110161 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
253	0.138421	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=111521 Win=262144 Len=0
254	0.139407	10.202.70.61	192.168.1.101	TCP	1414	80 → 13156 [ACK] Seq=111521 Ack=657 Win=7168 Len=1360 [TCP segment of a reassembled PDU]
255	0.139422	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=112881 Win=262144 Len=0
256	0.140253	10.202.70.61	192.168.1.101	HTTP	987	HTTP/1.1 200 OK (JPEG JFIF image)
257	0.140277	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [ACK] Seq=657 Ack=113815 Win=261120 Len=0
258	0.140337	192.168.1.101	10.202.70.61	TCP	54	13156 → 80 [FIN, ACK] Seq=657 Ack=113815 Win=261120 Len=0

Eq 7：



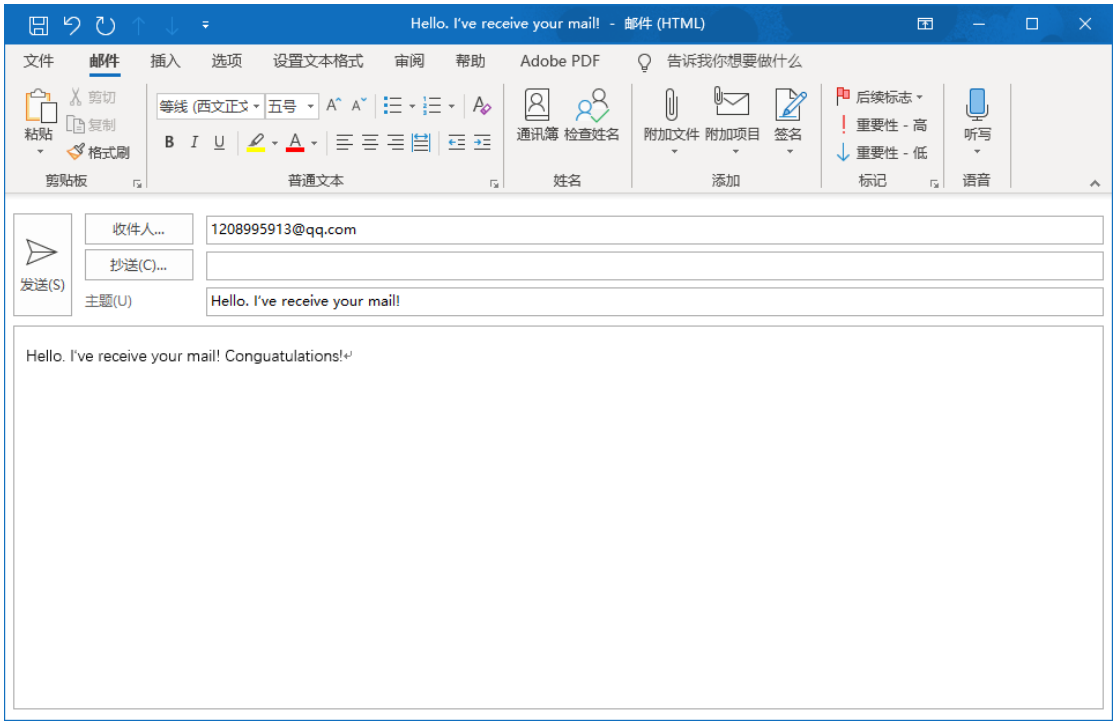
No.	Time	Source	Destination	Protocol	Length	Info
83	0.045523	192.168.1.101	10.202.70.61	TCP	66	13157 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
94	0.051404	10.202.70.61	192.168.1.101	TCP	66	80 → 13157 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1360 SACK_PERM=1 WS=128
96	0.051458	192.168.1.101	10.202.70.61	TCP	54	13157 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
97	0.051537	192.168.1.101	10.202.70.61	HTTP	328	GET /favicon.ico HTTP/1.1
117	0.063130	10.202.70.61	192.168.1.101	TCP	60	80 → 13157 [ACK] Seq=1 Ack=275 Win=6912 Len=0
118	0.063130	10.202.70.61	192.168.1.101	HTTP	427	HTTP/1.1 404 Not Found (text/html)
119	0.063131	10.202.70.61	192.168.1.101	TCP	60	80 → 13157 [FIN, ACK] Seq=374 Ack=275 Win=6912 Len=0
121	0.063240	192.168.1.101	10.202.70.61	TCP	54	13157 → 80 [ACK] Seq=275 Ack=375 Win=261632 Len=0
122	0.063263	192.168.1.101	10.202.70.61	TCP	54	13157 → 80 [FIN, ACK] Seq=275 Ack=375 Win=261632 Len=0
226	0.123410	10.202.70.61	192.168.1.101	TCP	60	80 → 13157 [ACK] Seq=375 Ack=276 Win=6912 Len=0

到 Eq 8 的时候，不再有新的包被筛选出来，这证明总共捕获到了 8 个 TCP 连接。其中，总共有 8 次 HTTP 会话。

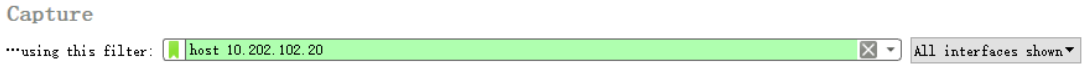
✧ Part Four

- **打开邮件客户端 Foxmail 或 Outlook，写一封电子邮件（建议采用直接送达方式），并捕获这次的数据包。捕获到的数据包由几层协议构成？分别是什么协议？标出数据包的源和目标 IP 地址、源和目标端口。**

打开 Outlook 客户端，使用 zijinshen@zju.edu.cn 给地址 1208995913@qq.com 发送一条邮件：



在发送前，如此配置 Wireshark 的捕获过滤器：



该 IP 为 mail.zju.edu.cn 的 IP 地址。

发送邮件后，wireshark 捕获到以下数据包，全部为本地请求 zju 邮件服务器的请求包和 zju 邮件服务器的响应包：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	10.202.102.20	TCP	66	1525 → 25 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
2	0.001224	10.202.102.20	192.168.1.101	TCP	66	25 → 1525 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=
3	0.001268	192.168.1.101	10.202.102.20	TCP	54	1525 → 25 [ACK] Seq=1 Ack=1 Win=131840 Len=0
4	0.003580	10.202.102.20	192.168.1.101	SMTP	119	S: 220 zju.edu.cn Anti-spam GT for Coremail System (z
5	0.003685	192.168.1.101	10.202.102.20	SMTP	88	C: EHLO [IPv6:::ffff:192.168.1.101]
6	0.004377	10.202.102.20	192.168.1.101	TCP	60	25 → 1525 [ACK] Seq=66 Ack=35 Win=29312 Len=0
7	0.004998	10.202.102.20	192.168.1.101	SMTP	274	S: 250-mail | 250-PIPELINING | 250-AUTH LOGIN PLAIN
8	0.005077	192.168.1.101	10.202.102.20	SMTP	64	C: STARTTLS
9	0.006168	10.202.102.20	192.168.1.101	SMTP	78	S: 220 Ready to start TLS
10	0.006419	192.168.1.101	10.202.102.20	TLSv1.2	401	Client Hello
11	0.008014	10.202.102.20	192.168.1.101	TLSv1.2	1414	Server Hello
12	0.008015	10.202.102.20	192.168.1.101	TCP	1414	25 → 1525 [ACK] Seq=1670 Ack=392 Win=30336 Len=1360 [
13	0.008015	10.202.102.20	192.168.1.101	TLSv1.2	268	Certificate, Server Hello Done
14	0.008051	192.168.1.101	10.202.102.20	TCP	54	1525 → 25 [ACK] Seq=392 Ack=3244 Win=131840 Len=0
15	0.008856	192.168.1.101	10.202.102.20	TLSv1.2	372	Client Key Exchange, Change Cipher Spec, Encrypted Ha
16	0.011114	10.202.102.20	192.168.1.101	TLSv1.2	296	New Session Ticket, Change Cipher Spec, Encrypted Han
17	0.020704	192.168.1.101	10.202.102.20	TLSv1.2	117	Application Data
18	0.021648	10.202.102.20	192.168.1.101	TLSv1.2	303	Application Data
19	0.021838	192.168.1.101	10.202.102.20	TLSv1.2	95	Application Data
20	0.022631	10.202.102.20	192.168.1.101	TLSv1.2	101	Application Data
21	0.022681	192.168.1.101	10.202.102.20	TLSv1.2	113	Application Data
22	0.023586	10.202.102.20	192.168.1.101	TLSv1.2	101	Application Data
23	0.023628	192.168.1.101	10.202.102.20	TLSv1.2	97	Application Data
24	0.030624	10.202.102.20	192.168.1.101	TLSv1.2	114	Application Data
25	0.032605	192.168.1.101	10.202.102.20	TLSv1.2	117	Application Data
26	0.034320	10.202.102.20	192.168.1.101	TLSv1.2	96	Application Data
27	0.034456	192.168.1.101	10.202.102.20	TLSv1.2	112	Application Data
28	0.035583	10.202.102.20	192.168.1.101	TLSv1.2	96	Application Data
29	0.035638	192.168.1.101	10.202.102.20	TLSv1.2	89	Application Data
30	0.036578	10.202.102.20	192.168.1.101	TLSv1.2	120	Application Data
31	0.055847	192.168.1.101	10.202.102.20	TLSv1.2	3816	Application Data

可以看出我们一共捕获到了 SMTP、TCP、TLSv1.2 三种类型的包。

TCP 包的结构之前已有论述，在此略去不表。

SMTP 包的结构：

```
> Frame 9: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
> Internet Protocol Version 4, Src: 10.202.102.20, Dst: 192.168.1.101
> Transmission Control Protocol, Src Port: 25, Dst Port: 1525, Seq: 286, Ack: 45, Len: 24
> Simple Mail Transfer Protocol
```

可以看出 SMTP 包有四层协议：Ethernet II 协议、IPv4 协议、TCP 协议和 SMTP 协议。

TLSv1.2 包的结构：

```
> Frame 13: 268 bytes on wire (2144 bits), 268 bytes captured (2144 bits)
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
> Internet Protocol Version 4, Src: 10.202.102.20, Dst: 192.168.1.101
> Transmission Control Protocol, Src Port: 25, Dst Port: 1525, Seq: 3030, Ack: 392, Len: 214
v [3 Reassembled TCP Segments (2867 bytes): #11(1302), #12(1360), #13(205)]
    [Frame: 11, payload: 0-1301 (1302 bytes)]
    [Frame: 12, payload: 1302-2661 (1360 bytes)]
    [Frame: 13, payload: 2662-2866 (205 bytes)]
    [Segment count: 3]
    [Reassembled TCP length: 2867]
    [Reassembled TCP Data: 1603030b2e0b000b2a000b270006923082068e30820576a0...]
v Secure Sockets Layer
  > TLSv1.2 Record Layer: Handshake Protocol: Certificate
v Secure Sockets Layer
  > TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
```

可以看出 TLSv1.2 包有四层协议：Ethernet II 协议、IPv4 协议、TCP 协议和 SSL 协议。

以第一个 SMTP 包为例：

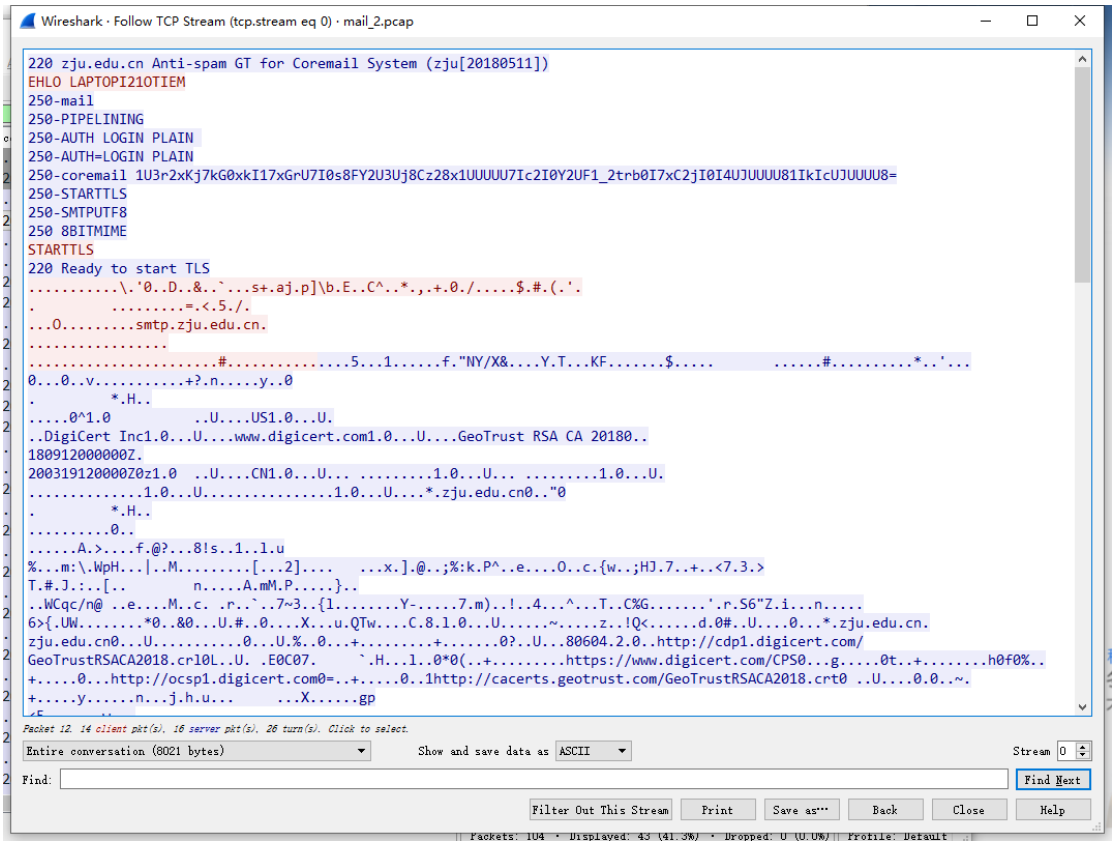
```
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
v Internet Protocol Version 4, Src: 10.202.102.20, Dst: 192.168.1.101
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 105
    Identification: 0xae0a (44554)
  > Flags: 0x4000, Don't fragment
    Time to live: 58
    Protocol: TCP (6)
    Header checksum: 0x5f99 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.202.102.20
    Destination: 192.168.1.101
v Transmission Control Protocol, Src Port: 25, Dst Port: 1525, Seq: 1, Ack: 1, Len: 65
    Source Port: 25
    Destination Port: 1525
    [Stream index: 0]
    [TCP Segment Len: 65]
    Sequence number: 1    (relative sequence number)
```

- **跟踪 TCP 流，查看 SMTP 握手消息采用的是什么（HELO 还是 EHLO）？标出 SMTP 协议层中的客户端机器名、发件人地址、收件人地址、认证的用户名和密码（如果是 EHLO 握手方式）、邮件正文（内容过长可截取关键部分）。**

```
220 zju.edu.cn Anti-spam GT for Coremail System (zju[20180511])
EHLO LAPTOPI21OTIEM
250-mail
250-PIPELINING
250-AUTH LOGIN PLAIN
250-AUTH=LOGIN PLAIN
250-coremail 1U3r2xKj7kG0xkI17xGrU7I0s8FY2U3Uj8Cz28x1UUUUU7Ic2I0Y2UF1_2trb0I7xC2jI0I4UJUUUU81IkIcUJUUUU8=
250-STARTTLS
250-SMTPUTF8
250 8BITMIME
STARTTLS
```

```
Simple Mail Transfer Protocol
  Command Line: EHLO LAPTOPI21OTIEM\r\n
    Command: EHLO
    Request parameter: LAPTOPI21OTIEM
```

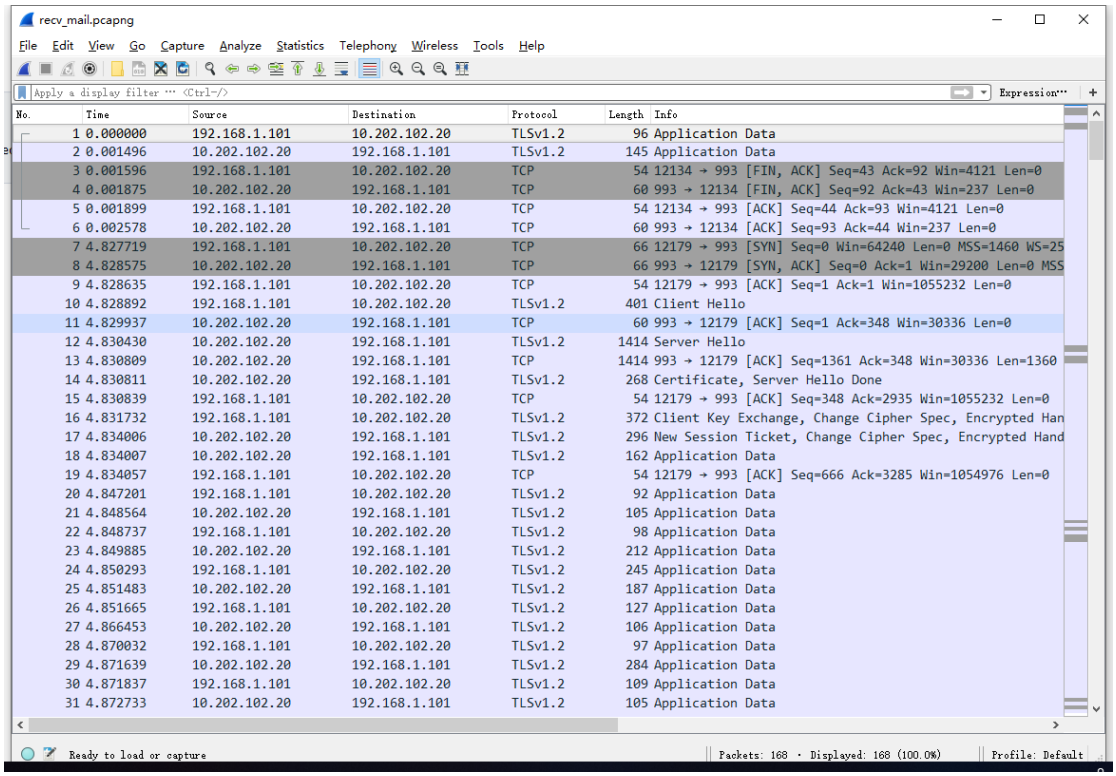
跟踪 TCP 流，发现 SMTP 握手消息采用的是 EHLO。客户端机器名为 LAPTOPI21OTIEM（本机）



其余信息都被 SSL 加密了，无法辨认。

- **打开邮件客户端 Foxmail 或 Outlook，收取自己邮箱中的邮件（请在邮件服务器中设置允许 POP3 或者 IMAP），并捕获这次的数据包。捕获到的数据包由几层协议构成？分别是什么协议？标出数据包的源和目标 IP 地址、源和目标端口。**

使用 QQ 邮箱给 zju 邮箱发一条邮件，使用客户端接收邮件，获得如下数据包：



可以看出 TLSv1.2 包有四层协议：Ethernet II 协议、IPv4 协议、TCP 协议和 SSL 协议。

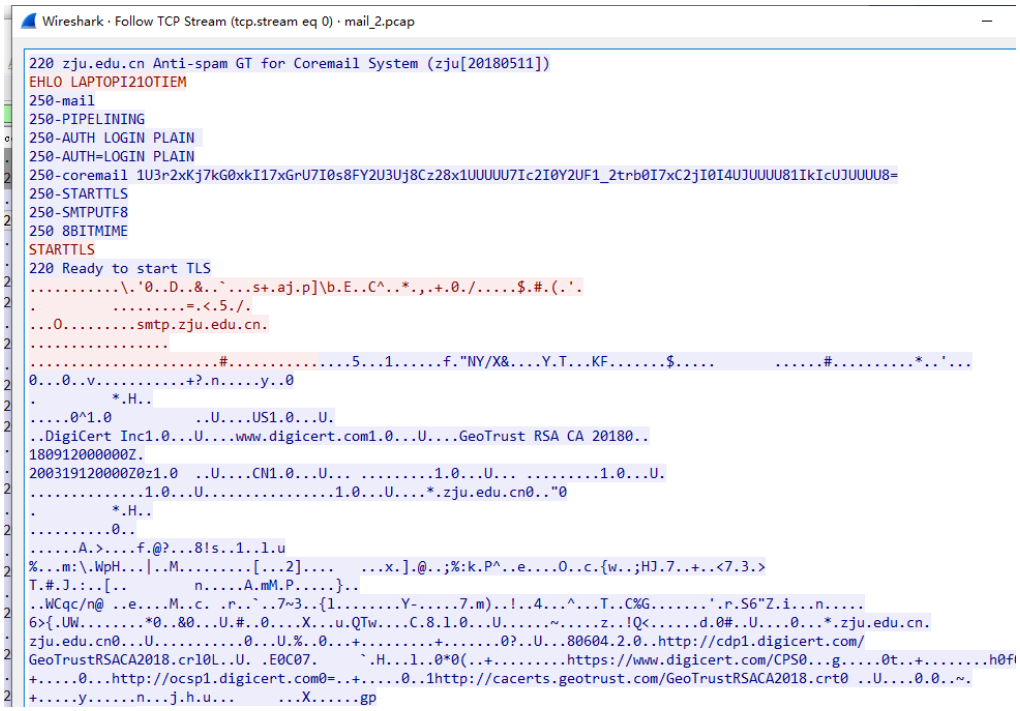
```
> Frame 2: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits) on interface 0
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
> Internet Protocol Version 4, Src: 10.202.102.20, Dst: 192.168.1.101
> Transmission Control Protocol, Src Port: 993, Dst Port: 12134, Seq: 1, Ack: 43, Len: 91
> Secure Sockets Layer
```

源/目的相关信息如下：

```
Header checksum: 0x428b [validation disabled]
[Header checksum status: Unverified]
Source: 10.202.102.20
Destination: 192.168.1.101
Transmission Control Protocol, Src Port: 993, Dst Port: 12134,
Source Port: 993
Destination Port: 12134
[Stream index: 0]
[TCP Segment Len: 91]
```

- **跟踪 TCP 流，标出 POP3 或 IMAP 协议层中的认证用户名和密码、以及接收的邮件正文（内容过长可截取关键部分）。**

协议层和正文由于使用 SSL 已经被全部加密。



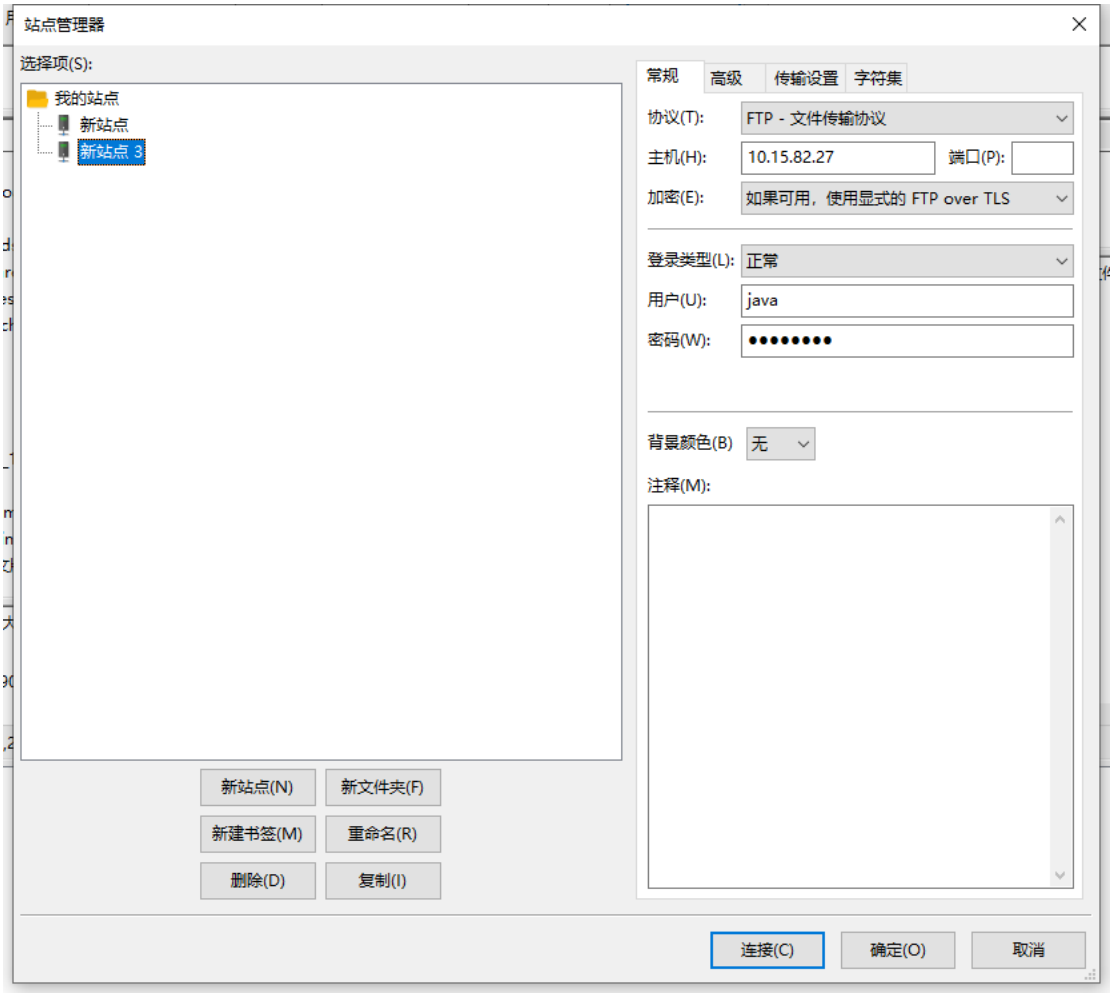
✧ Part Five

本部分需要边操作，边捕获，请在每次操作后暂停捕获，或者使用过滤器。建议通过 FTP 命令行进行实验，也可以使用 FTP 图形客户端。

- **运行 FTP xxx.com 命令，连接并登录服务器，输入用户名和帐号（如果是免费服务器，可以使用匿名帐号 Anonymous，密码是任意的邮箱）。捕获到的数据包由几层协议构成？分别是什么协议？标出数据包的源和目标 IP 地址、源和目标端口。**

由于进行实验的过程中计网 ftp 无法连接，故此处以 java 课程提交作业的 FTP 为例：

使用 filezilla 连接 java ftp：



捕获到如下数据包:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	10.15.82.27	TCP	66	13176 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=25
2	0.001678	10.15.82.27	192.168.1.101	TCP	66	21 → 13176 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=
3	0.001736	192.168.1.101	10.15.82.27	TCP	54	13176 → 21 [ACK] Seq=1 Ack=1 Win=131840 Len=0
4	0.013712	10.15.82.27	192.168.1.101	FTP	92	Response: 220 Serv-U FTP Server v15.0 ready...
5	0.013817	192.168.1.101	10.15.82.27	FTP	64	Request: AUTH TLS
6	0.016519	10.15.82.27	192.168.1.101	FTP	105	Response: 234 AUTH command OK. Initializing SSL conne
7	0.017558	192.168.1.101	10.15.82.27	FTP	248	Request: \026\003\001\000\275\001\000\000\271\003\003
8	0.019694	10.15.82.27	192.168.1.101	FTP	782	Response: \026\003\001\0005\002\000\0001\003\001\\035
9	0.020362	192.168.1.101	10.15.82.27	FTP	193	Request: \026\003\001\000\206\020\000\000\202\000\200
10	0.020380	192.168.1.101	10.15.82.27	FTP	60	Request: \024\003\001\000\001\001
11	0.020394	192.168.1.101	10.15.82.27	FTP	107	Request: \026\003\001\0000\246E\236\257EA9_o\034,\211
12	0.022947	10.15.82.27	192.168.1.101	TCP	60	21 → 13176 [ACK] Seq=818 Ack=403 Win=66048 Len=0
13	0.024243	10.15.82.27	192.168.1.101	FTP	288	Response: \026\003\001\000\252\004\000\000\246\000\00
14	0.031131	192.168.1.101	10.15.82.27	FTP	91	Request: \027\003\001\000 r\002 \227I~[9\202\23137\27
15	0.035774	10.15.82.27	192.168.1.101	FTP	160	Response: \027\003\001\000 \001%s;\272\230g\360\037\3
16	0.035977	192.168.1.101	10.15.82.27	FTP	107	Request: \027\003\001\0000\223Mo{\303\324\206\360\304
17	0.039760	10.15.82.27	192.168.1.101	FTP	160	Response: \027\003\001\000 \025
18	0.040005	192.168.1.101	10.15.82.27	FTP	91	Request: \027\003\001\000 ,\213e\367LR\234b\020\314\0
19	0.042636	10.15.82.27	192.168.1.101	FTP	144	Response: \027\003\001\000 \234-\226)\222cB\275%d\306
20	0.043093	192.168.1.101	10.15.82.27	FTP	91	Request: \027\003\001\000 \202\207~\210\346\357\364\3
21	0.050725	10.15.82.27	192.168.1.101	FTP	848	Response: \027\003\001\000 I\221\235\233\022\206"pAS\
22	0.050972	192.168.1.101	10.15.82.27	FTP	107	Request: \027\003\001\0000\326U\332\342\346Y\360\376\
23	0.053382	10.15.82.27	192.168.1.101	FTP	144	Response: \027\003\001\000 \020\245t\314eYy\352\327qk
24	0.053590	192.168.1.101	10.15.82.27	FTP	107	Request: \027\003\001\0000\236\365\257\021\345\353?\3
25	0.055736	10.15.82.27	192.168.1.101	FTP	160	Response: \027\003\001\000 \335\025\260#\343\344"\035
26	0.055958	192.168.1.101	10.15.82.27	FTP	91	Request: \027\003\001\000 \000\226\361\034\306\232r:\
27	0.057958	10.15.82.27	192.168.1.101	FTP	176	Response: \027\003\001\000 \324B\235\270\002!\256Y\30
28	0.058177	192.168.1.101	10.15.82.27	FTP	91	Request: \027\003\001\000 \220z\333\325$\246\323YguZq
29	0.060245	10.15.82.27	192.168.1.101	FTP	176	Response: \027\003\001\000 \237W2\021\320\017K\227w<\
30	0.060463	192.168.1.101	10.15.82.27	FTP	123	Request: \027\003\001\000@$\272V\272 s\315)R\252\340h

其中：

```
> Ethernet II, Src: Tp-LinkT_3f:10:5e (5c:63:bf:3f:10:5e), Dst: LcfcHefe_32:c2:f1 (8c:16:45:32:c2:f1)
v Internet Protocol Version 4, Src: 10.15.82.27, Dst: 192.168.1.101
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 78
    Identification: 0x236d (9069)
  > Flags: 0x4000, Don't fragment
    Time to live: 122
    Protocol: TCP (6)
    Header checksum: 0xbf05 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.15.82.27
    Destination: 192.168.1.101
v Transmission Control Protocol, Src Port: 21, Dst Port: 13176, Seq: 1, Ack: 1, Len: 38
    Source Port: 21
    Destination Port: 13176
    [Stream index: 0]
    [TCP Segment Len: 38]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 39    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 260
    [Calculated window size: 66560]
    [Window size scaling factor: 256]
    Checksum: 0xb2bf [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (38 bytes)
> File Transfer Protocol (FTP)
  [Current working directory: ]
```

FTP 包由四层协议构成：Ethernet II 协议、IPv4 协议、TCP 协议和 FTP 协议。其中：

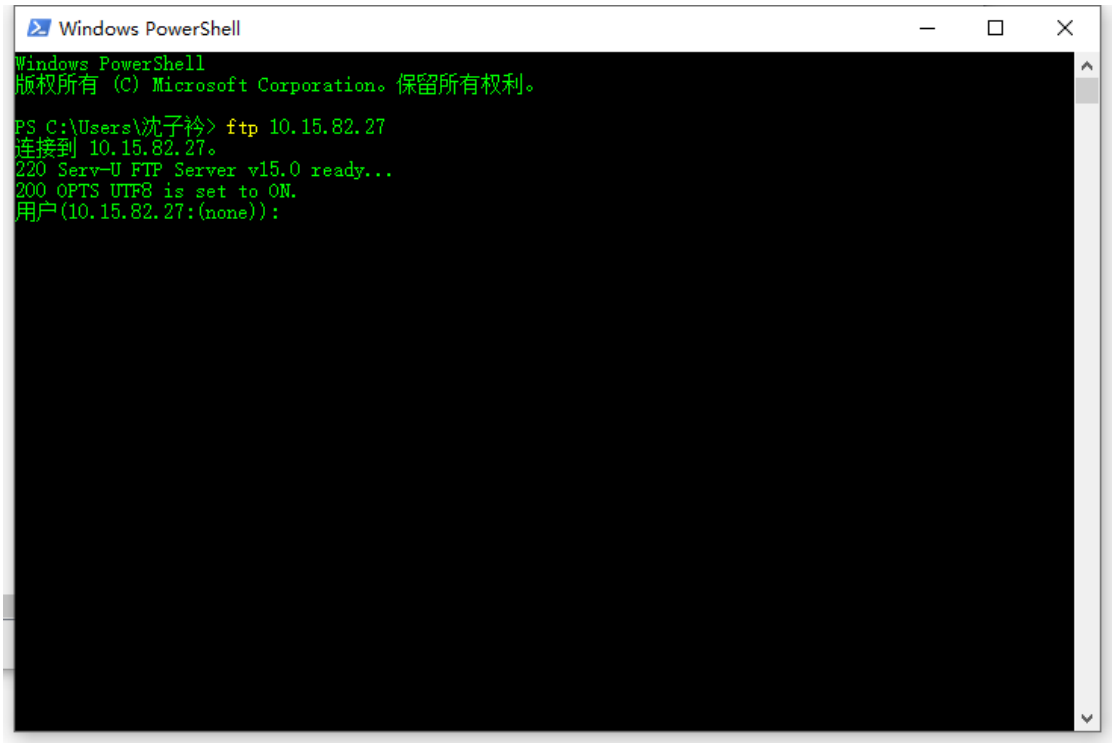
```
    Time to live: 122
    Protocol: TCP (6)
    Header checksum: 0xbf05 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.15.82.27
    Destination: 192.168.1.101
v Transmission Control Protocol, Src Port: 21, Dst Port: 13176, Seq: 1, Ack: 1, Len: 38
    Source Port: 21
    Destination Port: 13176
    [Stream index: 0]
    [TCP Segment Len: 38]
    Sequence number: 1    (relative sequence number)
```

来源和目的地的 IP 和端口如图所示。

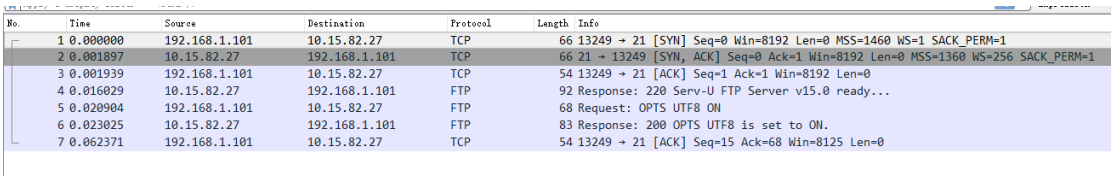
- **跟踪 TCP 流，标注客户端发出的登录命令、用户名、密码以及服务器的响应。**

考虑到使用客户端跟踪 TCP 流不甚清楚，加上大量数据被加密而无从辨认，故我们换用 Powershell 再实现一次：

首先连接 ftp：



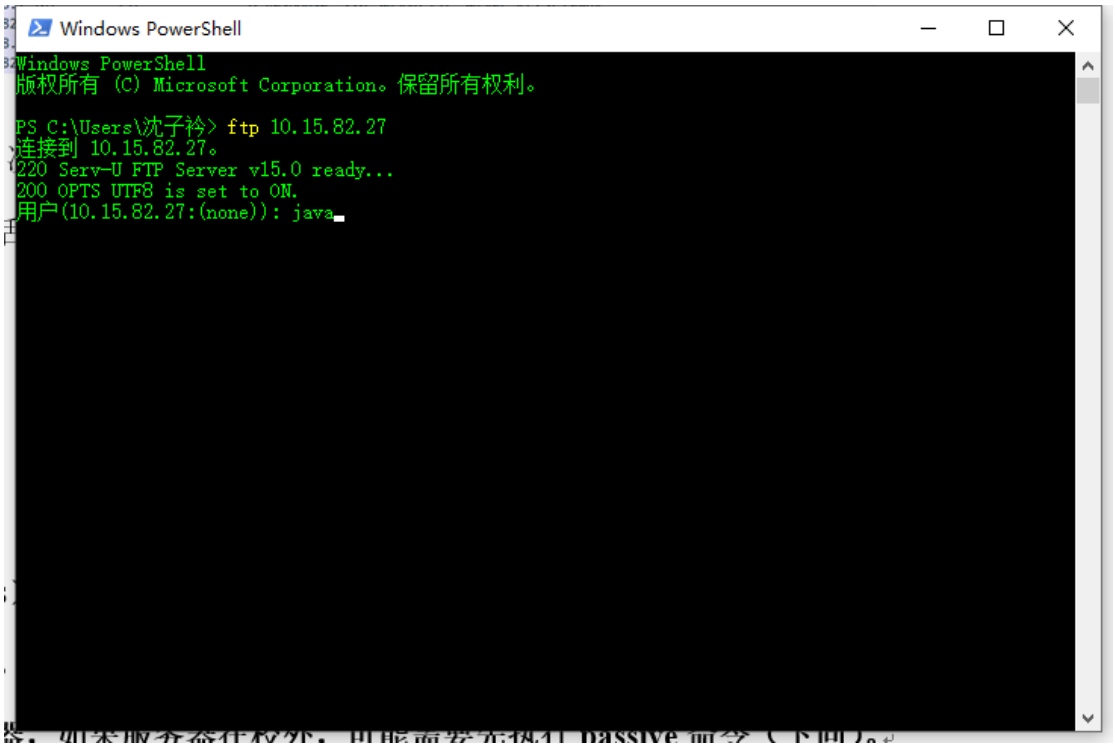
到这一步时，wireshark 抓取到如下数据包：



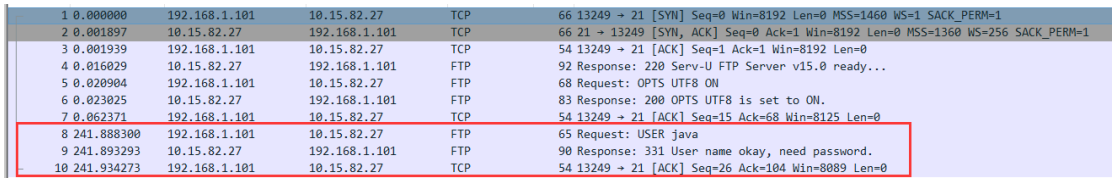
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.101	10.15.82.27	TCP	66	13249 → 21 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=1 SACK_PERM=1
2	0.001897	10.15.82.27	192.168.1.101	TCP	66	21 → 13249 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1360 WS=256 SACK_PERM=1
3	0.001939	192.168.1.101	10.15.82.27	TCP	54	13249 → 21 [ACK] Seq=1 Ack=1 Win=8192 Len=0
4	0.016029	10.15.82.27	192.168.1.101	FTP	92	Response: 220 Serv-U FTP Server v15.0 ready...
5	0.020904	192.168.1.101	10.15.82.27	FTP	68	Request: OPTS UTF8 ON
6	0.023025	10.15.82.27	192.168.1.101	FTP	83	Response: 200 OPTS UTF8 is set to ON.
7	0.062371	192.168.1.101	10.15.82.27	TCP	54	13249 → 21 [ACK] Seq=15 Ack=68 Win=8125 Len=0

前三个数据包，是 TCP 的三次握手；三次握手结束后，FTP Server 会给出一个连接成功的响应；然后客户端会请求激活 UTF8 编码，服务器也会给予相应响应。

然后我们输入用户名：



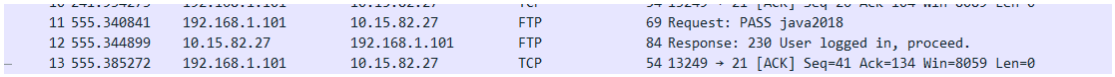
回车输入后。又收到如下数据包：



1	0.000000	192.168.1.101	10.15.82.27	TCP	66 13249 → 21 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=1 SACK_PERM=1
2	0.001897	10.15.82.27	192.168.1.101	TCP	66 21 → 13249 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1360 WS=256 SACK_PERM=1
3	0.001939	192.168.1.101	10.15.82.27	TCP	54 13249 → 21 [ACK] Seq=1 Ack=1 Win=8192 Len=0
4	0.016029	10.15.82.27	192.168.1.101	FTP	92 Response: 220 Serv-U FTP Server v15.0 ready...
5	0.020904	192.168.1.101	10.15.82.27	FTP	68 Request: OPTS UTF8 ON
6	0.023025	10.15.82.27	192.168.1.101	FTP	83 Response: 200 OPTS UTF8 is set to ON.
7	0.062371	192.168.1.101	10.15.82.27	TCP	54 13249 → 21 [ACK] Seq=15 Ack=68 Win=8125 Len=0
8	241.888300	192.168.1.101	10.15.82.27	FTP	65 Request: USER java
9	241.893293	10.15.82.27	192.168.1.101	FTP	90 Response: 331 User name okay, need password.
10	241.934273	192.168.1.101	10.15.82.27	TCP	54 13249 → 21 [ACK] Seq=26 Ack=104 Win=8089 Len=0

可以看出客户端向服务器发送了一个用户名，尔后服务器响应用户名有效；

然后再输入密码，回车提交：

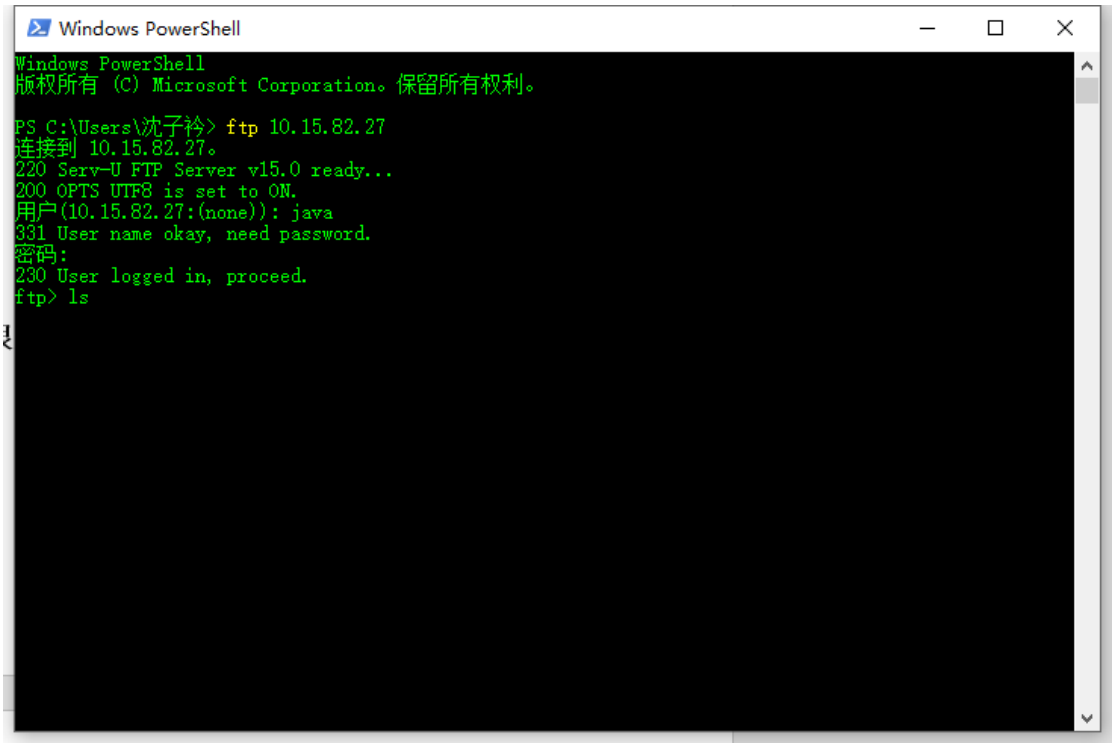


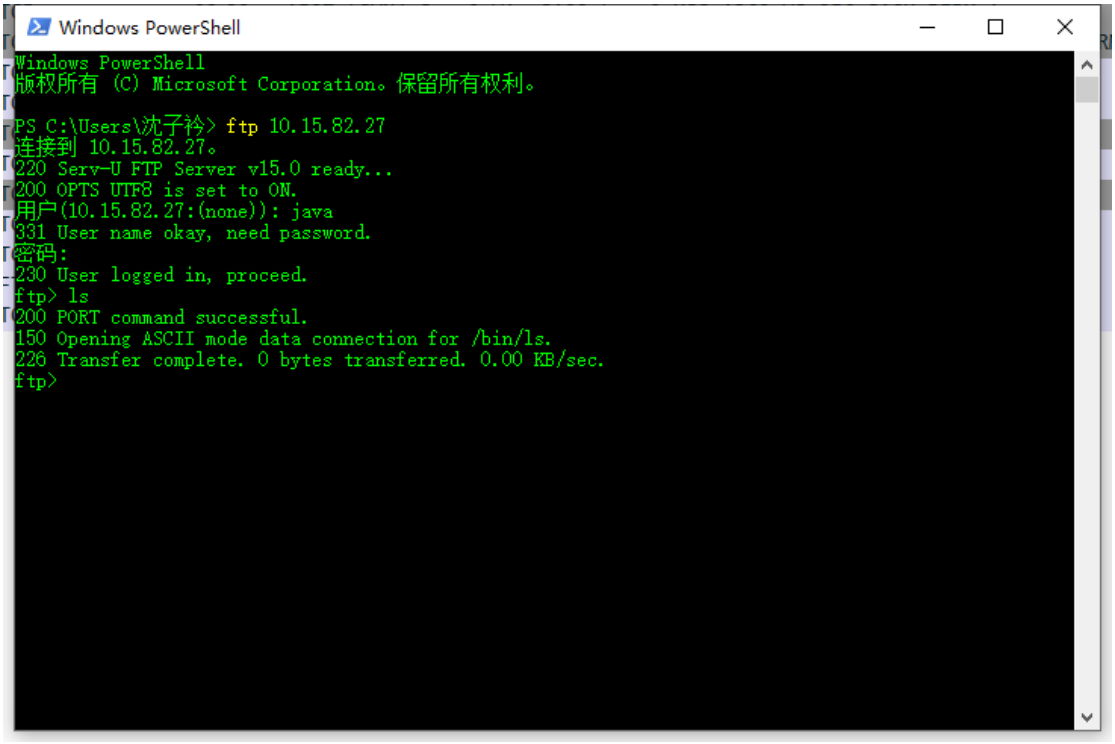
11	555.340841	192.168.1.101	10.15.82.27	FTP	69 Request: PASS java2018
12	555.344899	10.15.82.27	192.168.1.101	FTP	84 Response: 230 User logged in, proceed.
13	555.385272	192.168.1.101	10.15.82.27	TCP	54 13249 → 21 [ACK] Seq=41 Ack=134 Win=8059 Len=0

发现我们已经成功登录，其中密码使用明文传输，和客户端有所不同。

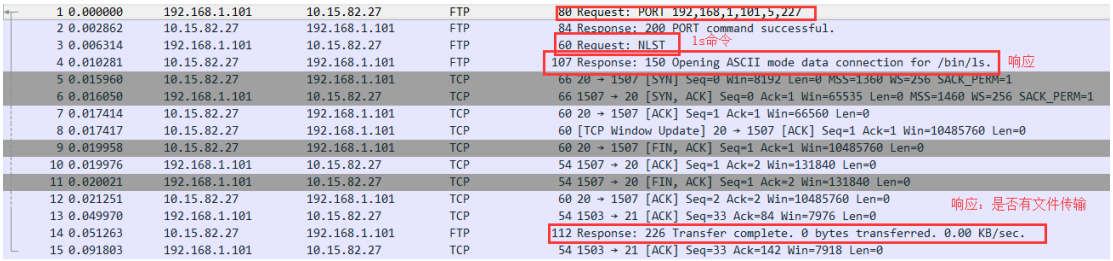
- **执行列目录操作（ls），在新捕获的数据包中跟踪 TCP 流，标注客户端发出的命令、以及服务器的响应。查看是否建立了一个新的 TCP 连接，跟踪该连接的 TCP 流。建议连接校内服务器，如果服务器在校外，可能需要先执行 passive 命令（下同）。**

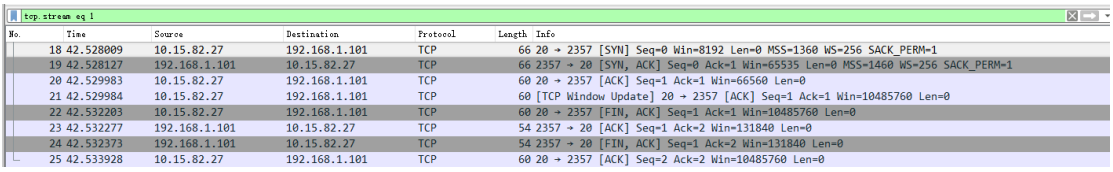
在命令行中执行列目录操作：





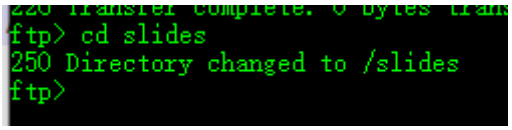
可以发现此时建立了一个新的 TCP 连接：

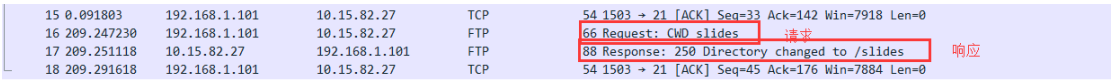




- **执行更换目录操作（cd），在新捕获的数据包中跟踪 TCP 流，标注客户端发出的命令、以及服务器的响应。**

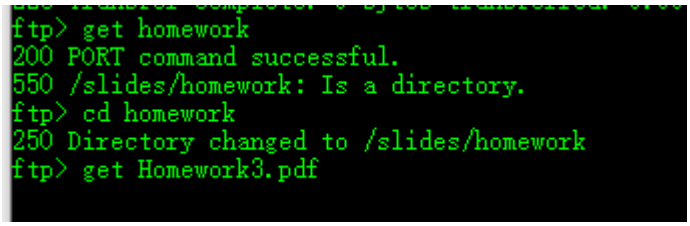
  执行 cd slides 命令，切换到 slides 文件夹：

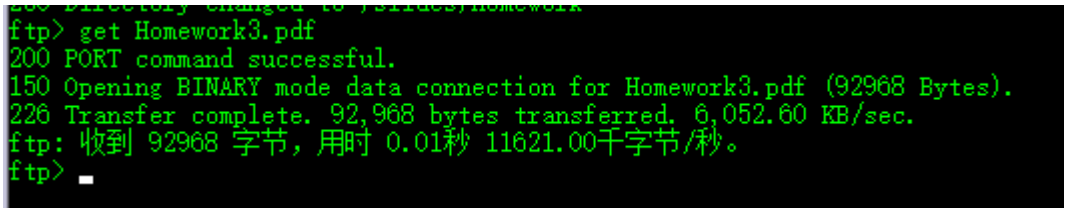




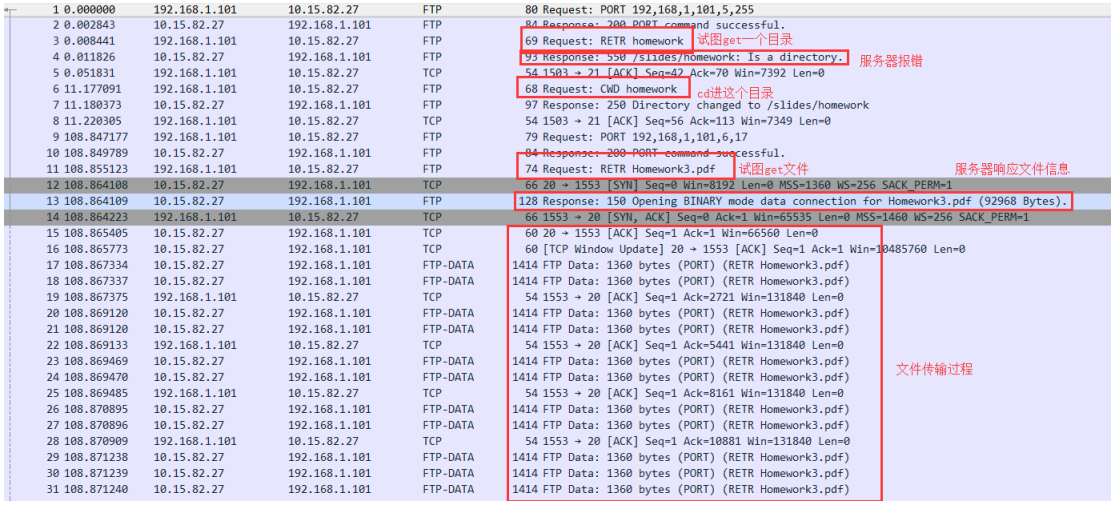
- **执行下载文件操作（get filename），如果是二进制文件，先执行 binary 命令。在新捕获的数据包中跟踪 TCP 流，标注客户端发出的命令、以及服务器的响应。查看是否建立了一个新的 TCP 连接，跟踪该连接的 TCP 流（内容较长时截取部分关键内容）。**

  首先试图尝试 get 目录，提示错误；后执行 cd 进入该目录，再对该目录下文件执行下载操作：

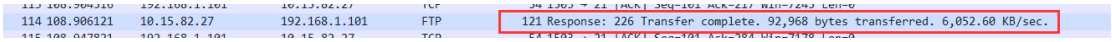




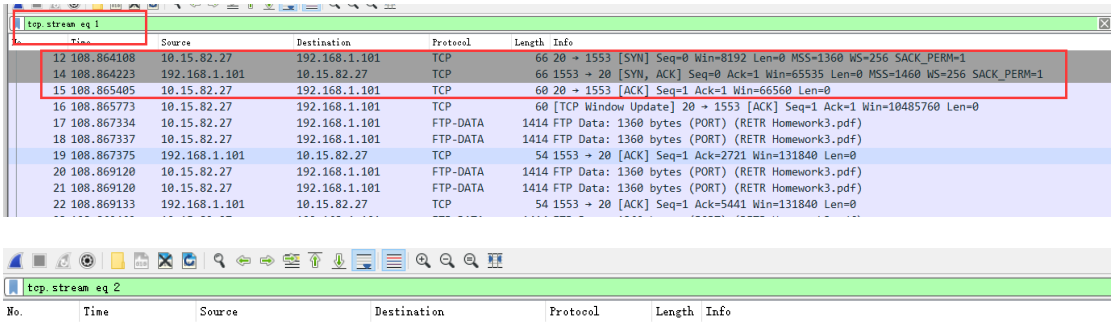
捕获数据包如下：



最后，服务器会响应传输完成，信息中包括传输文件大小：



进一步跟踪文件传输之前的 TCP 包，发现文件传输过程建立了一个新的 TCP 连接：



六、 实验结果与分析

- Ping 发送的是什么类型的协议数据包？什么时候会出现 ARP 消息？Ping 一个域名和 Ping 一个 IP 地址出现的数据包有什么不同？

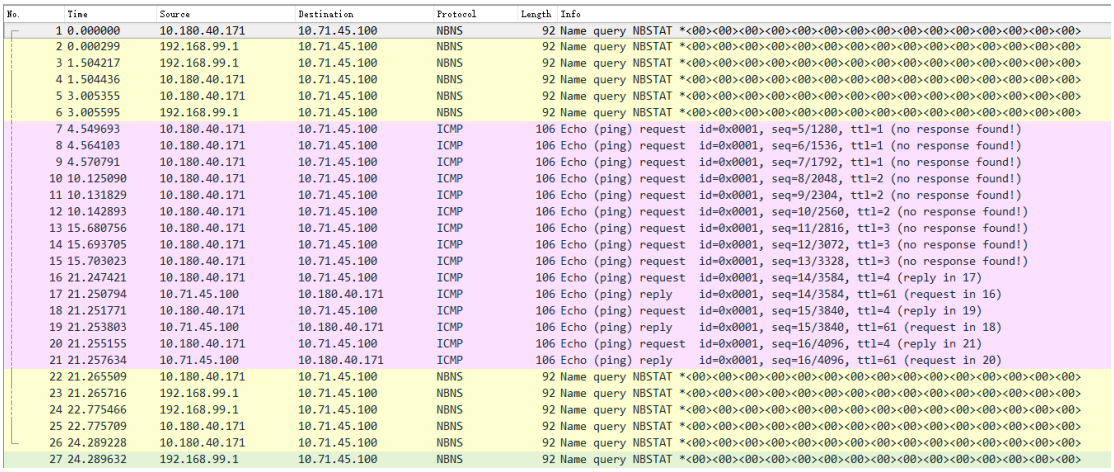
  答：①Ping 发送的是一个 ICMP 数据包；

  ②当缓存中不存在某一 IP 地址对应的 MAC 地址时，才会发送 ARP 请求到局域网查询，产生 ARP 消息；

  ③ping 一个 IP 地址只会在起点、终点及其中继节点之间产生 ICMP 数据包。但 ping 一个域名时，因解析域名需要，会在主机和 DNS 服务器之间产生 ICMP 数据包。

- Tracert/Traceroute 发送的是什么类型的协议数据包，整个路由跟踪过程是如何进行的？

  答：结合这张图：



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
2	0.000299	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
3	1.504217	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
4	1.504436	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
5	3.005355	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
6	3.005595	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
7	4.549693	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=5/1280, ttl=1 (no response found!)
8	4.564103	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=6/1536, ttl=1 (no response found!)
9	4.570791	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=7/1792, ttl=1 (no response found!)
10	10.125090	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=8/2048, ttl=2 (no response found!)
11	10.131829	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=9/2304, ttl=2 (no response found!)
12	10.142893	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=10/2560, ttl=2 (no response found!)
13	15.680756	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=11/2816, ttl=3 (no response found!)
14	15.693705	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=12/3072, ttl=3 (no response found!)
15	15.703023	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=13/3328, ttl=3 (no response found!)
16	21.247421	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=14/3584, ttl=4 (reply in 17)
17	21.250794	10.71.45.100	10.180.40.171	ICMP	106	Echo (ping) reply id=0x0001, seq=14/3584, ttl=61 (request in 16)
18	21.251771	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=15/3840, ttl=4 (reply in 19)
19	21.253803	10.71.45.100	10.180.40.171	ICMP	106	Echo (ping) reply id=0x0001, seq=15/3840, ttl=61 (request in 18)
20	21.255155	10.180.40.171	10.71.45.100	ICMP	106	Echo (ping) request id=0x0001, seq=16/4096, ttl=4 (reply in 21)
21	21.257634	10.71.45.100	10.180.40.171	ICMP	106	Echo (ping) reply id=0x0001, seq=16/4096, ttl=61 (request in 20)
22	21.265509	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
23	21.265716	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
24	22.775466	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
25	22.775709	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
26	24.289228	10.180.40.171	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
27	24.289632	192.168.99.1	10.71.45.100	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>

可以看出该命令发送的是 NBNS 和 ICMP 类型的包。

Tracert 命令用 IP 生存时间 (TTL)字段和 ICMP 错误消息来确定从一个主机到网络上其他主机的路由。

首先，tracert 送出一个 TTL 是 1 的 IP 数据包到目的地，当路径上的第一个路由器收到这个数据包时，它将 TTL 减 1。此时，TTL 变为 0，所以该路由器会将此数据包丢掉，并送回一个「ICMP time exceeded」消息（包括发 IP 包的源地址，IP 包的所有内容及路由器的 IP 地址），tracert 收到这个消息后，便知道这个路由器存在于这个路径上，接着 tracert 再送出另一个 TTL 是 2 的数据包，发现第 2 个路由器.....tracert 每次将送出的数据包的 TTL 加 1 来发现另一个路由器，这个重复的动作一直持续到某个数据 抵达目的地。当数据包到达目的地后，该主机则不会送回 ICMP。

- 建立 TCP 连接的数据包由几个构成？各自的 SYN 和 ACK 标志字段是什么？

答：建立 TCP 连接的数据包由三个构成（三次握手）。

标志字段的意义：

- 第一次：建立连接时，客户端发送 syn 包（syn=j）到服务器，并进入 SYN_SENT 状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。
- 第二次：服务器收到 syn 包，必须确认客户的 SYN（ack=j+1），同时自己也发送一个 SYN 包（syn=k），即 SYN+ACK(确认字符)包，此时服务器进入 SYN_RECV 状态；
- 第三次：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK(ack=k+1)，此包发送完毕，客户端和服务器进入 ESTABLISHED（TCP 连接成功）状态，完成三次握手。

- 浏览器打开一个网页，可能会看到多个 TCP 连接，多次 HTTP 会话。一个 TCP 连接上是否会存在多个 HTTP 会话？什么情况下会出现 DNS 数据包？

  答：一个 TCP 连接可能存在多个 HTTP 会话。当存在对域名进行解析的需要时，会出现 DNS 数据包。

- 邮件客户端发送一封电子邮件，需要几次请求、响应消息的交互？消息的一般格式是什么？邮件正文结束的标记是什么？

  答：之前在做实验时，我使用的时经过 SSL 加密处理的客户端，因此观察效果很差。之后查看教程取消设定 SSL 加密后，观察到 9 次请求、响应信息的交互。邮件正文结束的标记为两个换行符加上字符‘.’.

- 邮件客户端接收一封电子邮件，需要几次请求、响应消息的交互？消息的一般格式是什么？用户名和密码是否经过了加密处理？

  答：我观察到 20 次请求、响应信息的交互。格式为内容加换行符结尾。用户名和密码在关闭 SSL 加密设置的情况下是没有经过加密处理的，但在打开时是经过加密的。

- 登录 FTP 服务器时，会产生几个 TCP 连接？列目录和上传或者下载文件时，会产生几个 TCP 连接？

  答：登录时会产生一个 TCP 连接，列目录和上传或者下载文件时，各会产生一个新的 TCP 连接。实验结束时，可以抓取到和 3 组和 TCP 连接相关的包。

七、 讨论、心得

本次实验是计算机网络的第七次实验，难度不大，但量较大。通过本次实验，我进一步了解了 Wireshark 的功能和基本用法、过滤器原语的使用以及基本网络命令的功能与使用方法，受益匪浅。

本次实验对自学的要求较高。为了做好每一题，我不得不查阅大量参考文献，但这也帮助我对基本概念有了更好的理解。

因为学校特殊的网络环境和部分第三方程序的干扰（比如邮件客户端和 FTP 客户端自带的加密策略），本次实验也遇到了不少困难，但我凭借查阅资料和生活经验妥善解决了它们。我想，这将对未来工作的开展提供极大的帮助。

在未来的实验中，我会再接再厉。