



## 计算机网络实验报告

实验名称：实验二 数据帧和IP包分析

系别：

班号：

实验者姓名：

学号：

实验日期：2019年10月23日

实验报告完成日期：2019年10月25日

指导老师意见：

## 一、实验目的及要求

掌握以太网MAC帧和IP数据包的构成，了解字段的含义

学习如何捕获和分析网络数据包，掌握ICMP和ARP协议的请求/响应机理深入

学习ping、arp和tracert指令

学习使用工具构造IP数据报

## 二、实验内容

1、学习工具Wireshark

2、以太网数据帧分析

ping指令：观察以太网帧格式，解释ICMP报文

arp指令：记录和解释ARP协议工作原理

tracert指令：记录和解释tracert的工作原理

3. 无线网络抓取802.11数据报并分析

构建适合的wifi环境，用Wireshark捕获无线数据报

过滤这些数据包中的802.11数据帧（重点说明这些帧的作用）

## 三、实验环境

Windows7（虚拟机）

Mac OS（链接XMUNET+无线网络）

虚拟机网络接口：以太网

Windows本机ip：10.24.90.140

Windows网关ip：10.24.90.1

## 四、实验原理

Wireshark是网络封包分析软件（前称Ethereal）

捕获网络封包

显示网络封包的详细信息

使用WinPcap作为接口，直接与网卡进行数据报文交换丰富的统计功能

## 五、实验过程

### （一）以太网帧分析——帧格式

实验要求：在本机上发起网络命令：ping IP/域名地址

观察 Wireshark 捕获的数据，查看 ICMP 请求帧和回应帧，分析其结构组成

#### （1）请求帧

##### 1、记录和解释一个 MAC 帧

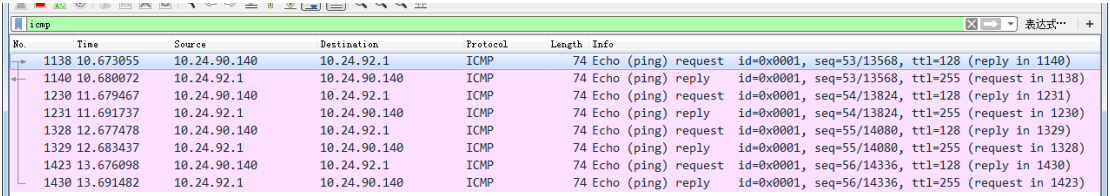
首先，打开控制台程序，ping 通 10.24.90.1，如图所示。

```
C:\Windows\system32>ping 10.24.90.1

正在 Ping 10.24.90.1 具有 32 字节的数据:
来自 10.24.90.1 的回复: 字节=32 时间=16ms TTL=255
来自 10.24.90.1 的回复: 字节=32 时间=56ms TTL=255
来自 10.24.90.1 的回复: 字节=32 时间=4ms TTL=255
来自 10.24.90.1 的回复: 字节=32 时间=46ms TTL=255

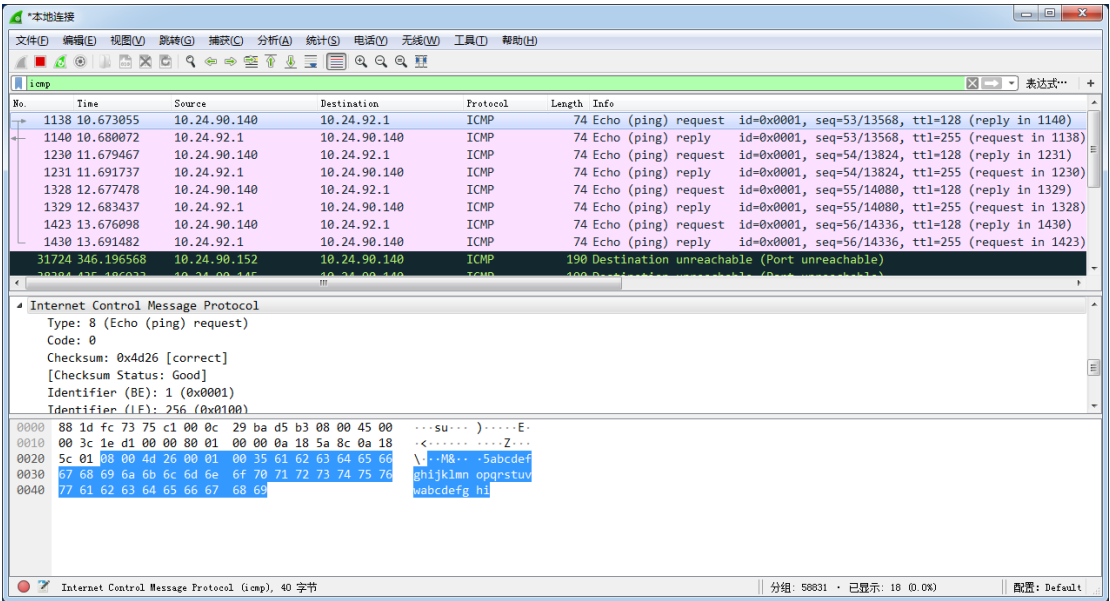
10.24.90.1 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 4ms, 最长 = 56ms, 平均 = 30ms
```

与此同时，利用 Wireshark 捕获数据帧。在 Filter 文本框内输入过滤条件，筛选出 ICMP 协议的数据帧，如图所示，就是刚刚 ping 通另一台主机时发送的请求和响应数据帧。



No.	Time	Source	Destination	Protocol	Length	Info
1138	10.673055	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=53/13568, ttl=128 (reply in 1140)
1140	10.680072	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=53/13568, ttl=255 (request in 1138)
1230	11.679467	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=54/13824, ttl=128 (reply in 1231)
1231	11.691737	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=54/13824, ttl=255 (request in 1230)
1328	12.677478	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=55/14080, ttl=128 (reply in 1329)
1329	12.683437	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=55/14080, ttl=255 (request in 1328)
1423	13.676098	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=56/14336, ttl=128 (reply in 1430)
1430	13.691482	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=56/14336, ttl=255 (request in 1423)

如图，一共获得了八个 ICMP 数据包，其中请求帧和回应帧各 4 个。



No.	Time	Source	Destination	Protocol	Length	Info
1138	10.673055	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=53/13568, ttl=128 (reply in 1140)
1140	10.680072	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=53/13568, ttl=255 (request in 1138)
1230	11.679467	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=54/13824, ttl=128 (reply in 1231)
1231	11.691737	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=54/13824, ttl=255 (request in 1230)
1328	12.677478	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=55/14080, ttl=128 (reply in 1329)
1329	12.683437	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=55/14080, ttl=255 (request in 1328)
1423	13.676098	10.24.90.140	10.24.92.1	ICMP	74	Echo (ping) request id=0x0001, seq=56/14336, ttl=128 (reply in 1430)
1430	13.691482	10.24.92.1	10.24.90.140	ICMP	74	Echo (ping) reply id=0x0001, seq=56/14336, ttl=255 (request in 1423)

Internet Control Message Protocol	
Type: 8 (Echo (ping) request)	
Code: 0	
Checksum: 0x4d26 [correct]	
[Checksum Status: Good]	
Identifier (BE): 1 (0x0001)	
Identifier (LE): 256 (0x0100)	

Raw Data (Hex)	
0000	88 1d fc 73 75 c1 00 0c 29 ba d5 b3 08 00 45 00
0010	00 3c 1e d1 00 00 00 01 00 00 0a 18 5a 8c 0a 18
0020	5c 01 08 00 4d 26 00 01 00 35 61 62 63 64 65 66
0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
0040	77 61 62 63 64 65 66 67 68 69

如图，第一个请求包共长度为 74 字节，592 位。  
开头的 6 个字节 88 1d fc 73 75，表示目的地址 (MAC 地址) 为 88:1d:fc:73:75。  
紧接着的 6 个字节 00 0c 29 ba d5 b3，表示源地址 (MAC 地址) 为 00:0c:29:ba:d5:b3。  
最后两个字节 08 00 为类型字段，表示上层使用的是 IP 数据报，以太网帧类型：IPV4 // Type: IPv4 (0x0800)

## 2、ipv4 数据包分析

接着向上层分析，如图所示，对 IP 数据报进行解析。

如图，共有共 20 个字节。源：10.24.92.140，目标：10.24.92.1。

第 1 字节：0x45，表示 IPV 协议版本号为 4 以及头部长度为 20byte。

0100 .... = Version: 4, .... 0101 = Header Length: 20 bytes (5)

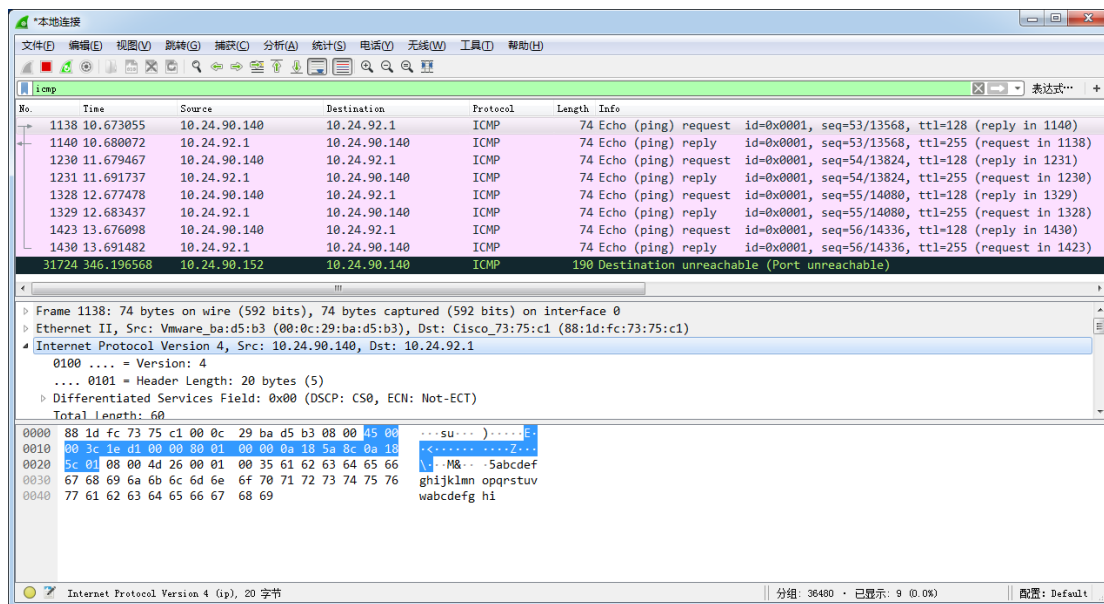
第 2 字节：0x00，表示 DSCP (IP 优先级字段)，值为 0，表示“尽力服务”。

DSCP 使用 6 个 bit，DSCP 的值得范围为 0~63。

DSCP 是“IP 优先”和“服务类型”字段的组合。为了利用只支持“IP 优先”的旧路由器，会使用 DSCP 值，因为 DSCP 值与“IP 优先”字段兼容。

第 3-4 字节：0x003c，表示报文总长度 60 字节 (头部 20 字节+icmp 报文 40 个字节=40 个字节)

Total Length: 84



第 5-6 字节：0x1ed1，用来唯一地标识一个报文的所有分片，因为分片不一定按序到达，所以在重组时需要知道分片所属的报文。每产生一个数据报，计数器加 1，并赋值给此字段。

第 7-8 字节：0x0000，前 3 位为标志位，后 13 位为分片位移

标志位：这个 3 位字段用于控制和识别分片，它们是：

位 0：保留，必须为 0；

位 1：禁止分片（Don't Fragment，DF），当 DF=0 时才允许分片；

位 2：更多分片（More Fragment，MF），MF=1 代表后面还有分片，MF=0 代表已经是最后一个分片。

如果 DF 标志被设置为 1，但路由要求必须分片报文，此报文会被丢弃。这个标志可被用于发往没有能力组装分片的主机。

当一个报文被分片，除了最后一块外的所有分片都设置 MF 为 1。最后一个片段具有非零 片段偏移字段，将其与未分片数据包区分开，未分片的偏移字段为 0。

分片偏移（Fragment Offset）

这个 13 位字段指明了每个分片相对于原始报文开头的偏移量，以 8 字节作单位。

第 9 字节：0x80，存活时间为 128（Time To Live，TTL）

这个 8 位字段避免报文在互联网中永远存在（例如陷入路由环路）。存活时间以秒为单位，但小于一秒的时间均向上取整到一秒。在现实中，这实际上成了一个跳数计数器：报文经过的每个路由器都将此字段减 1，当此字段等于 0 时，报文不再向下一跳传送并被丢弃，最大值是 255。

第 10 字节：0x01，协议（Protocol）

占 8bit，这个字段定义了该报文数据区使用的协议。IANA 维护着一份协议列表（最初由 RFC790 定义），详细参见 IP 协议号列表。

第 11-12 字节：0x0000，首部检验和（Header Checksum）

这个 16 位检验和字段只对首部查错，不包括数据部分。在每一跳，路由器都要重新计算出的首部检验和并与此字段进行比对，如果不一致，此报文将会被丢弃。重新计算的必要性是因为每一跳的一些首部字段（如 TTL、Flag、Offset 等）都有可能发生变化，不检查数据部分是为了减少工作量。

### \*Ipv4 的 Checksum 算法

对 Ipv4 首部字节,按字节划分,跳过 checksum 的字节,其余字节求和,最高位进位加到最低位上,得到的和,按位取反即得到 Checksum 的值 以该数据包为例,该 ip 报文头部为

4500 003c 1ed1 0000 8001 0000 0a18 5a8c 0a18 5c01

其中 85f8 为 checksum,故计算 checksum 之前,该字节未知,跳过该字节,把剩余的字节加和

$4500+003c+1ed1+0000+8001+0a18+5a8c+0a18+5c01=1af34$

最高位的进位加到最低位上去:  $af34+1 = af35$

取反得到 checksum 值  $FFFF-af34=50ca$

说明并没有进行首部检验。

第 13-16 字节: 0x0a18 5a8c, 源 ip 地址 10.24.92.140。

一个 IPv4 地址由四个字节共 32 位构成,此字段的值是将每个字节转为二进制并拼在一起所得到的 32 位值。

第 17-20 字节: 0x0a18 5c01, 目的 ip 地址 10.24.92.1

与源地址格式相同,但指出报文的接收端。

### 3、数据段

IP 数据包首部之后为数据段,数据字段不是首部的一部分,因此并不被包含在首部检验和中。数据的格式在协议首部字段中被指明,并可以是任意的传输层协议。一些常见协议的协议字段值被列在下面:

协议字段值	协议名	缩写
1	互联网控制消息协议	ICMP
2	互联网组管理协议	IGMP
6	传输控制协议	TCP
17	用户数据报协议	UDP
41	IPv6 封装	ENCAP
89	开放式最短路径优先	OSPF
132	流控制传输协议	SCTP

### 4、ICMP 包分析

ICMP 报头从 IP 报头的第 160 位开始 (IP 首部 20 字节) (除非使用了 IP 报头的可选部分)。

160-167	168-175	176-191	192-199	200-207
Type	Code	校验码	ID	序号

Type- ICMP 的类型,标识生成的错误报文;

Code- 进一步划分 ICMP 的类型,该字段用来查找产生错误的原因.;

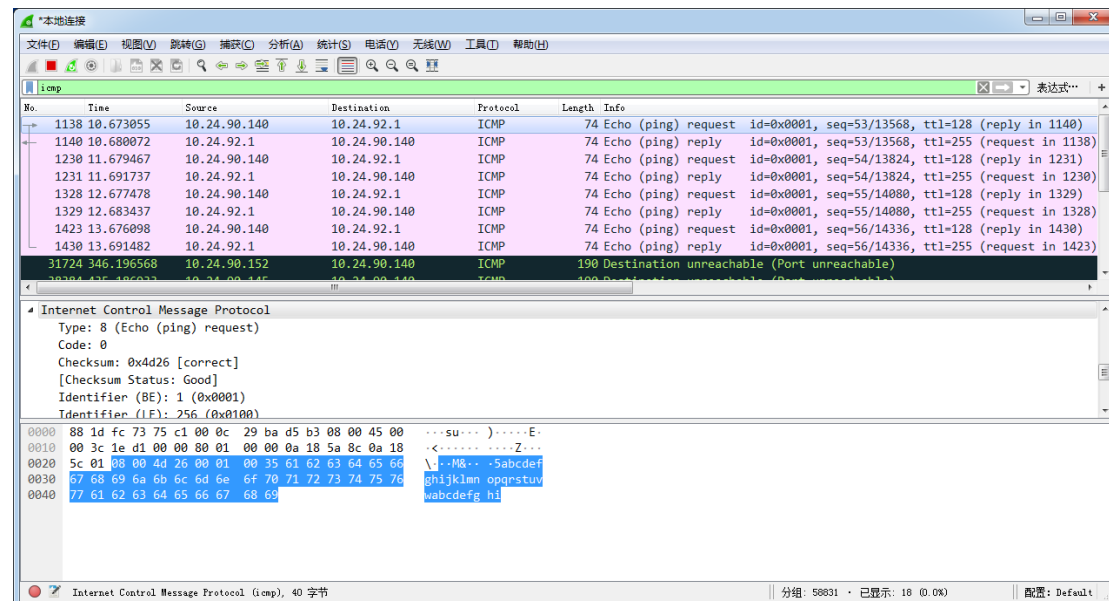
例如,ICMP 的目标不可达类型可以把这个位设为 1 至 15 等来表示不同的意思。

Checksum- 校验码部分,这个字段包含有从 ICMP 报头和数据部分计算得来的,用于检查错误的数据,其中此校验码字段的值视为 0。

ID- 这个字段包含了 ID 值,在 Echo Reply 类型的消息中要返回这个字段。

Sequence- 这个字段包含一个序号,同样要在 Echo Reply 类型的消息中要返回这个字段。

填充数据：填充的数据紧接在 ICMP 报头的后面（以 8 位为一组）：  
Windows 的“ping.exe”填充的 ICMP 除了 8 个 8 位组的报头以外，默认情况下还另外填充数据使得总大小为 40 字节。



基本信息： 共 40 个字节

第 1-2 字节：0x0800, Type (ICMP 的类型)+code (ICMP 子类型)

Echo request (used to ping)

第 3-4 字节：0x4d26, 为校验和。

第 5-6 字节：0x0001, 标识符，用于匹配发出包与回应包，同一对发出包与回应包的该位置的标识符完全相同。

第 7-8 字节：0x0035, Sequence number (报文序号)

这个字段包含一个序号，同样要在 Echo Reply 类型的消息中要返回这个字段。

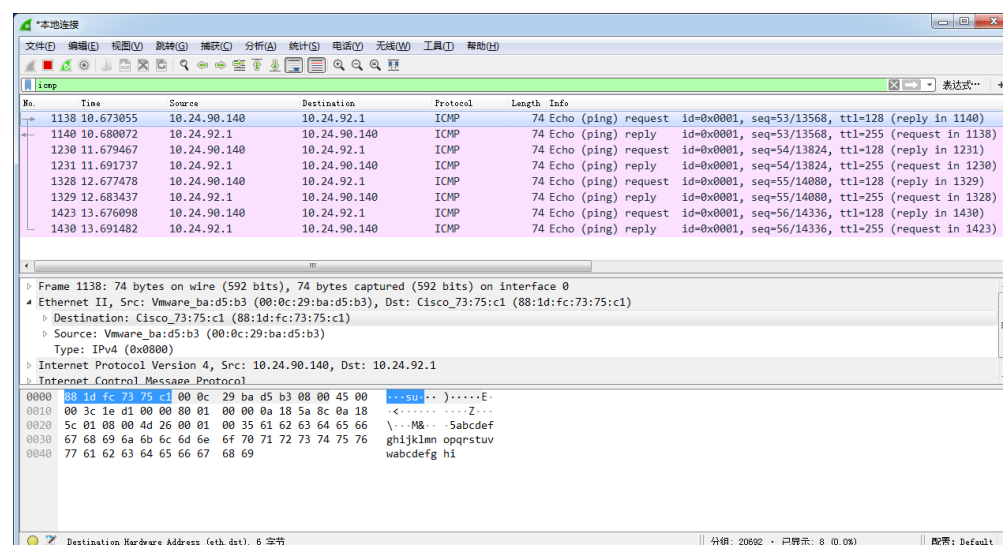
第 9-40 字节：数据域

(2) ping 请求帧和响应帧首部比较

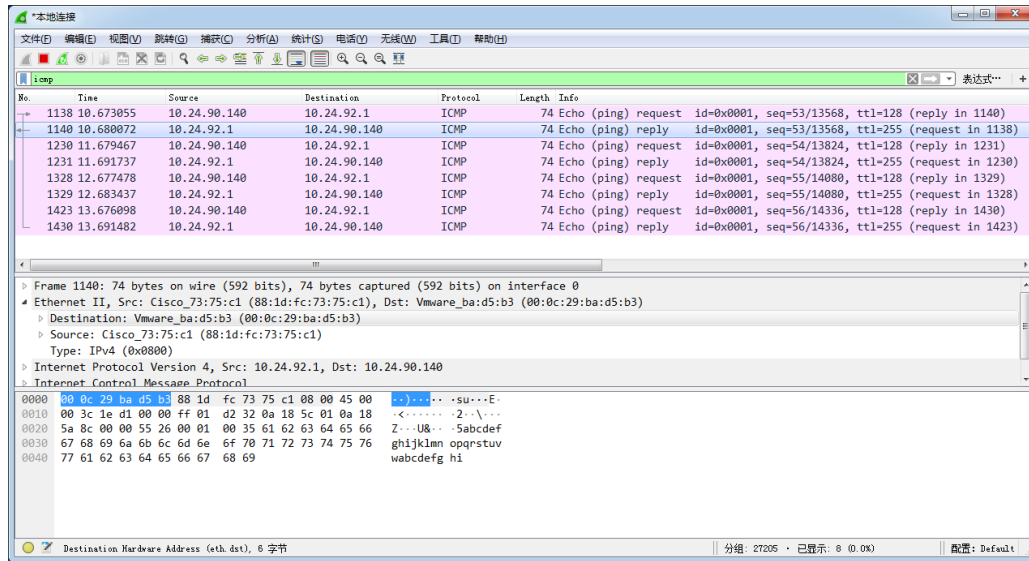
1、MAC 帧首部比较

不同： MAC 帧的首部的源与目的 mac 地址对调

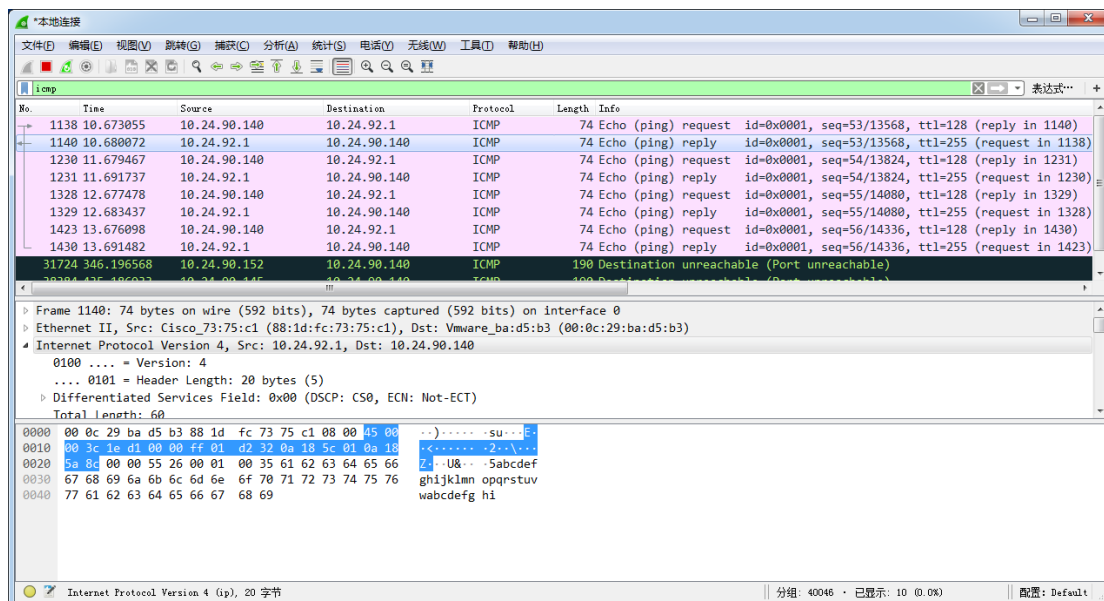
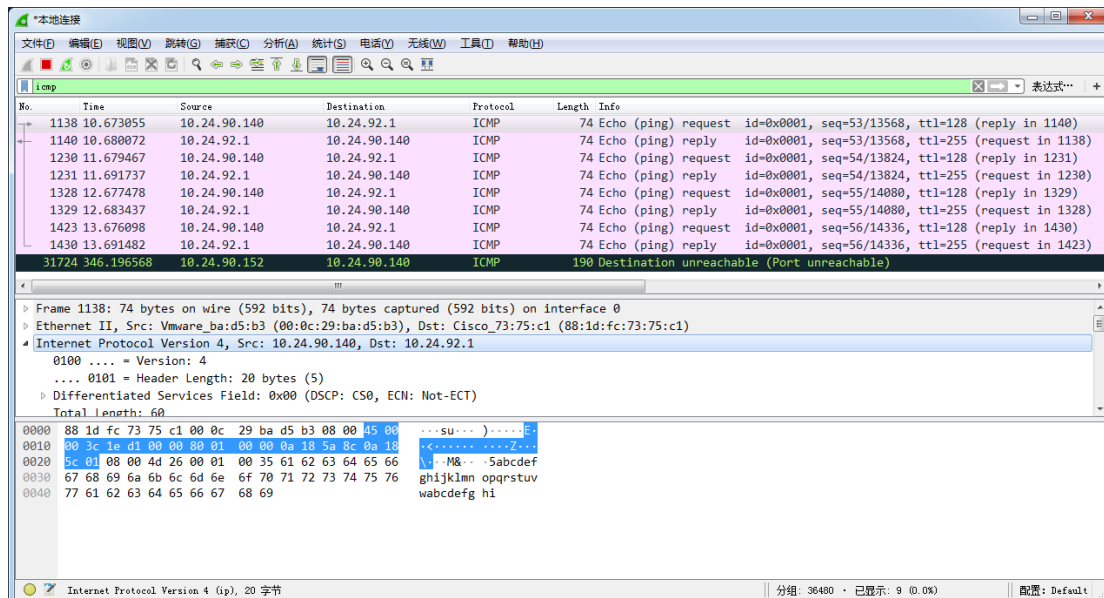
相同： 协议类型字段不变，都为 0x0800，即 Ipv4





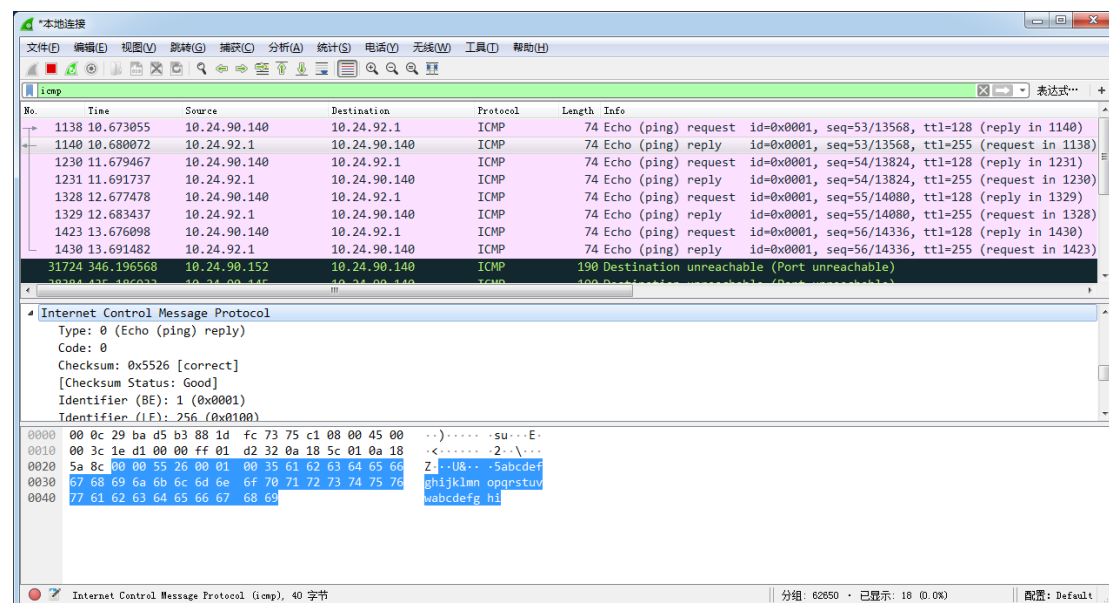
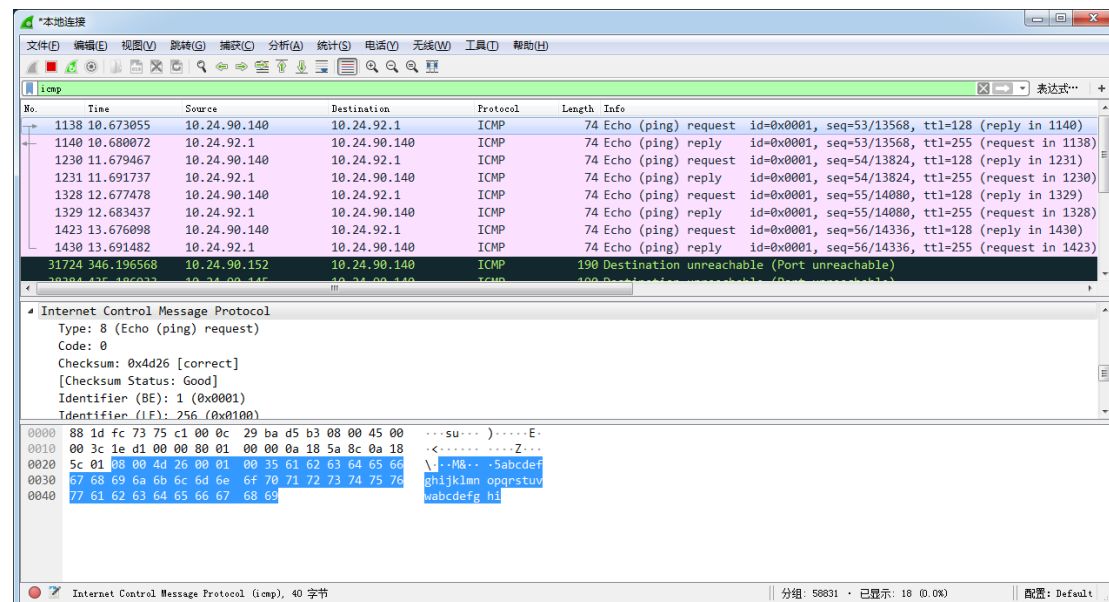


## 2、IP 首部比较



不同：  
TTL 字段，request 为 128，request 为 255；  
checksum 字段；  
source 字段，destination 字段值互换；  
其余字段均相同。

### 3、ICMP 报文首部



不同：  
Type 字段：request 为 08，reply 为 0；  
checksum 字段。

## （二）ARP 协议工作原理的分析

- 1、使用 `arp -d` 命令，清空本机已有 ARP 缓存；
  - 2、发起网络活动，如 ping 命令、打开网页、文件上传下载；
- 在清空本机的 ARP 记录后，仿照先前过程，ping 通 10.24.92.139（隔壁主机）



以获取数据帧，筛选出使用 ARP 协议的数据帧。

3、观察 Wireshark 捕获的 ARP 报文，解释各字段意义，并描述请求/响应的过程；

### (1) ARP 数据包结构

#### A. 以太网链路层

目标以太网地址：目标 MAC 地址。FF:FF:FF:FF:FF:FF （二进制全 1）为广播地址。

源以太网地址：发送方 MAC 地址。

帧类型：以太类型，ARP 为 0x0806。

#### B. 以太网报文数据

硬件类型：如以太网（0x0001）、分组无线网。

协议类型：如网际协议(IP)（0x0800）、IPv6（0x86DD）。

硬件地址长度：每种硬件地址的字节长度，一般为 6（以太网）。

协议地址长度：每种协议地址的字节长度，一般为 4（IPv4）。

操作码：1 为 ARP 请求，2 为 ARP 应答，3 为 RARP 请求，4 为 RARP 应答。

源硬件地址：n 个字节，n 由硬件地址长度得到，一般为发送方 MAC 地址。

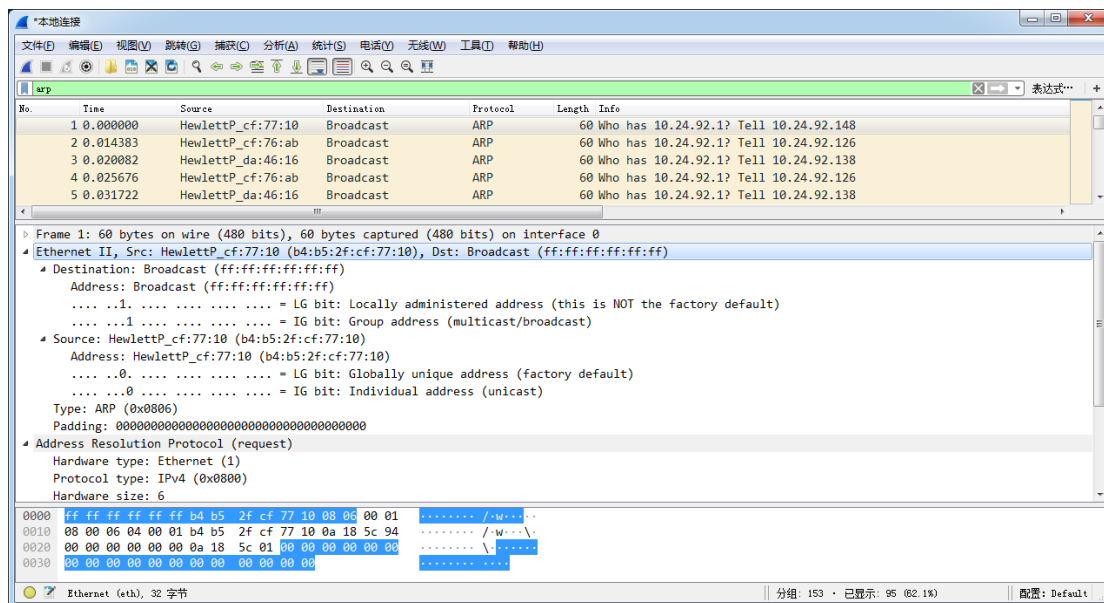
源协议地址：m 个字节，m 由协议地址长度得到，一般为发送方 IP 地址。

目标硬件地址：n 个字节，n 由硬件地址长度得到，一般为目标 MAC 地址。

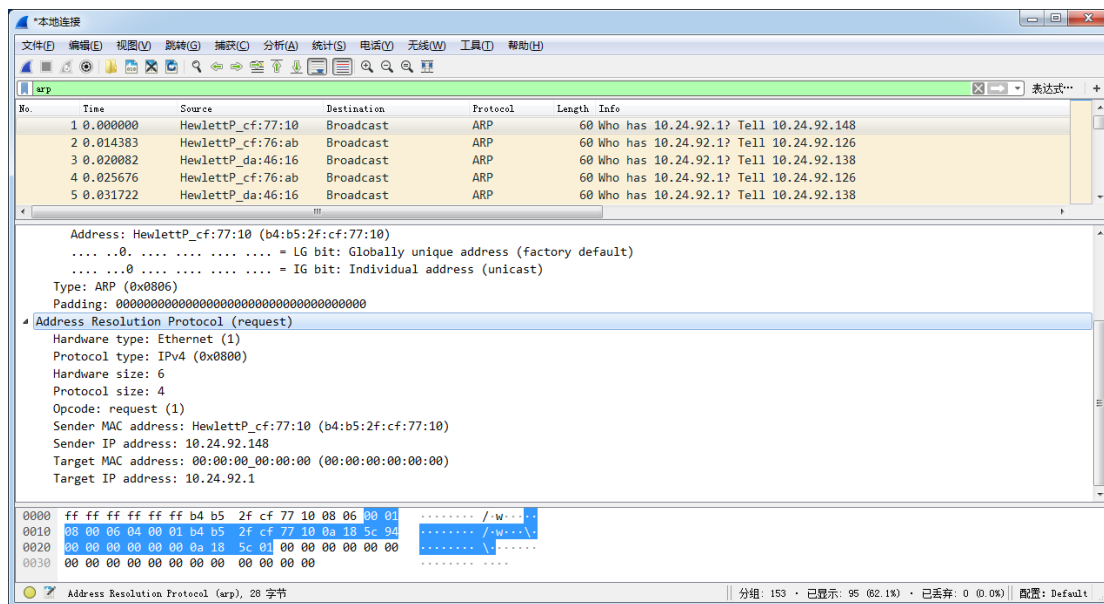
目标协议地址：m 个字节，m 由协议地址长度得到，一般为目标 IP 地址。

### (2) 分析捕获的 ARP 数据包

先解释请求数据帧，如图：



使用 ARP 协议的请求数据帧的 MAC 头



### 使用 ARP 协议的请求数据帧的 ARP 报文

- 第 1-6 字节： 目标主机 MAC 地址，若为 FF FF FF FF FF FF（二进制全一）表示广播地址；
  - 第 7-12 字节： 源以太网地址：发送放主机 MAC 地址；
  - 第 13 字节以后为报文数据；
  - 第 13-14 字节： 0x0806，报文类型，0x0806 表示该数据包为 ARP 包；
  - 第 15-16 字节： 0x0001，硬件类型，0x0001 表示以太网；
  - 第 17-18 字节： 0x0800，表示协议烈性，0x0800 表示 Ipv4；
  - 第 19 字节： 0x06，硬件地址的字节长度，以太网为 6；
  - 第 20 字节： 0x04，协议地址长度，Ipv4 协议为 4；
  - 第 21-22 字节： 0x0001，操作码：1 为 ARP 请求，2 为 ARP 应答，3 为 RARP 请求，4 为 RARP 应答；
  - 第 23-28 字节： 0xb4 b5 2f cf 77 10，发送方主机的 mac 地址；
  - 第 28-32 字节： 0x0a 18 5c 94，发送方的 IP 地址；
  - 第 33-38 字节： 0x00 00 00 00 00，目标硬件地址；
  - 第 39-40 字节： 0x0a 18 5c 91，目标主机 IP 地址。
- 而在报文末尾，还进行了若干填充，使得报文长度符合规定下限。

### （三）记录和解释 tracert 的工作原理

#### 1、tracert 工作原理

分组网间探测 PING，是用来测试两个主机之间的连通性的。PING 使用 ICMP 回送请求与回送回答报文。PC 机上运行 PING，就会向待测试主机一连发送四个 ICMP 回送请求报文。在服务器正常工作且响应这个 ICMP 回送请求报文，就会发回 ICMP 回送回答报文。由于往返的 ICMP 报文上都有时间戳，因此就可以得出往返时间。而 tracert 是用来跟踪一个分组从源点到终点的路径。过程如下：

- A. 从源地址发出一个 ICMP 请求回显(ICMP Echo Request)数据包到目的地址，并将 TTL 设置为 1；
- B. 到达路由器时，将 TTL 减 1；
- C. 当 TTL 变为 0 时，包被丢弃，路由器向源地址发回一个 ICMP 超时通知

(ICMP Time Exceeded Message)，内含发送 IP 包的源地址，IP 包的所有内容及路由器的 IP 地址；

D. 当源地址收到该 ICMP 包时，显示这一跳路由信息；

E. 重复 1~5，并每次设置 TTL 加 1；

F. 直至目标地址收到探测数据包，并返回 ICMP 回应答复 (ICMPEcho Reply)；

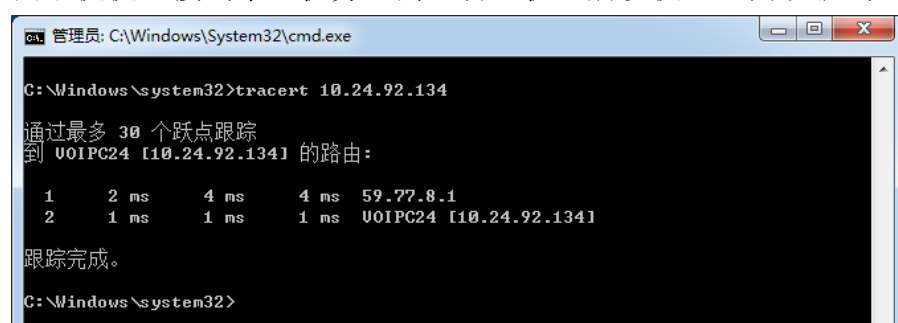
G. 当源地址收到 ICMP Echo Reply 包时停止 tracert。

这样，源主机到达了自己的目的，因为这些路由器和 1 最后目的主机发来的 ICMP 报文正好给出了源主机想知道的路由信息——到达目的主机所经过的路由器的 IP 地址，以及到达其中的每一个路由器的往返时间。

## 2、连接到实验室内部另一台主机

结果如下：

由于仅仅连接到本主机旁边的一台主机，所以仅经过两跳就到达了目的主机。



```
C:\Windows\system32>tracert 10.24.92.134

通过最多 30 个跃点跟踪
到 001PC24 [10.24.92.134] 的路由:

 1    2 ms    4 ms    4 ms  59.77.8.1
 2    1 ms    1 ms    1 ms  001PC24 [10.24.92.134]

跟踪完成。

C:\Windows\system32>
```

Wireshark 捕获结果如下图：

1	0.000000	10.24.90.140	10.24.92.134	ICMP	106 Echo (ping) request id=0x0001, seq=81/20736, ttl=1 (no response found!)
2	0.002894	59.77.8.1	10.24.90.140	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
3	0.003223	10.24.90.140	10.24.92.134	ICMP	106 Echo (ping) request id=0x0001, seq=82/20992, ttl=1 (no response found!)
4	0.007240	59.77.8.1	10.24.90.140	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
5	0.007498	10.24.90.140	10.24.92.134	ICMP	106 Echo (ping) request id=0x0001, seq=83/21248, ttl=1 (no response found!)
6	0.011485	59.77.8.1	10.24.90.140	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
7	5.536068	10.24.90.140	10.24.92.134	ICMP	106 Echo (ping) request id=0x0001, seq=84/21504, ttl=2 (reply in 8)
8	5.537338	10.24.92.134	10.24.90.140	ICMP	106 Echo (ping) reply id=0x0001, seq=84/21504, ttl=127 (request in 7)
9	5.538594	10.24.90.140	10.24.92.134	ICMP	106 Echo (ping) request id=0x0001, seq=85/21760, ttl=2 (reply in 10)
10	5.539862	10.24.92.134	10.24.90.140	ICMP	106 Echo (ping) reply id=0x0001, seq=85/21760, ttl=127 (request in 9)
11	5.540155	10.24.90.140	10.24.92.134	ICMP	106 Echo (ping) request id=0x0001, seq=86/22016, ttl=2 (reply in 12)
12	5.541378	10.24.92.134	10.24.90.140	ICMP	106 Echo (ping) reply id=0x0001, seq=86/22016, ttl=127 (request in 11)

数据包中可以看出，一开始本机发出 3 个 ping 包，源地址 10.24.90.140，目标主机地址为 10.24.92.134，这 3 个 ICMP 包的 TTL 值都被设置为 1。故当这三个包发送到第一个路由时，会被丢弃，同时路由器向源地址发回一个 ICMP 超时通知，这样本机就可以知道第一跳路由的 ip 地址为 59.77.8.1，这里发出的头 3 个 ping 包均收到了 request，所以在命令行窗口中没有返回\*。

分析请求报文：

先解析请求报文的 MAC 头：

开头 6 个字节 88 1d fc 73 75 c1，表示目的地址(MAC 地址)为 88:1d:fc:73:75:c1。之后 6 个字节 00 0c 29 ba d5 b3，表示源地址(MAC 地址)为 00:0c:29:ba:d5:b3。最后两个字节 0x0800，表示上一层使用的是 IP 协议。

下面再对 IP 头进行解析：

开头的 4 位对应的十六进制数为 0x4，表示 IP 协议版本号为 4，即 IPv4。

之后的 4 位对应的十六进制数位 0x5，表示首部长度为 5 个单位，即 5\*4=20 字节。

下面 1 个字节 0x00 表示区分服务，但是这个字段并未使用过。

之后的 2 个字节，为 0x005c，表示数据报的总长度为 92 字节。

往后的 2 个字节 0x2081，表示标识。  
之后的 3 位为标志，其中 MF=0，表示该数据报是若干数据报片中的最后一个；  
DF=0，表示允许分片。

No.	Time	Source	Destination	Protocol	L
1	0.000000	10.24.90.140	10.24.92.134	ICMP	16
2	0.002894	59.77.8.1	10.24.90.140	ICMP	7

> Frame 1: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface

▼ Ethernet II, Src: Vmware\_ba:d5:b3 (00:0c:29:ba:d5:b3), Dst: Cisco\_73:75:c1

- > Destination: Cisco\_73:75:c1 (88:1d:fc:73:75:c1)
- > Source: Vmware\_ba:d5:b3 (00:0c:29:ba:d5:b3)
- Type: IPv4 (0x0800)

▼ Internet Protocol Version 4, Src: 10.24.90.140, Dst: 10.24.92.134

- 0100 .... = Version: 4
- .... 0101 = Header Length: 20 bytes (5)
- > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 92
- Identification: 0x2081 (8321)
- > Flags: 0x0000
- > Time to live: 1
- Protocol: ICMP (1)

Offset	Hex	ASCII
0000	88 1d fc 73 75 c1 00 0c 29 ba d5 b3 08 00 45 00	...su... ).....E.
0010	00 5c 20 81 00 00 01 01 00 00 0a 18 5a 8c 0a 18	.\ ....Z...
0020	5c 86 08 00 f7 ad 00 01 00 51 00 00 00 00 00 00	\.....Q.....
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
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	.....

下一个字节 0x01，表示生存时间 TTL 为 1。  
下一个字节 0x01，表示数据报携带的数据是使用 ICMP 协议的。  
紧接着 2 个字节 0x0000，表示首部校验和为 0。  
下面 4 个字节 0a 18 5a 8c，表示源地址为 10.24.90.140。  
最后 4 个字节 0a 18 5a 86，表示目的地址为 10.24.92.134。  
**最后分析 ICMP 报文格式：**

No.	Time	Source	Destination	Protocol	L
1	0.000000	10.24.90.140	10.24.92.134	ICMP	16
2	0.002894	59.77.8.1	10.24.90.140	ICMP	7

> Source: Vmware\_ba:d5:b3 (00:0c:29:ba:d5:b3)

Type: IPv4 (0x0800)

> Internet Protocol Version 4, Src: 10.24.90.140, Dst: 10.24.92.134

▼ Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0xf7ad [correct]
- [Checksum Status: Good]
- Identifier (BE): 1 (0x0001)
- Identifier (LE): 256 (0x0100)
- Sequence number (BE): 81 (0x0051)
- Sequence number (LE): 20736 (0x5100)
- > [No response seen]
- > Data (64 bytes)

Offset	Hex	ASCII
0000	88 1d fc 73 75 c1 00 0c 29 ba d5 b3 08 00 45 00	...su... ).....E.
0010	00 5c 20 81 00 00 01 01 00 00 0a 18 5a 8c 0a 18	.\ ....Z...
0020	5c 86 08 00 f7 ad 00 01 00 51 00 00 00 00 00 00	\.....Q.....
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
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	.....

开头 1 个字节 0x08，表示 ICMP 报文类型为回送请求。  
之后 1 个字节 0x00 为代码字段，以进一步区分某种类型中出现的几种不同情况。

之后 2 个字节 0xf7ad 为校验和。  
 之后 2 个字节 0x0001 为标识符。  
 接着 2 个字节 0x0051 为序列号。  
 之后的部分均为数据，这里全部为 0。  
 下面我们分析获取的响应报文，如图：

No.	Time	Source	Destination	Protocol
1	0.000000	10.24.90.140	10.24.92.134	ICMP
2	0.002894	59.77.8.1	10.24.90.140	ICMP
3	0.003223	10.24.90.140	10.24.92.134	ICMP
4	0.007240	59.77.8.1	10.24.90.140	ICMP
5	0.007498	10.24.90.140	10.24.92.134	ICMP
6	0.011485	59.77.8.1	10.24.90.140	ICMP

> Frame 2: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on i	
> Ethernet II, Src: Cisco_73:75:c1 (88:1d:fc:73:75:c1), Dst: Vmware_ba:d5	
> Internet Protocol Version 4, Src: 59.77.8.1, Dst: 10.24.90.140	
▼ 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	

0000	00 0c 29 ba d5 b3 88 1d fc 73 75 c1 08 00 45 c0	..). . . . . su . . . E .
0010	00 38 a4 d0 00 00 ff 01 6e 42 3b 4d 08 01 0a 18	.8 . . . . . nB ; M . . . .
0020	5a 8c 0b 00 f4 ff 00 00 00 00 45 00 00 5c 20 81	Z . . . . . . . E . . \ .
0030	00 00 01 01 cd de 0a 18 5a 8c 0a 18 5c 86 08 00	. . . . . Z . . . \ . . .
0040	f7 ad 00 01 00 51	. . . . . Q

### 先解析响应报文的 MAC 头

开头 6 个字节 00 0c 29 ba d5 b3, 表示目的地址(MAC 地址)为 00:0c:29:ba:d5:b3。  
 之后 6 个字节 88 1d fc 73 75 c1, 表示源地址(MAC 地址)为 88:1d:fc:73:75:c1。  
 最后两个字节 0x0800, 表示上一层使用的是 IP 协议。

### 下面再对 IP 头进行解析：

开头的 4 位对应的十六进制数为 0x4, 表示 IP 协议版本号为 4, 即 IPv4。  
 之后的 4 位对应的十六进制数位 0x5, 表示首部长度为 5 个单位, 即 5\*4=20 字节。

下面 1 个字节 0x00 表示区分服务, 但是这个字段并未使用过。

之后的 2 个字节, 为 0x0038, 表示数据报的总长度为 56 字节。

往后的 2 个字节 0xa4d0, 表示标识。

之后的 3 位为标志, 其中 MF=0, 表示该数据报是若干数据报片中的最后一个;  
 DF=0, 表示允许分片。

下一个字节 0xff, 表示生存时间 TTL 为 255。

下一个字节 0x01, 表示数据报携带的数据是使用 ICMP 协议的。

紧接着 2 个字节 0x6e42, 表示首部校验和。

下面 4 个字节 3b 4d 08 01, 表示源地址为 59.77.8.1。

最后 4 个字节 0a 18 5a 8c, 表示目的地址为 10.24.90.140。



### 最后分析 ICMP 报文格式：

开头 1 个字节 0xb0，表示 ICMP 报文差错报告为时间超过 (Time-to-live exceeded)。

之后 1 个字节 0x00 为代码字段，以进一步区分某种类型中出现的几种不同情况。

之后 2 个字节 0xf4ff 为校验和。

之后 2 个字节 0x0000 为标识符。

接着 2 个字节 0x0000 为序列号。

之后的部分为报告错误的 IP 数据报的首部部分。

第二跳，成功到达目标地址，所发送数据帧 TTL=2，所以请求报文源地址为 10.24.90.140，目的地址为 10.24.90.134，响应报文源地址为 10.24.90.134，目的地址为 10.24.90.140，不再做具体分析。

7	5.536068	10.24.90.140	10.24.92.134	ICMP
8	5.537338	10.24.92.134	10.24.90.140	ICMP
9	5.538594	10.24.90.140	10.24.92.134	ICMP
10	5.539862	10.24.92.134	10.24.90.140	ICMP
11	5.540155	10.24.90.140	10.24.92.134	ICMP
12	5.541378	10.24.92.134	10.24.90.140	ICMP

.0..	....	....	....	= Don't fragment: Not set
..0.	....	....	....	= More fragments: Not set
...0	0000	0000	0000	= Fragment offset: 0

> Time to live: 2			
-------------------	--	--	--

0000	88 1d fc 73 75 c1 00 0c	29 ba d5 b3 08 00 45 00	...su... )....E.
0010	00 5c 20 87 00 00 02 01	00 00 0a 18 5a 8c 0a 18	.\ ... . ....Z...
0020	5c 86 08 00 f7 aa 00 01	00 54 00 00 00 00 00 00	\.....T.....
0030	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
0040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
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	.....

### 3、连接到外部网络

本次实验使用的是以太网，链接到 www.baidu.com 共经过 10 跳：

```
C:\WINDOWS\system32>tracert www.baidu.com

通过最多 30 个跃点跟踪
到 www.a.shifen.com [14.215.177.39] 的路由:

 1    3 ms    3 ms    2 ms  1.160.160.220.broad.xm.fj.dynamic.163data.com.cn [220.160.160.1]
 2    3 ms    1 ms    6 ms  218.85.117.41
 3    3 ms    6 ms    2 ms  61.154.238.73
 4    *      *      *      请求超时。
 5   15 ms   15 ms   21 ms  113.96.4.122
 6   14 ms   14 ms   14 ms  98.96.135.219.broad.fs.gd.dynamic.163data.com.cn [219.135.96.98]
 7   20 ms   20 ms   21 ms  14.29.121.198
 8    *      *      *      请求超时。
 9    *      *      *      请求超时。
10   13 ms   13 ms   13 ms  14.215.177.39

跟踪完成。
```

Wireshark 捕获结果如上图。

命令行中，没有收到 request 的 ping 包，在显示延时的地方，使用\*来表示没有收到该 ping 包的 request。



以第四跳为例，如下图所示，在 Wireshark 捕获结果中，这里发出的头 3 个 ping 包都没有收到 request，在命令行中返回“请求超时”。

TTL 值为 n， $n < 10$  时，经过持续不断的寻找路由，故在经过第 n 跳的路由时会被丢弃，同时第二跳的路由向本机发送一个 ICMP 超时通知（ICMP Time Exceeded Message），其中的源地址就是第 n 跳路由的地址，由此可以知道第 n 跳路由的地址。

在第十跳时，收到了目的 IP 地址的返回报告。

13	2.012617	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=9/2304, ttl=1 (no response found!)
14	2.016187	220.160.160.1	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
15	2.016975	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=10/2560, ttl=1 (no response found!)
16	2.019961	220.160.160.1	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
17	2.020336	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=11/2816, ttl=1 (no response found!)
18	2.023063	220.160.160.1	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
17	2.048517	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=12/3072, ttl=2 (no response found!)
48	12.051566	218.85.117.41	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
49	12.054278	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=13/3328, ttl=2 (no response found!)
50	12.056151	218.85.117.41	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
51	12.058070	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=14/3584, ttl=2 (no response found!)
52	12.064275	218.85.117.41	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
81	22.071166	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=15/3840, ttl=3 (no response found!)
82	22.074302	61.154.238.73	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
83	22.075694	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=16/4096, ttl=3 (no response found!)
84	22.082438	61.154.238.73	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
85	22.085785	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=17/4352, ttl=3 (no response found!)
86	22.088196	61.154.238.73	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
109	32.103662	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=18/4608, ttl=4 (no response found!)
114	36.074725	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=19/4864, ttl=4 (no response found!)
120	40.076395	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=20/5120, ttl=4 (no response found!)
135	44.080276	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=21/5376, ttl=5 (no response found!)
136	44.095495	113.96.4.122	220.160.160.122	ICMP	118 Time-to-live exceeded	(Time to live exceeded in transit)
137	44.096457	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=22/5632, ttl=5 (no response found!)
138	44.111589	113.96.4.122	220.160.160.122	ICMP	118 Time-to-live exceeded	(Time to live exceeded in transit)
139	44.114345	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=23/5888, ttl=5 (no response found!)
140	44.135505	113.96.4.122	220.160.160.122	ICMP	118 Time-to-live exceeded	(Time to live exceeded in transit)
160	54.130875	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=24/6144, ttl=6 (no response found!)
161	54.144856	219.135.96.98	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
162	54.147648	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=25/6400, ttl=6 (no response found!)
163	54.161605	219.135.96.98	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
164	54.164392	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=26/6656, ttl=6 (no response found!)
165	54.178779	219.135.96.98	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
183	64.204088	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=27/6912, ttl=7 (no response found!)
184	64.224755	14.29.121.198	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
185	64.227534	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=28/7168, ttl=7 (no response found!)
186	64.248342	14.29.121.198	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
187	64.249447	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=29/7424, ttl=7 (no response found!)
188	64.270527	14.29.121.198	220.160.160.122	ICMP	78 Time-to-live exceeded	(Time to live exceeded in transit)
208	74.266442	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=30/7680, ttl=8 (no response found!)
217	78.077792	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=31/7936, ttl=8 (no response found!)
223	82.079214	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=32/8192, ttl=8 (no response found!)
228	86.079980	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=33/8448, ttl=9 (no response found!)
234	90.074077	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=34/8704, ttl=9 (no response found!)
241	94.073800	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=35/8960, ttl=9 (no response found!)
248	98.077891	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=36/9216, ttl=10 (reply in 249)
249	98.090937	14.215.177.39	220.160.160.122	ICMP	114 Echo (ping) reply	id=0x0001, seq=36/9216, ttl=56 (request in 248)
250	98.093664	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=37/9472, ttl=10 (reply in 251)
251	98.106658	14.215.177.39	220.160.160.122	ICMP	114 Echo (ping) reply	id=0x0001, seq=37/9472, ttl=56 (request in 250)
252	98.107771	220.160.160.122	14.215.177.39	ICMP	114 Echo (ping) request	id=0x0001, seq=38/9728, ttl=10 (reply in 253)
253	98.121014	14.215.177.39	220.160.160.122	ICMP	114 Echo (ping) reply	id=0x0001, seq=38/9728, ttl=56 (request in 252)

## （四）抓取 802.11 数据报并分析

### 1、使用自己的笔记本连接 WLAN，捕获无线数据包

在 macOS 下链接 XMUNET+，使用 Wireshark，打开监控模式，用来捕获 802.11 数据报。

### 2、802.11 帧的各字段

802.11 帧共有三种类型，即控制帧、数据帧和管理帧。802.11 局域网的数据帧和三种控制帧的主要字段，可以进一步了解 802.11 局域网的 MAC 帧的特点。

802.11 数据帧由以下三大部分组成：

（1）MAC 首部，共 30 字节。帧的复杂性都在帧的 MAC 首部。

（2）帧主体，也就是帧的数据部分，不超过 2312 字节。这个数值比以太网的最大长度长很多。不过 802.11 帧的长度通常都小于 1500 字节。

（3）帧检验序列 FCS 是 MAC 尾部，共 4 字节。

802.11 数据帧有四个地址字段。前三个地址的内容取决于帧控制字段中的“去往 AP”（发送到接入点）和“来自 AP”（从接入点发出）这两个子字段的数值。这两个子字段各占 1 位，合起来共有 4 种组合，用于定义 802.11 帧中的几个地址字段的含义。当然，这些地址都是 MAC 地址（在数据链路层不可能使用 IP 地址）。

下表给出的是 802.11 帧的地址字段最常用的两种情况（在有基础设施的网络中只使用前三种地址，而不使用仅在自组移动网络中使用的地址 4）。

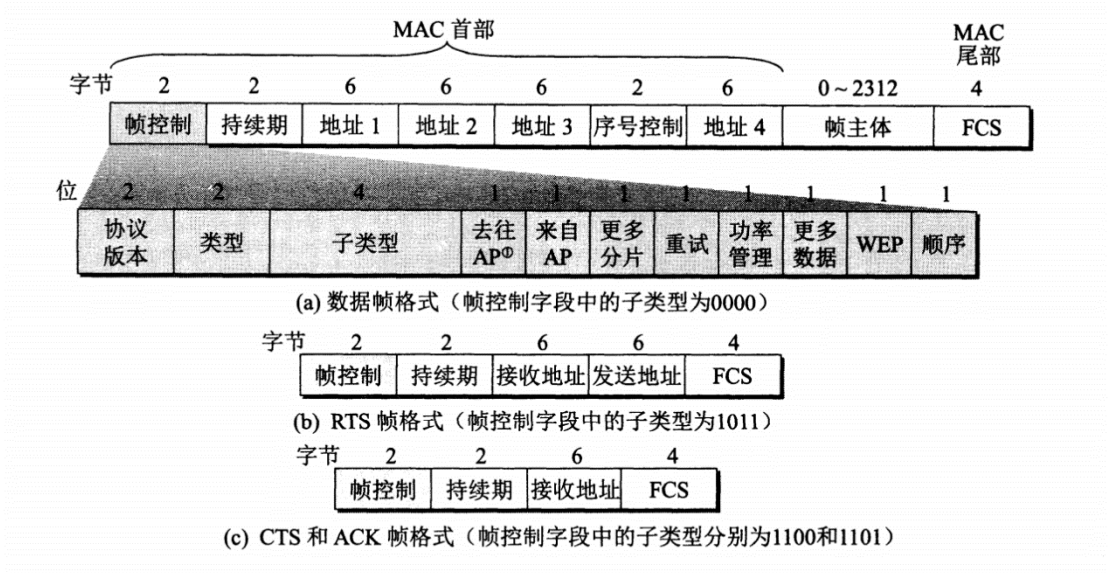


图 9-10 802.11 局域网的帧格式

去往 AP	来自 AP	地址 1	地址 2	地址 3	地址 4
0	1	目的地址	AP 地址	源地址	——
1	0	AP 地址	源地址	目的地址	——

表：802.11 帧的地址字段最常用的两种情况

序号控制字段占 16 位，作用是使接收方能够区分是新传送的帧还是因出现差错而重传的帧。这和运输层讨论的序号的概念是相似的。

持续期字段占 16 位。只有最高位为 0 时才表示持续期。这样，持续期不能超过  $2^{15}-1=32767$ ，单位是微秒。

帧控制字段共分为 11 个子字段。

协议版本字段现在是 0。

类型字段和子类型字段用来区分帧的功能。802.11 帧共有三种类型：控制帧、数据帧和管理帧，而每一种帧又分为若干种子类型。

更多分片字段置为 1 时表明这个帧属于一个帧的多个分片之一。为了提高传输效率，在信道质量较差时，需要把一个较长的帧划分为许多较短的分片。这时可以在一次使用 RTS 和 CTS 帧预约信道后连续发送这些分片。当然这仍然要使用停止等待协议，即发送一个分片，等到收到确认后再发送下一个分片，不过后面的分片都不需要用 RTS 和 CTS 重新预约信道。

### 3、观察 Wireshark 捕获的无线数据帧，过滤出 802.11 的各种帧，重点说明这些帧的作用

(1) 使用 wlan.fc.type\_subtype == 0x08 筛选出 Beacon 帧，选择一条进行分

析。  
帧首部:

```
> Frame 1: 210 bytes on wire (1680 bits), 210 bytes captured (1680 bits) on interface 0
> Radiotap Header v0, Length 25
> 802.11 radio information
  IEEE 802.11 Beacon frame, Flags: .....C
    Type/Subtype: Beacon frame (0x0008)
    > Frame Control Field: 0x8000
      .000 0000 0000 0000 = Duration: 0 microseconds
      Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
      Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
      Transmitter address: ArubaAHe_ad:37:91 (18:64:72:ad:37:91)
      Source address: ArubaAHe_ad:37:91 (18:64:72:ad:37:91)
      BSS Id: ArubaAHe_ad:37:91 (18:64:72:ad:37:91)
      .... .... 0000 = Fragment number: 0
      0010 1011 0011 .... = Sequence number: 691
      Frame check sequence: 0xb9ca4215 [unverified]
      [FCS Status: Unverified]
    > IEEE 802.11 wireless LAN
```

0000	00 00 19 00 6f 08 00 00 e6 96 2e 51 00 00 00 00	....o... ..Q....
0010	12 18 8c 14 40 01 ac a1 01 80 00 00 00 ff ff ff	...@... .....
0020	ff ff ff 18 64 72 ad 37 91 18 64 72 ad 37 91 30	....dr.7 ..dr.7.0
0030	2b 3b e0 cc 20 00 00 00 00 64 00 11 04 00 07 58	+;... ..d....X
0040	4d 55 4e 45 54 2b 01 06 98 24 b0 48 60 6c 03 01	MUNET+... \$.H`1..
0050	34 05 04 00 01 00 00 30 14 01 00 00 0f ac 04 01	4.....0 .....
0060	00 00 0f ac 04 01 00 00 0f ac 01 28 00 2d 1a ef	..... (....
0070	01 1b ff ff ff 00 00 00 00 00 00 00 00 00 00	.....
0080	00 00 00 00 00 00 00 00 00 3d 16 34 05 08 00 00	..... =.4....
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00a0	00 7f 08 00 00 08 00 00 00 00 00 dd 18 00 50 f2	..... ..P..
00b0	02 01 01 80 00 03 a4 00 00 27 a4 00 00 42 43 5e	..... '...BC^
00c0	00 62 32 2f 00 dd 07 00 0b 86 01 04 08 12 15 42	..b2/.... ....B
00d0	ca b9	..

第 1 字节 0x80: 前 4 个 bit 为 1000, 用以显示该帧 subtype 为 1000, 表示为 Beacon 帧。接着 2bit 为 00, 表示该帧帧类型为管理帧。最后 2bit 表示所使用的 MAC 版本: 0。

第 2 字节 0x00: 在这 16bit 中, 最末 2bit 为 00, 表示 To DS =0, From DS =0, 所有管理与控制帧 IBSS (非基础型数据帧)。第 6bit 为 More fragments bit, 表示是否为分段帧。若较上层的封包经过 MAC 分段处理, 最后一个片段除外, 其他片段均会将此 bit 设定为 1。本例为 0, 说明这是最后一个片段 (或只有一个片段)。第 5bit 为 Retrybit (重传帧)。任何重传的帧会将此 bit 设定为 1, 以协助接收端剔除重复的帧。这里为 0, 为非重传帧。第 4bit 为 Powermanagement bit, 此 bit 用来指出传送端在完成目前的基本帧交换之后是否进入省电模式。1 代表工作站即将进入省电模式, 而 0 则代表工作站会一直保持在清醒状态。基站必须行使一系列重要的管理功能, 所以不允许进入省电模式, 因此基站所传送的帧中, 此 bit 必然为 0。第 3bit 为 Moredata bit, 表示为了服务处于省电模式的工作站, 基站会将这些由“传输系统”接收而来的帧加以暂存。基站如果设定此 bit, 即代表至少有一个帧待传给休眠中的工作站。第 2bit 为 ProtectedFrame bit, 相对于有线网络, 无线传输本质上就比较容易遭受拦截。如果帧受到链路层安全协议的保护, 此 bit 会被设定为 1, 而且该帧会略有不同。第 1 位为 Orderbit 帧与帧片段可依序传送, 不过发送端与接收端的 MAC 必须付出额外的代价。一旦进行“严格依序”传送, 此 bit 被设定为 1。

第 3-4 字节 0x00, duration 位, 用来预定一段介质使用时间。

第 5-10 字节 0xff ff ff ff ff ff, 表示该帧为广播。

第 11-16 字节 0x18 64 72 ad 37 91, 表示源地址。

第 17-22 字节 0x18 64 72 ad 37 91, 表示发射机地址。

第 23-24 字节 0x30 2c, 为顺序控制位, 此位的长度为 16 个 bit, 用来重组帧片段以及丢弃重复帧。它由 4 个 bit 的 fragment number (片段编号) 位以及 12 个 bit 的 sequencenumber (顺序编号) 位所组成。控制帧未使用顺序编号, 因此并无 sequence control 位。

接下去为 Frame Body (帧主体), 亦称为数据位, 负责在工作站间传送上层数据 (payload)。

最后为帧检验序列 (FCS)。和以太网一样, 802.11 帧也是以帧检验序列作为结束。在以太网上, 如果帧的 FCS 有误, 则随即予以丢弃, 否则就会传送给上层协议处理。在 802.11 网络上, 通过完整性检验的帧还需接收端送出应答。例如, 接收无误的数据帧必须得到正面应答, 否则就必须重传。对于未能通过 FCS 检验的帧, 802.11 并未提供负面应答机制; 在重传之前, 工作站就必须等候应答超时。

(2) 用 wlan.fc.type\_subtype == 0x1B 和 wlan.fc.type\_subtype == 0x1C 过滤出 RTS/CTS, 并分析。

RTS/CTS : 可选的 RTS/CTS (request-to-send 和 clear-to-send) 功能为 AP 提供了对媒介的控制。对于大部分的 NIC, 用户在激活 RTS/CTS 的时候可以设置一个最大的帧长门限值。RTS/CTS 功能缓解了隐藏终端问题: 两个或者更多的 NIC 在关联到同一个 AP 的时候不能够互相知道对方。如果 NIC 激活了 RTS/CTS, 在发送数据帧之前首先发送一个 RTS 帧, AP 将响应一个 CTS 帧, 表示可以发送帧数据。在 CTS 帧的 duration 域中, 包含了站点可以发送数据帧的时间。AP 将会保留此时间直到站点发送数据完毕。这种方法避免了隐藏终端之间的冲突。对于每个帧, RTS/CTS 的握手过程将继续, 直到帧长超过对应的门限值。

5 0.045813	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
20 0.337814	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
24 0.341216	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
29 0.345564	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
46 0.555013	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
49 0.555352	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
70 1.050420	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
139 2.633599	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
142 2.633834	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
165 2.822332	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
168 2.834412	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
225 4.061440	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
265 4.882756	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
268 4.883175	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
288 5.159614	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C
290 5.159938	Apple_6c:cb:e1 (78:... 802.11	39 Clear-to-send, Flags=.....C

选择一个 CTS 帧并分析帧首部:

```
> Frame 5: 39 bytes on wire (312 bits), 39 bytes captured (312 bits) on interface 0
> Radiotap Header v0, Length 25
> 802.11 radio information
▼ IEEE 802.11 Clear-to-send, Flags: .....C
  Type/Subtype: Clear-to-send (0x001c)
  ▼ Frame Control Field: 0xc400
    .... ..00 = Version: 0
    .... 01.. = Type: Control frame (1)
    1100 .... = Subtype: 12
    ▼ Flags: 0x00
      .... ..00 = DS status: Not leaving DS or network is operating in AD-HOC mode (To DS: 0 From DS: 0) (0x0)
      .... .0.. = More Fragments: This is the last fragment
0000 00 00 19 00 6f 08 00 00 4c 4a 2f 51 00 00 00 00  ....o... LJ/Q...
0010 12 30 8c 14 40 01 d1 a1 01 c4 00 6e 00 78 4f 43  .0..@... ..n.xOC
0020 6c cb e1 ec 2d 02 5d          l...-...]
```



第1字节 0xc4: 前4个bit为1100, 用以显示该帧 subtype 为1100, 表示为 Deauthentication (解除认证) 帧。接着2bit为01, 表示该帧类型为控制帧。最后2bit表示所使用的MAC版本: 0。

第2字节 0x00, 与上类似, 不再赘述。

第3-4字节 0x6e00, duration 位, 设定为110 微秒。

第5-10字节 0x78 4f 43 6c cb e1, 表示接受方地址为78:4f:43:6c:cb:e1。

最末4字节为FCS 校验和, 为0xec 2d 02 5d。

### (3) 用 wlan.fc.type == 2 过滤出数据帧

23 0.338681	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	247 QoS Data, SN=142, FN=0, Flags=.p...F.C
28 0.344178	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	251 QoS Data, SN=143, FN=0, Flags=.p...F.C
43 0.554064	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=144, FN=0, Flags=.pm...F.C
44 0.554164	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=145, FN=0, Flags=.pm...F.C
45 0.554216	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=146, FN=0, Flags=.p...F.C
163 2.804879	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	206 QoS Data, SN=29, FN=0, Flags=.p...F.C
164 2.821432	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=147, FN=0, Flags=.p...F.C
167 2.832629	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=148, FN=0, Flags=.p...F.C
174 2.914973	XiaomiCo_8c:41:c1	ArubaAHe_ad:3b:f1	802.11	55 QoS Null function (No data), SN=131, FN=0, Flags=.....TC
176 2.916212	NewH3CTe_68:82:01	XiaomiCo_8c:41:c1	802.11	206 QoS Data, SN=50, FN=0, Flags=.p...F.C
179 2.966611	XiaomiCo_8c:41:c1	ArubaAHe_ad:3b:f1	802.11	55 QoS Null function (No data), SN=132, FN=0, Flags=...P...TC
285 5.158422	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=149, FN=0, Flags=.pm...F.C
286 5.158532	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=150, FN=0, Flags=.pm...F.C
287 5.158636	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	148 QoS Data, SN=151, FN=0, Flags=.p...F.C
325 5.986927	NewH3CTe_68:82:01	Apple_6c:cb:e1	802.11	206 QoS Data, SN=30, FN=0, Flags=.p...F.C
326 5.989758	XiaomiCo_8c:41:c1	ArubaAHe_ad:3b:f1	802.11	55 QoS Null function (No data), SN=133, FN=0, Flags=.....TC
328 5.990646	NewH3CTe_68:82:01	XiaomiCo_8c:41:c1	802.11	206 QoS Data, SN=51, FN=0, Flags=.p...F.C

选择一个 QoS 帧分析:

> Frame 23: 247 bytes on wire (1976 bits), 247 bytes captured (1976 bits) on interface 0

> Radiotap Header v0, Length 48

> 802.11 radio information

> IEEE 802.11 QoS Data, Flags: .p...F.C

Type/Subtype: QoS Data (0x0028)

> Frame Control Field: 0x8842

.... ..00 = Version: 0

.... 10.. = Type: Data frame (2)

1000 .... = Subtype: 8

> Flags: 0x42

.... ..10 = DS status: Frame from DS to a STA via AP(To DS: 0 From DS: 1) (0x2)

.... .0.. = More Fragments: This is the last fragment

.... 0... = Retry: Frame is not being retransmitted

...0 .... = PWR MGT: STA will stay up

..0. .... = More Data: No data buffered

0000 00 00 30 00 6b 08 1c 00 0f c2 33 51 00 00 00 00 ..0.k... ..3Q...

0010 14 00 8c 14 40 01 d0 a1 01 00 00 00 40 01 02 00 ...@... ..@...

0020 8c 14 34 22 1f 15 17 00 8e 00 00 00 00 00 00 00 ..4".....

0030 88 42 2c 00 78 4f 43 6c cb e1 18 64 72 ad 3b f1 .B,xOCl...dr.;.

0040 04 40 a9 68 82 01 e0 08 00 00 a7 00 00 20 00 00 .@.h..... ..

0050 00 00 aa aa 03 00 00 00 08 00 45 00 00 91 45 04 ..E...E.

0060 00 00 3c 11 51 69 d2 22 00 12 0a 1e 0b 9d 00 35 ..<.Qi." .....5

0070 e6 f0 00 7d 09 bd 51 80 81 80 00 01 00 02 00 00 ...}.Q.....

0080 00 00 0f 63 6f 6d 6d 6e 61 74 2d 6d 61 69 6e 2d ..commn at-main-

0090 67 63 03 65 73 73 05 61 70 70 6c 65 03 63 6f 6d gc-ess-a pple.com

00a0 00 00 01 00 01 c0 0c 00 05 00 01 00 01 51 4d 00 .....QM.

00b0 2a 0f 63 6f 6d 6d 6e 61 74 2d 6d 61 69 6e 2d 67 \*.commna t-main-g

00c0 63 09 65 73 73 2d 61 70 70 6c 65 03 63 6f 6d 06 c-ess-ap ple.com

00d0 61 6b 61 64 6e 73 03 6e 65 74 00 c0 3b 00 01 00 akadns-n et...;

00e0 01 00 00 00 09 00 04 11 b2 68 b6 05 37 37 b2 37 .....h..77.7

00f0 86 67 4c 5b 72 7b 04 .gL[r{.

与上面类似,

第1字节 0x88: 前4个bit为1000, 用以显示该帧 subtype 为1000, 表示为 QoS Data 帧。接着2bit为10, 表示该帧类型为数据帧。最后2bit表示所使用的MAC版本: 0。

第2字节 0x42: 在这16bit中, 最末2bit为00, 表示 To DS =0, From DS =1, 表示基础网络里无线工作站所收到的数据帧。第2bit为1, 表示该帧被保护。

第 3-4 字节 0x2c00, duration 位, 设定为 44 微秒。

接下来是该帧的三个地址段:

第 5-10 字节 0x78 4f 43 6c cb e1, 表示接受方地址为 78:4f:43:6c:cb:e1。

第 11-16 字节 0x18 64 72 ad 3b e1, 表示发射机地址 18:64:72:ad:3b:e1。

第 17-22 字节 0x04 40 a9 68 82 01, 表示源地址 04:40:a9:68:82:01。

第 23-24 字节 0xe0 08, 为顺序控制位, 表示分片及片序号。

接下来是报文数据部分。

最后 4 字节为 FCS 作为结尾。

#### 4、我们可以总结出连接 WLAN 的过程:

(1) AP 发送 Beacon 广播管理帧

因为 AP 发送的这个 Beacon 管理帧数据包是广播地址, 所以我们的 PC/MIA 内置网卡、或者 USB 外界网卡会接收到这个数据包, 然后在我们的“无线连接列表”中显示出来

(2) 客户端向承载指定 SSID 的 AP 发送 Probe Request (探测请求) 帧

当我们点击“连接”的时候, 无线网卡就会发送一个 Prob 数据帧, 用来向 AP 请求连接

(3) AP 接入点对客户端的 SSID 连接请求进行应答

AP 对客户端的连接作出了回应, 并表示不接受任何形式的“帧有效负载加密 (frame-payload-encryption)”

(4) 客户端对目标 AP 请求进行身份认证 (Authentication)

(5) AP 对客户端的身份认证 (Authentication) 请求作出回应

AP 回应, 表示接收身份认证

(6) 客户端向 AP 发送连接 (Association) 请求

身份认证通过之后, 所有的准备工作都做完了, 客户端这个时候可以向 WLAN AP 发起正式的连接请求, 请求接入 WLAN

(7) AP 对连接 (Association) 请求进行回应

AP 对客户端的连接请求 (Association) 予以了回应 (包括 SSID、性能、加密设置等)。至此, Wi-Fi 的连接身份认证交互就全部结束了, 之后就可以正常进行数据发送了

(8) 客户端向 AP 请求断开连接 (Disassociation)

## 六、实验心得

通过本次实验, 我通过使用 Wireshark 抓包分析, 熟悉了 MAC 帧的结构、IP 数据报的结构、ARP 报文格式、ICMP 报文格式, 了解了 ping 和 tracert 的原理和使用方式, 将书本理论知识同工程实际联系了起来, 对巩固理论知识有非常大的帮助。在编程实现 IP 头的过程中, 我熟悉了位操作, 更加清楚信息在 IP 头中的存储方式。

(1) 在编程中遇到了这样几个问题并得到解决:

1、由于数据在内存中的存储满足“高位高字节、低位低字节”, 所以在存储一个字时, C 语言自动将高 16 位存储在高字节, 而低 16 位存储在低字节, 这样就和 Wireshark 抓取的数据形式不同了。因此需要将相邻两个字节进行对调。

2、由于书上 IP 头的字段分布是按照从低位向高位摆放, 而内存中一个字节的位



序是从第 7 位至第 0 位，这和书上的恰好相反。因此在位序上，需要特别注意结构体中各字段的摆放顺序，且不可随意调动，否则就不能正确存储信息了。

(2) 未解决问题：

在抓取 802.11 数据帧时，发现在每一种数据帧首部之前，还有许多字节的“radiotap header”，长度不等但是格式基本固定。我查阅资料，但是并没有得到较好的解释。希望在以后的学习中加深了解。

总之，通过这次实验，对于数据链路层和网络层的了解更加深入了。

## 参考资料

Checksum 算法

<https://blog.csdn.net/zjli321/article/details/74908451/>

802.11 帧格式解析

<https://blog.csdn.net/dxpqxb/article/details/79665569>

[802.11] IEEE 802.11 的帧格式介绍

<https://blog.csdn.net/u012503786/article/details/78783874>

802.11 帧数据详细讲解

[https://blog.csdn.net/Small\\_dong\\_/article/details/50365382](https://blog.csdn.net/Small_dong_/article/details/50365382)

802.11 协议中的一些帧的理解

<https://blog.csdn.net/zhaomininternational/article/details/51541160>