

北京邮电大学

计算机学院（国家示范性软件学院）

## 计算机网络实验二

课程名称：\_\_\_\_\_计算机网络\_\_\_\_\_

项目名称：\_\_\_\_\_IP 和 TCP 数据分组的捕获和解析\_\_\_\_\_

项目成员：\_\_\_\_\_陈韵涵\_\_\_\_\_

时间：2024 年 6 月 5 日

# 目录

一、实验内容及环境描述 .....	1
1.1 实验内容和目的 .....	1
1.2 实验环境描述 .....	1
二、DHCP .....	1
2.1 前置知识 .....	1
2.2 捕获 DHCP 报文 .....	2
2.3 RELEASE: 释放 IP .....	3
2.3.1 DHCP 报文 .....	4
2.3.2 传输层头部 .....	7
2.3.3 网络层头部 .....	8
2.3.4 链路层头部 .....	9
2.4 RENEW: 获取 IP .....	9
2.4.1 DHCP Discover .....	10
2.4.2 DHCP Offer .....	11
2.4.3 DHCP Request .....	12
2.4.4 DHCP ACK .....	13
2.4.5 DHCP 四次握手全过程 .....	13
2.5 观察 PING 命令 .....	14
2.5.1 ARP Request .....	14
2.5.2 ARP Reply .....	16
2.5.3 ICMP .....	16
三、分析数据分组的分片传输过程 .....	17
3.1 使用 PING 命令并抓包 .....	17
3.2 分析分段 .....	18
3.2.1 分片字段分析 .....	18
3.2.2 分片字段对比 .....	18
四、分析 TCP 通信过程 .....	19
4.1 建立连接并拆除连接的抓包过程 .....	19
4.2 三次握手 .....	19
4.2.1 第一次握手 .....	20
4.2.2 第二次握手 .....	21
4.2.3 第三次握手 .....	22
4.3 数据通信 .....	23
4.4 四次挥手 .....	24
4.4.1 第一次挥手: 客户端>服务器端 .....	24
4.4.2 第二次挥手: 服务器端>客户端 .....	25
4.4.3 第三次挥手: 服务器端>客户端 .....	26
4.4.4 第四次挥手: 客户端>服务器端 .....	26
4.4.5 整个过程示意图 .....	27

# 一、实验内容及环境描述

## 1.1 实验内容和目的

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。

## 1.2 实验环境描述

1. 操作系统：Windows 11
2. 选用网卡：MediaTek Wi-Fi 6 MT7921 Wireless
3. 使用 wireshark 4.2.5

# 二、DHCP

## 2.1 前置知识

### 1. DHCP

DHCP（Dynamic Host Configuration Protocol）是一种网络协议，其作用是自动分配 IP 地址和其他网络配置参数给计算机或设备，以便它们可以在网络上进行通信。DHCP 协议工作在网络协议栈中的应用层和传输层之间，通常使用 UDP 协议进行传输。因此，DHCP 分组是在传输层进行封装和传输的。

总结起来，DHCP 分组的作用是在网络中自动分配 IP 地址和其他配置参数，以简化网络设备的配置过程。

DHCP 的报文结构：

链路层头	IP头 20bytes	UDP头	DHCP报文
------	-------------	------	--------

### 2. ARP（地址解析协议）

#### 1. 功能：

- ARP 的主要功能是将 IP 地址转换为物理地址（MAC 地址）。在一个局域网

中，设备通信需要知道对方的 MAC 地址，但设备通常只知道对方的 IP 地址，这时就需要 ARP 来进行地址解析。

## 2. 工作流程：

- 主机发送一个 ARP 请求广播，询问网络上哪个设备拥有某个特定的 IP 地址。
- 具有该 IP 地址的设备回复其 MAC 地址。
- 请求设备将 IP 地址和 MAC 地址的映射关系缓存起来，以便下次使用。

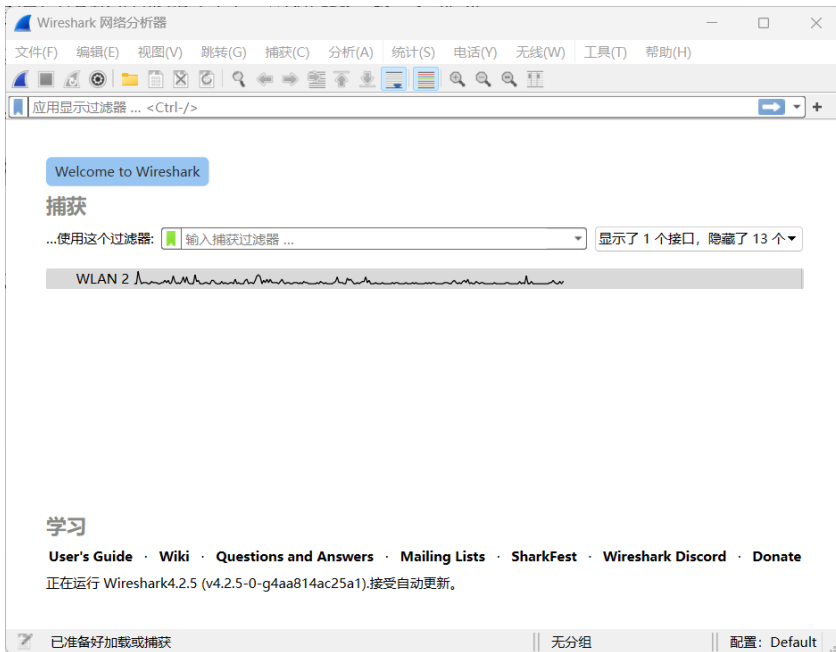
## 2.2 捕获DHCP报文

开启 Wireshark 监控，设置捕获过滤器，仅捕获 UDP 报文

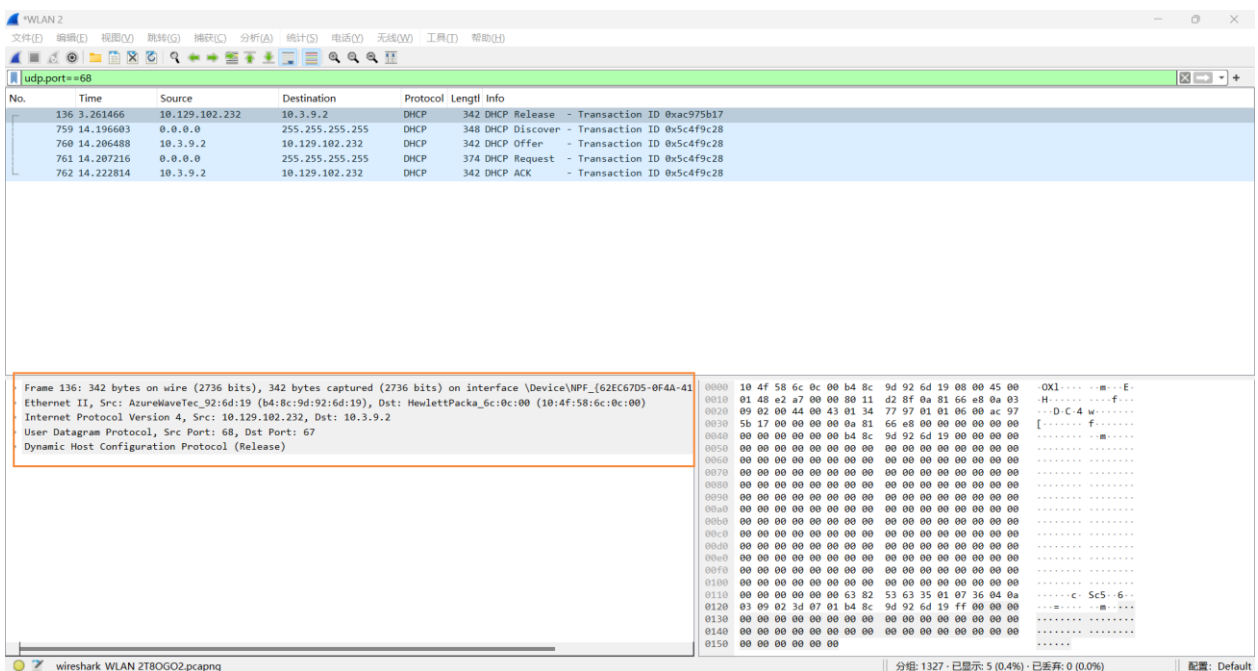
所用网卡：

WLAN 2 属性		
IP 分配:	自动(DHCP)	编辑
DNS 服务器分配:	自动(DHCP)	编辑
SSID:	BUPT-portal	复制
协议:	Wi-Fi 6 (802.11ax)	
安全类型:	开放	
制造商:	MediaTek, Inc.	
描述:	MediaTek Wi-Fi 6 MT7921 Wireless LAN Card	
驱动程序版本:	3.0.1.1294	
网络频带:	5 GHz	
网络通道:	149	
链接速度(接收/传输):	573/573 (Mbps)	
IPv6 地址:	2001:da8:215:3c0a:761:b5c5:fba6:f6c4	
本地链接 IPv6 地址:	fe80::1eed:6ae3:93ef:f5d8%13	
IPv4 地址:	10.129.102.232	
IPv4 DNS 服务器:	223.5.5.5 (未加密) 223.6.6.6 (未加密)	
物理地址(MAC):	B4-8C-9D-92-6D-19	

选择这个接口进行捕获：



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：



可以看到 wireshark 已经将包里面的数据分层显示

## 2.3 release: 释放IP

分析这一步捕获到的 DHCP Release 包：

Wireshark · 分组 136 · WLAN 2		
0000	10 4f 58 6c 0c 00 b4 8c 9d 92 6d 19 08 00 45 00	.OX1... ..m...E.
0010	01 48 e2 a7 00 00 80 11 d2 8f 0a 81 66 e8 0a 03	.H... ..f...
0020	09 02 00 44 00 43 01 34 77 97 01 01 06 00 ac 97	...D.C.4 w.....
0030	5b 17 00 00 00 00 0a 81 66 e8 00 00 00 00 00 00	[..... f.....
0040	00 00 00 00 00 00 b4 8c 9d 92 6d 19 00 00 00 00	..... .m.....
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0110	00 00 00 00 00 00 63 82 53 63 35 01 07 36 04 0a	.....c Sc5..6..
0120	03 09 02 3d 07 01 b4 8c 9d 92 6d 19 ff 00 00 00	...=.....m.....
0130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

2.3.1 DHCP 报文

Wireshark · 分组 136 · WLAN 2																																																																				
> Frame 136: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits) on interface \Device\NPF_{62EC67D5-0F4A}																																																																				
> Ethernet II, Src: AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19), Dst: HewlettPacka_6c:0c:00 (10:4f:58:6c:0c:00)																																																																				
> Internet Protocol Version 4, Src: 10.129.102.232, Dst: 10.3.9.2																																																																				
> User Datagram Protocol, Src Port: 68, Dst Port: 67																																																																				
> Dynamic Host Configuration Protocol (Release)																																																																				
<table> <tr><td>0000</td><td>10 4f 58 6c 0c 00 b4 8c 9d 92 6d 19 08 00 45 00</td><td>.OX1... ..m...E.</td></tr> <tr><td>0010</td><td>01 48 e2 a7 00 00 80 11 d2 8f 0a 81 66 e8 0a 03</td><td>.H... ..f...</td></tr> <tr><td>0020</td><td>09 02 00 44 00 43 01 34 77 97 01 01 06 00 ac 97</td><td>...D.C.4 w.....</td></tr> <tr><td>0030</td><td>5b 17 00 00 00 00 0a 81 66 e8 00 00 00 00 00 00</td><td>[..... f.....</td></tr> <tr><td>0040</td><td>00 00 00 00 00 00 b4 8c 9d 92 6d 19 00 00 00 00</td><td>..... .m.....</td></tr> <tr><td>0050</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0060</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0070</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0080</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0090</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>00a0</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>00b0</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>00c0</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>00d0</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>00e0</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>00f0</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0100</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0110</td><td>00 00 00 00 00 00 63 82 53 63 35 01 07 36 04 0a</td><td>.....c Sc5..6..</td></tr> <tr><td>0120</td><td>03 09 02 3d 07 01 b4 8c 9d 92 6d 19 ff 00 00 00</td><td>...=.....m.....</td></tr> <tr><td>0130</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0140</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> <tr><td>0150</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td><td>.....</td></tr> </table>			0000	10 4f 58 6c 0c 00 b4 8c 9d 92 6d 19 08 00 45 00	.OX1... ..m...E.	0010	01 48 e2 a7 00 00 80 11 d2 8f 0a 81 66 e8 0a 03	.H... ..f...	0020	09 02 00 44 00 43 01 34 77 97 01 01 06 00 ac 97	...D.C.4 w.....	0030	5b 17 00 00 00 00 0a 81 66 e8 00 00 00 00 00 00	[..... f.....	0040	00 00 00 00 00 00 b4 8c 9d 92 6d 19 00 00 00 00	..... .m.....	0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	00b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	00c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	00d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	00e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	00f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0110	00 00 00 00 00 00 63 82 53 63 35 01 07 36 04 0a	.....c Sc5..6..	0120	03 09 02 3d 07 01 b4 8c 9d 92 6d 19 ff 00 00 00	...=.....m.....	0130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0000	10 4f 58 6c 0c 00 b4 8c 9d 92 6d 19 08 00 45 00	.OX1... ..m...E.																																																																		
0010	01 48 e2 a7 00 00 80 11 d2 8f 0a 81 66 e8 0a 03	.H... ..f...																																																																		
0020	09 02 00 44 00 43 01 34 77 97 01 01 06 00 ac 97	...D.C.4 w.....																																																																		
0030	5b 17 00 00 00 00 0a 81 66 e8 00 00 00 00 00 00	[..... f.....																																																																		
0040	00 00 00 00 00 00 b4 8c 9d 92 6d 19 00 00 00 00	..... .m.....																																																																		
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
00b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
00c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
00d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
00e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
00f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0110	00 00 00 00 00 00 63 82 53 63 35 01 07 36 04 0a	.....c Sc5..6..																																																																		
0120	03 09 02 3d 07 01 b4 8c 9d 92 6d 19 ff 00 00 00	...=.....m.....																																																																		
0130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
0150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....																																																																		
No.: 136 · Time: 3.261466 · Source: 10.129.102.232 · Destination: 10.3.9.2 · Protocol: DHCP · Length: 342 · Info: DHCP Release · Transaction ID 0xac975b17																																																																				
<input checked="" type="checkbox"/> Show packet bytes																																																																				
<div> <div>关闭</div> <div>帮助</div> </div>																																																																				

分析含义：

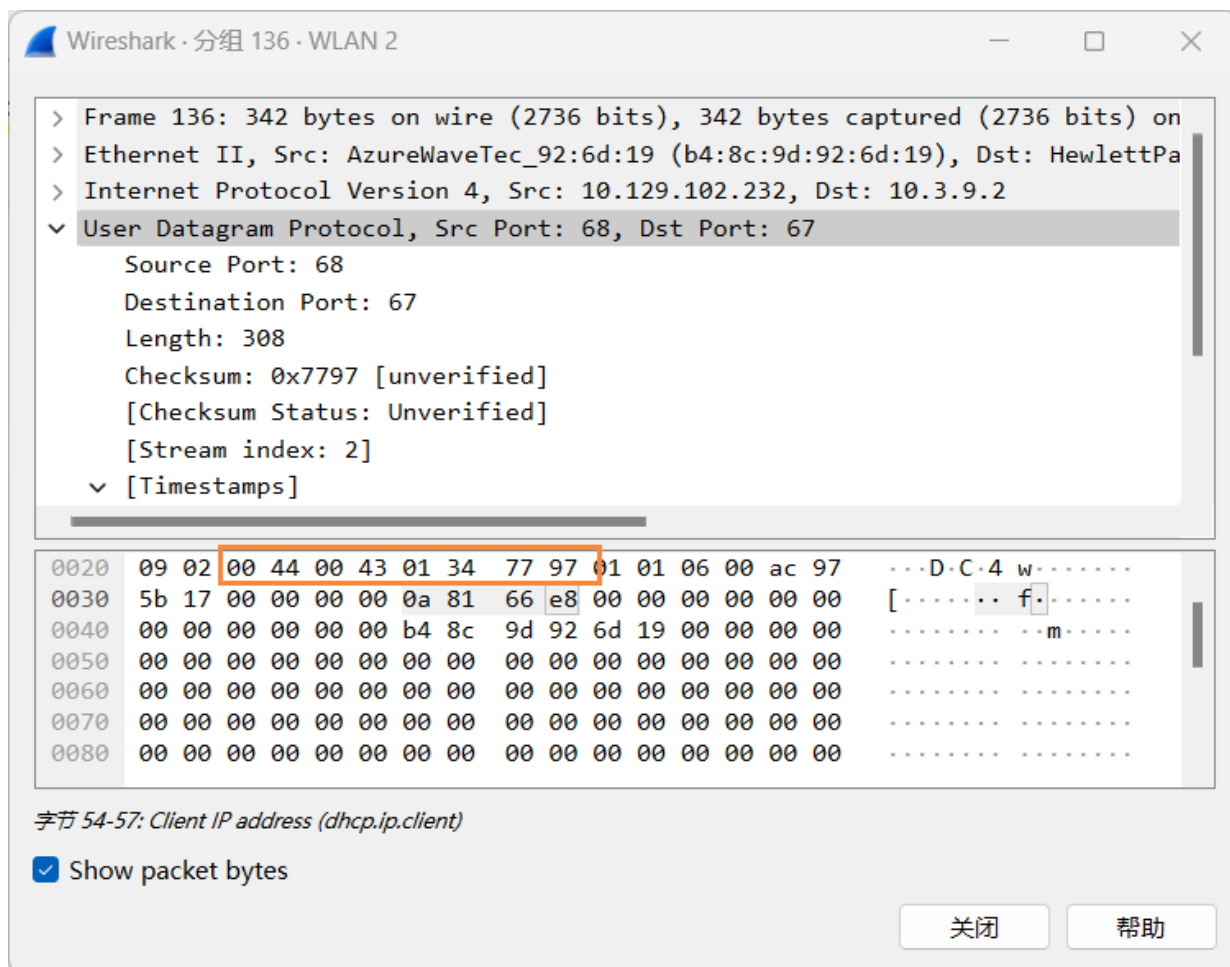
16 进制值	字段类型	含义
01	OP：操作类型	01：发送给服务器的操作报文； 若为 02，则是服务器发送的应答报文



00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	DHCP 客户端指定的启动 配置文件名称及路径信息	（不一 定有）
.....	Options: 可选项字段	长度可变，格式为"代码+长度+ 数据"



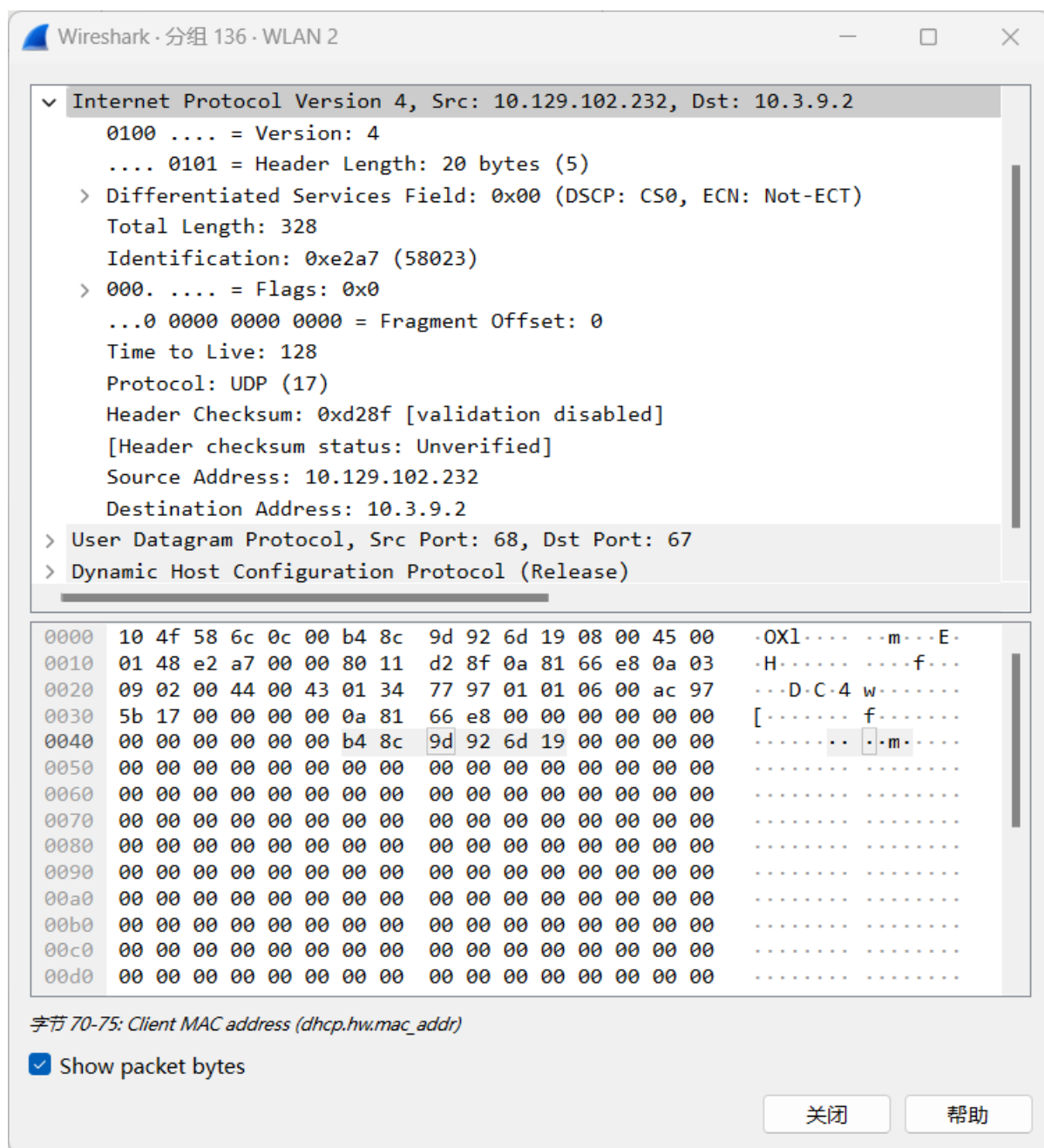
### 2.3.2 传输层头部



协议类型：UDP

16 进制值	字段类型	含义
00 44	源端口	端口 68
00 43	目的端口	端口 67
01 34	UDP 包长度	包长度为 308
77 97	校验和	0x7797

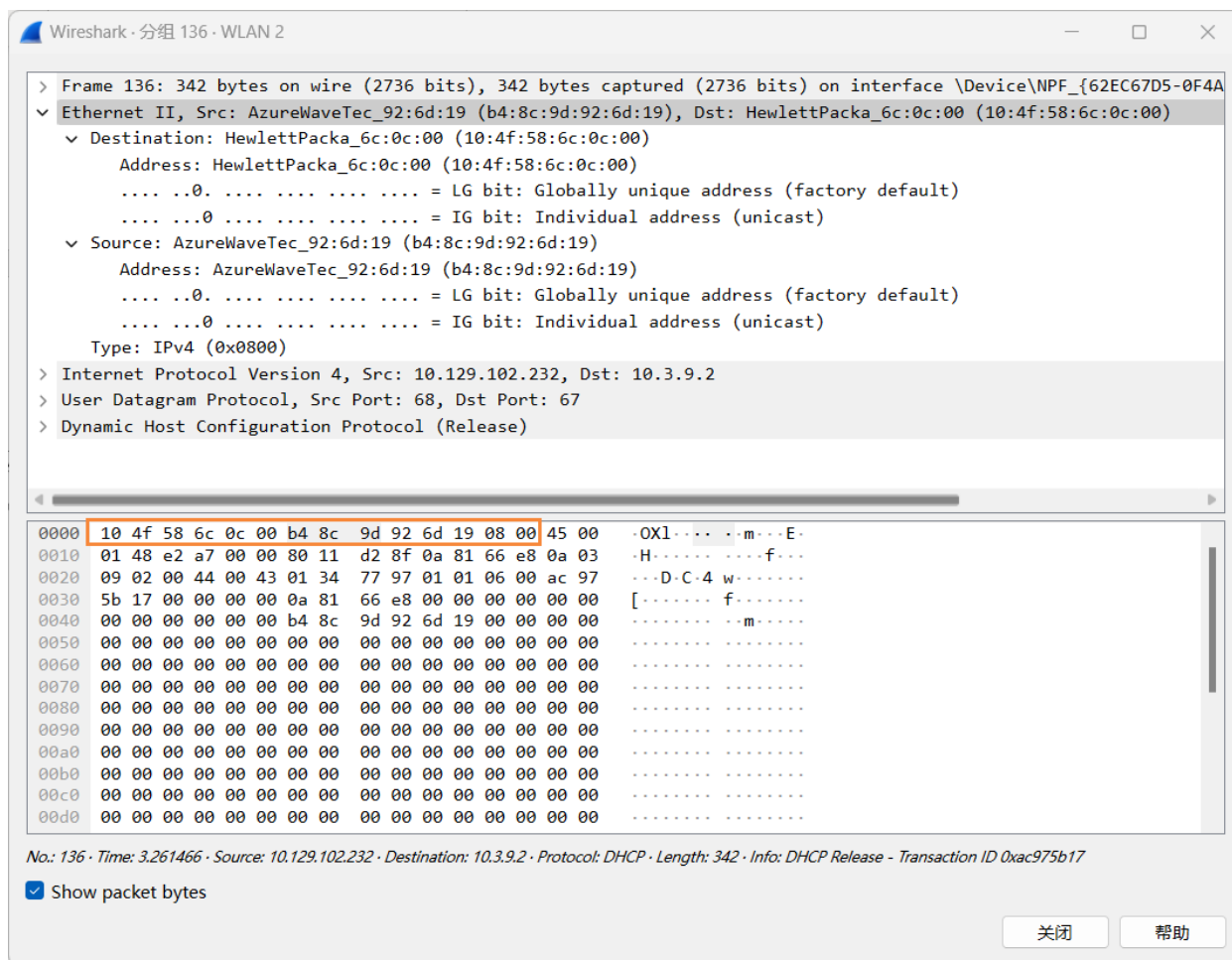
### 2.3.3 网络层头部



协议类型：IPv4

16 进制值	字段类型	含义
0a 81 66 e8	源 IP	10.129.102.232
0a 03 09 02	目的 IP	10.3.9.2
80	TTL	数据包在网络中最大跳数为 80

## 2.3.4 链路层头部



协议类型：Ethernet II

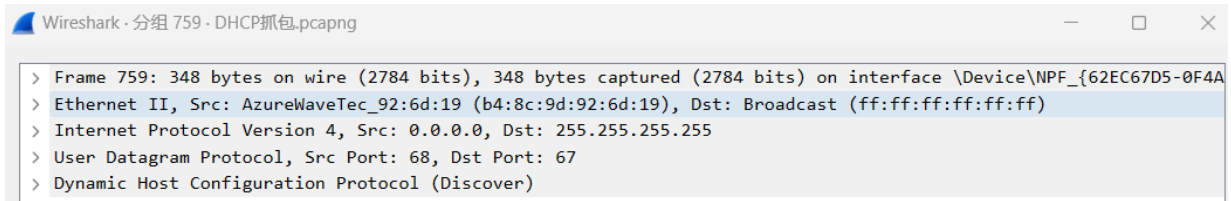
16 进制值	字段类型	含义
10 4f 58 6c 0c 00	目的 MAC 地址	48 位 MAC 地址：10:4f:58:6c:0c:00
B4 8c 9d 92 6d 19	源 MAC 地址	48 位 MAC 地址：b4:8c:9d:92:6d:19
08 00	协议类型	上层使用 IPv4

## 2.4 renew：获取IP

使用 DHCP 获取 IP 地址是通过 Discover、Offer、Request、ACK 四个报文完成的。下面通过对 DHCP 包进行具体的分析，来探究 DHCP 通过四次握手获得 IP 地址，缺省路由 DNS 等参数的过程。各个字段的含义已经在前面分析过，这里只分析有意义的字段。

759	14.196603	0.0.0.0	255.255.255.255	DHCP	348 DHCP Discover	- Transaction ID 0x5c4f9c28
760	14.206488	10.3.9.2	10.129.102.232	DHCP	342 DHCP Offer	- Transaction ID 0x5c4f9c28
761	14.207216	0.0.0.0	255.255.255.255	DHCP	374 DHCP Request	- Transaction ID 0x5c4f9c28
762	14.222814	10.3.9.2	10.129.102.232	DHCP	342 DHCP ACK	- Transaction ID 0x5c4f9c28

### 2.4.1 DHCP DISCOVER



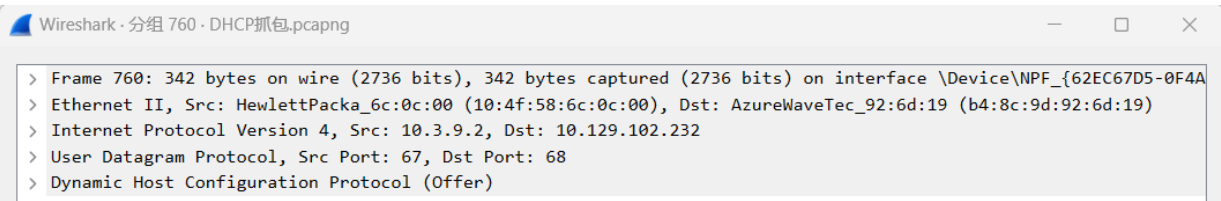
字段名称	值	含义
<b>eth.dst 目的 MAC</b>	ff:ff:ff:ff:ff:ff	Destination: Broadcast (ff:ff:ff:ff:ff:ff) 此时我的电脑是没有获取 IP 地址的，也不知道该向谁请求 IP 分配，因此设置以太网的目 的为全 f，即广播地址
<b>ip.src 源 IP</b>	0.0.0.0	原因同上，这整个流程中，客户端发出的 DHCP 包的 source IP 都是 0.0.0.0
<b>Source port</b>	68	基于 UDP 的源端口号 68 来发送 DHCP Discover 发现信息来寻找 DHCP 服务器
<b>Destination port</b>	67	基于目的端口号 67 来发送 DHCP Discover 发现信息来寻找 DHCP 服务器
<b>Bootp Flag</b>	0x0000	DHCP 字段中，设置发送方式为广播
<b>dhcp.hw.mac_addr</b>	b4:8c:9d:92:6d:19	向服务器说明自己的 MAC 地址
<b>Option</b>	50	Option: (50) Requested IP Address (10.129.102.232). 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
<b>Option</b>	55	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

结论：

DHCP Discover（广播）是第一个阶段即 DHCP 客户端寻找 DHCP 服务端的阶段。

由于 DHCP 服务端的 IP 地址等信息对于 DHCP 客户端来说是未知，此时就需要使用广播的方式进行发送消息，基于 UDP 的源端口号 68，目的端口号 67 来发送 DHCP Discover

2.4.2 DHCP OFFER

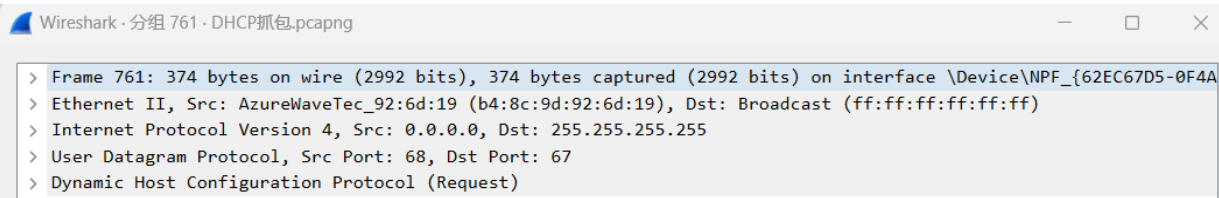


字段名称	值	含义
eth.dst 目的 MAC	b4:8c:9d:92:6d:19	以太网字段中，目的 MAC 地址是之前 Discover 中填写的本机 MAC
dhcp.ip.your	10.129.102.232	Your (client) IP address: 10.129.102.232。表示服务端提供的 IP 地址
Option	54	DHCP Server Identifier 表示 DHCP 服务器的信息，在这里就是 DHCP 服务器的 IP 地址。这个字段将被用于区分多个服务器
Option	51	IP Address Lease Time 表示租约时长，展开后可看到时长为两小时
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）。

结论：  
这个阶段是 DHCP 服务器向 DHCP 客户端提供预分配 IP 地址的阶段。网络中的所有 DHCP 服务器接收到客户端的 DHCP Discover 报文后都会根据自己的 DHCP 地址池中 IP 地址分配的优先次序选出一个 IP 地址，然后与其他参数一起通过传输层的 UDP67 号端口在 DHCP Offer 报文中以单播/广播方式发送给客户端的 UDP68 号端口 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。客户端通过封装在帧中的目的 MAC 地址（也就是 DHCP Discover 报文中的 CHADDR 字段值）的比对来确定是否接收该帧。但这样以来，理论上 DHCP 客户端可能会收到多个 DHCP Offer 报文（当网络中存在多个 DHCP 服务器时），但 DHCP 客户端只接收第一个到来的 DHCP Offer 报文。

2.4.3 DHCP REQUEST



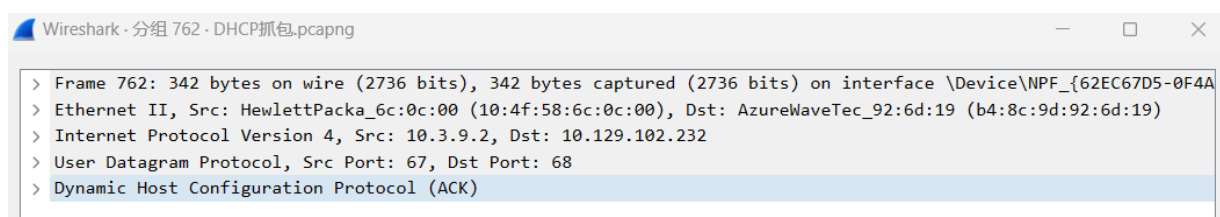
字段名称	值	含义
ip.dst 目的 IP	255.255.255.255	采用广播的形式，将这个 DHCP 包广播给所有 DHCP 服务器。由于这个报文需要告知可能的多个服务器说自己选择了哪个 ip，因此目的地全部是广播
ip.src 源 IP	0.0.0.0	Source Address: 0.0.0.0 说明此时并没有真正被分配 IP
Option	50	Option: (50) Requested IP Address (10.129.102.232). 指示客户端要使用的 IP 地址
Option	54	Option: (54) DHCP Server Identifier (10.3.9.2). 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
Option	12	Option: (12) Host Name. 告知自己的主机名

结论：  
选择阶段，即 DHCP 客户端选择 IP 地址的阶段。  
如果有多台 DHCP 服务器向该客户端发来 DHCP OFFER 报文，客户端只接收第一个收

到的 DHCP Offer 报文，然后以广播方式发送 DHCP Request 报文。在该报文的 Requested Address 选项中包含 DHCP 服务器在 DHCP Offer 报文中预分配的 IP 地址、对应的 DHCP 服务器 IP 地址等。这样也就相当于同时告诉其他 DHCP 服务器，它们可以释放已提供的地址并将这些地址返回到可用的地址池中。

在 DHCP REQUEST 报文封装的 IP 协议头部中，客户端的 Source Address 仍然是 0.0.0.0，数据包的 Destination 仍然是 255.255.255.255。但在 DHCP Request 报文中 Ciaddr、Yiaddr、Siaddr、Giaddr 字段的地址均为 0.0.0.0

#### 2.4.4 DHCP ACK



字段名称	值	含义
dhcp.ip.your	10.129.102.232	最后一次确认，告知客户端它的 ip
ip.dst	10.129.102.232	此时 DHCP 服务器发送的包中的目的 IP 已经是分配的这个 IP 了

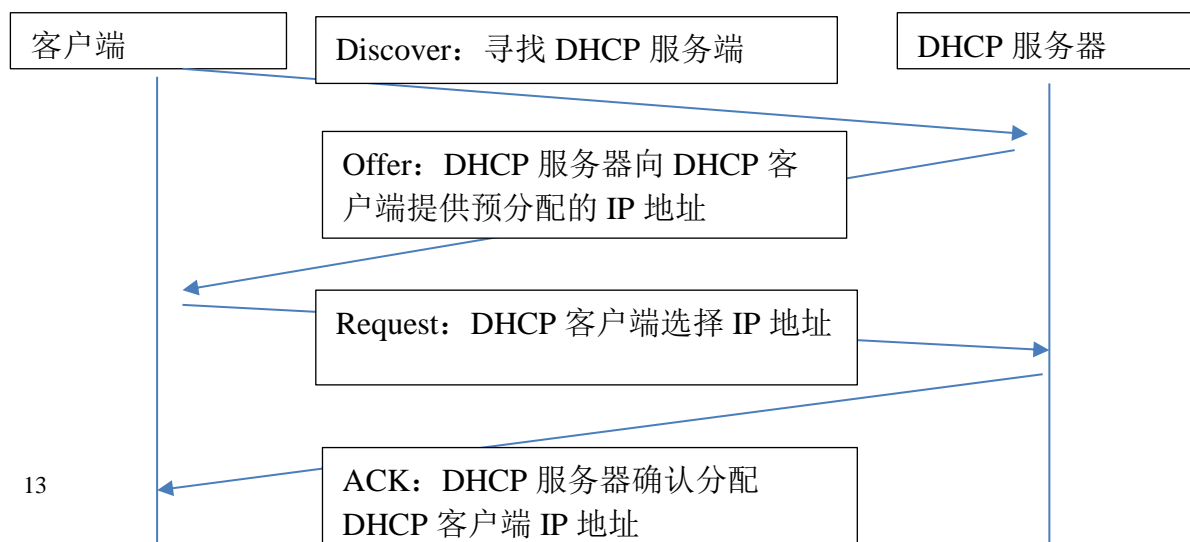
结论：

确认阶段，即 DHCP 服务器确认分配级 DHCP 客户端 IP 地址的阶段。

某个 DHCP 服务器在收到 DHCP 客户端发来的 DHCP REQUEST 报文后，只有 DHCP 客户端选择的服务器会进行如下操作：如果确认将地址分配给该客户端，则以单播/广播方式返回 DHCP ACK 报文；否则返回 DHCP NAK 报文，表明地址不能分配给该客户端。这里抓到的是 ACK 报文。

#### 2.4.5 DHCP 四次握手全过程

结合以上，可得出 DHCP 四次握手的全过程：





## 2.5 观察 ping 命令

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执行 ping，这里直接选择 ping 我的手机 10.129.81.76，因为手机没有防火墙。

无线局域网 BUCT-portal

IPv4 地址

配置 IP 自动 >

IP 地址 10.129.81.76

子网掩码 255.255.0.0

路由器 10.129.0.1

```
C:\Users\cyh>ping 10.129.81.76

正在 Ping 10.129.81.76 具有 32 字节的数据:
来自 10.129.81.76 的回复: 字节=32 时间=13ms TTL=64
来自 10.129.81.76 的回复: 字节=32 时间=80ms TTL=64
来自 10.129.81.76 的回复: 字节=32 时间=99ms TTL=64
来自 10.129.81.76 的回复: 字节=32 时间=14ms TTL=64

10.129.81.76 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 = 13ms, 最长 = 99ms, 平均 = 51ms
```

该设备此前不在本机的 ARP 表中，因此 Wireshark 抓包结果：

No.	Time	Source	Destination	Protocol	Length	Info
1037	1.567658	AzureWaveTec_92:6d:19	Broadcast	ARP	42	Who has 10.129.81.76? Tell 10.129.102.232
1042	1.575405	32:be:96:dc:29:c8	AzureWaveTec_92:6d:19	ARP	42	10.129.81.76 is at 32:be:96:dc:29:c8
1043	1.575415	10.129.102.232	10.129.81.76	ICMP	74	Echo (ping) request id=0x0001, seq=401/37121, ttl=128 (reply in 1049)
1049	1.580936	10.129.81.76	10.129.102.232	ICMP	74	Echo (ping) reply id=0x0001, seq=401/37121, ttl=64 (request in 1043)
1671	2.581482	10.129.102.232	10.129.81.76	ICMP	74	Echo (ping) request id=0x0001, seq=402/37377, ttl=128 (reply in 1715)
1715	2.661961	10.129.81.76	10.129.102.232	ICMP	74	Echo (ping) reply id=0x0001, seq=402/37377, ttl=64 (request in 1671)
2280	3.586887	10.129.102.232	10.129.81.76	ICMP	74	Echo (ping) request id=0x0001, seq=403/37633, ttl=128 (reply in 2347)
2347	3.686562	10.129.81.76	10.129.102.232	ICMP	74	Echo (ping) reply id=0x0001, seq=403/37633, ttl=64 (request in 2280)
2932	4.591579	10.129.102.232	10.129.81.76	ICMP	74	Echo (ping) request id=0x0001, seq=404/37889, ttl=128 (reply in 2942)
2942	4.605605	10.129.81.76	10.129.102.232	ICMP	74	Echo (ping) reply id=0x0001, seq=404/37889, ttl=64 (request in 2932)

### 2.5.1 ARP REQUEST

Wireshark · 分组 1037 · WLAN 2	
> Frame 1037: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{62EC67D5-0F4A-41...}	
Ethernet II, Src: AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19), Dst: Broadcast (ff:ff:ff:ff:ff:ff)	
> Destination: Broadcast (ff:ff:ff:ff:ff:ff)	
> Source: AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19)	
Type: ARP (0x0806)	
Address Resolution Protocol (request)	
Hardware type: Ethernet (1)	
Protocol type: IPv4 (0x0800)	
Hardware size: 6	
Protocol size: 4	
Opcode: request (1)	
Sender MAC address: AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19)	
Sender IP address: 10.129.102.232	
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)	
Target IP address: 10.129.81.76	

对有意义的字段进行解析：

字段名称	值	含义
arp	Address Resolution Protocol (request)	表示这是一个 ARP 请求包
arp.src.hw_mac	b4:8c:9d:92:6d:19	Sender MAC address:



		AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19)表示发送方 MAC 地址，也就是本机的 MAC 地址
arp.src.proto_ipv4	10.129.102.232	Sender IP address: 10.129.102.232, 表示发送方的 ip 地址，也就是本机的 ip 地址
arp.dst.hw_mac	00:00:00:00:00:00	Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)表示目标 MAC 地址，请求时目标的 MAC 地址还不知道，因此用全 0 占位。
arp.dst.proto_ipv4	10.129.81.76	表示目标 IP 地址，这个 ARP 包正是想要查询这个 IP 地址对应的主机的 MAC 地址
eth.dst	ff:ff:ff:ff:ff:ff	这是链路层的头部字段，这里的目标地址为广播地址，是因为主机 PC2 不知道 PC1 主机的 MAC 地址。这样，局域网中所有设备都会收到该数据包

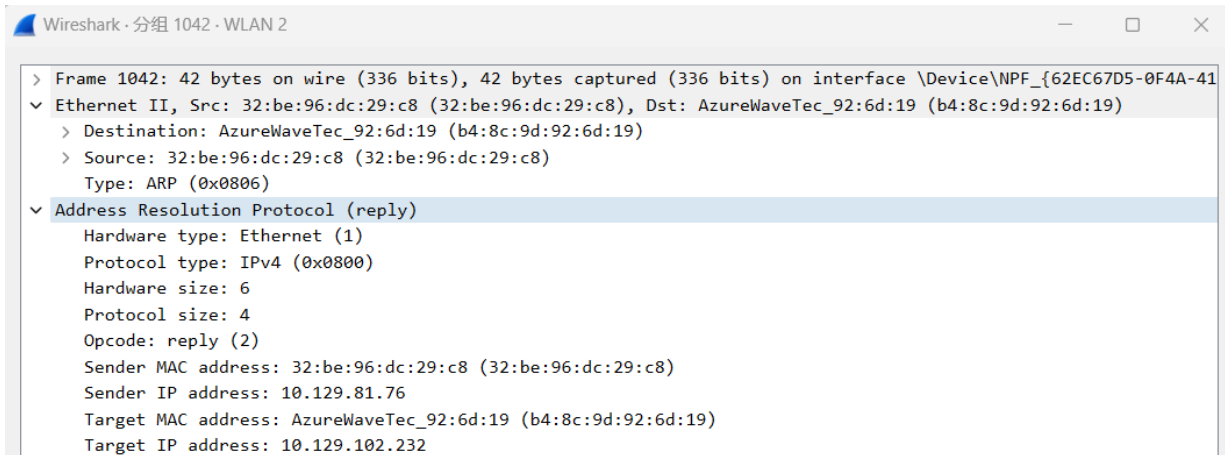
结论：

ARP 请求包(ARP Request)的作用是用于获取局域网内某 IP 对应的 MAC 地址

当主机 A 需要与主机 B 进行通讯时，它会先查一下本机的 ARP 缓存中，有没有主机 B 的 MAC 地址。如果有就可以直接通讯。如果没有，主机 A 就需要通过 ARP 协议来获取主机 B 的 MAC 地址，具体做法相当于主机 A 向局域网内所有主机喊一嗓子：“喂～谁是 10.129.81.76？我是 10.129.102.232，我的 MAC 地址是 b4:8c:9d:92:6d:19，你的 MAC 地址是什么，快告诉我”，这时候主机 A 发的数据包类型为：广播-请求  
这个 info 也很形象：

Info
Who has 10.129.81.76? Tell 10.129.102.232

## 2.5.2 ARP REPLY



对有意义的字段进行分析：

字段名称	值	含义
arp	Address Resolution Protocol (reply)	表示这是一个 ARP reply 包，里面有要查询的那个 MAC 地址
arp.src.hw_mac	32:be:96:dc:29:c8	Sender MAC address: 32:be:96:dc:29:c8 (32:be:96:dc:29:c8)，发送方 MAC 地址，也就是想要查询的那个 MAC 地址

结论：

ARP 回复包(ARP Reply)的作用是告知别的主机，本机的 IP 地址和 MAC 是什么。当主机 B 接收到来自主机 A 的“ARP 广播-请求”数据包后，它会先把主机 A 的 IP 地址和 MAC 地址对应关系保存/更新到本机的 ARP 缓存表中，然后它会给主机 A 发送一个“ARP 非广播-回复”数据包，其作用相当于告诉主机 A：“嘿，我是 10.129.81.76，我的 MAC 地址是 32:be:96:dc:29:c8”。当主机 A 接收到主机 B 的回复后，它会把主机 B 的 IP 地址和 MAC 地址对应关系保存/更新到本机的 ARP 缓存表中，之后主机 A 和 B 就可以进行通讯了。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。

## 2.5.3 ICMP

1043	1.575415	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=401/37121, ttl=128 (reply in 1049)
1049	1.580936	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=401/37121, ttl=64 (request in 1043)
1671	2.581482	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=402/37377, ttl=128 (reply in 1715)
1715	2.661961	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=402/37377, ttl=64 (request in 1671)
2280	3.586887	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=403/37633, ttl=128 (reply in 2347)
2347	3.686562	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=403/37633, ttl=64 (request in 2280)
2932	4.591579	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=404/37889, ttl=128 (reply in 2942)
2942	4.605605	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=404/37889, ttl=64 (request in 2932)

>	0000	32	be	96	dc	29	c8	b4	8c	9d	92	6d	19	08	00	45	00	2...	...	m...	E.
>	0010	00	3c	bb	2b	00	00	80	01	b2	5f	0a	81	66	e8	0a	81	<+...	...	f...	
>	0020	51	4c	08	00	4b	ca	00	01	01	91	61	62	63	64	65	66	QL..K...	...	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							wabdefgh	hi		

1049	1.580936	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=401/37121, ttl=64 (request in 1043)
1671	2.581482	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=402/37377, ttl=128 (reply in 1715)
1715	2.661961	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=402/37377, ttl=64 (request in 1671)
2280	3.586887	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=403/37633, ttl=128 (reply in 2347)
2347	3.686562	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=403/37633, ttl=64 (request in 2280)
2932	4.591579	10.129.102.232	10.129.81.76	ICMP	74 Echo (ping) request	id=0x0001, seq=404/37889, ttl=128 (reply in 2942)
2942	4.605605	10.129.81.76	10.129.102.232	ICMP	74 Echo (ping) reply	id=0x0001, seq=404/37889, ttl=64 (request in 2932)

>	0000	b4 8c 9d 92 6d 19 32 be 96 dc 29 c8 08 00 45 00	....m 2...).E.
>	0010	00 3c ad 1d 00 00 40 01 00 6e 0a 81 51 4c 0a 81	<...@...nQL..
>	0020	66 e8 00 00 53 ca 00 01 01 91 61 62 63 64 65 66	f...S...-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

结论：

IMCP 包也分为 reply 和 request 两种，这是两个类型为 8（Echo）的 ICMP 报文，携带的数据长度为 32 字节。里面的数据是随机的，只是用来表示两台主机可以进行通信。

并且发现：reply 和 request 里面的报文内容必须完全相同

ping 命令的工作原理就是基于 ICMP 协议，通过发送 ICMP 回显请求（Echo Request）并接收 ICMP 回显应答（Echo Reply）来测试网络连通性和测量网络性能。

## 三、分析数据分组的分片传输过程

### 3.1 使用ping命令并抓包

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。使用 Windows 中 ping 命令的 -l 选项，例如：ping -l 8000 [www.bupt.edu.cn](http://www.bupt.edu.cn)

【-lPreload】：在进入正常行为模式(每秒 1 个)前尽快发送 Preload 变量指定数量的信息包。-l 标志是小写的 L

执行命令后，wireshark 抓包内容如下：

The screenshot shows a Wireshark capture of a ping command using the -l 8000 option. The capture shows several fragmented IP packets (protocol 1514) and ICMP Echo (ping) request and reply packets (protocol 1). The packets are captured on the wlan2 interface. The ICMP Echo (ping) request packet (No. 27) has a length of 1514 bytes and is fragmented into 1514-byte fragments. The ICMP Echo (ping) reply packet (No. 33) also has a length of 1514 bytes and is fragmented into 1514-byte fragments. The capture shows the reassembly of these fragments into the original ICMP Echo (ping) request and reply packets.

一共 ping 了 4 次，每次包含一个 ICMP request 和一个 ICMP reply，因此一共抓到 8 个 ICMP 包

3.2 分析分段

以第一个分段为例来分析

ip.addr==10.3.9.161					
No.	Time	Source	Destination	Protocol	Length Info
22	1.907500	10.129.102.232	10.3.9.161	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fac4) [Reassembled in #27]
23	1.907500	10.129.102.232	10.3.9.161	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fac4) [Reassembled in #27]
24	1.907500	10.129.102.232	10.3.9.161	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=2960, ID=fac4) [Reassembled in #27]
25	1.907500	10.129.102.232	10.3.9.161	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=4440, ID=fac4) [Reassembled in #27]
26	1.907500	10.129.102.232	10.3.9.161	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=5920, ID=fac4) [Reassembled in #27]
27	1.907500	10.129.102.232	10.3.9.161	ICMP	642 Echo (ping) request id=0x0001, seq=405/38145, ttl=128 (reply in 33)

3.2.1 分片字段分析

以第一个分片为例来分析

Wireshark - 分组 22 - WLAN 2	
> Frame 22: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{62EC67D5-0...}	
> Ethernet II, Src: AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19), Dst: HewlettPacka_6c:0c:00 (10:4f:58:6c:0c:00)	
▼ Internet Protocol Version 4, Src: 10.129.102.232, 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: 0xfac4 (64196)	
▼ 001. .... = Flags: 0x1, More fragments	
0... .... = Reserved bit: Not set	
.0.. .... = Don't fragment: Not set	
..1. .... = More fragments: Set	
...0 0000 0000 0000 = Fragment Offset: 0	
Time to Live: 128	
Protocol: ICMP (1)	
Header Checksum: 0x954f [validation disabled]	
[Header checksum status: Unverified]	
Source Address: 10.129.102.232	
Destination Address: 10.3.9.161	
<a href="#">[Reassembled IPv4 in frame: 27]</a>	
> Data (1480 bytes)	

字段名称	值	含义
ip.flags.mf	True	MF 字段为 true，表示后面还有分片
ip.reassembled_in	27	[Reassembled IPv4 in frame: 27]，表示这个分片属于 frame27
ip.frag_offset	0	偏移量为 0，表示这个是第一个分片
ip.id	0xfac4	Identification:0xfac4 (64196)，表示标识

3.2.2 分片字段对比

分片	MF	ip.reassembled_in	Offset	id
22	True	27	0	0xfac4

23	True	27	1480	0xfac4
24	True	27	2960	0xfac4
25	True	27	4440	0xfac4
26	True	27	5920	0xfac4
27	False		7400	0xfac4

结论:

- 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。
- 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。
- MF=false 的表示最后一个分片, 意思是后面没有 fragment 了

## 四、分析 TCP 通信过程

### 4.1 建立连接并拆除连接的抓包过程

输入命令 `curl baidu.com` 并按回车。这会发起一个 HTTP 请求到 `baidu.com`, 触发 TCP 连接的三次握手过程。

连接完成后, `curl` 会自动关闭连接, 触发 TCP 四次挥手过程。

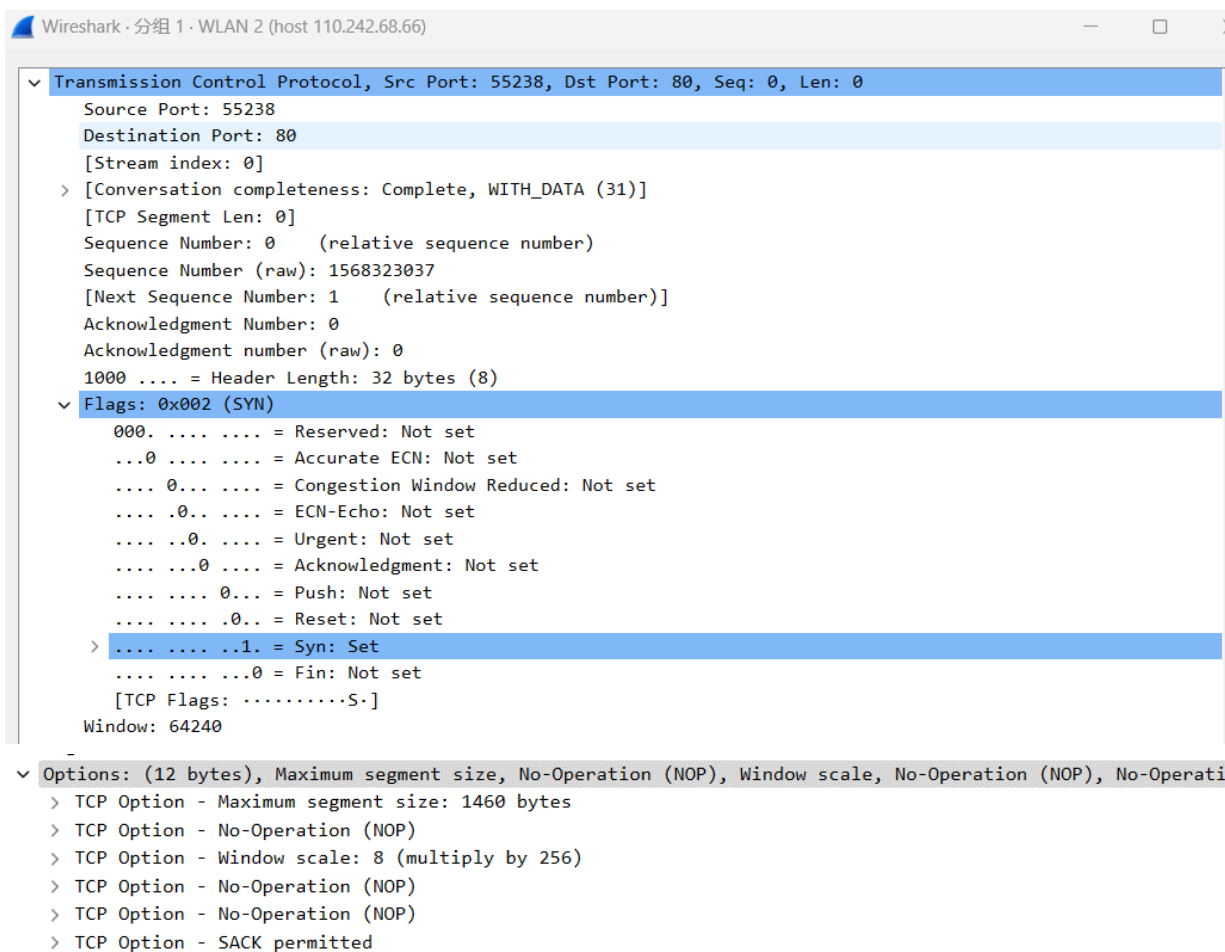
```
C:\Users\cyh>curl baidu.com
<html>
<meta http-equiv="refresh" content="0;url=http://www.baidu.com/">
</html>
```

Wireshark 提前设置好捕获条件为 host 110.242.68.66(使用 ping 得到的 `baidu.com` 的 ip), 抓包结果:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.129.102.232	110.242.68.66	TCP	66	55238 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.035060	110.242.68.66	10.129.102.232	TCP	66	80 → 55238 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1382 WS=32 SACK_PERM
3	0.035122	10.129.102.232	110.242.68.66	TCP	54	55238 → 80 [ACK] Seq=1 Ack=1 Win=131072 Len=0
4	0.053131	10.129.102.232	110.242.68.66	HTTP	126	GET / HTTP/1.1
5	0.132484	110.242.68.66	10.129.102.232	TCP	60	80 → 55238 [ACK] Seq=1 Ack=73 Win=24704 Len=0
6	0.132484	110.242.68.66	10.129.102.232	TCP	359	80 → 55238 [PSH, ACK] Seq=1 Ack=73 Win=24704 Len=305 [TCP segment of a reassembled ...
7	0.132484	110.242.68.66	10.129.102.232	HTTP	135	HTTP/1.1 200 OK (text/html)
8	0.132571	10.129.102.232	110.242.68.66	TCP	54	55238 → 80 [ACK] Seq=73 Ack=387 Win=130816 Len=0
9	0.132959	10.129.102.232	110.242.68.66	TCP	54	55238 → 80 [FIN, ACK] Seq=73 Ack=387 Win=130816 Len=0
10	0.180396	110.242.68.66	10.129.102.232	TCP	60	80 → 55238 [ACK] Seq=387 Ack=74 Win=24704 Len=0
11	0.180396	110.242.68.66	10.129.102.232	TCP	60	80 → 55238 [FIN, ACK] Seq=387 Ack=74 Win=24704 Len=0
12	0.180436	10.129.102.232	110.242.68.66	TCP	54	55238 → 80 [ACK] Seq=74 Ack=388 Win=130816 Len=0

### 4.2 三次握手

### 4.2.1 第一次握手



55238 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK\_PERM

由端口 55238（本机端口）发给端口 80（服务器端口）

对有意义的字段进行分析：

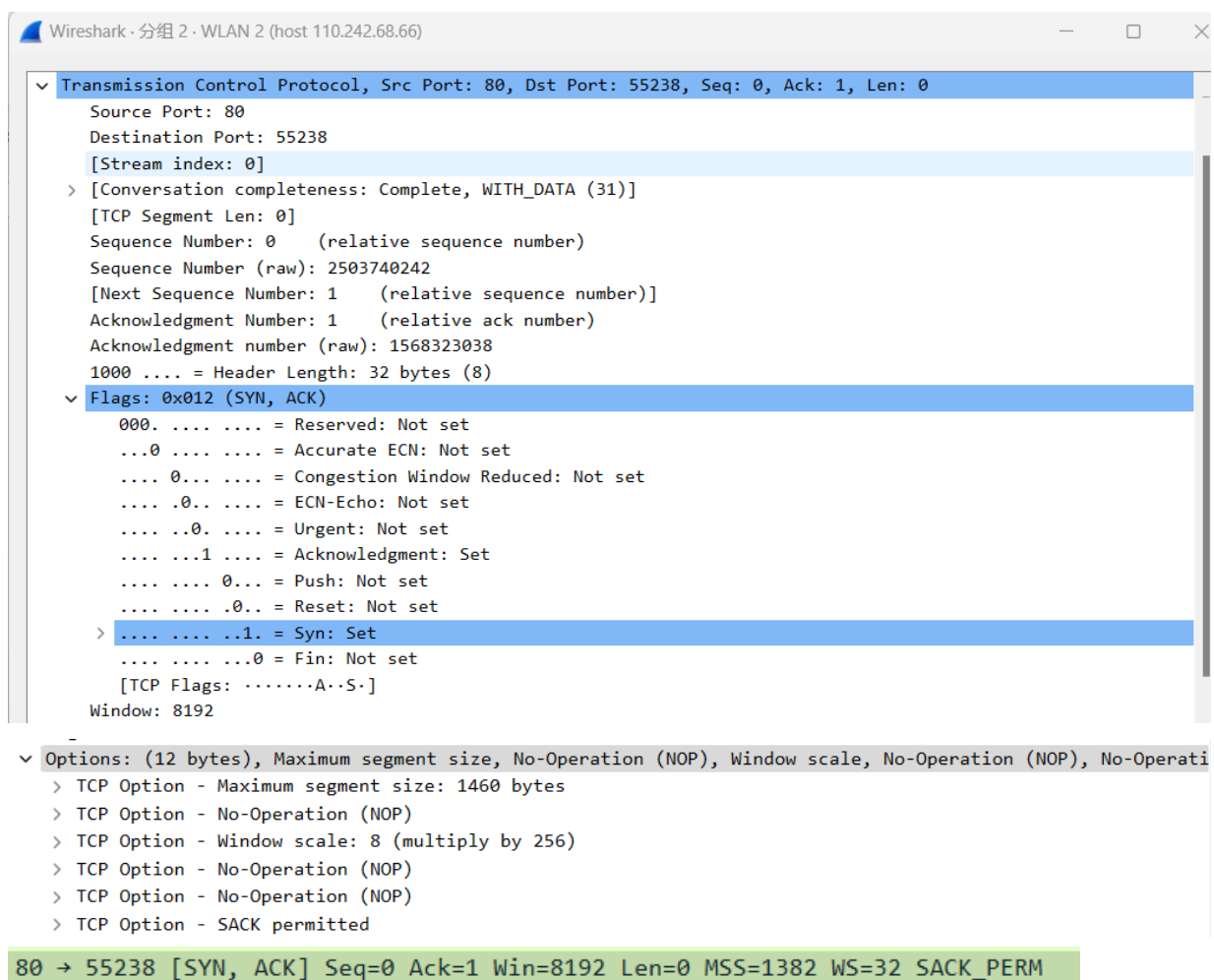
字段名称	值	含义
tcp.flags	0x0002	Flags: 0x002 (SYN)，连接由客户端发起，客户端发出第一个段，置 SYN 位，表示想建立连接
tcp.seq	0	Sequence number=0 表示这是发送的第一个段，是相对的序列号而不是真实的序列号
tcp.seq_raw	1568323037	这个表示真实的初始 sequence number，出于安全考虑通常是一个随机的数
tcp.ack	0	初始 ACK=0

<b>tcp.window_size_value</b>	64240	客户端接收窗口大小，但是由于此时连接还未协商好，这个并不是真实的接收窗口大小
<b>tcp.options</b>	020405660103030501010402	TCP 选项内容

结论：

连接请求由客户端发送，客户端发出第一个段，置 SYN 位，并给出了自己的初始 SEQ 编号，Window 字段给出了接收窗口的大小。

#### 4.2.2 第二次握手



Wireshark · 分组 2 · WLAN 2 (host 110.242.68.66)

Transmission Control Protocol, Src Port: 80, Dst Port: 55238, Seq: 0, Ack: 1, Len: 0

- Source Port: 80
- Destination Port: 55238
- [Stream index: 0]
- > [Conversation completeness: Complete, WITH\_DATA (31)]
- [TCP Segment Len: 0]
- Sequence Number: 0 (relative sequence number)
- Sequence Number (raw): 2503740242
- [Next Sequence Number: 1 (relative sequence number)]
- Acknowledgment Number: 1 (relative ack number)
- Acknowledgment number (raw): 1568323038
- 1000 .... = Header Length: 32 bytes (8)
- ✓ Flags: 0x012 (SYN, ACK)
  - 000. .... = Reserved: Not set
  - ...0 .... = Accurate ECN: Not set
  - .... 0... = Congestion Window Reduced: Not set
  - .... .0.. = ECN-Echo: Not set
  - .... ..0. = Urgent: Not set
  - .... ...1 = Acknowledgment: Set
  - .... .... 0... = Push: Not set
  - .... ..... 0.. = Reset: Not set
  - > .... .... ..1. = Syn: Set
  - .... .... ...0 = Fin: Not set
  - [TCP Flags: .....A..S.]
- Window: 8192
- Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP)
  - > TCP Option - Maximum segment size: 1460 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

80 → 55238 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1382 WS=32 SACK\_PERM

由端口 80（服务器）发给端口 55238（本机端口）

对有意义的字段进行分析：

字段名称	值	含义
<b>tcp.flags</b>	0x0012	Flags: 0x012 (SYN, ACK), 置 SYN 和 ACK
<b>tcp.seq</b>	0	这个段是服务器发出的第一个段，因此相对序列号为 0

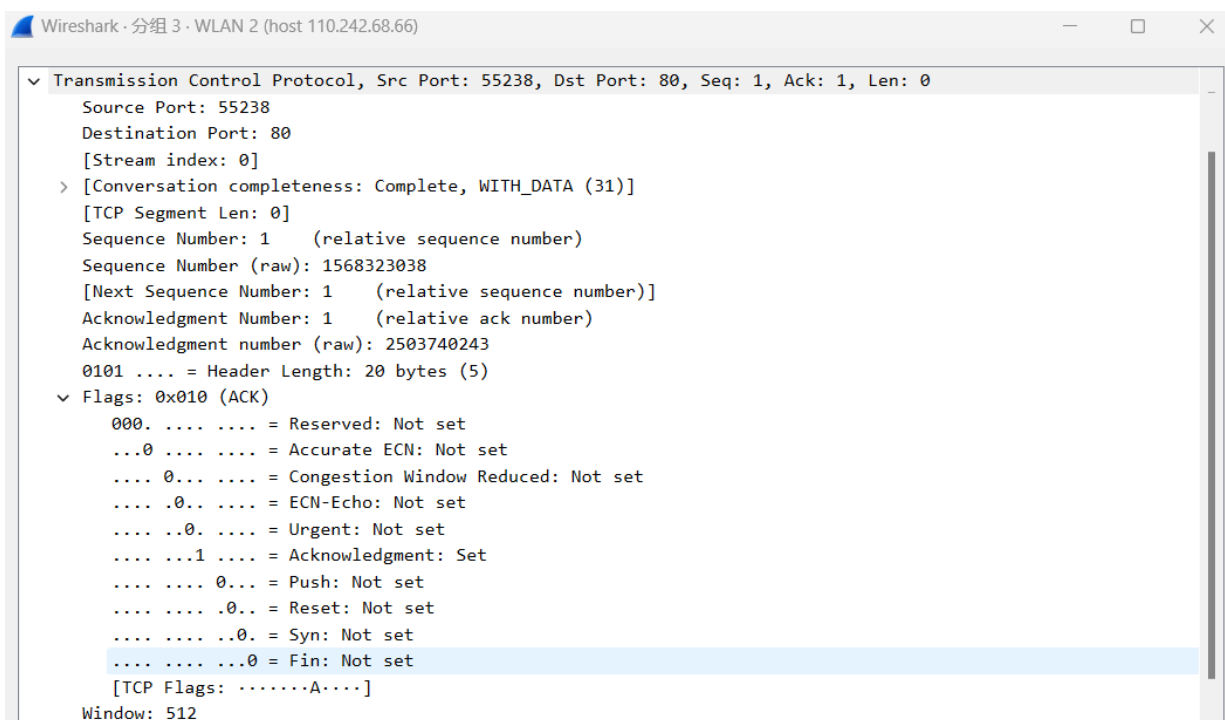


tcp.seq_raw	2503740242	表示真实的序列号
tcp.ack	1	表示期望接收到的下一个段的序列号，采用累计确认，说明刚刚客户端发送的序列号为 0 的那个段（SYN）已经被收到了
tcp.window_size_value	8192	服务器端的接收窗口大小
tcp.options	020405b40103030801010402	TCP 选项内容

结论：

第二个段由服务器发出，即“第二次握手”，置 SYN 和 ACK 位。这个段的 TCP 选项内容和第一个段相同，表示双方协商成功。

### 4.2.3 第三次握手



55238 → 80 [ACK] Seq=1 Ack=1 Win=131072 Len=0

由端口 55238（本机端口）发给端口 80（服务器）

分析有意义的字段：

字段名称	值	含义
tcp.flags	0x0010	Flags: 0x0010 (ACK)
tcp.seq	1	这个段是客户端发送的第二个段
tcp.seq_raw	1568323038	真实的序列号，也是相对序列号为 0 的段的真实序列号加一
tcp.ack	1	表示已经接收到了服务器



		端序列号为 0 的段，下一个想接收的是服务器端序列号为 1 的段
<b>tcp.window_size_value</b>	512	客户端给出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。

结论：

第三次握手标志位为：ACK，表示确认收到了连接回复，三次握手就建立连接完毕

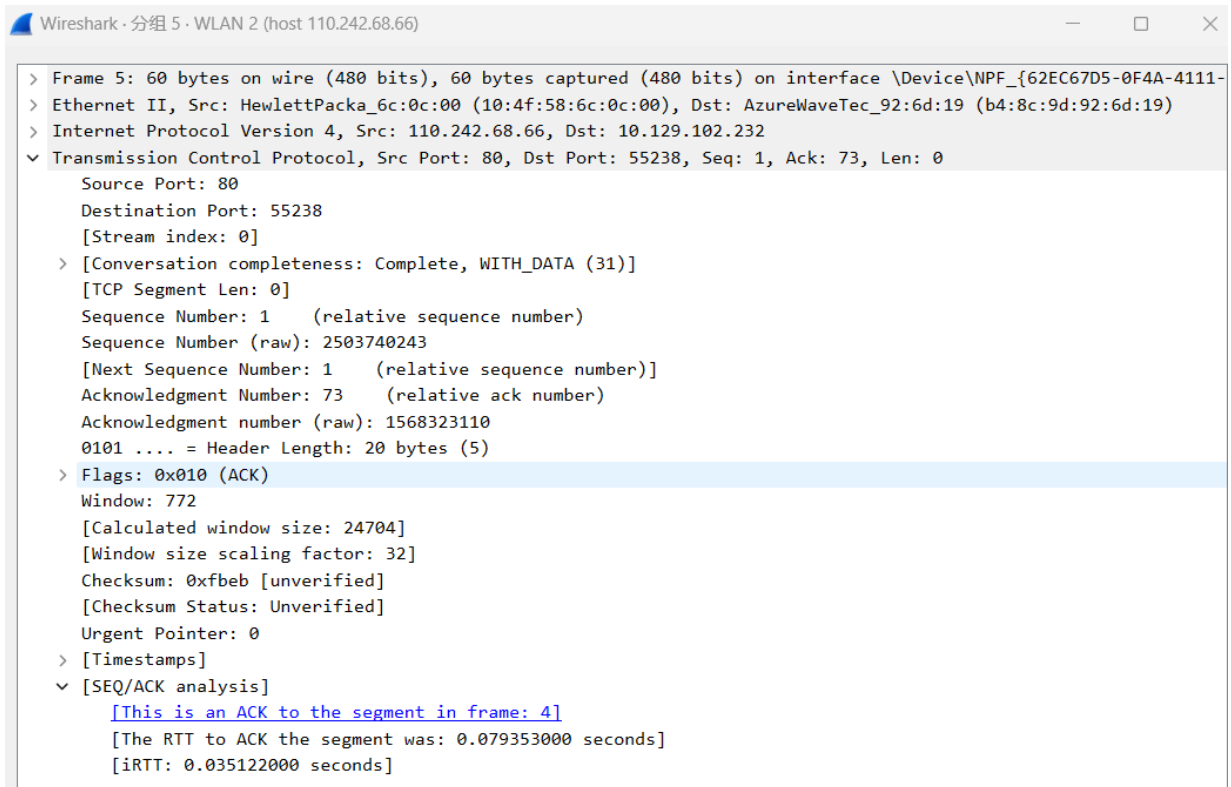
## 4.3 数据通信

```

HTTP      126 GET / HTTP/1.1
Wireshark · 分组 4 · WLAN 2 (host 110.242.68.66)
> Ethernet II, Src: AzureWaveTec_92:6d:19 (b4:8c:9d:92:6d:19), Dst: HewlettPacka_6c:0c:00 (10:4f:58:6c:0c:00)
> Internet Protocol Version 4, Src: 10.129.102.232, Dst: 110.242.68.66
✓ Transmission Control Protocol, Src Port: 55238, Dst Port: 80, Seq: 1, Ack: 1, Len: 72
  Source Port: 55238
  Destination Port: 80
  [Stream index: 0]
  > [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 72]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 1568323038
  [Next Sequence Number: 73 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 2503740243
  0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
  Window: 512
  [Calculated window size: 131072]
  [Window size scaling factor: 256]
  Checksum: 0xc9bd [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > [Timestamps]
  ✓ [SEQ/ACK analysis]
    [iRTT: 0.035122000 seconds]
    [Bytes in flight: 72]
    [Bytes sent since last PSH flag: 72]
  TCP payload (72 bytes)

```

第四个段是建立 http 请求，开始传输数据，SEQ=1，ack=1，由客户端发往服务器端，置 PSH 和 ACK 位。这一段的 SEQ 仍为 1，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。



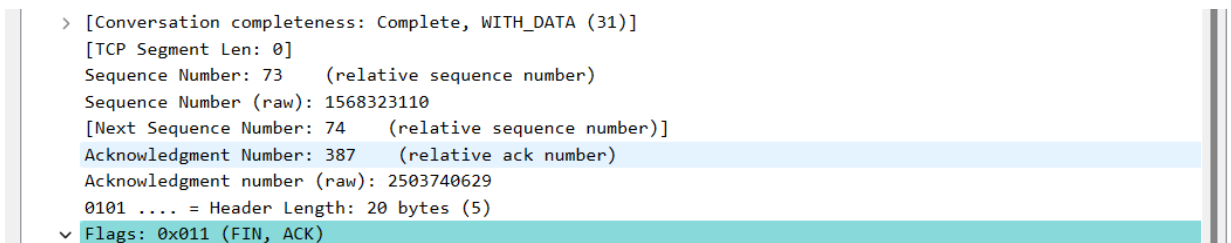
第五段是服务器相应 http 请求，SEQ=1，ACK=73，由服务器端发往客户端，置 ACK，表示第 73 字节之前的字节都收到了，下一次期望收到序列号为 73 的段。

后面的数据通信过程都跟这两段的类似，只是变为服务器发送数据，客户端确认数据，序列号都为上一次的序列号+数据长度，ACK 都为上一次的 ACK+确认接收的数据长度。

4 0.053131	10.129.102.232	110.242.68.66	HTTP	126 GET / HTTP/1.1
5 0.132484	110.242.68.66	10.129.102.232	TCP	60 80 → 55238 [ACK] Seq=1 Ack=73 Win=24704 Len=0
6 0.132484	110.242.68.66	10.129.102.232	TCP	359 80 → 55238 [PSH, ACK] Seq=1 Ack=73 Win=24704 Len=305 [TCP segment of a reassembled ...
7 0.132484	110.242.68.66	10.129.102.232	HTTP	135 HTTP/1.1 200 OK (text/html)
8 0.132571	10.129.102.232	110.242.68.66	TCP	54 55238 → 80 [ACK] Seq=73 Ack=387 Win=130816 Len=0

## 4.4 四次挥手

### 4.4.1 第一次挥手：客户端->服务器端



分析有意义的字段：

字段名称	值	含义
tcp.flags	0x0011	Flags: 0x011 (FIN, ACK),

		置 ACK 和 FIN，表示确认服务器端上一次发送的数据同时向服务器端表明想要拆除连接
tcp.seq	73	当前段的 SEQ，但客户端从此再也没有数据发送，这个值会在后续过程一直不变

结论：

客户端想要拆除连接，因此发送给服务器端一个 FIN 标志位置 1 的段  
至此客户端进入 FIN\_WAIT\_1 阶段，服务器端进入 CLOSE\_WAIT 阶段

#### 4.4.2 第二次挥手：服务器端->客户端

```
> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 387 (relative sequence number)
Sequence Number (raw): 2503740629
[Next Sequence Number: 387 (relative sequence number)]
Acknowledgment Number: 74 (relative ack number)
Acknowledgment number (raw): 1568323111
0101 .... = Header Length: 20 bytes (5)
v Flags: 0x010 (ACK)
```

分析有意义的字段：

字段名称	值	含义
tcp.flags	0x0010	Flags: 0x010 (ACK)
tcp.ack	74	表示已经收到了刚刚客户端发送的 FIN 为 1 的段（序列号为 73）
tcp.seq	387	

结论：

服务器端向客户端发送一个 FIN=0 的段，表示服务器端已经接收到客户端想要关闭连接的请求，但服务器端还没完全关闭连接，在此期间服务器仍然可以单方面发送数据。

至此客户端进入 FIN\_WAIT\_2 阶段

#### 4.4.3 第三次挥手：服务器端->客户端

```
> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 387      (relative sequence number)
Sequence Number (raw): 2503740629
[Next Sequence Number: 388      (relative sequence number)]
Acknowledgment Number: 74      (relative ack number)
Acknowledgment number (raw): 1568323111
0101 .... = Header Length: 20 bytes (5)
✓ Flags: 0x011 (FIN, ACK)
```

分析有意义的字段：

字段名称	值	含义
<b>tcp.flags</b>	0x0011	Flags: 0x011 (FIN, ACK), 此时服务器端已经确定没有数据要发了，因此发送给客户端一个 FIN=1 的段
<b>tcp.ack</b>	74	跟刚刚的一样
<b>tcp.seq</b>	387	跟刚刚的一样，说明刚刚的 ACK 没有被算作一个段

结论：

服务器端已经确定没有数据要发了，因此发送给客户端一个 FIN=1 的段。

至此客户端进入 TIME\_WAIT 阶段，服务器端进入 LAST\_ACK 阶段，服务器端的连接已经关闭。

#### 4.4.4 第四次挥手：客户端->服务器端

```
> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 74      (relative sequence number)
Sequence Number (raw): 1568323111
[Next Sequence Number: 74      (relative sequence number)]
Acknowledgment Number: 388      (relative ack number)
Acknowledgment number (raw): 2503740630
0101 .... = Header Length: 20 bytes (5)
> Flags: 0x010 (ACK)
```

分析有意义的字段：

字段名称	值	含义
<b>tcp.flags</b>	0x0010	Flags: 0x010 (ACK), 表示客户端已经收到了服务器端发送的连接关闭信号，现在发送最后一个 ACK
<b>tcp.ack</b>	388	表示接收到服务器端发过

结论:

至此客户端进入等待阶段直到超时，然后连接关闭，服务器端收到 ACK 后彻底关闭连接。

#### 4.4.5 整个过程示意图

