北京郵電大學

实验报告

题目：IP 和 TCP 数据分组的捕获和解析

班　　级：2021211301

学　　号：2021210724

姓　　名：任飞

学　　院：计算机学院

2023 年 6 月 15 日

一、实验内容和实验目的

1. 捕获在连接 Internet 过程中产生的网络层分组：DHCP 分组，ARP 分组，IP 数据分组，ICMP 分组。
2. 分析各种分组的格式，说明各种分组在建立网络连接过程中的作用。
3. 分析 IP 数据分组分片的结构。通过本次实验了解计算机上网的工作过程，学习各种网络层分组的格式及其作用，理解长度大于 1500 字节 IP 数据组分片传输的结构。
4. 分析 TCP 建立连接，拆除连接和数据通信的流程。

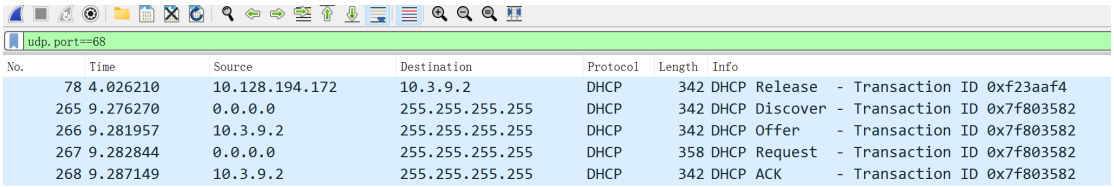
二、实验设备环境

操作系统为 Windows 11
笔记本网卡为 MediaTek MT7921
使用 Wireshark 4.0.6

三、实验过程

1. 捕获 DHCP 报文并分析、观察 ping 命令

将 Wireshark 显示过滤器设置为 udp.port==68，先使用 ipconfig /release 命令释放已经申请的 IP 地址，然后执行 ipconfig /renew 获取 IP 地址。在 Wireshark 中抓到以下报文：



第一个 Release 报文为一开始释放 IP 地址的报文，在此忽略。通过 DHCP 获取 IP 地址通过 Discover Offer Request ACK 四个报文完成。下面进行具体的分析。

首先，客户端广播一个 DHCP Discover 报文：

```
> Frame 265: 342 bytes on wire (2736 bits), 342 bytes captured (2736
> Ethernet II, Src: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19), Dst: Broa
> Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
> User Datagram Protocol, Src Port: 68, Dst Port: 67
v Dynamic Host Configuration Protocol (Discover)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x7f803582
    Seconds elapsed: 0
  > Bootp flags: 0x8000, Broadcast flag (Broadcast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  > Option: (53) DHCP Message Type (Discover)
  > Option: (61) Client identifier
  > Option: (50) Requested IP Address (10.128.194.172)
  > Option: (12) Host Name
  > Option: (60) Vendor class identifier
  > Option: (55) Parameter Request List
  > Option: (255) End
    Padding: 00000000
```

注意到 IP 包头中源地址为 0.0.0.0，这是由于客户端当前还没有分配到 IP 地址，故使用 0.0.0.0 占位；IP 包头中目的地址为 255.255.255.255，这是由于客户端不知道 DHCP 服务器的 IP 地址，必须进行广播。

下面分析该报文中的一些值得讨论的字段：

- Message type 只能为 1 或 2，1 表示客户发出的请求，2 表示服务端的应答。
- Transaction ID 为一个随机标识符，服务器的应答必须带上相同的标识符，以区分不同的 DHCP 请求。
- Bootp flags 字段目前只有最高一位被用于指定应答报文是否广播，其他位都保留备用。在本报文中最高位为 1，表示客户端希望服务器使用广播的方式回应。

DHCP 报文中还附带了若干选项（Option），下面是一些值得讨论的选项：

- (Option 53) DHCP Message Type 指示了该报文的类型为 Discover
- (Option 50) Requested IP Address 指示了客户端希望获得的 IP 地址，为了尽可能维持网络的稳定，客户端会尽可能要求获得和之前一样的 IP 地址，服务器会尽可能满足客户端的要求。
- (Option 55) Parameter Request List 指示了客户端希望获取的额外信息，包括但不限于子网掩码、默认网关、DNS 服务器等。

接下来是服务器发来的 DHCP Offer 报文。Offer 报文表示服务器给客户端提供了一个可供使用的 IP 地址，但最终是否使用仍需要客户端自己决定。

```
> Frame 266: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bi
> Ethernet II, Src: ArubaaHe_6c:0c:00 (10:4f:58:6c:0c:00), Dst: CloudNe
> Internet Protocol Version 4, Src: 10.3.9.2, Dst: 255.255.255.255
> User Datagram Protocol, Src Port: 67, Dst Port: 68
v Dynamic Host Configuration Protocol (Offer)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 1
    Transaction ID: 0x7f803582
    Seconds elapsed: 0
  > Bootp flags: 0x8000, Broadcast flag (Broadcast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 10.128.194.172
    Next server IP address: 0.0.0.0
    Relay agent IP address: 10.128.128.1
    Client MAC address: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  > Option: (53) DHCP Message Type (Offer)
  > Option: (54) DHCP Server Identifier (10.3.9.2)
  > Option: (51) IP Address Lease Time
  > Option: (1) Subnet Mask (255.255.128.0)
  > Option: (3) Router
  > Option: (6) Domain Name Server
  > Option: (255) End
    Padding: 0000000000000000000000000000000000000000000000000000
```

下面分析在这个报文中值得讨论的字段和选项：

- Hops 表示 DHCP 报文经过中继的次数。在这里 Hops 为 1，表示经过了一次中继。与之有关的字段是 Relay agent IP address，表示 DHCP 中继的地址，在这里实际上就是校园网的默认网关。默认网关将 DHCP 报文中继给真正的 DHCP 服务器，然后再回复给客户端。
- Your (client) IP address 表示服务端提供的 IP 地址。

- Option(54) DHCP Server Identifier 表示 DHCP 服务器的信息，在这里就是 DHCP 服务器的 IP 地址。这个字段将被用于区分多个服务器。
- Option(51) IP Address Lease Time 表示租约时长，展开后可看到时长为两小时。
- Option(1) Subnet Mask, Option(3) Router, Option(6) Domain Name Server 提供了客户端需要的额外信息，分别为子网掩码（255.255.128.0）、默认网关（10.128.128.1）和 DNS 服务器（10.3.9.44 和 10.3.9.45）。

一个有趣之处在于，这个报文的 IP 包头的目的地址是 255.255.255.255，遵循了客户端的使用广播的要求。但是如果看二层帧头，会发现目的 MAC 地址实际上是单播地址而不是广播地址。也就是说这是一个在三层“伪装”成广播但在二层实际是单播的报文。猜测这样的设计在满足协议要求的前提下减少了网络中实际的广播流量。

接下来客户端确认使用这个 IP 地址，发送 DHCP Request 报文向服务器请求使用这个 IP 地址。

```
> Frame 267: 358 bytes on wire (2864 bits), 358 bytes captured (286
> Ethernet II, Src: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19), Dst: Bro
> Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
> User Datagram Protocol, Src Port: 68, Dst Port: 67
v Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x7f803582
    Seconds elapsed: 0
  > Bootp flags: 0x8000, Broadcast flag (Broadcast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  > Option: (53) DHCP Message Type (Request)
  > Option: (61) Client identifier
  > Option: (50) Requested IP Address (10.128.194.172)
  > Option: (54) DHCP Server Identifier (10.3.9.2)
  > Option: (12) Host Name
  > Option: (81) Client Fully Qualified Domain Name
  > Option: (60) Vendor class identifier
  > Option: (55) Parameter Request List
  > Option: (255) End
```

下面分析在这个报文中值得讨论的选项：

- Option(50) Requested IP Address 指示客户端要使用的 IP 地址。
- Option(54) DHCP Server Identifier 指示 DHCP 服务器的信息。由于 Request 报文仍然是广播的，需要用这个字段来确定在向哪个服务器发出请求，而其他 DHCP 服务器收到这个广播包后也能知道自己的 Offer 被拒绝了。

最后服务器回应 DHCP ACK 报文，表示确认成功使用该 IP 地址。

```
> Frame 268: 342 bytes on wire (2736 bits), 342 bytes captured (2736 b
> Ethernet II, Src: ArubaaHe_6c:0c:00 (10:4f:58:6c:0c:00), Dst: CloudN
> Internet Protocol Version 4, Src: 10.3.9.2, Dst: 255.255.255.255
> User Datagram Protocol, Src Port: 67, Dst Port: 68
v Dynamic Host Configuration Protocol (ACK)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 1
    Transaction ID: 0x7f803582
    Seconds elapsed: 0
  > Bootp flags: 0x8000, Broadcast flag (Broadcast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 10.128.194.172
    Next server IP address: 0.0.0.0
    Relay agent IP address: 10.128.128.1
    Client MAC address: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  > Option: (53) DHCP Message Type (ACK)
  > Option: (54) DHCP Server Identifier (10.3.9.2)
  > Option: (51) IP Address Lease Time
  > Option: (1) Subnet Mask (255.255.128.0)
  > Option: (3) Router
  > Option: (6) Domain Name Server
  > Option: (255) End
    Padding: 000000000000000000000000000000000000000000000000
```

这个报文中没有新的值得讨论的字段或选项。

至此通过 DHCP 获取 IP 地址的全过程完成。此后客户端就可以使用分配到的 IP 地址了。

下面进行观察 ping 命令的实验。

重新设置 Wireshark 的显示过滤器为 icmp or arp。对同一无线网中另一台设备（10.128.205.138）执行 ping，该设备此前不在本设备的 ARP 表中。

Wireshark 抓包得到以下结果：

icmp or arp

No.	Time	Source	Destination	Protocol	Length	Info
7	0.897610	CloudNet_a0:1e:19	Broadcast	ARP	42	Who has 10.128.205.138? Tell 10.128.194.172
8	1.007240	LiteonTe_a5:cd:cb	CloudNet_a0:1e:19	ARP	42	10.128.205.138 is at 74:4c:a1:a5:cd:cb
9	1.007270	10.128.194.172	10.128.205.138	ICMP	74	Echo (ping) request id=0x0001, seq=142/36352, ttl=128 (reply in 10)
10	1.013803	10.128.205.138	10.128.194.172	ICMP	74	Echo (ping) reply id=0x0001, seq=142/36352, ttl=128 (request in 9)
11	1.913078	10.128.194.172	10.128.205.138	ICMP	74	Echo (ping) request id=0x0001, seq=143/36608, ttl=128 (reply in 12)
12	1.928915	10.128.205.138	10.128.194.172	ICMP	74	Echo (ping) reply id=0x0001, seq=143/36608, ttl=128 (request in 11)
14	2.927185	10.128.194.172	10.128.205.138	ICMP	74	Echo (ping) request id=0x0001, seq=144/36864, ttl=128 (reply in 15)
15	2.952684	10.128.205.138	10.128.194.172	ICMP	74	Echo (ping) reply id=0x0001, seq=144/36864, ttl=128 (request in 14)
23	3.939040	10.128.194.172	10.128.205.138	ICMP	74	Echo (ping) request id=0x0001, seq=145/37120, ttl=128 (reply in 24)
24	3.976627	10.128.205.138	10.128.194.172	ICMP	74	Echo (ping) reply id=0x0001, seq=145/37120, ttl=128 (request in 23)

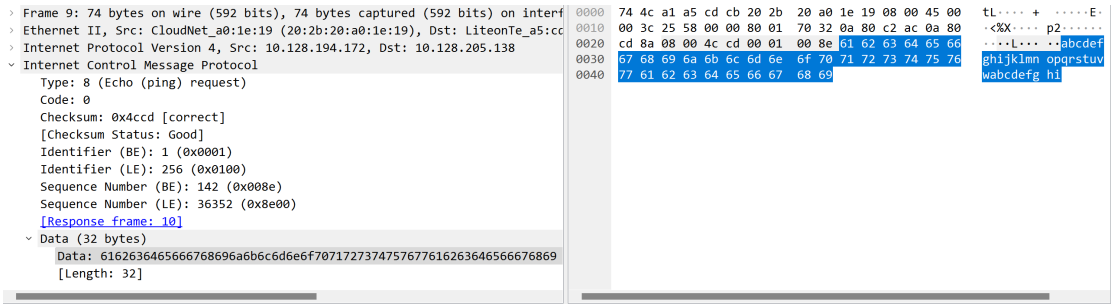
首先，本设备的 ARP 表中没有 10.128.205.138 的记录，因此发出一个 ARP 请求报文。ARP 请求报文为广播报文，内容包括发送方的 IP 地址、MAC 地址，以及目标的 IP 地址、MAC 地址（请求时目标的 MAC 地址还不知道，因此用全 0 占位）。

```
> Frame 7: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on i
> Ethernet II, Src: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19), Dst: Broadcast
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19)
    Sender IP address: 10.128.194.172
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.128.205.138
```

目标设备收到 ARP 请求后返回一个单播的 ARP 回应。内容同样包括发送方的 IP 地址、MAC 地址，以及目标的 IP 地址、MAC 地址。本设备收到 ARP 回应后就获知了目标设备的 MAC 地址，并写入本地的 ARP 表。

```
> Frame 8: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) (
> Ethernet II, Src: LiteonTe_a5:cd:cb (74:4c:a1:a5:cd:cb), Dst: CloudI
v Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: LiteonTe_a5:cd:cb (74:4c:a1:a5:cd:cb)
    Sender IP address: 10.128.205.138
    Target MAC address: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19)
    Target IP address: 10.128.194.172
```

接下来是 4 次 ping 产生的一共 8 个 ICMP 报文。以第一个报文为例：



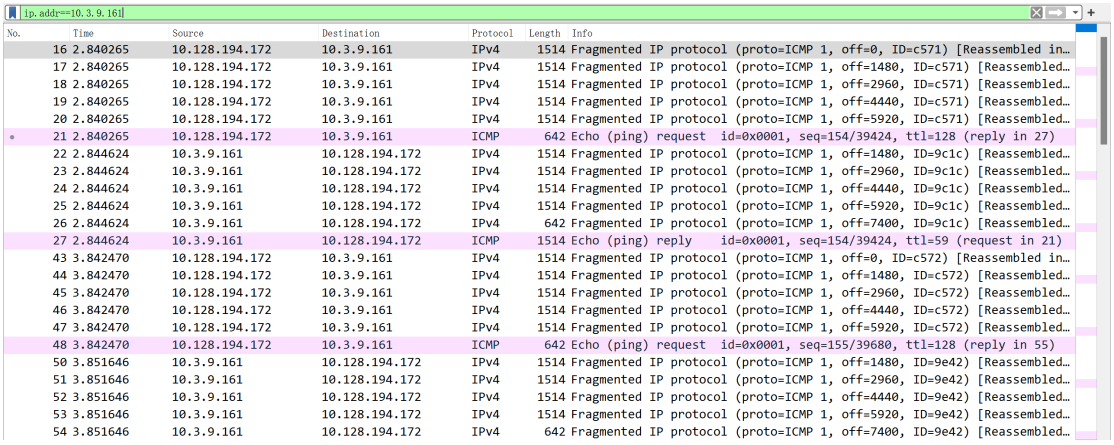
这是一个类型为 8（Echo）的 ICMP 报文，携带的数据长度为 32 字节。有趣的是携带的数据内容是 abcdefg…，对方回应的 ICMP 报文携带的内容也要完全相同。

2．分析数据分组的分片传输过程

制作大于 8000 字节的 IP 数据分组并发送，捕获后分析其分片传输的分组结构。

本实验选择 ping 北邮官网服务器，使用-4 选项强制使用 IPv4。

输入命令 ping -l 8000 -4 www.bupt.edu.cn，设置 Wireshark 的显示过滤器为 ip.addr==10.3.9.161（10.3.9.161 即为 www.bupt.edu.cn 的 ip 地址）。捕获到如下内容：



可以看到，每个 ICMP 包都被拆分为了 6 片，并且可以观察到链路的 MTU 是 1500。除去 IP 头后，前五个分片每个包含 1480 字节，最后一个分片包含 608 字节，一共 8008 字

节，正好是 8 字节的 ICMP 头加上 8000 字节的数据。

下面观察第一个分片的 IP 头信息：

```
> Frame 16: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
> Ethernet II, Src: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19), Dst: ArubaaHe_6c:0c
v Internet Protocol Version 4, Src: 10.128.194.172, Dst: 10.3.9.161
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xc571 (50545)
  > 001. .... = Flags: 0x1, More fragments
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x6edf [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.128.194.172
    Destination Address: 10.3.9.161
    [Reassembled IPv4 in frame: 21]
v Data (1480 bytes)
    Data: 0800eb5a0001009a6162636465666768696a6b6c6d6e6f7071727374757677616 26
    [Length: 1480]
```

该分片中 MF 位为 1，表示这不是一个完整的包，后面还有更多分片。Fragment Offset 为 0，表示这个分片在包中的偏移量为 0，即第一个分片。

第一个包的第二个分片与之类似，但 Fragment Offset 变为 1480，表示这个分片在包中的偏移量为 1480。

```
> Frame 17: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
> Ethernet II, Src: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19), Dst: ArubaaHe_6c:0
v Internet Protocol Version 4, Src: 10.128.194.172, Dst: 10.3.9.161
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xc571 (50545)
  > 001. .... = Flags: 0x1, More fragments
    ...0 0000 1011 1001 = Fragment Offset: 1480
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x6e26 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.128.194.172
    Destination Address: 10.3.9.161
    [Reassembled IPv4 in frame: 21]
v Data (1480 bytes)
    Data: 6162636465666768696a6b6c6d6e6f707172737475767761626364656667686 6a6
    [Length: 1480]
```

下面直接看第一个包的最后一个分片。这里的 MF 位变为 0，表示后面不再有更多的分片。Fragment Offset 为 7400，表示这个分片在包中的偏移量为 7400。

```
> Frame 21: 642 bytes on wire (5136 bits), 642 bytes captured (5136 bits) on
> Ethernet II, Src: CloudNet_a0:1e:19 (20:2b:20:a0:1e:19), Dst: ArubaaHe_6c:0
v Internet Protocol Version 4, Src: 10.128.194.172, Dst: 10.3.9.161
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 628
    Identification: 0xc571 (50545)
  > 000. .... = Flags: 0x0
    ...0 0011 1001 1101 = Fragment Offset: 7400
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x8eaa [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.128.194.172
    Destination Address: 10.3.9.161
  > [6 IPv4 Fragments (8008 bytes): #16(1480), #17(1480), #18(1480), #19(1480
> Internet Control Message Protocol
```

同一个包的所有分片都有相同的 Identification 字段，从而可以确认哪些分片属于同一个包。

3．分析 TCP 通信过程

由于实际网络中 TCP 的通信过程都比较复杂，不方便观察，本实验利用了我在“面向对象程序设计实践（C++）”课程中的大作业“单词消除游戏”完成，“单词消除游戏”的服务端和客户端使用简单的 TCP 连接，可以简洁但完整地展示 TCP 的连接建立、数据传输和连接拆除过程。

由于客户端和服务端运行在同一台机器上，使用 127.0.0.1 连接，走的是本地环回，因此设置 Wireshark 监听本地环回网卡。

首先在本地启动服务端，监听端口 1764，然后启动客户端，连接后进行简单操作后退出客户端。设置 Wireshark 的显示过滤器位 tcp.port==1764，捕获到如下内容。注意在这里源地址和目的地址都是 127.0.0.1，可以通过端口号来判断是服务端发给客户端还是客户端发给服务端。

tcp.port==1764

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	10651 → 1764 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000060	127.0.0.1	127.0.0.1	TCP	56	1764 → 10651 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_…
3	0.000091	127.0.0.1	127.0.0.1	TCP	44	10651 → 1764 [ACK] Seq=1 Ack=1 Win=327424 Len=0
17	17.077173	127.0.0.1	127.0.0.1	TCP	55	10651 → 1764 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=11
18	17.077217	127.0.0.1	127.0.0.1	TCP	44	1764 → 10651 [ACK] Seq=1 Ack=12 Win=2161152 Len=0
19	17.077511	127.0.0.1	127.0.0.1	TCP	78	1764 → 10651 [PSH, ACK] Seq=1 Ack=12 Win=2161152 Len=34
20	17.077546	127.0.0.1	127.0.0.1	TCP	44	10651 → 1764 [ACK] Seq=12 Ack=35 Win=327424 Len=0
21	20.680835	127.0.0.1	127.0.0.1	TCP	44	10651 → 1764 [FIN, ACK] Seq=12 Ack=35 Win=327424 Len=0
22	20.680880	127.0.0.1	127.0.0.1	TCP	44	1764 → 10651 [ACK] Seq=35 Ack=13 Win=2161152 Len=0
23	20.681113	127.0.0.1	127.0.0.1	TCP	44	1764 → 10651 [FIN, ACK] Seq=35 Ack=13 Win=2161152 Len=0
24	20.681146	127.0.0.1	127.0.0.1	TCP	44	10651 → 1764 [ACK] Seq=13 Ack=36 Win=327424 Len=0

其中前三个段为“三次握手”，最后四个段为“四次挥手”，其余包为数据传输。

首先分析连接建立的过程。

连接由客户端发起，客户端发出第一个段，置 SYN 位，并给出了自己的初始 SEQ 编号为 4200338096。Window 字段给出了接收窗口的大小，但由于现在还没有协商好窗口大小的单位，这个字段的单位默认为 1 字节，因此只能填到 65535 字节了。TCP 选项中包含了希望与对方协商的内容，包括 MSS 大小、窗口大小单位、是否允许选择确认。

```
> Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on inter
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 10651, Dst Port: 1764, Seq: 0, Len
    Source Port: 10651
    Destination Port: 1764
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 4200338096
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x395e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  v Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window sc
      > TCP Option - Maximum segment size: 65495 bytes
      > TCP Option - No-Operation (NOP)
      > TCP Option - Window scale: 8 (multiply by 256)
      > TCP Option - No-Operation (NOP)
      > TCP Option - No-Operation (NOP)
      > TCP Option - SACK permitted
  > [Timestamps]
```

第二个段由服务器发出，即“第二次握手”，置 SYN 和 ACK 位。这个段给出了服务端的初始 SEQ 编号为 1130267958。同时其 ACK 字段为 4200338097，表示对客户端的起始 SEQ 编号 4200338096 的确认。和第一段相同，服务端告诉客户端自己的接收窗口大小为 65535 字节。这个段的 TCP 选项内容和第一个段相同，双方协商成功。

```
> Frame 2: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on inter
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 1764, Dst Port: 10651, Seq: 0, Ack
    Source Port: 1764
    Destination Port: 10651
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1130267958
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 4200338097
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x70b8 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  v Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window sca
      > TCP Option - Maximum segment size: 65495 bytes
      > TCP Option - No-Operation (NOP)
      > TCP Option - Window scale: 8 (multiply by 256)
      > TCP Option - No-Operation (NOP)
      > TCP Option - No-Operation (NOP)
      > TCP Option - SACK permitted
  > [Timestamps]
  > [SEQ/ACK analysis]
```

第三个段由客户端发出，即“第三次握手”，置 ACK 位。这个段的 SEQ 为此前的起始 SEQ 编号加 1，即为 4200338097，可见 SYN 本身被视作一个字节的内容。同时其 ACK 字段为 1130267959，表示对客户端的起始 SEQ 编号 1130267958 的确认。此外客户端也同时给出了其当前接收窗口的大小。从这个段开始窗口大小的单位已经协商完成了，因此现在客户端可以告诉服务器其真实的接受窗口大小。从这个段开始不再携带 TCP 选项信息。

```
> Frame 3: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on in
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 10651, Dst Port: 1764, Seq: 1,
    Source Port: 10651
    Destination Port: 1764
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 4200338097
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1130267959
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 1279
    [Calculated window size: 327424]
    [Window size scaling factor: 256]
    Checksum: 0xa6b0 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
```

下面简单分析数据传输过程。

第四段，客户端给服务端发送数据，内容长度为 11 字节，置 PSH 和 ACK 位。这一段的 SEQ 仍为 4200338097，因为第三次握手没有包含实际内容。这一段的 ACK 其实没有必要，因为客户端刚刚才给服务端确认过其起始 SEQ。而 PSH 位则表示对方应该尽快发送确认，不要等待。

```
> Frame 17: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on inte
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 10651, Dst Port: 1764, Seq: 1, Ack
    Source Port: 10651
    Destination Port: 1764
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 11]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 4200338097
    [Next Sequence Number: 12    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1130267959
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 1279
    [Calculated window size: 327424]
    [Window size scaling factor: 256]
    Checksum: 0x9ea5 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (11 bytes)
> Data (11 bytes)
```

第五段，服务端发送确认。由于客户端设置了 PSH 位，服务端没有等待捎带 ACK，而是立即发送了一个单独的 ACK。其 SEQ 编号为 1130267959，是服务端起始 SEQ 编号加 1。其 ACK 编号为 4200338108，表示刚才的 11 字节数据全部确认收到。

```
> Frame 18: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on inte
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 1764, Dst Port: 10651, Seq: 1, Ack
    Source Port: 1764
    Destination Port: 10651
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1130267959
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 12    (relative ack number)
    Acknowledgment number (raw): 4200338108
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]
    Checksum: 0x8aaa [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
```

第六、七段与第四、五段类似，只不过变成了服务端发数据，客户端确认。

最后分析拆除连接的过程。

连接的拆除同样由客户端发起。客户端发出“第一次挥手”，置 FIN 和 ACK 位。由于此前已经确认过，这里的 ACK 同样是没有必要的。FIN 位指示了客户端关闭连接。

```
> Frame 21: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on inte
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 10651, Dst Port: 1764, Seq: 12, Ac
    Source Port: 10651
    Destination Port: 1764
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 12    (relative sequence number)
    Sequence Number (raw): 4200338108
    [Next Sequence Number: 13    (relative sequence number)]
    Acknowledgment Number: 35    (relative ack number)
    Acknowledgment number (raw): 1130267993
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x011 (FIN, ACK)
    Window: 1279
    [Calculated window size: 327424]
    [Window size scaling factor: 256]
    Checksum: 0xa682 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

服务端发出“第二次挥手”，置 ACK 位。这一段的 ACK 为 4200338109，是“第一次挥手”的 SEQ 加 1，表示对客户端关闭连接的确认，由此可见 FIN 也被视作一个字节的内容。

```
> Frame 22: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on inte
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 1764, Dst Port: 10651, Seq: 35, Ac
    Source Port: 1764
    Destination Port: 10651
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 35    (relative sequence number)
    Sequence Number (raw): 1130267993
    [Next Sequence Number: 35    (relative sequence number)]
    Acknowledgment Number: 13    (relative ack number)
    Acknowledgment number (raw): 4200338109
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]
    Checksum: 0x8a87 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
```

至此，客户端到服务端这一方向上的连接已经被关闭，客户端进入 FIN WAIT 2 状态，服务端进入 CLOSE WAIT 状态。接下来服务端也要关闭服务端到客户端这一方向的连接。“第三次挥手”和“第四次挥手”与前两次类似，只是变成了从服务端发起，客户端确认。全部完成后，服务端关闭 TCP 连接，而客户端进入 TIMED WAIT 状态，等待一段时间后关闭 TCP 连接。

四、心得体会

本次实验难度不大，整体进展顺利。在 DHCP 协议方面我进行了比较多的资料搜寻，查阅了 RFC 文档，做了较多课外的拓展。对 TCP 协议的分析让我对 TCP 连接建立和关闭过程中的细节有了更深入的理解。