

# 浙江大学

## 本科实验报告

课程名称：	计算机网络
实验名称：	网络协议分析
姓 名：	
学 院：	计算机学院
系：	计算机系
专 业：	信息安全
学 号：	
指导教师：	张泉方

2020 年 09 月 26 日

# 浙江大学实验报告

## 一、 实验目的

- 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

- Wireshark 是 PC 上使用最广泛的免费抓包工具，可以分析大多数常见的协议数据包。有 Windows 版本和 Mac 版本，可以免费从网上下载。
- 掌握网络协议分析软件 Wireshark 的使用，学会配置过滤器
- 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、Windows、Linux 或 Mac 操作系统、浏览器软件
- WireShark 协议分析软件

## 四、 操作方法与实验步骤

- 安装网络包捕获软件 Wireshark
- 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
  - ✓ TRACE ROUTE：跟踪一个目标地址的途经路由
  - ✓ NSLOOKUP：查询一个域名
  - ✓ HTTP：访问一个网页

五、实验数据记录和处理

✧ Part One

1. 运行 Wireshark 软件，开始捕获数据包，列出你看到的协议名字（至少 5 个）。

协议名： ARP, SSDP, IGMPv2, DTLs, MDNS, NBNS

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9e:74:2b:cc:8f:d5	Broadcast	ARP	56	who has 10.189.133.214? Tell 10.189.138.135
2	0.000210	52:68:81:3c:3c:06	Broadcast	ARP	56	who has 10.189.142.131? Tell 10.189.128.93
3	0.000510	10.189.1.30	239.255.255.250	SSDP	216	M-SEARCH * HTTP/1.1
4	0.000775	10.186.1.8	239.255.255.250	IGMPv2	56	Membership Report group 239.255.255.250
5	0.001153	10.186.1.8	239.255.255.250	IGMPv2	56	Membership Report group 239.255.255.250
6	0.001833	10.189.138.135	10.189.143.255	DTLS	396	Continuation Data
7	0.002082	fe80::30ca:4291:838_	ff02::fb	MDNS	98	Standard query 0x0000 A eqzhadnypuk.local, "QM" question
8	0.002468	10.189.1.236	10.189.15.255	NBNS	92	Name query NB EQZHADNYPUK<00>
9	0.002648	10.189.1.236	224.0.0.251	MDNS	78	Standard query 0x0000 A eqzhadnypuk.local, "QM" question
10	0.003009	10.189.1.236	10.189.15.255	NBNS	92	Name query NB RMEUHOBHXPDPY<00>
11	0.003086	10.189.1.236	10.189.15.255	NBNS	92	Name query NB ABLCWUSPZL<00>

2. 找一个包含 IP 的数据包,这个数据包有 4 层？最高层协议是 TCP 协议，从 Ethernet 开始往上，各层协议的名字分别为 Ethernet II, IPv4, TCP。

展开 IP 层协议，标出源 IP 地址、目标 IP 地址及其在数据包中的具体位置，展开 Ethernet 层，标出源 MAC 地址和目标 MAC 地址及其在数据包中的具体位置。

42.26.010178168.62.58.13010.186.0.220TCP1514443 → 11032 [ACK] Seq=1 Ack=209 Win=263424 Len=1448 TSval=1783037547

> Frame 42: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF\_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0

> Ethernet II, Src: JuniperN\_60:6f:c2 (2c:21:72:60:6f:c2), Dst: RivetNet\_b9:cc:cb (9c:b6:d0:b9:cc:cb)

> Destination: RivetNet\_b9:cc:cb (9c:b6:d0:b9:cc:cb) 目标MAC地址

> Source: JuniperN\_60:6f:c2 (2c:21:72:60:6f:c2) 源MAC地址

Type: IPv4 (0x0000)

> Internet Protocol Version 4, Src: 168.62.58.130, Dst: 10.186.0.220

0100 ... = Version: 4

... 0101 = Header Length: 20 bytes (5)

> Differentiated Services Field: 0x40 (DSCP: CS2, ECN: Not-ECT)

Total Length: 1500

Identification: 0x7ec3 (32451)

> Flags: 0x4000, Don't fragment

Fragment offset: 0

Time to live: 106

Protocol: TCP (6)

Header checksum: 0x9dc2 [validation disabled]

[Header checksum status: Unverified]

Source: 168.62.58.130 源MAC地址

Destination: 10.186.0.220 目标MAC地址

> Transmission Control Protocol, Src Port: 443, Dst Port: 11032, Seq: 1, Ack: 209, Len: 1448

0000 9c b6 d0 b9 cc cb 2c 21 72 60 6f c2 08 00 45 40 .....l r'o---E@

0010 05 dc 7e c3 40 00 6a 06 9d c2 a8 3e 3a 82 0a ba ...~@:j- ->:~

0020 00 dc 01 bb 2b 18 a0 30 a4 d3 b0 27 ab fb 80 10 ...+~0 ...'...

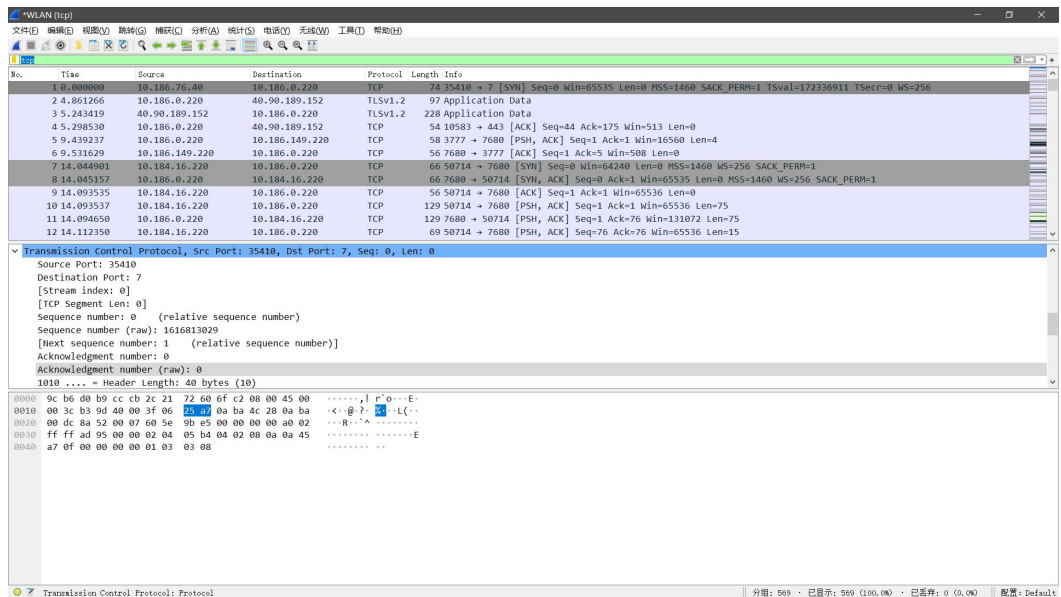
0030 04 05 ed f8 00 00 01 01 08 0a 6a 46 fe 6b 02 ee ...-jF-k-

0040 ad 6b 16 03 03 0f 9c 02 00 00 55 03 03 5f 86 a6 .k-.....U-\_-

3. 配置应用显示过滤器，让界面只显示某一协议类型的数据包（输入协议名称）。

使用的过滤器： tcp ， 希望显示的协议类型： TCP。

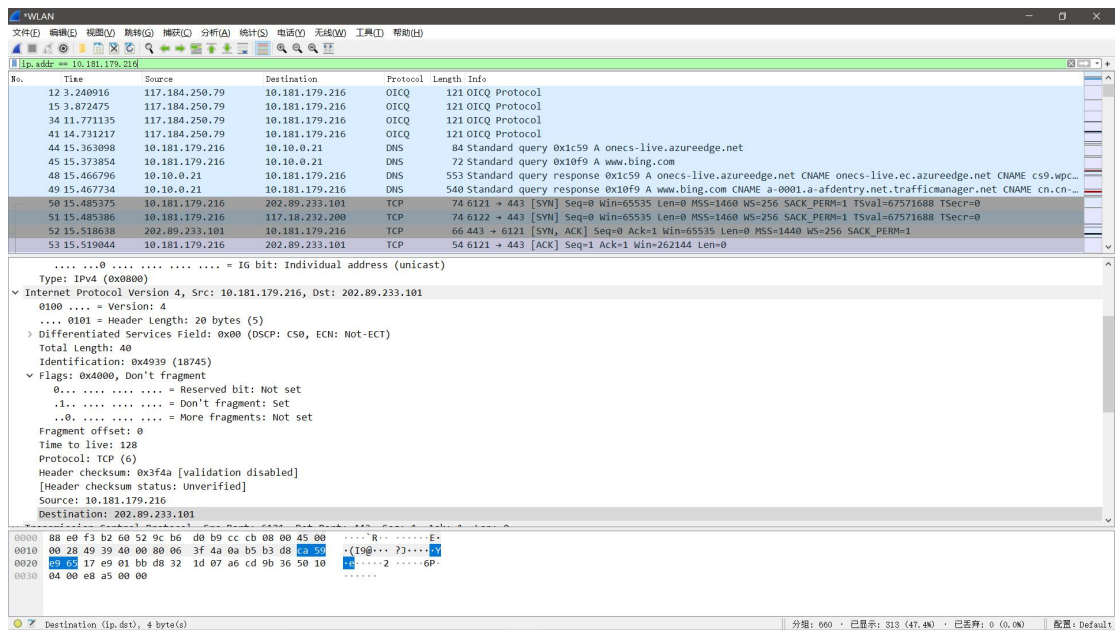
截图：



#### 4. 配置应用显示过滤器，让界面只显示某个 IP 地址的数据包（ip.addr==x.x.x.x）。

使用的过滤器：ip.addr == 10.181.179.216，希望显示的 IP 地址：10.181.179.216。

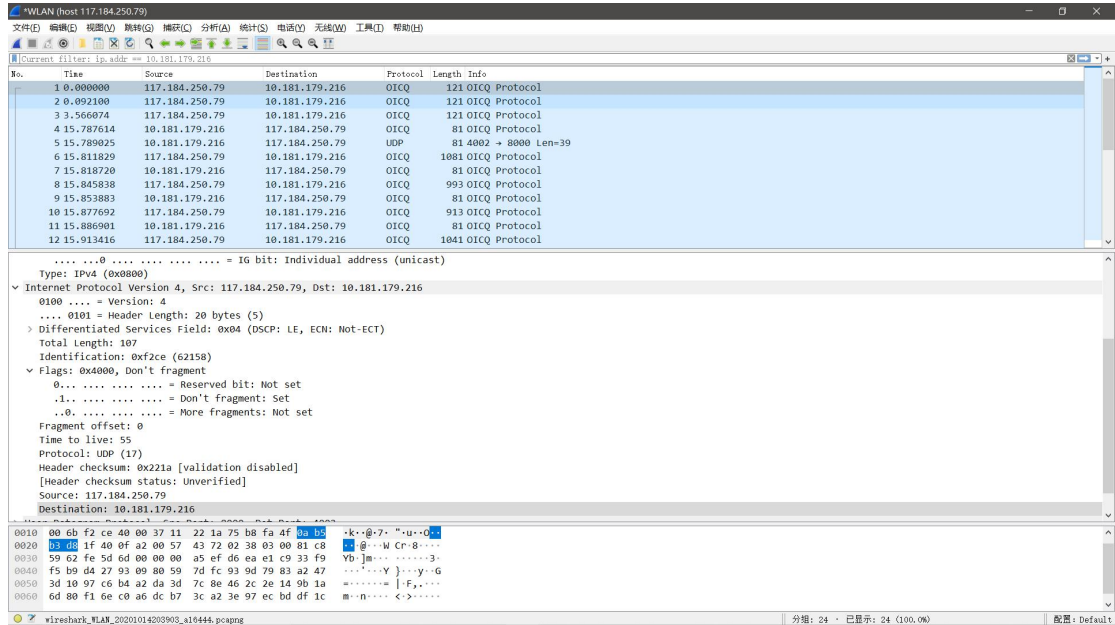
截图：



## 5. 配置捕获过滤器，只捕获某个 IP 地址的数据包（host x.x.x.x）。

使用的过滤器： host 117.184.250.79 ，希望捕获的 IP 地址： 117.184.250.79 。

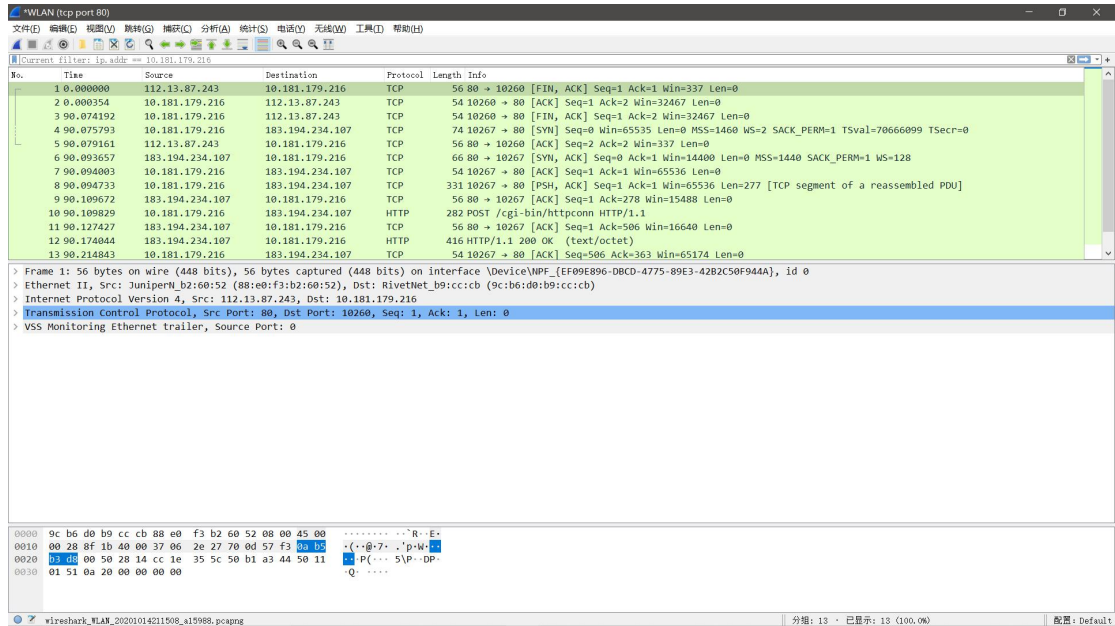
截图：



## 6. 配置捕获过滤器，只捕获某类协议的数据包（tcp port xx 或者 udp port xx）。

使用的过滤器： tcp port 80 ，希望捕获的协议类型： TCP 。

截图：



## ✧ Part Two

任务 1: 使用 nslookup 命令, 查询某个域名, 并捕获这次的数据包。DNS 数据包由哪几层协议构成? Ethernet II, IPv4, UDP, DNS。使用的服务方端口是: 53。

分别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, 标出交易 ID、查询类型、查询的域名内容以及查询结果。

请求包:

3	0.007551	10.181.241.168	10.10.0.21	DNS	75 Standard query 0xef9a A rwsk.zju.edu.cn
> Frame 3: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0					
> Ethernet II, Src: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb), Dst: JuniperN_67:28:52 (88:e0:f3:67:28:52)					
> Internet Protocol Version 4, Src: 10.181.241.168, Dst: 10.10.0.21					
> User Datagram Protocol, Src Port: 53578, Dst Port: 53					
▼ Domain Name System (query)					
Transaction ID: 0xef9a 交易ID					
> Flags: 0x0100 Standard query					
Questions: 1					
Answer RRs: 0					
Authority RRs: 0					
Additional RRs: 0					
▼ Queries					
▼ rwsk.zju.edu.cn: type A, class IN					
Name: rwsk.zju.edu.cn 查询的域名内容					
[Name Length: 15]					
[Label Count: 4]					
Type: A (Host Address) (1) 查询类型					
Class: IN (0x0001)					
[Response In: 4]					
0000	88 e0 f3 67 28 52 9c b6 d0 b9 cc cb 08 00 45 00	...g(R... ..E...			
0010	00 3d d6 49 00 00 80 11 5d ea 0a b5 f1 a8 0a 0a	...I... ].....			
0020	00 15 d1 4a 00 35 00 29 20 d8 ef 9a 01 00 00 01	...J-5... ..			
0030	00 00 00 00 00 00 04 72 77 73 6b 03 7a 6a 75 03	...r wsk-zju-			
0040	65 64 75 02 63 6e 00 00 01 00 01	edu.cn. ....			

响应包:

4	0.010484	10.10.0.21	10.181.241.168	DNS	126 Standard query response 0xef9a A rwsk.zju.edu.cn A 10.203.4.15 NS dns1.zju.edu.cn A 10.10.0.8
> Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0					
> Ethernet II, Src: JuniperN_67:28:52 (88:e0:f3:67:28:52), Dst: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)					
> Internet Protocol Version 4, Src: 10.10.0.21, Dst: 10.181.241.168					
> User Datagram Protocol, Src Port: 53, Dst Port: 53578					
▼ Domain Name System (response)					
Transaction ID: 0xef9a					
> Flags: 0x8500 Standard query response, No error					
Questions: 1					
Answer RRs: 1					
Authority RRs: 1					
Additional RRs: 1					
▼ Queries					
▼ rwsk.zju.edu.cn: type A, class IN					
Name: rwsk.zju.edu.cn					
[Name Length: 15]					
[Label Count: 4]					
Type: A (Host Address) (1)					
Class: IN (0x0001)					
> Answers					
0000	9c b6 d0 b9 cc cb 88 e0 f3 67 28 52 08 00 45 00	...g(R... ..E...			
0010	00 70 3b 8d 00 00 3b 11 3d 74 0a 0a 00 15 0a b5	...p;...;...=t.....			
0020	f1 a8 00 35 d1 4a 00 5c dd 2f ef 9a 85 80 00 01	...5-J\... ..			
0030	00 01 00 01 00 01 04 72 77 73 6b 03 7a 6a 75 03	...r wsk-zju-			
0040	65 64 75 02 63 6e 00 00 01 00 01 c0 0c 00 01 00	edu.cn. ....			
0050	01 00 01 51 80 00 04 0a cb 04 0f c0 11 00 02 00	...Q.....			
0060	01 00 01 51 80 00 07 04 64 6e 73 31 c0 11 c0 3d	...Q..... dns1....=			
0070	00 01 00 01 00 01 51 80 00 04 0a 0a 00 08	...Q.....			



任务 2: 使用 Ping 命令, 分别测试某个 IP 地址和某个域名的连通性, 并捕获数据包。

捕获到了哪些相关协议数据包?

Ping IP 地址时: \_\_\_\_\_ ARP, ICMP

Ping 域名时: \_\_\_\_\_ ARP, ICMP, DNS, TCP, DHCP

ICMP 数据包分别由哪几层协议构成? \_\_\_\_\_ Ethernet II, IPv4, ICMP

分别选择一个 ARP 请求和响应数据包, 展开最高层协议的详细内容, 标出操作码、发送者 IP 地址、发送者 MAC 地址、查询的目标 IP 地址、Ethernet 层的目标 MAC 地址以及查询结果。

```
C:\WINDOWS\system32>ping 202.89.233.100

正在 Ping 202.89.233.100 具有 32 字节的数据:
来自 202.89.233.100 的回复: 字节=32 时间=35ms TTL=117
来自 202.89.233.100 的回复: 字节=32 时间=37ms TTL=117
来自 202.89.233.100 的回复: 字节=32 时间=35ms TTL=117
来自 202.89.233.100 的回复: 字节=32 时间=36ms TTL=117

202.89.233.100 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 35ms, 最长 = 37ms, 平均 = 35ms

C:\WINDOWS\system32>ping www.bing.com

正在 Ping china.bing123.com [202.89.233.101] 具有 32 字节的数据:
来自 202.89.233.101 的回复: 字节=32 时间=36ms TTL=117
来自 202.89.233.101 的回复: 字节=32 时间=36ms TTL=117
来自 202.89.233.101 的回复: 字节=32 时间=37ms TTL=117
来自 202.89.233.101 的回复: 字节=32 时间=36ms TTL=117

202.89.233.101 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 36ms, 最长 = 37ms, 平均 = 36ms
```

请求:

```
112|19.074287|RivetNet_b9:cc:cb|JuniperN_b2:60:52|ARP|42 Who has 10.181.128.1? Tell 10.181.171.142
> Frame 112: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0
  Ethernet II, Src: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb), Dst: JuniperN_b2:60:52 (88:e0:f3:b2:60:52) Ethernet层的目标MAC地址
    Destination: JuniperN_b2:60:52 (88:e0:f3:b2:60:52)
    Source: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)
    Type: ARP (0x0806)
  Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1) 操作码
    Sender MAC address: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb) 发送者MAC地址
    Sender IP address: 10.181.171.142 发送者IP地址
    Target MAC address: JuniperN_b2:60:52 (88:e0:f3:b2:60:52) 查询的目标MAC地址
    Target IP address: 10.181.128.1 查询的目标IP地址

0000 88 e0 f3 b2 60 52 9c b6 d0 b9 cc cb 08 06 00 01 .....R.....
0010 08 00 06 04 00 01 9c b6 d0 b9 cc cb 0a b5 ab 8e .....:..
0020 88 e0 f3 b2 60 52 0a b5 80 01 .....R.....
```

响应:

113	19.076523	JuniperN_b2:60:52	RivetNet_b9:cc:cb	ARP	60	10.181.128.1	is at 88:e0:f3:b2:60:52
> Frame 113: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0							
Ethernet II, Src: JuniperN_b2:60:52 (88:e0:f3:b2:60:52), Dst: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)							
> Destination: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)							
> Source: JuniperN_b2:60:52 (88:e0:f3:b2:60:52)							
Type: ARP (0x0806)							
Padding: 00000000000000000000000000000000							
Address Resolution Protocol (reply)							
Hardware type: Ethernet (1)							
Protocol type: IPv4 (0x0800)							
Hardware size: 6							
Protocol size: 4							
Opcode: reply (2)							
Sender MAC address: JuniperN_b2:60:52 (88:e0:f3:b2:60:52)							
Sender IP address: 10.181.128.1							
Target MAC address: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)							
Target IP address: 10.181.171.142							
0000	9c b6 d0 b9 cc cb	88 e0 f3 b2 60 52	08 06 00 01	.....R....			
0010	08 00 06 04 00 02	88 e0 f3 b2 60 52	0a b5 80 01	.....R....			
0020	9c b6 d0 b9 cc cb	0a b5 ab 8e 00 00	00 00 00 00	.....			
0030	00 00 00 00 00 00	00 00 00 00 00 00		.....			

分别选择一个 ICMP 请求和响应数据包，展开最高层协议的详细内容，标出类型、序号。

请求:

8	3.029815	10.181.171.142	202.89.233.100	ICMP	74	Echo (ping) request	id=0x0001, seq=28/7168, ttl=64 (reply in 9)
> Frame 8: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0							
> Ethernet II, Src: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb), Dst: JuniperN_b2:60:52 (88:e0:f3:b2:60:52)							
> Internet Protocol Version 4, Src: 10.181.171.142, Dst: 202.89.233.100							
Internet Control Message Protocol							
Type: 8 (Echo (ping) request) 类型							
Code: 0							
Checksum: 0x4d3f [correct]							
[Checksum Status: Good]							
Identifier (BE): 1 (0x0001)							
Identifier (LE): 256 (0x0100)							
Sequence number (BE): 28 (0x001c) 序号							
Sequence number (LE): 7168 (0x1c00)							
[Response frame: 9]							
Data (32 bytes)							
Data: 6162636465666768696a6b6c6d6e6f707172737475767761...							
[Length: 32]							
0000	88 e0 f3 b2 60 52 9c b6	d0 b9 cc cb 08 00 45 00	.....R.....E..				
0010	00 3c 1b 3a 00 00 40 01	f5 85 0a b5 ab 8e ca 59	-<:-@.....Y				
0020	e9 64 08 00 4d 3f 00 01	00 1c 61 62 63 64 65 66	-d-M?.. 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				

响应:

225	17.737280	115.239.210.27	172.16.14.124	ICMP	74	Echo (ping) reply	id=0x0001, seq=26/6656, ttl=56 (request in 224)
> Frame 225: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0							
> Ethernet II, Src: JuniperN_e1:07:00 (00:26:88:e1:07:00), Dst: Giga-Byt_4d:e7:0a (90:2b:34:4d:e7:0a)							
> Internet Protocol Version 4, Src: 115.239.210.27, Dst: 172.16.14.124							
Internet Control Message Protocol							
Type: 0 (Echo (ping) reply)							
Code: 0							
Checksum: 0x5541 [correct]							
[Checksum Status: Good]							
Identifier (BE): 1 (0x0001)							
Identifier (LE): 256 (0x0100)							
Sequence number (BE): 26 (0x001a)							
Sequence number (LE): 6656 (0x1a00)							
[Request frame: 224]							
[Response time: 4.538 ms]							
Data (32 bytes)							
Data: 6162636465666768696a6b6c6d6e6f707172737475767761...							
[Length: 32]							
0000	90 2b 34 4d e7 0a 00 26	88 e1 07 00 08 00 45 00	..+M...& .....E.				
0010	00 3c 2d 3c 00 00 38 01	54 ee 73 ef d2 1b ac 10	.-<-..8. T.s.....				
0020	0e 7c 00 00 55 41 00 01	00 1a 61 62 63 64 65 66	.. .UA.. 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				



任务 3：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。跟踪路由使用的数据包协议类型是： ICMP ，数据包由几层协议构成？ 3 层，分别是 EthernetII, IPv4, ICMP 。

```
C:\WINDOWS\system32>Tracert 10.203.6.126

通过最多 30 个跃点跟踪到 10.203.6.126 的路由

  1          5 ms      2 ms      2 ms    10.0.2.1
  2         103 ms     3 ms      2 ms    10.3.1.18
  3          3 ms      3 ms     10 ms    10.3.7.78
  4          72 ms     21 ms     3 ms    10.3.7.226
  5          3 ms      9 ms      5 ms    10.203.6.126

跟踪完成。
```

观察并记录请求包中 IP 协议层的 TTL 字段变化规律，第一个请求的 TTL 等于 1 ，同样 TTL 的请求连续发送了 3 个，然后每次 TTL 增加了 1 ，最后一个请求的 TTL 等于 5 。附上截图：

No.	Time	Source	Destination	Protocol	Length	Info
28	9.735203	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=102/26112, ttl=1 (no response found!)
29	9.740296	10.0.2.1	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
30	9.743372	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=103/26368, ttl=1 (no response found!)
31	9.745873	10.0.2.1	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
32	9.750776	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=104/26624, ttl=1 (no response found!)
33	9.753233	10.0.2.1	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
61	19.822508	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=105/26880, ttl=2 (no response found!)
62	19.925454	10.3.1.18	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
63	19.928950	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=106/27136, ttl=2 (no response found!)
64	19.931976	10.3.1.18	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
65	19.934989	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=107/27392, ttl=2 (no response found!)
66	19.937450	10.3.1.18	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
106	30.007115	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=108/27648, ttl=3 (no response found!)
107	30.010129	10.3.7.78	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
108	30.013063	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=109/27904, ttl=3 (no response found!)
109	30.016078	10.3.7.78	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
110	30.019047	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=110/28160, ttl=3 (no response found!)
111	30.029373	10.3.7.78	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
169	40.083025	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=111/28416, ttl=4 (no response found!)
170	40.155286	10.3.7.226	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
171	40.158745	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=112/28672, ttl=4 (no response found!)
172	40.180070	10.3.7.226	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
173	40.183201	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=113/28928, ttl=4 (no response found!)
174	40.186969	10.3.7.226	10.181.171.142	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
205	50.252119	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=114/29184, ttl=5 (reply in 206)
206	50.255171	10.203.6.126	10.181.171.142	ICMP	106	Echo (ping) reply id=0x0001, seq=114/29184, ttl=60 (request in 205)
207	50.258298	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=115/29440, ttl=5 (reply in 208)
208	50.267598	10.203.6.126	10.181.171.142	ICMP	106	Echo (ping) reply id=0x0001, seq=115/29440, ttl=60 (request in 207)
209	50.270648	10.181.171.142	10.203.6.126	ICMP	106	Echo (ping) request id=0x0001, seq=116/29696, ttl=5 (reply in 210)
210	50.276353	10.203.6.126	10.181.171.142	ICMP	106	Echo (ping) reply id=0x0001, seq=116/29696, ttl=60 (request in 209)

观察并记录响应包的信息，第一组响应包的发送者 IP 是： 10.0.2.1 ，标记 ICMP 层的类型字段。最后一组响应包的发送者 IP 是： 10.203.6.126 ，标记 ICMP 层的类型字段。附上截图：

第一组：

29	9.740296	10.0.2.1	10.181.171.142	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
>	Frame 29: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0				
>	Ethernet II, Src: JuniperN_b2:60:52 (88:e0:f3:b2:60:52), Dst: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)				
>	Internet Protocol Version 4, Src: 10.0.2.1, Dst: 10.181.171.142				
√	Internet Control Message Protocol				
	Type: 11 (Time-to-live exceeded)				
	Code: 0 (Time to live exceeded in transit)				
	Checksum: 0xf4ff [correct]				
	[Checksum Status: Good]				
	Unused: 00000000				
>	Internet Protocol Version 4, Src: 10.181.171.142, Dst: 10.203.6.126				
√	Internet Control Message Protocol				
	Type: 8 (Echo (ping) request)				
	Code: 0				
	Checksum: 0xf798 [unverified] [in ICMP error packet]				
	[Checksum Status: Unverified]				
	Identifier (BE): 1 (0x0001)				
	Identifier (LE): 256 (0x0100)				
	Sequence number (BE): 102 (0x0066)				
	Sequence number (LE): 26112 (0x6600)				

最后一组：

210	50.276353	10.203.6.126	10.181.171.142	ICMP	106 Echo (ping) reply id=0x0001, seq=116/29696, ttl=60 (request in 209)
>	Frame 210: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0				
>	Ethernet II, Src: JuniperN_b2:60:52 (88:e0:f3:b2:60:52), Dst: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb)				
>	Internet Protocol Version 4, Src: 10.203.6.126, Dst: 10.181.171.142				
√	Internet Control Message Protocol				
	Type: 0 (Echo (ping) reply)				
	Code: 0				
	Checksum: 0xff8a [correct]				
	[Checksum Status: Good]				
	Identifier (BE): 1 (0x0001)				
	Identifier (LE): 256 (0x0100)				
	Sequence number (BE): 116 (0x0074)				
	Sequence number (LE): 29696 (0x7400)				
	[Request frame: 209]				
	[Response time: 5.705 ms]				
>	Data (64 bytes)				

✧ Part Three

1. 运行 `ipconfig /flushdns` 命令清空 DNS 缓存，然后打开浏览器，访问 `www.zju.edu.cn`，并使用捕获过滤器只捕获访问该网站的数据（过滤器设置：`tcp port 80 or udp port 53`），网页完全打开后，停止捕获。

捕获到的这些最高层的协议数据包分别由哪几层协议构成？

DNS: Ethernet II, IPv4, UDP, DNS

HTTP: Ethernet II, IPv4, TCP, HTTP

每种协议选取一个代表展开后截图，并标出源和目标 IP 地址、源和目标端口）

DNS:

21	0.581313	10.181.171.142	10.10.0.21	DNS	76 Standard query 0xd970 A zdzsc.zju.edu.cn
>	Frame 21: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0				
>	Ethernet II, Src: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb), Dst: JuniperN_b2:60:52 (88:e0:f3:b2:60:52)				
>	Internet Protocol Version 4, Src: 10.181.171.142, Dst: 10.10.0.21				
>	User Datagram Protocol, Src Port: 63023, Dst Port: 53				
>	Domain Name System (query)				

HTTP:

No.	Time	Source	Destination	Protocol	Length	Info
28	1.281369	10.181.171.142	10.203.6.126	HTTP	734	GET /_upload/tpl/05/e5/1509/template1509/images/favicon.ico HTTP/1.1
>	Frame 28: 734 bytes on wire (5872 bits), 734 bytes captured (5872 bits) on interface \Device\NPF_{EF09E896-DBCD-4775-89E3-42B2C50F944A}, id 0					
>	Ethernet II, Src: RivetNet_b9:cc:cb (9c:b6:d0:b9:cc:cb), Dst: JuniperN_b2:60:52 (88:e0:f3:b2:60:52)					
>	Internet Protocol Version 4, Src: 10.181.171.142, Dst: 10.203.6.126					
>	Transmission Control Protocol, Src Port: 1163, Dst Port: 80, Seq: 1295, Ack: 13380, Len: 668					
>	Hypertext Transfer Protocol					

2. 为了打开网页，浏览器查询了哪些相关的域名？

域名列表： zdzsc.zju.edu.cn www.beian.miit.gov.cn weibo.com

3. 使用显示过滤器 `tcp.stream eq X`，让 X 从 0 开始变化，直到没有数据。分析浏览器为了获取网页数据，总共建立了几个连接？（一个 TCP 流对应一个 TCP 连接）

TCP 连接数： 3

4. 右键点击某个 HTTP 数据包，选择跟踪 TCP 流，可以看到 HTTP 会话的数据。分析浏览器与 WEB 服务器之间进行了几次 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。

HTTP 会话数： stream0,1,2 分别进行了 0,5,1 次对话，共 6 次

5. 选择一个 HTTP 的 TCP 流进行截图，标出请求和响应部分（最好有多个 HTTP 会话的）：



```
GET / HTTP/1.1
Host: www.zju.edu.cn
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: route=42bbc6a24acf9f019080162ce2732518; Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1602897938,1602921569,1602921594,1602921652; JSESSIONID=8A1C3A2FD90FA3AF9B16B9824246F1D4; Hm_lpvt_fe30bbc1ee45421ec1679d1b8d8f8453=1602921682

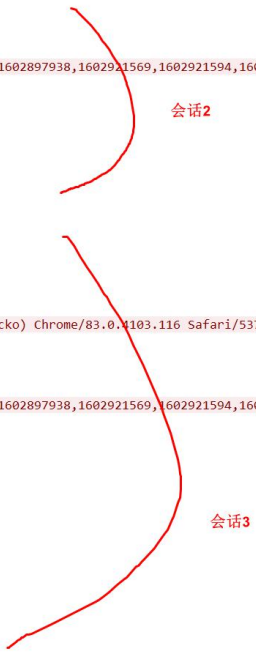
HTTP/1.1 200 OK
Server: nginx
Date: Sat, 17 Oct 2020 08:09:11 GMT
Content-Type: text/html
Content-Length: 12941
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Frame-Options: SAMEORIGIN
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
```

```
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: http://www.zju.edu.cn/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: route=42bbc6a24acf9f019080162ce2732518; Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1602897938,1602921569,1602921594,1602921652; JSESSIONID=8A1C3A2FD90FA3AF9B16B9824246F1D4; Hm_lpvt_fe30bbc1ee45421ec1679d1b8d8f8453=1602921724

HTTP/1.1 200 OK
Server: nginx
Date: Sat, 17 Oct 2020 08:09:12 GMT
Content-Length: 0
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Frame-Options: SAMEORIGIN

GET /_upload/tpl/05/e5/1509/template1509/images/favicon.ico HTTP/1.1
Host: www.zju.edu.cn
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: http://www.zju.edu.cn/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: route=42bbc6a24acf9f019080162ce2732518; Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1602897938,1602921569,1602921594,1602921652; JSESSIONID=8A1C3A2FD90FA3AF9B16B9824246F1D4; Hm_lpvt_fe30bbc1ee45421ec1679d1b8d8f8453=1602921724

HTTP/1.1 200 OK
Server: nginx
Date: Sat, 17 Oct 2020 08:09:12 GMT
Content-Type: image/x-icon
Content-Length: 432
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Frame-Options: SAMEORIGIN
Last-Modified: Tue, 30 Jul 2019 15:40:43 GMT
ETag: "57e-58ee7d4d6f4c0-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
```



## 六、实验结果分析与思考

- 如果只想捕获某个特定 WEB 服务器 IP 地址相关的 HTTP 数据包，捕获过滤器应该怎么写？

ip host x.x.x.x && http port 80

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

Ping 发送的是 ICMP 类型的协议数据包。

ARP 是消息解析协议，当需要根据 IP 地址获取物理地址且在 ARP 缓存中没有找到映射的时候就会产生 ARP 消息。

Ping IP 地址时采用的是 ICMP 协议，Ping 域名时采用的是 ICMP 和 TCP 协议。如果计算机内没有缓存 Ping 域名对应的 IP 地址时，命令会先对域名进行解析。域名和 IP 并不是一一对应的关系，因此当 Ping 域名的时候可以得到一个 IP 为目标，也可能得到域名的多负载不同的 IP；但是 Ping ip 的时候得到的是整个服务器，而不是某一个域名。

- 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？

Tracert/Traceroute 发送的是 ICMP 类型的协议数据包。

在路由跟踪过程中，通过向目标发送不同 IP 生存时间 (TTL) 值的 “Internet 控制消息协议 (ICMP)” 回应数据包，Tracert 诊断程序确定到目标所采取的路由。要求路径上的每个路由器在转发数据包之前至少将数据包上的 TTL 递减 1。数据包上的 TTL 减为 0 时，路由器应该将“ICMP 已超时”的消息发回源系统。Tracert 先发送 TTL 为 1 的回应数据包，并在随后的每次发送过程将 TTL 递增 1，直到目标响应或 TTL 达到最大值，从而确定路由。通过检查中间路由器发回的“ICMP 已超时”的消息确定路由。

- 如何理解 TCP 连接和 HTTP 会话？他们之间存在什么关系？

#### **TCP 连接：**

在 HTTP 的规范内，两台计算机的交互被视为 request 和 response 的传递。而在实际的 TCP 操作中，信息传递会比单纯的传递 request 和 response 要复杂。通过 TCP 建立的通讯往往需要计算机之间多次的交换信息才能完成一次 request 或 response。

TCP 的传输数据的核心是在于将数据分为若干段并将每段数据按顺序标记。标记后的顺序可以以不同的顺序被另一方接收并集成回完整的数据。计算机对每一段数据的成功接收都会做出相应，确保所有数据的完整性。

TCP 在传递数据时依赖于实现定义好的几个标记 (Flags) 去向另一方表态传达数据和连接的状态。也是基于这些标志 TCP 可以实现三次 (three ways handshake) 和四次握手 (four ways tear down)。三次握手是初步建立连接的机制，而四次握手则是断开链接。

#### **HTTP 会话：**

1. HTTP 存在于应用层之外，它的标准建立在将两台计算机视为不同的角色：客户端和服务端。客户端会向服务器传送不同的请求(request)，而服务器会对应每个请求给出回应(response)。

2. HTTP 属于无状态协议(Stateless)。这表示每一个请求之间是没有相关性的。在该协议的规则中服务器是不会记录任何客户端操作，每一次请求都是独立的。(记录用户浏览行为会通过其他技术实现)

3. 客户端的请求被定义在几个动词意义范围内。最常用到的是 GET 和 POST，

其他动词还包括 DELETE, HEAD 等等。

4. 服务器的回应被定义在几个状态码之间：5 开头表示服务器错误，4 开头表示客户端错误，3 开头表示需要做进一步处理，2 开头表示成功，1 开头表示在请求被接受处理的同时提供的额外信息。

5. 不管是客户端的请求信息还是服务器的回应，双方都拥有一块头部信息 (Header)。头部信息是自定义，其用途在于传递额外信息（浏览器信息、请求的内容类型、相应的语言）。

#### **关系：**

TCP 是传输层通讯协议，定义的是数据传输和连接方式的规范，HTTP 是应用层协议，定义的是传输数据的内容的规范。

HTTP 协议中的数据是利用 TCP 协议传输的，TCP 保证 HTTP 层能拿到正确格式的包。HTTP 必须运行在 TCP 上，由 TCP 帮他可靠传输。这样他可以专心处理收到的信息，无需关心收到的信息是不是有问题，不能解析。

HTTP 规定了每段数据以什么形式表达才是能够被另外一台计算机理解。而 TCP 所要规定的是数据应该怎么传输才能稳定且高效的传递与计算机之间。

#### ● DNS 为什么选择使用 UDP 协议进行传输？而 HTTP 为什么选择使用 TCP 协议？

DNS 使用 UDP 协议作为传输层协议的主要原因是为了避免使用 TCP 协议时造成的连接时延。因为为了得到一个域名的 IP 地址，往往会向多个域名服务器查询，如果使用 TCP 协议，那么每次请求都会存在连接时延，这样使 DNS 服务变得很慢，因为大多数的地址查询请求，都是浏览器请求页面时发出的，这样会造成网页的等待时间过长。而 UDP 协议的额外开销小、有着更好的性能表现。

HTTP 选择使用 TCP 协议则是因为 UDP 不安全不可靠，会产生丢包等，因此它只能主要应用在不安全性要求不高、效率要求比较高的应用程序比如聊天程序上，而 HTTP 要处理电子商务的应用。HTTP 协议只定义了应用层的东西，下层的可靠性要传输层来保证，而 TCP 可以保证传输的可靠性。



## 七、 讨论、心得

在完成本实验后，你可能会有很多待解答的问题，你可以把它们记在这里，接下来的学习中，你也许会逐渐得到答案的，同时也可以让老师了解到你有哪些困惑，老师在课堂可以安排针对性地解惑。等到课程结束后，你再回头看看这些问题时你或许会有不同的见解：

1. TCP 三次握手的实现过程；
2. ARP 协议的工作原理；

在实验过程中你可能会遇到的困难，并得到了宝贵的经验教训，请把它们记录下来，提供给其他人参考吧：

1. WireShark 的两种过滤器是不同的，显示过滤器只针对已经捕获的报文，过滤出符合过滤规则的报文；而捕获过滤器则是提前设置好过滤规则，只捕获符合过滤规则的报文。它们的语法也不同。
2. 在配置应用显示过滤器的时候，要注意用协议名称的小写。
3. 需要以管理员身份运行命令提示符，通过 `net start npf` 命令重启网卡服务。

你对本实验安排有哪些更好的建议呢？欢迎献计献策：

本实验的实验提示比较详细，总体难度并不高，也许可以提出更多探究性的问题让我们自行解决，这样印象也会更加深刻。