Tina Linux Wi-Fi 抓包
使用指南

版本号: 1.3
发布日期: 2024.12.03

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.05.20	AWA1381	初始版本。
1.1	2023.06.20	AWA1381	第六章: 添加 TCP 和 HTTPS 流程分析。
1.2	2024.06.20	KPA0568	增加 buildroot 打开 tcpdump。
1.3	2024.12.03	AWA2255	1. 完善 tcpdump 命令使用。2. 完善 Wireshark 过滤语法说明。3. 修正部分标点符号使用。

目　录

插　图

1 前言

1.1 文档简介

本文档主要介绍无线网络常用的抓包工具，将详细介绍抓包的方法，以及相关抓包文件的分析。

1.2 目标读者

所有对无线网络感兴趣的朋友。

1.3 适用范围

Tina Linux

1.4 术语列表

表 1-1: WLAN 模块的规格特性

类型	说明
Omnipeek	无线抓包工具
Wireshark	无线抓包工具
tcpdump	无线抓包工具
ARP	地址解析协议
DNS	域名解析协议
HTTPS	超文本传输协议 + 安全协议 (TLS)

2 抓包概念基础介绍

2.1 抓包概述

在应用的开发调试中，查看软件实际运行时网络通信的请求数据和返回数据，从而分析问题的过程就叫做抓包 (packet capture)，也叫数据包嗅探或者协议分析。

抓包 (packet capture) 就是将网络传输发送与接收的数据包进行截获、重发、编辑、转存等操作，也用来检查网络安全，进行数据截取等。空口包指的是空中的无线包。

2.2 抓包原理

抓包网卡在 sniffer 模式下，可以捕获所有的空口包，无过滤的接收周围的所有数据流。

无线网络信号在传播过程中是以发射点为中心，像波纹一样往外辐射。理论上来讲，一个接收器处于无线信号经过的地方，就可以收到 (听到) 任何经过它的信号，只是它可能 “听不懂”(无法解析报文内容)。

2.3 抓包作用

1) 通过抓包可以查看隐藏字段，了解网络特征，进行功能测试。

2) 通过抓包工具了解协议内容，查看通信主体，方便开展接口和性能测试。

3) 通过抓包定位网络协议问题，查找不安全和滥用网络的应用，检查数据加密。

4) 通过抓包对数据进行截取，修改，伪造，重发，也可以识别攻击和恶意活动。

5) 通过抓包分析，可以更好的学习网络协议，理解整个网络系统。

2.4 抓包工具

表 2-1: 抓包工具

工具	跨平台	免费	易用	支持 HTTPS	界面操作
Omnipeek	是	否	是	否	是
Wireshark	是	是	是	否	是
Httpwatch	是	是	是	否	是
Fiddler	是	是	是	是	是
Firebug	是	是	是	否	是
Charles	否 (mac)	否	是	是	是
tcpdump	是	是	是	否	否

3 Omnipeek 抓包工具使用

3.1 下载和安装

官网：

https://www.liveaction.com/products/omnipeek/

3.2 抓包流程介绍

1) 以管理员身份打开 Omnipeek 软件，New capture。

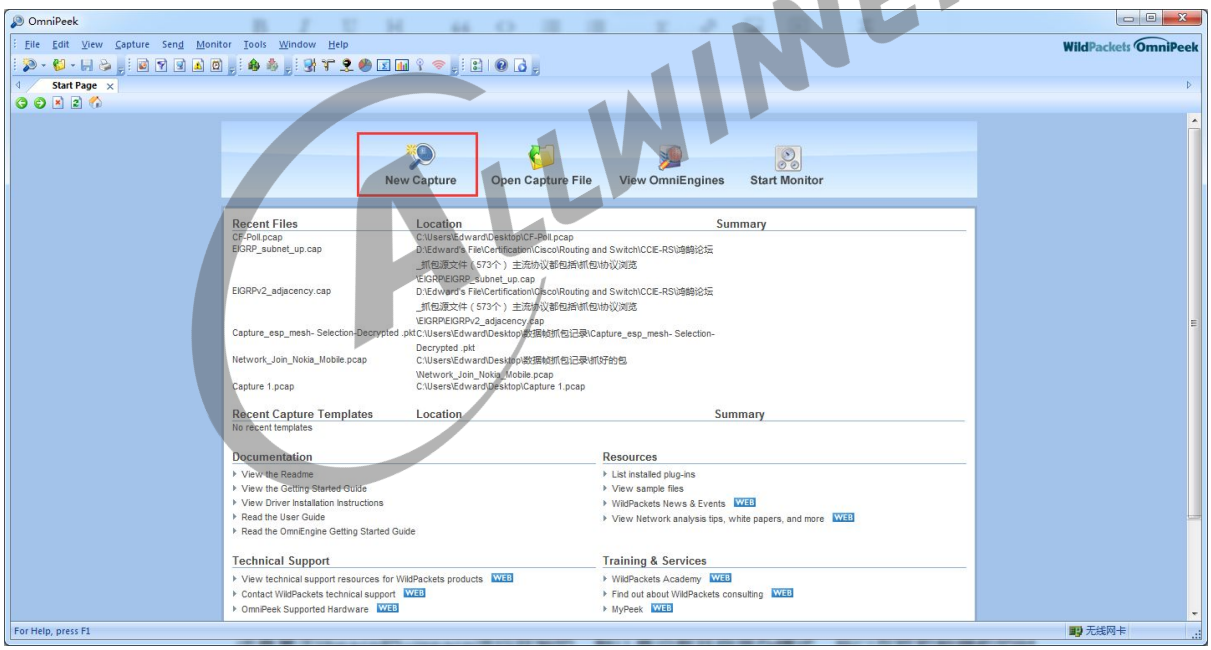


图 3-1: Omnipeek 打开界面

2) 配置打开的 capture，文件名，保存路径等。

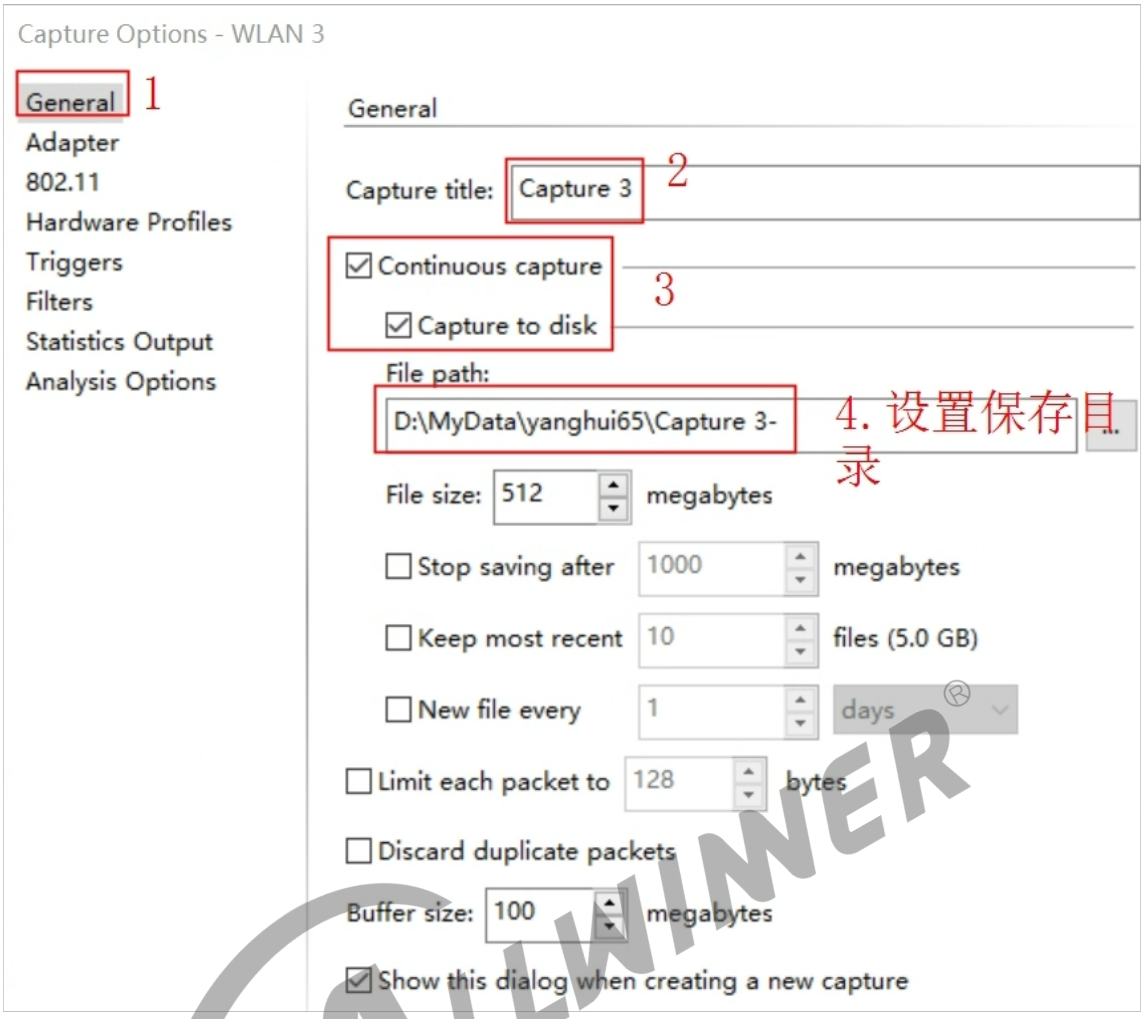


图 3-2: Omnipeek 配置界面

3) 设置使用的抓包网卡。

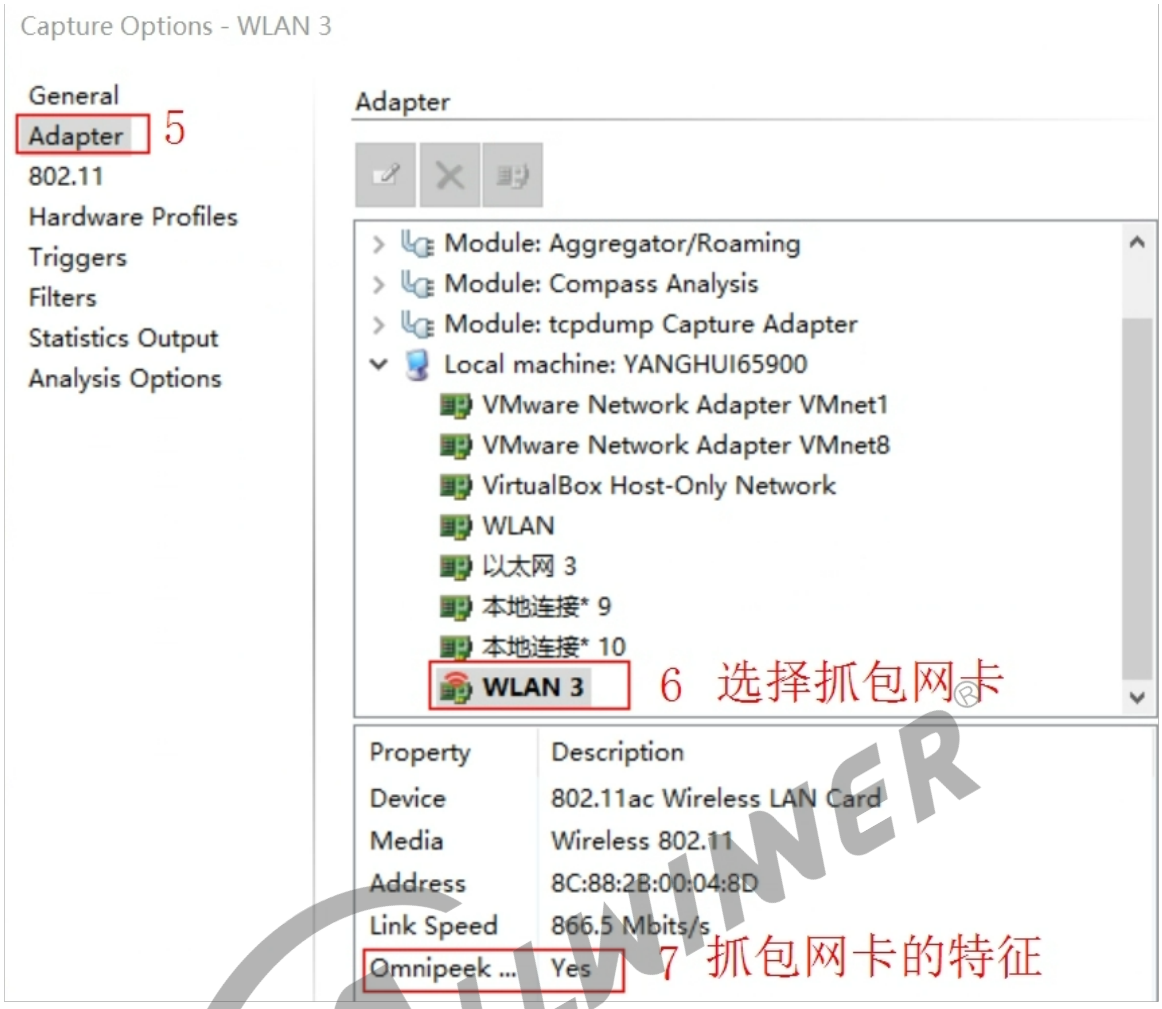


图 3-3: Omnipeek 网卡选择界面

4) 设置抓包信道以及解密无线包。

抓包信道需要与路由器信道一致，才能抓到包。(一般抓 2.4G 的设备使用的模式为 bgn) 解密秘钥正确，才能解密带密码的无线网络包。

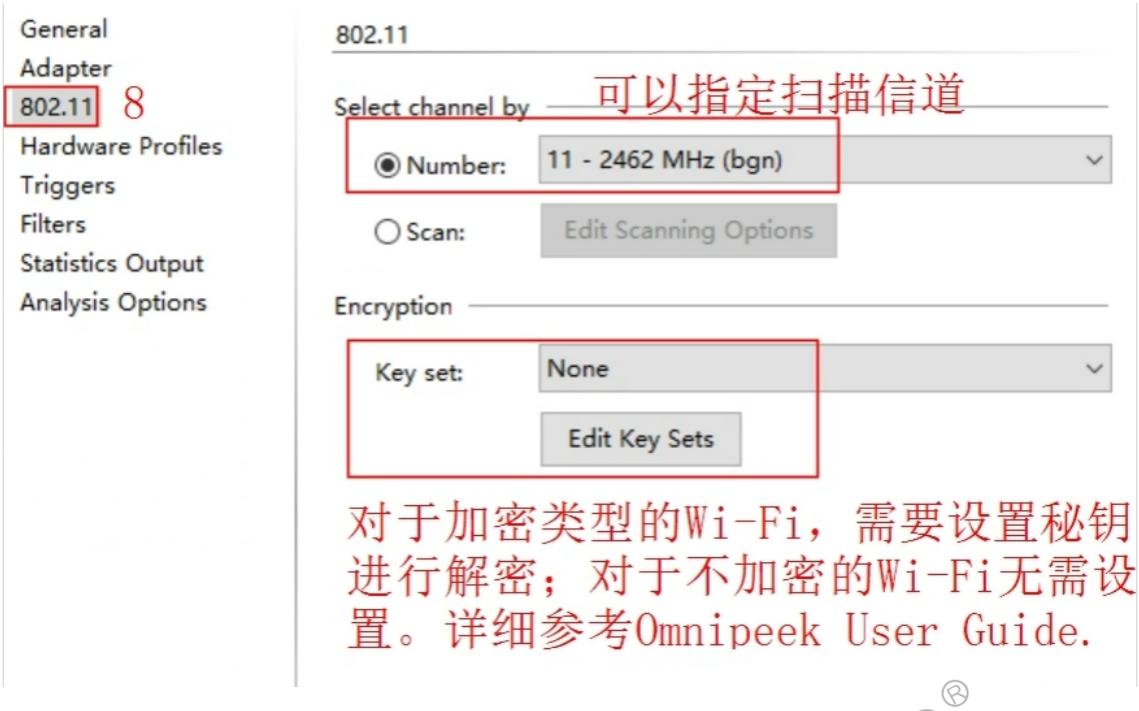


图 3-4: Omnipeek 信道设置界面

5) 解密包设置

特别注意：对于 Encryption 选项，如果路由器设置成加密类型，则 Omnipeek 需要设置正确的秘钥，才能解密包。

点击步骤 4 中

Edit Key Sets —— Insert

说明

Omnipeek 不支持中文 SSID 解密，故采用其他方式。

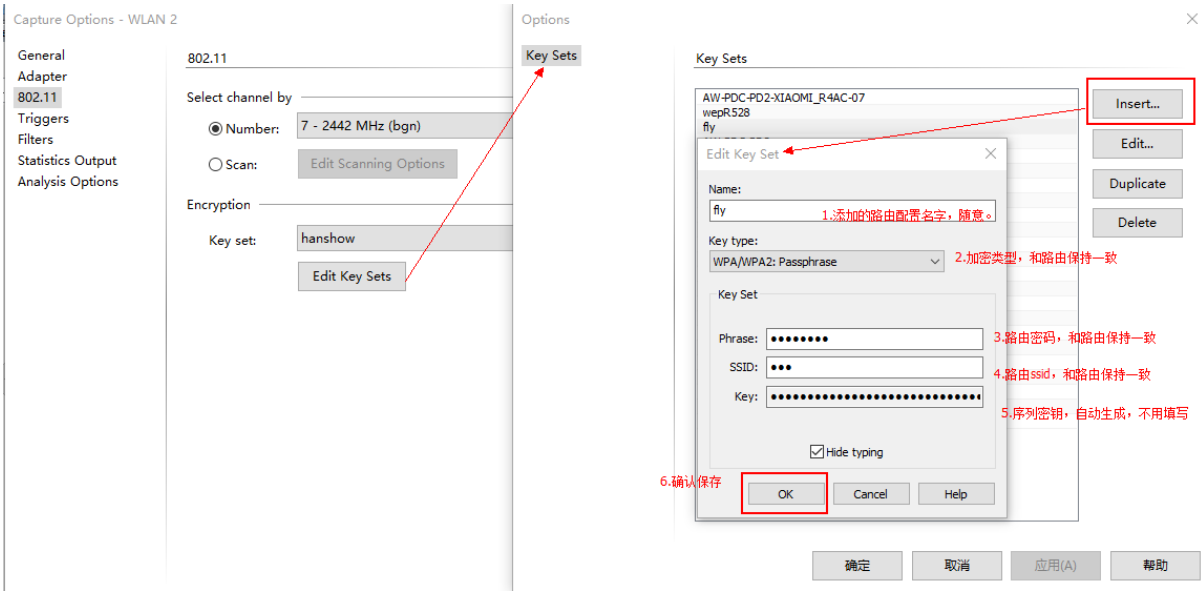


图 3-5: Omnipeek 密钥设置界面

说明

解析加密报文，需在前期抓取 WiFi 的连接过程，但受环境干扰或者工具的限制，有时候不一定能恰好抓到 EAPOL 帧四次握手，成功解析出密钥。可能需要重复启动多次抓取连接过程才能解密，或者开始抓包后过一会再发起连接，可以提高解析成功的概率。正常解密时可以看到最后的抓包 DHCP 的过程。且之后进行网络测试，会成功解析出 ICMP，HTTP 等协议字段，而不只是显示 Encrypte。

6) 开始抓包

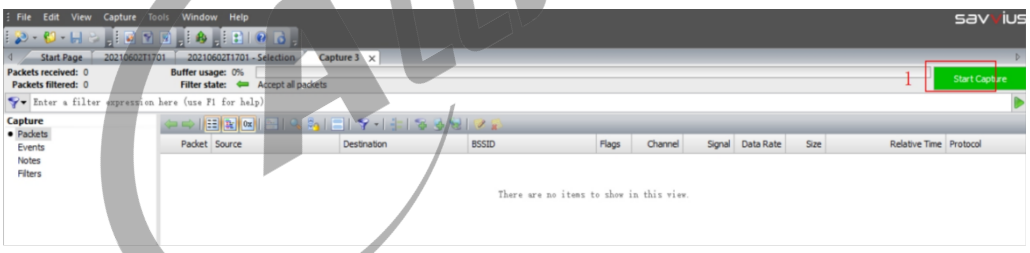


图 3-6: Omnipeek 抓包按钮

7) 抓包中

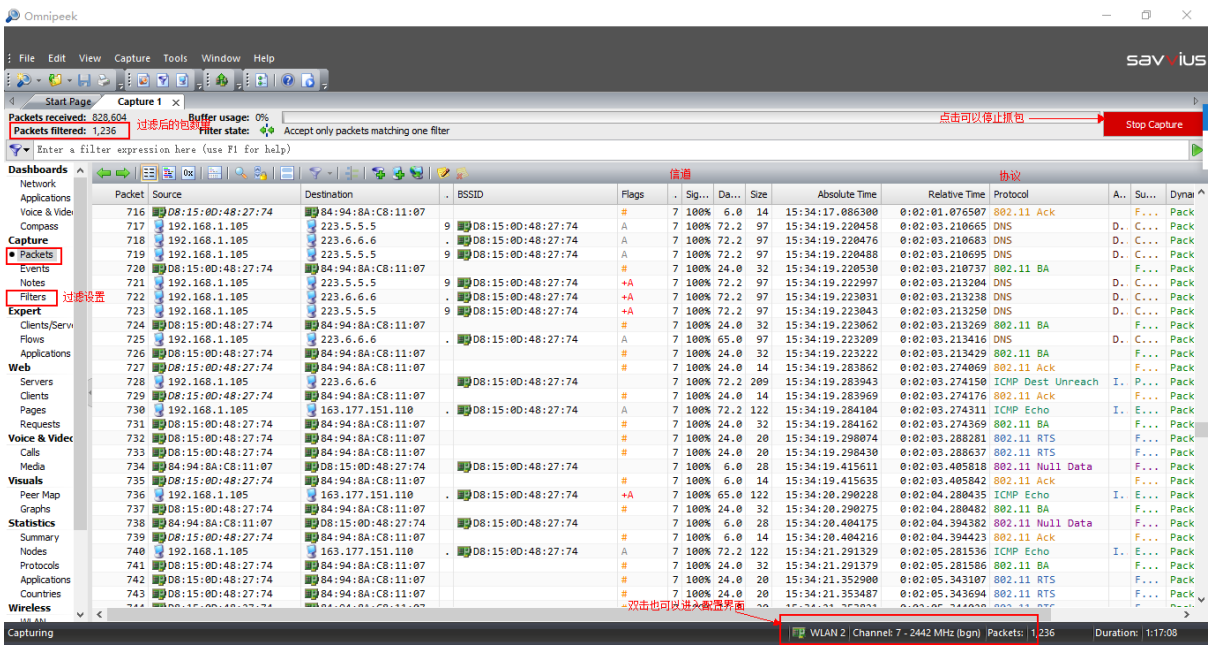


图 3-7: Omnipeek 抓包界面

8) 使用过滤器

筛选特定模组与特定路由器之间的包

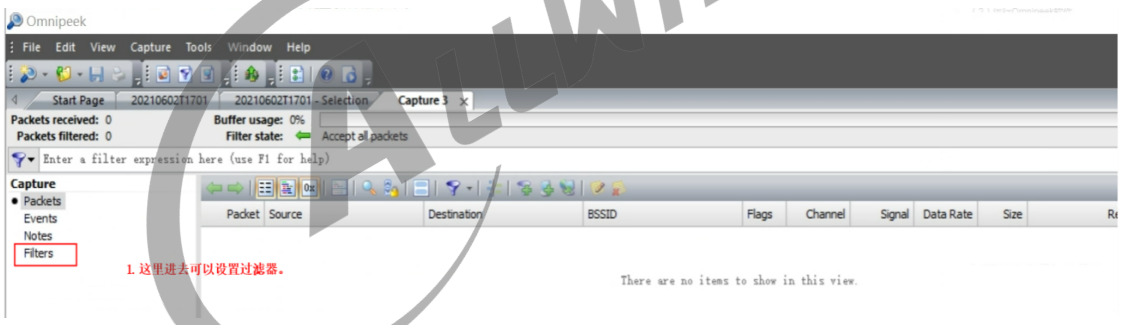


图 3-8: Omnipeek 使用过滤器

简单过滤器可以通过地址、协议和端口三种方式配置过滤条件。

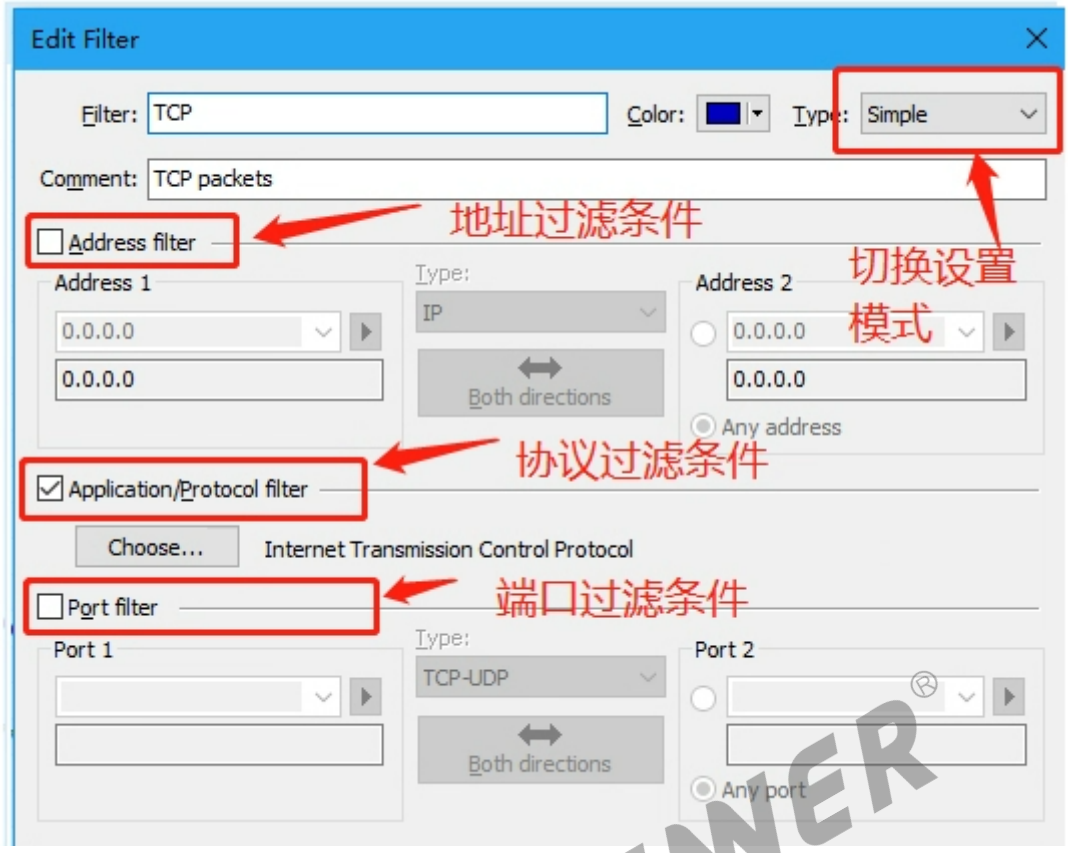


图 3-9: Omnipeek 简单模式过滤设置界面

也可以切换到高级设置界面，如图 3-10 在高级模式下可以对已有的过滤条件进行操作，增加、删除过滤条件，可以设置多个条件 (串行、并行)。

这样更助于数据分析。可以在 Filter 界面进行插入、删除等操作。点击 Insert 图标，进入过滤器创建界面。通过简单模式设置数据包的 MAC、端口号、通信协议来过滤，基本可以达到要求。

根据特定数据包来创建过滤器：

选择特定数据包，在数据包报文解析中找到作为过滤调节的条目，右键选择 Make Filter，会自动进入过滤器创建界面。后面步骤同上。

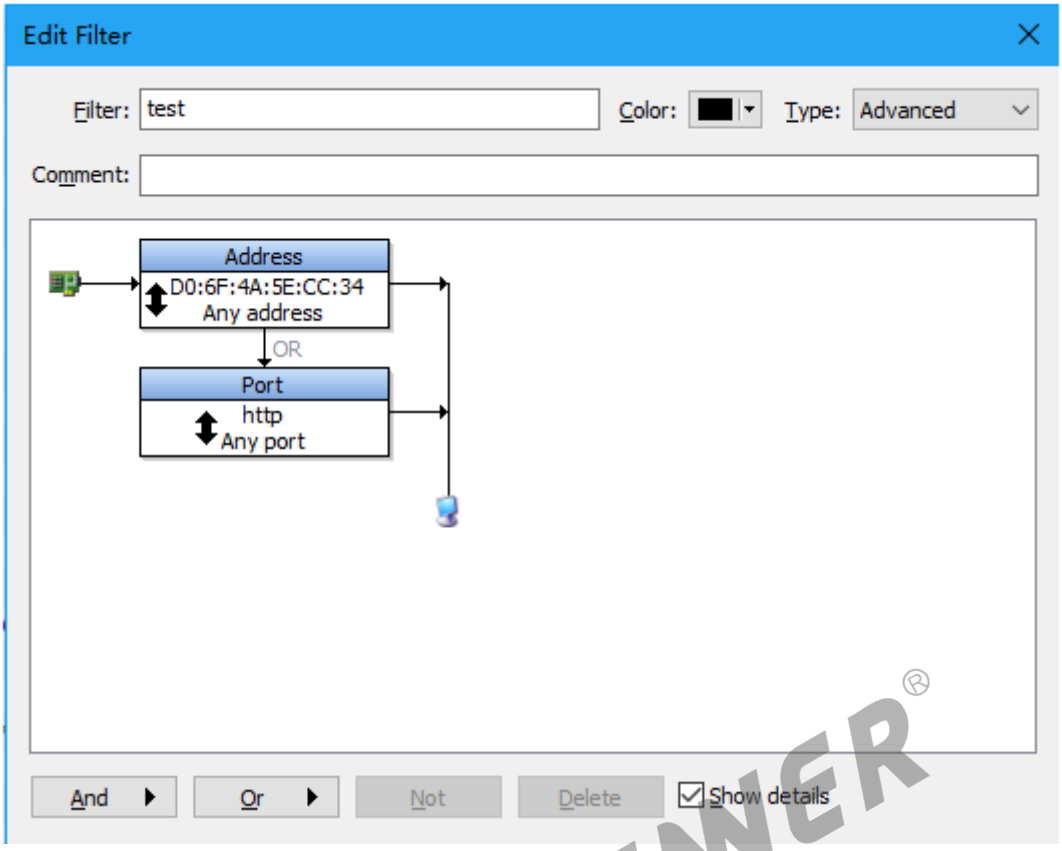


图 3-10: Omnipeek 高级过滤设置界面

下面介绍几中常用的过滤设置：

(1) 地址过滤——mac 地址过滤

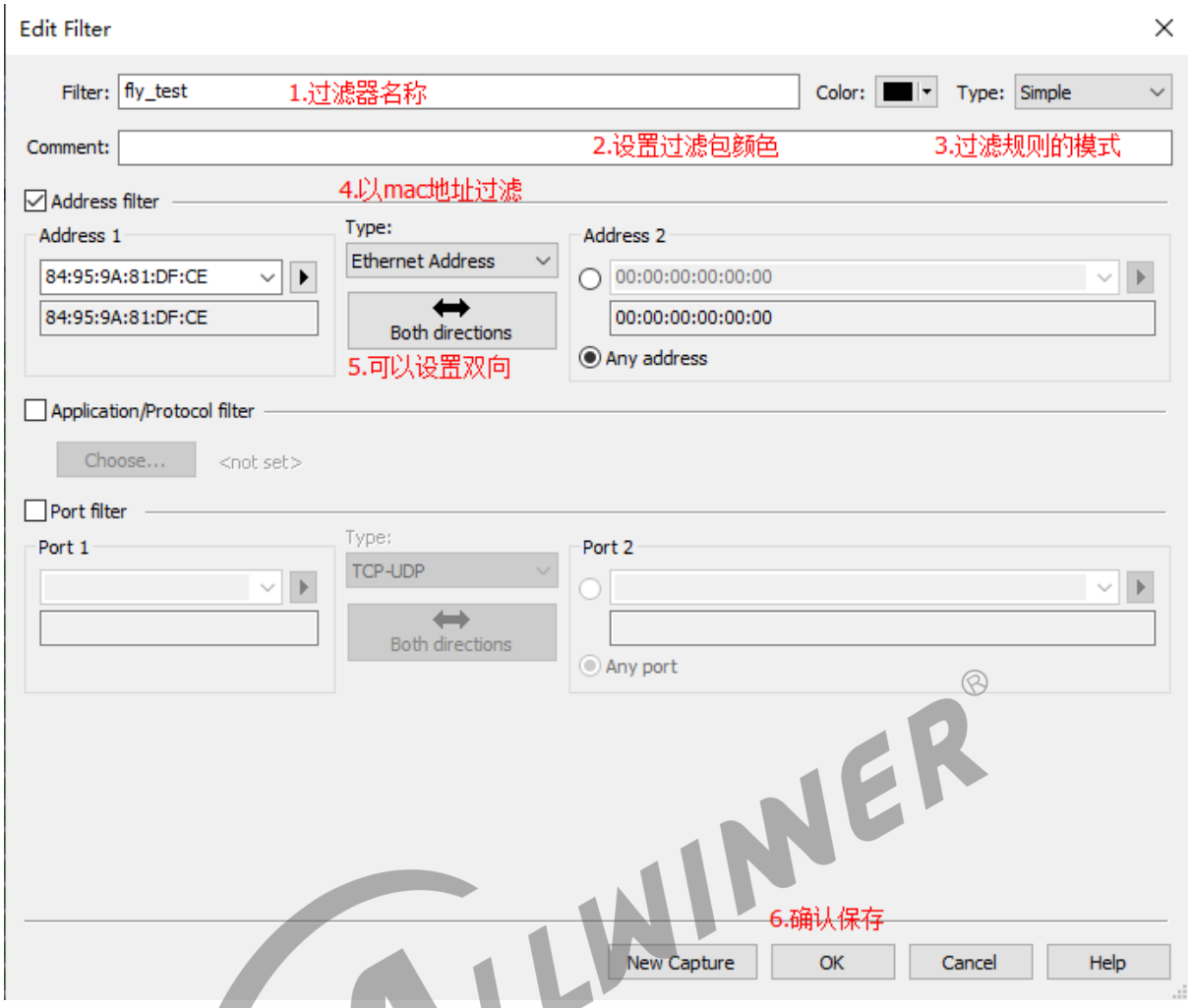


图 3-11: Omnipeekmac 地址过滤

(2) 帧类型过滤——EAPOL-key

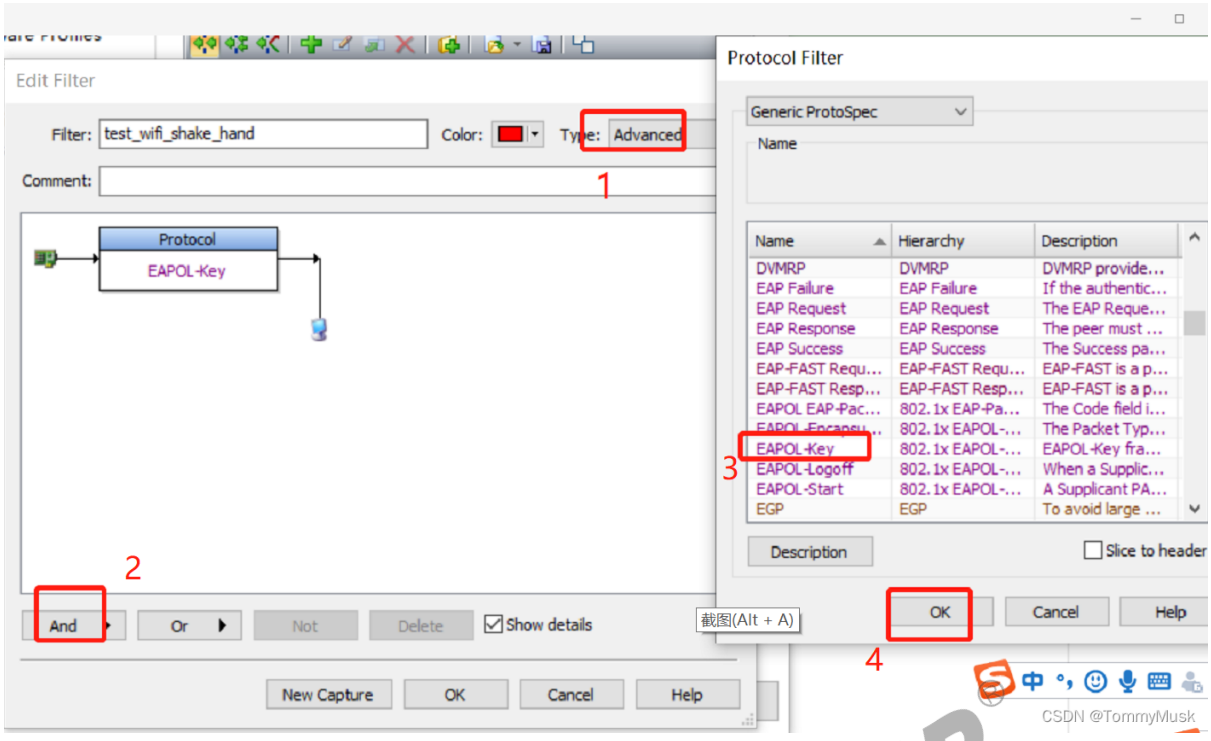


图 3-12: Omnipeek 帧类型过滤设置

(3) 正则表达式过滤

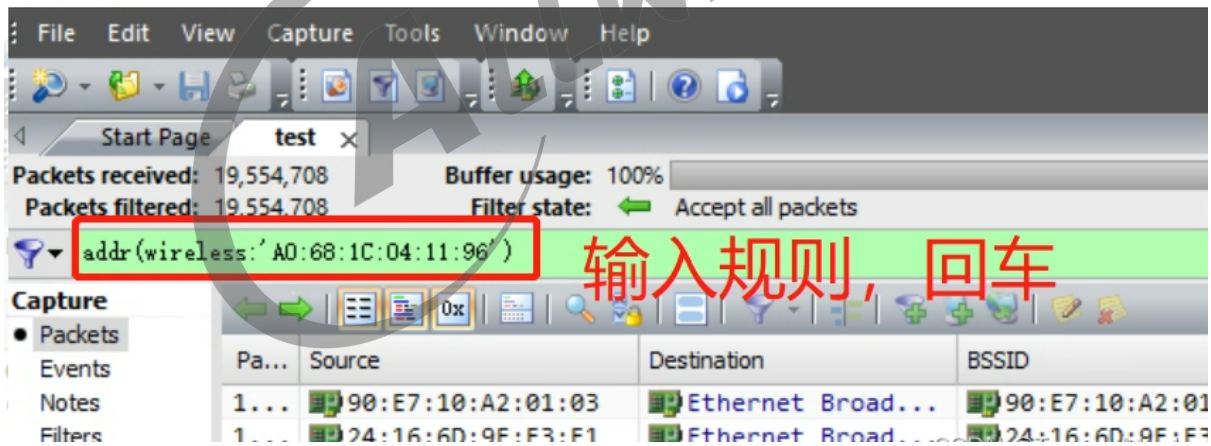


图 3-13: Omnipeek 正则表达式过滤

常用的正则表达式：

a. 筛选出某个 mac 地址双向数据包。

```
addr(wireless:'A0:68:1C:04:11:96')
addr(wireless:'*.*.*.19.4F:B6')
```

b. 使用某条规则。

```
filter('xxx')
```

c. 过滤网卡 mac 地址。

```
eth.addr == 00:e0:4c:68:01:f0
wlan.addr==04:95:E6:E6:3A:71
```

d. 通过 ip 地址筛选单方向包。

```
ip.src == 192.168.3.10 && ip.dst == 101.37.194.26
```

e. 通过 ip 地址筛选双向包。

```
(ip.src == 192.168.3.10 && ip.dst == 101.37.194.26) || (ip.dst == 192.168.3.10 && ip.src == 101.37.194.26)
```

4 Wireshark 抓包工具使用

4.1 下载和安装

官网：

https://www.wireshark.org

4.2 抓包流程介绍

在学习使用 wireshark 之前，首先要了解该工具相关界面的作用。

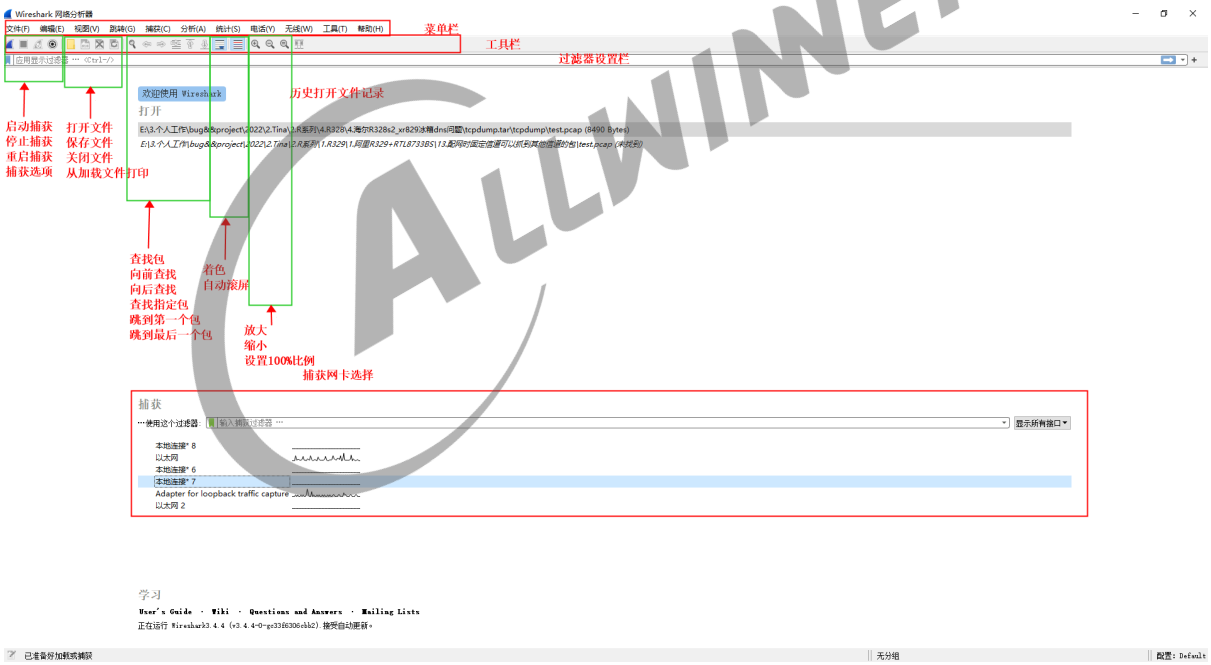


图 4-1: Wireshark 打开界面

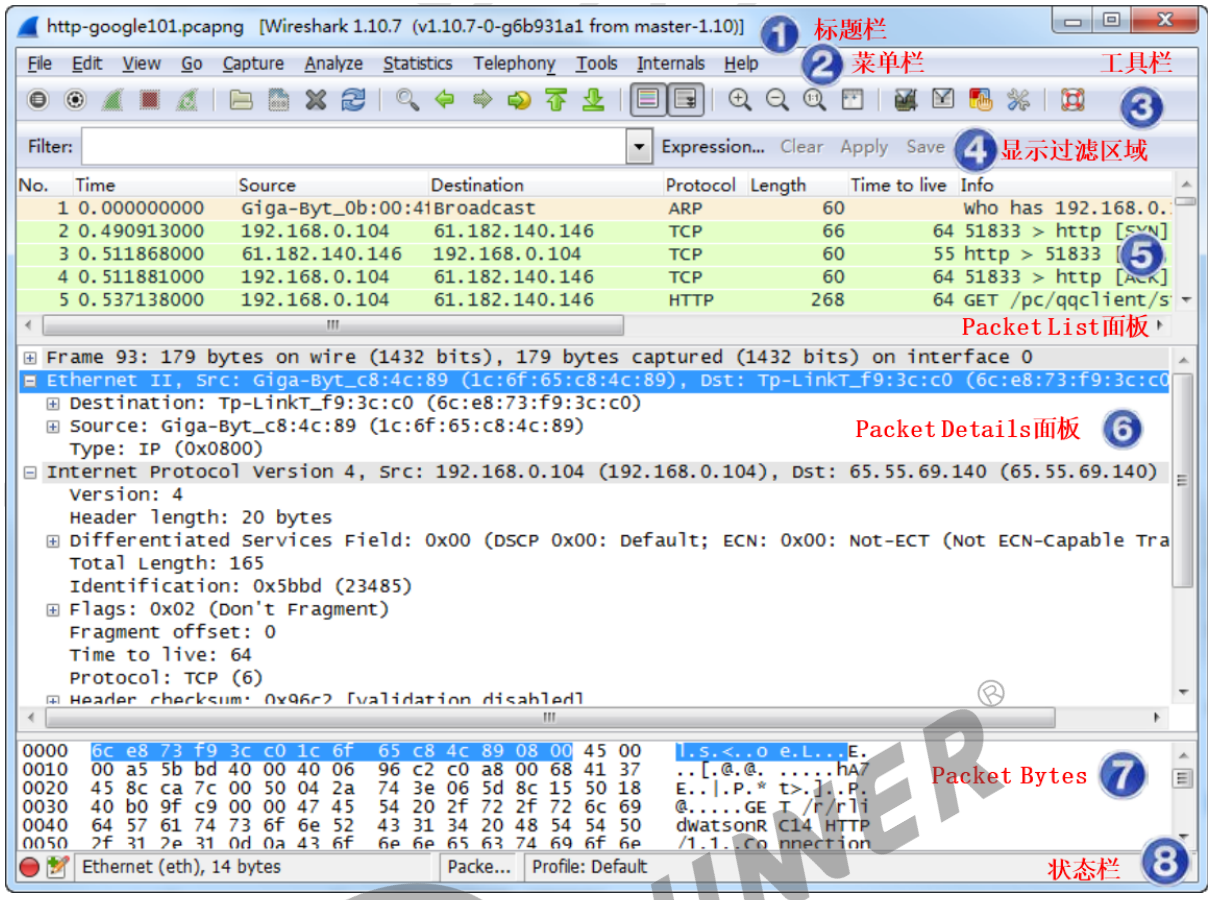


图 4-2: Wireshark 主窗口界面

在 Wireshark 主窗口界面中，以编号的形式将 Wireshark 每部分标出。下面分别介绍每部分的含义，

(1) 标题栏——用于显示文件名称、捕获的设备名称、Wireshark 版本号。

(2) 菜单栏——Wireshark 的标准菜单栏。

(3) 工具栏——常用功能快捷图标按钮，比如开始抓包、停止抓包、滚动等。

(4) 显示过滤区域——设置过滤规则，例如按 MAC 地址过滤、按协议过滤等，减少查看数据的复杂度。

(5) Packet List 面板——显示每个数据帧的摘要。

(6) Packet Details 面板——分析封包的详细信息。

(7) Packet Bytes 面板——以十六进制和 ASCII 格式显示数据包的细节。

(8) 状态栏——专家信息、注释、包数、和 Profile。

1) 选择菜单栏上捕获——> 选项，取消混杂模式，勾选 WLAN，开始 (你也可以直接双击上图的 WLAN 开始)

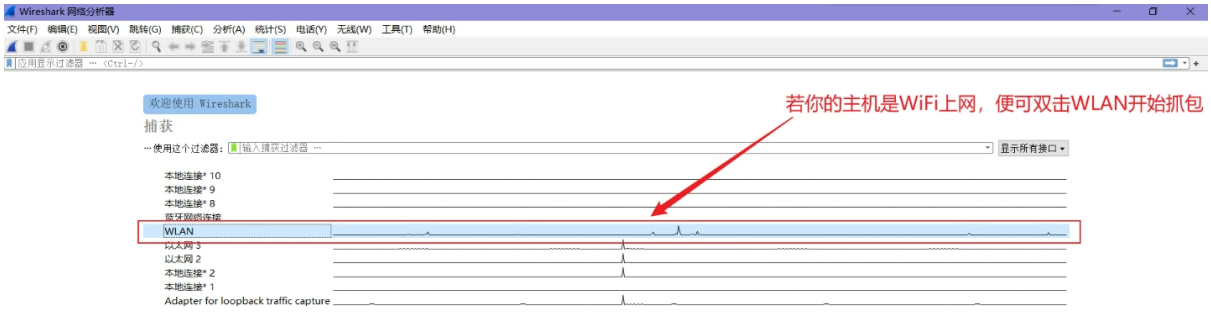


图 4-3: wireshark 接口显示 1

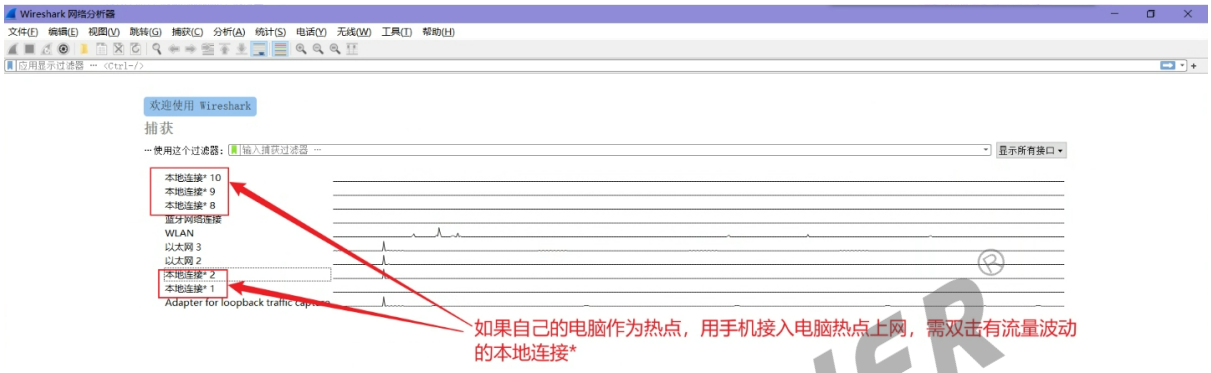


图 4-4: wireshark 接口显示 2

2）此时 wireshark 已经开始工作，可以查看抓包情况。

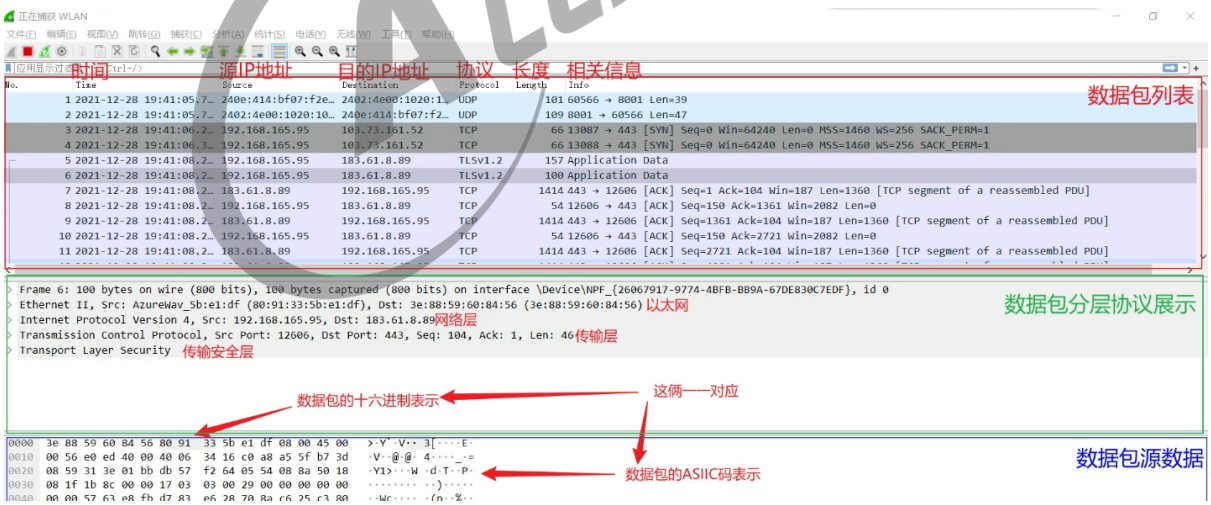


图 4-5: wireshark 抓包中 1

点击不同协议的数据包，数据包分层协议展示的内容就不一样。

例如：

- TLS：TLS 协议 (Transport Layer Security) 是一个在传输层为数据提供保密和完整性的一个安全协议，说白了就是保障传输层数据包的安全，大家所熟知的 HTTPS 中的 S 就是 TLS。

- HTTP：HTTP 协议 (Hypertext Transfer Protocol) 是一种用于从网络传输超文本到本地浏览器的传输协议。

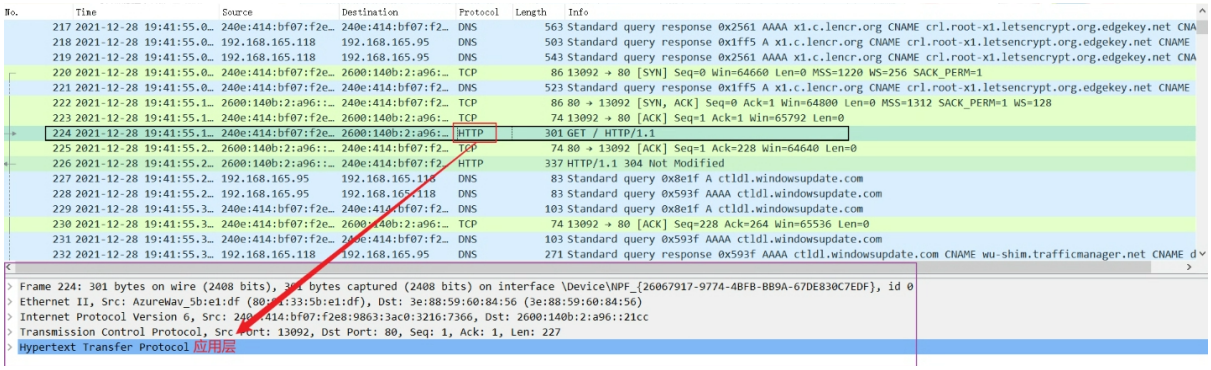


图 4-6: wireshark 抓包中 2

3) 数据包的详细信息。

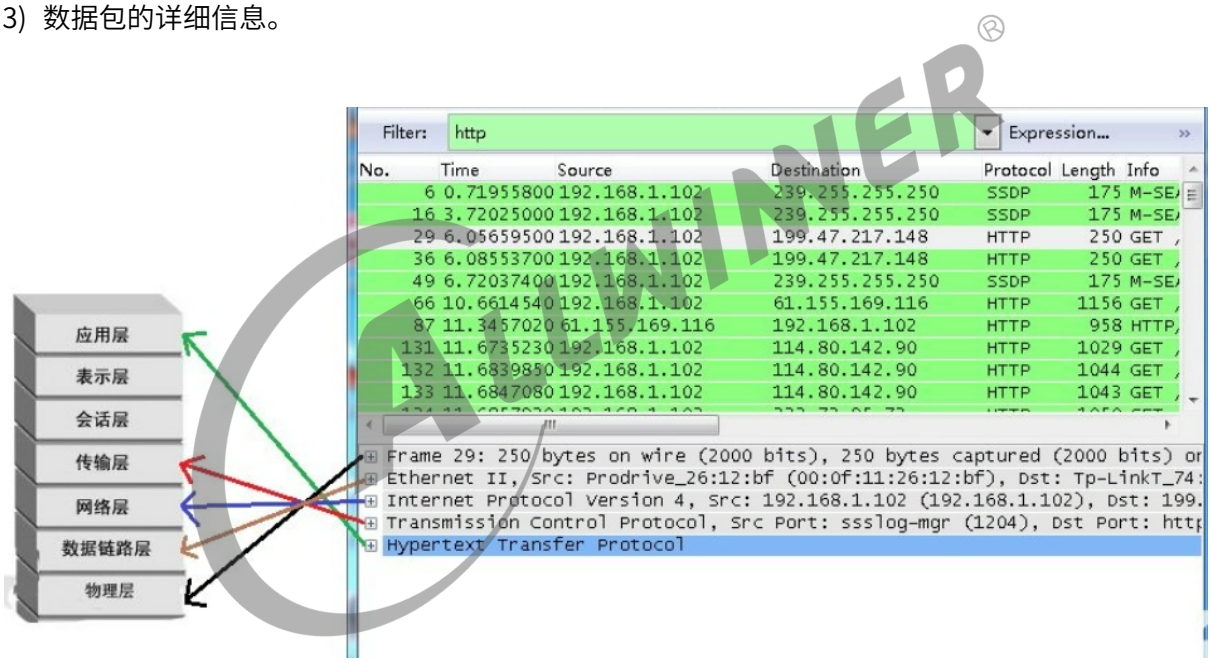


图 4-7: wireshark 显示数据包信息 1

IP 数据报信息：

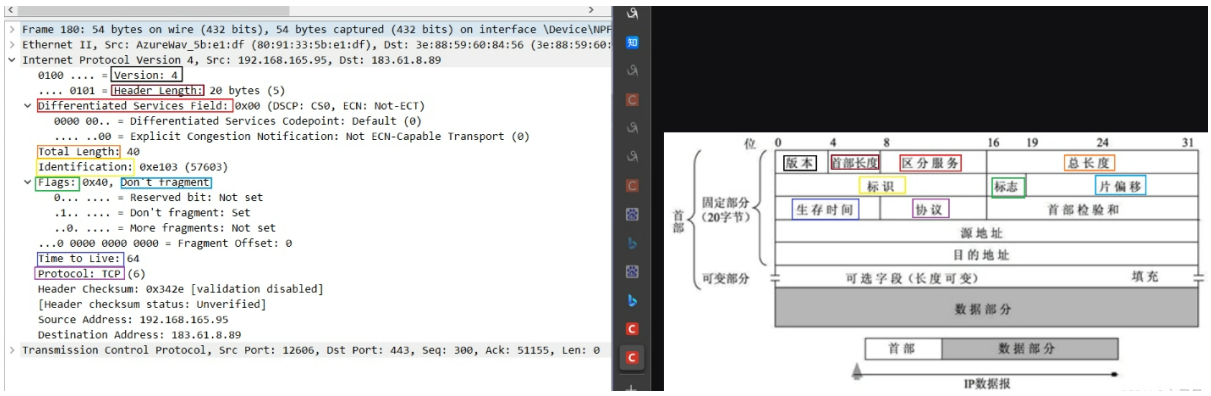


图 4-8: wireshark 显示数据包信息 2

TCP 数据报信息：

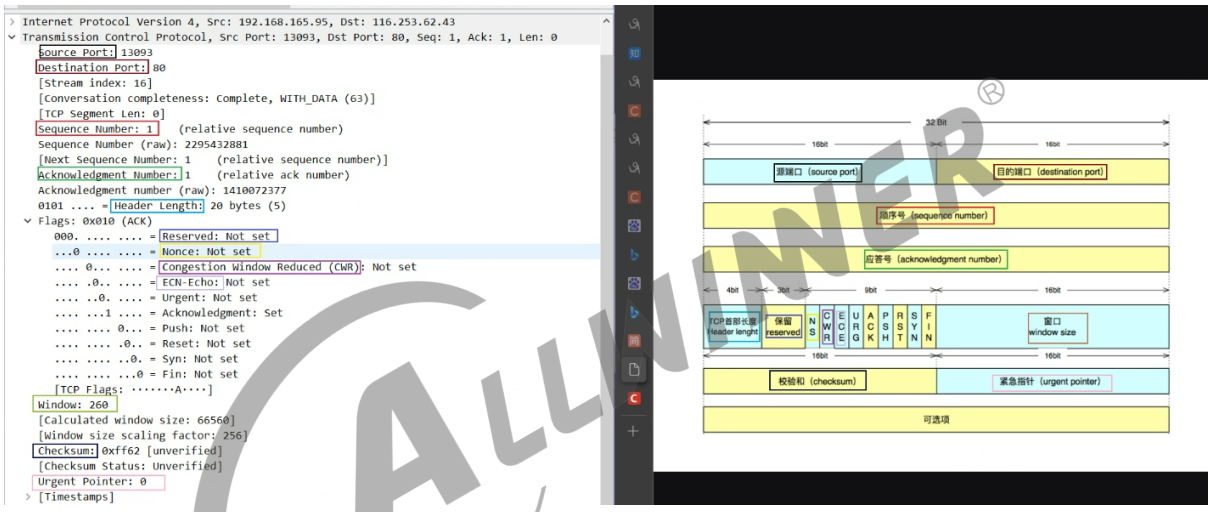


图 4-9: wireshark 显示数据包信息 3

4）捕获过滤器表达式。

捕获过滤器表达式作用在 wireshark 开始捕获数据包之前，只捕获符合条件的数据包，不记录不符合条件的数据包。

捕获过滤器表达式没有像显示过滤器表达式那样明显的规律，但写法不多所以也不难。而且除非全部捕获要占用的磁盘空间实现太大，且你非常明确过滤掉的数据包是你不需要的，不然一般都不用捕获过滤器表达式而用显示过滤器表达式。

如何打开捕获过滤器：

图 4-10: wireshark 捕获器设置

语法说明：

```
Protocol  Direction  Host(s) Value  Logical Operations  Other Expression
tcp dst 10.1.1.1  80  and tcp dst 10.1.1.2.3289
```

(1) Protocol(协议)

可能的值: ether、fddi、ip、arp、rarp、decnet、tcp and udp 等等。如果没有特别指明是什么协议，则默认使用所有支持的协议

(2) Direction(方向)

可能的值: src、dst、src and dst、src or dst 等等。如果没有特别指明来源或目的地，则默认使用“src or dst” 作为关键字

比如：“host 10.2.2.2” 与 “src or dst host 10.2.2.2” 是一样的

(3) Host(s)

可能的值：net、port、host、portrange 等等。如果没有指定此值，则默认使用 “host” 关键字

比如：“src 10.1.1.1” 与 “src host 10.1.1.1” 是一样的

(4) Logical Operations(逻辑运算)

可能的值：not、and、or。

否 (“not”) 具有最高的优先级。或 (“or”) 和与 (“and”) 具有相同的优先级，运算时从左至右进行。

例如，

"not tcp port 3128 and tcp port 23"与"(not tcp port 3128) and tcp port 23"相同。
"not tcp port 3128 and tcp port 23"与"not (tcp port 3128 and tcp port 23)"不同。

捕获过滤器例子

a. 捕获目的 TCP 端口为 3128 的封包

```
tcp dst port 3128
```

b. 捕获来源 IP 地址为 10.1.1.1 的封包

```
ip src host 10.1.1.1
```

c. 捕获目的或来源 IP 地址为 10.1.2.3 的封包

```
host 10.1.2.3
```

d. 捕获来源为 UDP 或 TCP，并且端口号在 2000 至 2500 范围内的封包

```
src portrange 2000-2500
```

e. 捕获除了 icmp 以外的所有封包

```
not icmp
```

f. 捕获来源 IP 地址为 10.7.2.12，但目的地不是 10.200.0.0/16 的封包

```
src host 10.7.2.12 and not dst net 10.200.0.0/16
```

g. 捕获源 IP 为 10.4.1.12 或源网络为 10.6.0.0/16，目的 TCP 端口号在 200 至 10000 之间，并且目的位于网络 10.0.0.0/8 内所有封包

```
(src host 10.4.1.12 or scr net 10.6.0.0/16) and tcp dst portrange 200-10000 and dst net 10.0.0.0/8
```

h. 捕获广播封包

```
broadcast
```

4) 显示过滤器表达式

显示过滤器表达式作用在 wireshark 捕获数据包之后，从已捕获的所有数据包中显示出符合条件的数据包，隐藏不符合条件的数据包。显示过滤表达示在工具栏下方的 “显示过滤器” 输入框输入即可生效。

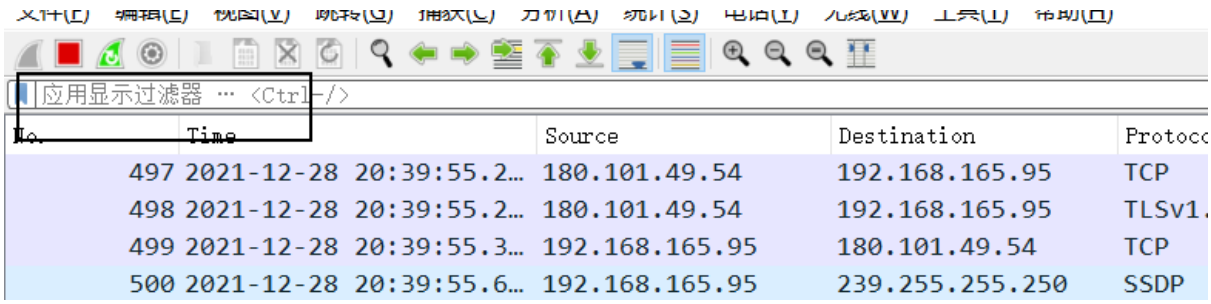


图 4-11: wireshark 显示过滤器

语法说明：

一条基本的表达式由过滤项、过滤关系、过滤值三项组成。

比如 ip.addr == 192.168.31.1，这条表达式中:

- ip.addr 就是过滤项，wireshark 的过滤项是 “协议” + “.” + “协议字段” 的模式，区分大小写，只能使用小写。
- == 就是过滤关系，即大于、小于、等于等几种等式关系，英文或是 C-Like 形式均可。
- 192.168.31.1 就是过滤值，具体根据过滤项而定。

整条表达式的意思是找出所有 ip 协议中源或目标 ip 等于 192.168.31.1 的数据包。

复合过滤表达式，就是指由多条基本过滤表达式组合而成的表达式，通过逻辑运算符相连。

```
Protocol . String1 . String2 Comparison operator value Logical Operations   Other Expression
ftp.passive.ip == 10.1.1.1 xor icmp.type
```

(1) Protocol(协议)

位于 OSI 模型第 2 至 7 层的协议，如：IP、TCP、DNS 等。

(2) String1，String2(可选项)

协议的子类。

(3) Comparison Operator(比较运算符)

表 4-1: 比较运算符

英文写法	c 语言写法	含义
eq	==	等于
ne	!=	不等于
gt	>	大于
lt	<	小于
ge	>=	大于等于
le	<=	小于等于
bitwise_and	&	位与操作

(4) Logical Operations(逻辑运算符)

表 4-2: 比较运算符

英文写法	c 语言写法	含义
and	&&	逻辑与
or	||	逻辑或
xor	^^	逻辑异或
not	!	逻辑非
[...]	无	切片操作符

逻辑异或是一种排除性的或。当其被用在过滤器的两个条件之间时，只有当且仅当其中的一个条件满足时，这样的结果才会被显示在屏幕上。

例如：

```
tcp.dstport 80 xor tcp.dstport 1025
```

只有当目的 TCP 端口为 80 或者来源于端口 1025(但又不能同时满足这两点) 时，这样的数据包才会被显示。

显示过滤器例子：

a. 显示 SNMP 或 DNS 或 ICMP 封包

```
snmp || dns || icmp
```

b. 显示来源或目的 IP 地址为 10.1.1.1 的封包

```
ip.addr == 10.1.1.1
```

c. 显示来源不为 10.1.2.3 或目的不为 10.4.5.6 的包

```
ip.src != 10.1.2.3 or ip.dst != 10.4.5.6
```

d. 显示来源或目的 TCP 端口号为 25 的封包

```
tcp.port == 25
```

e. 显示目的 TCP 端口号为 25 的封包

```
tcp.dstport == 25
```

f. 显示包含 TCP 标志的封包

```
tcp.flags
```

g. 显示包含 TCP SYN 标志的封包

```
tcp.flags.syn == 1
```

h. 排除 arp 流量

```
!arp
```

i. 文本管理流量 (telnet 或 tftp)

```
tcp.port == 23 || tcp.port == 21
```

5 tcpdump 抓包工具使用

5.1 下载和安装

官网：

```
https://www.tcpdump.org/
```

5.2 抓包流程介绍

tcpdump 主要通过监测网络接口，抓取 TCP/IP 网络层的数据包，包括 tcp、ip 和 ICMP 数据包等，也能抓取 wpa_supplicant 通过 netdevice 与 UMAC 的交互包。使用 tcpdump 抓包指令之后可通过 omnipeek、wireshark 等软件来查看和分析数据包的。

1) 配置

Tina 执行 make menuconfig

```
Network--->
   <*> tcpdump..... Network monitoring and data acquisition tool
```

针对 buildroot 文件系统，执行./build.sh buildroot_menuconfig

```
 -> Networking applications
  -> tcpdump (BR2_PACKAGE_TCPDUMP [=y])
```

2) 使用

```
1.Tina系统联网

2.tcpdump -i wlan0 -s 0 -w tmp/a.pcap &  //后台运行tcpdump抓包

3.进行网络操作，如ping、dhcp、测吞吐...

4.adb pull /tmp/a.pcap .    //拉取tcpdump生成的pcap文件到本地用wireshark等工具打开分析。
```

3)tcpdump 命令语法

```
tcpdump <option> <proto> <dir> <type>
```

- option: 随后介绍。
- proto: 根据协议进行过滤，可识别的关键词有: tcp、udp、icmp、ip、ip6、arp、rarp、ether、wlan、fddi、tr、decnet 等。

- dir: 根据数据流向进行过滤，可识别的关键字有：src、dst，同时你可以使用逻辑运算符进行组合，比如 “src or dst”。
- type: 可识别的关键词有: host、net、port、portrange 等，这些词后边需要再接参数。

tcpdump 也支持类似 wireshark 的逻辑运算符，以及多条表达式组成复杂表达式。

4)option 参数

```
-a 将网络地址和广播地址转变成容易识别的名字
-d 将已截获的数据包的代码以人容易理解的格式输出
-dd 将已截获的数据包的代码以C程式的格式输出
-ddd 将已截获的数据包的代码以十进制格式输出
-e 输出数据链路层的头部信息
-f 将internet地址以数字形式输出
-l 将标准输出变为行缓冲方式
-n 不将网络地址转换成易识别的主机名，只以数字形式列出主机地址(如IP地址)，能够避免DNS查询
-nn 指定将每个监听到的数据包中的域名转换成IP、端口从应用名称转换成端口号后显示
-t 每行不输出时间戳
-tt 每行输出时间戳
-ttt 输出每两行打印的时间间隔(以毫秒为单位)
-tttt 在每行打印的时间戳之前添加日期的打印，输出的时间最直观
-v 输出较周详的信息，例如IP包中的TTL和服务类型信息
-vv 输出详尽的报文信息
-c 在捕获指定个数的数据包后退出
-F 从指定的文档中读取过滤规则，忽略命令行中指定的其他过滤规则
-i 指定监听的网络接口
-r 从指定的文档中读取数据包(该文档一般通过-w选项产生)
-s 表示从一个包中截取的字节数，0表示包不截断，抓取完整的数据包，默认只显示68字节
-w 将截获的数据包直接写入指定的文档中，不对其进行分析和输出
-T 将截获的数据包直接解释为指定类型的报文，现在支持的类型有cnfp、rpc、rtp、snmp、vat和wb
-x 以16进制的形式打印每个包的头部数据(但不包括数据链路层的头部)
-X 以16进制和ASCII码的形式打印出每个包的数据(但不包括连接层的头部)
```

5) 常规过滤规则

a. 基于 IP 地址过滤

```
tcpdump host 192.168.31.100
```

b. 基于网段进行过滤

```
tcpdump net 192.168.31.0/24
```

c. 基于端口进行过滤

```
tcpdump port 8080
```

d. 基于协议进行过滤

```
tcpdump icmp
```

6) 组合过滤规则

组合过滤规则是通过使用运算符进行组合的，例如：

- 逻辑运算符: and、or、not

- 条件判断符: =、==、!=

a. 抓取来自 192.168.31.99 发往目标主机的 80 端口的数据包

```
tcpdump src 192.168.31.99 and dst port 80
```

当在使用多个过滤器进行组合时，有可能需要用到括号，而括号在 shell 中是特殊符号，所以需要使用引号将其包含。例子如下:

```
tcpdump 'src 192.168.31.99 and (dst port 80 or 22)'
```

b. 使用条件判断符的时候，可以搭配一些关键字使用

- if：表示网卡接口名
- proc：表示进程名
- pid：表示进程 id
- svc：表示 service class
- dir：表示方向，in 和 out
- eproc：表示 effective process name
- epid：表示 effective process ID

过滤来自进程名为 nc 发出的流经 en0 网卡的数据包，或者不流经 en0 的入方向的数据包

```
tcpdump "( if=en0 and proc =nc ) || (if != en0 and dir=in)"
```

7) 特殊过滤规则

a. 基于 [...] 进行过滤

```
tcpdump -i eth0 "tcp[tcpflags] & tcp-syn != 0"
```

抓取 eth0 上 syn 位为 1 的包。

b. 基于包大小进行过滤

```
tcpdump less 32
tcpdump greater 64
tcpdump <= 128
```

c. 基于 mac 地址进行过滤

```
tcpdump ether host [ehost]
```

d. 基于广播/多播数据包进行过滤

```
tcpdump ether broadcast
tcpdump ether multicast
tcpdump ip broadcast
tcpdump ip multicast
```

6 抓包文件协议分析

6.1 扫描过程

wifi 扫描流程分为两个步骤，设备发起 probe request 请求，路由回复 probe response。

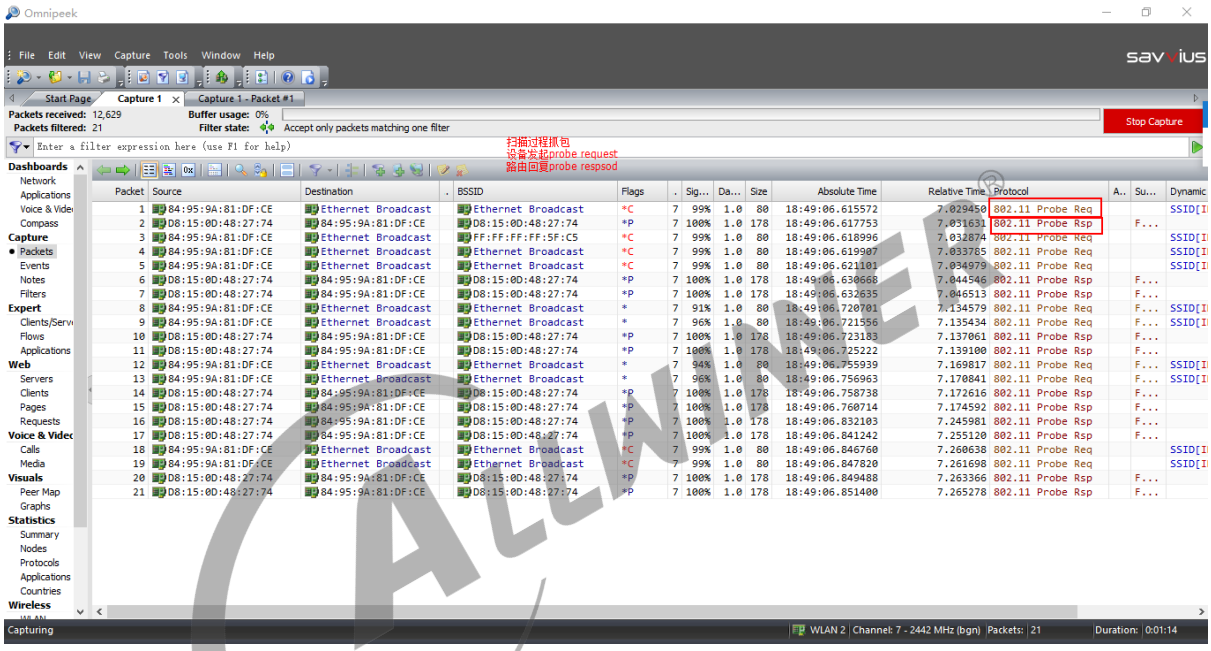


图 6-1: 扫描流程

图 6-2: probe-request 帧

图 6-3: probe-response 帧

6.2 连接流程

wifi 连接流程包括: 扫描、认证、关联、四次握手、DHCP 等阶段。

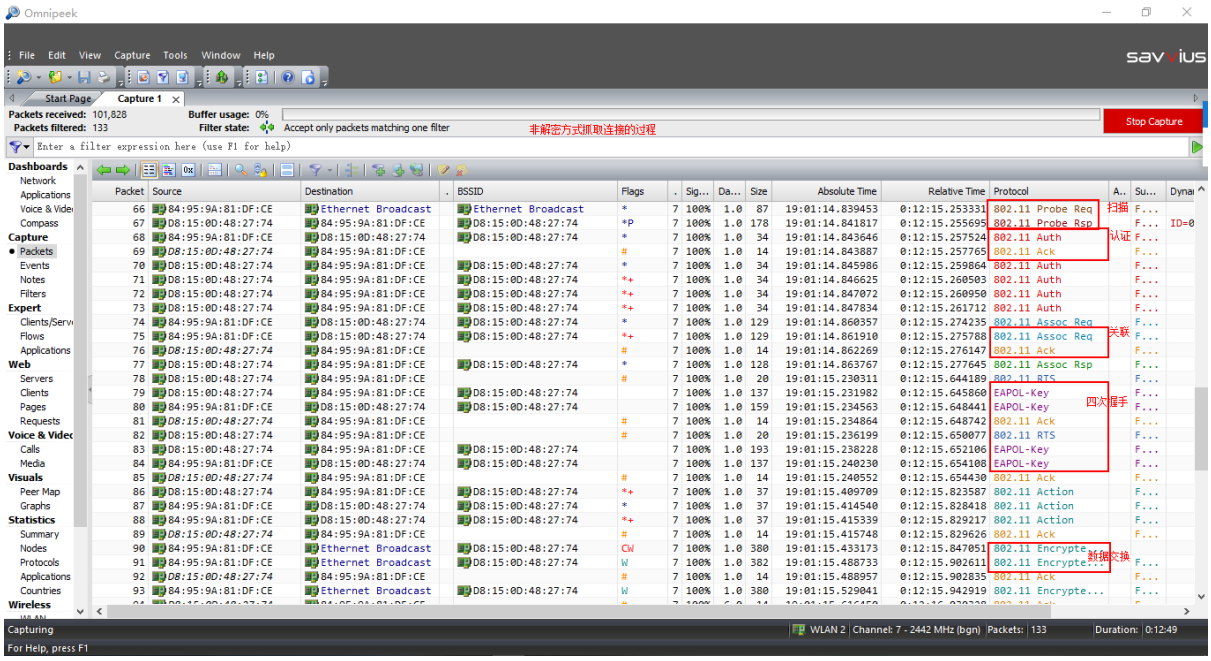


图 6-4: 非加密连接过程

如果没有配置路由解密，我们抓包是看不到完整的连接流程的，后续的数据交换流程也是以加密方式呈现出来的。

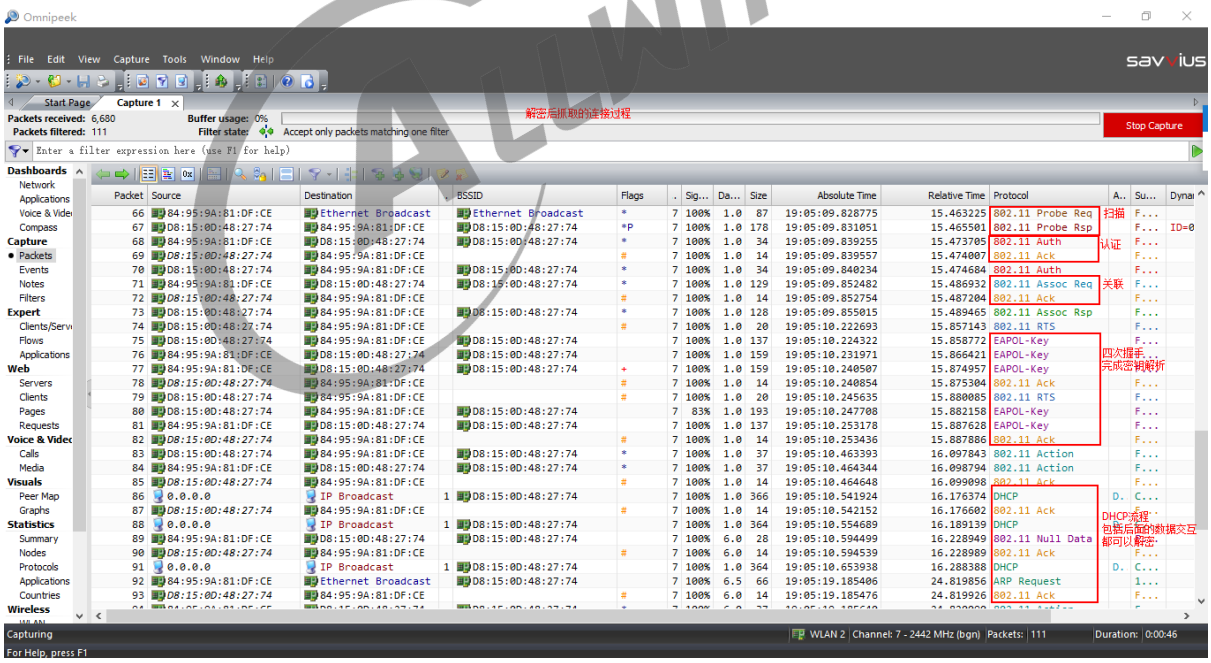


图 6-5: 加密连接过程

6.3 ARP 流程

地址解析协议 ARP 原理：

ARP(Address Resolution Protocol，地址解析协议) 是根据 IP 地址获取物理地址的一个 TCP/IP 协议。由于 OSI 模型把网络工作分为七层，IP 地址在 OSI 模型的第三层，MAC 地址在第二层，彼此不直接通信。在通过以太网发送 IP 数据包时，需要先封装第三层 (32 位 IP 地址)、第二层 (48 位 MAC 地址) 的报头。但由于发送数据包时只知道目标 IP 地址，不知道其 MAC 地址，而又不能跨越第二、三层，所以需要使用地址解析协议。

ARP 的基本功能就是负责将一个已知的 IP 地址解析成 MAC 地址，以便主机间能正常进行通信。

<table>
<tr><td colspan="3">广播 MAC 地址(全 1)</td></tr>
<tr><td colspan="2">目标 MAC 地址(广播 MAC 地址)</td><td>源 MAC 地址</td></tr>
<tr><td colspan="3">源 MAC 地址</td></tr>
<tr><td colspan="3">协议类型</td></tr>
<tr><td colspan="2">硬件类型</td><td>协议类型</td></tr>
<tr><td>硬件地址长度</td><td>协议长度</td><td>操作(请求 1)</td></tr>
<tr><td colspan="3">发送方硬件地址(前 32 位)</td></tr>
<tr><td colspan="2">发送方硬件地址(后 16 位)</td><td>发送方 IP 地址(前 16 位)</td></tr>
<tr><td colspan="2">发送方 IP 地址(后 16 位)</td><td>目标硬件地址(前 16 位)</td></tr>
<tr><td colspan="3">目标硬件地址(后 32 位)</td></tr>
<tr><td colspan="3">目标 IP 地址(32 位)</td></tr>
</table>

图 6-6: ARP 头部

抓包示例

ARP 请求报文

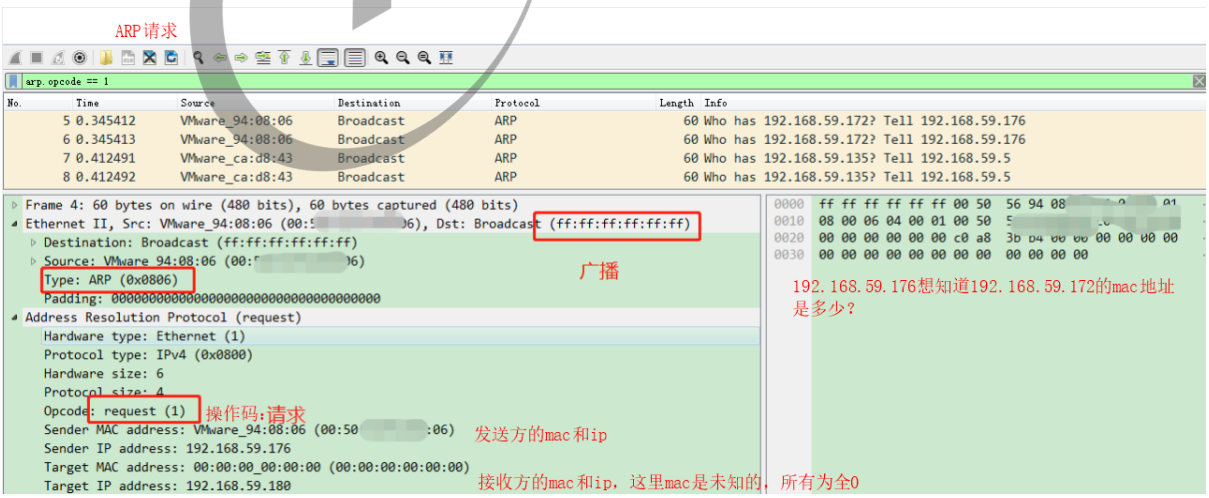


图 6-7: ARP 请求

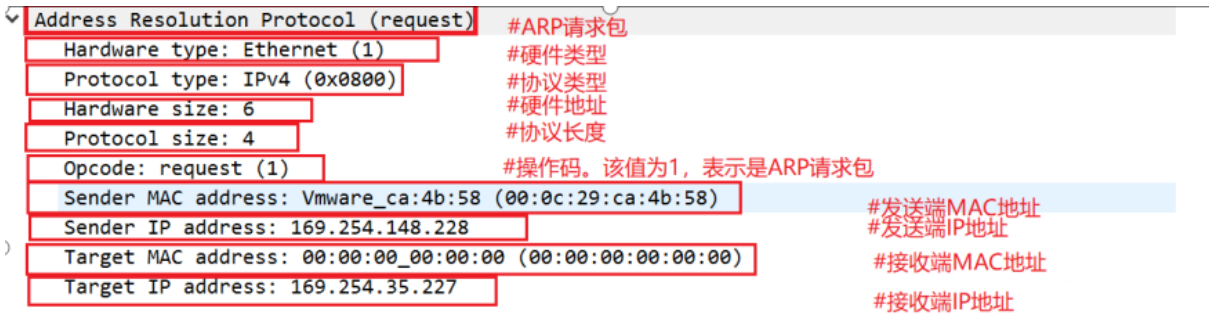


图 6-8: ARP 请求地址段说明

长度：8 位/字节，MAC 地址 48 位，即 6 字节，IP 地址 32 位，即 4 字节。

ARP 响应报文

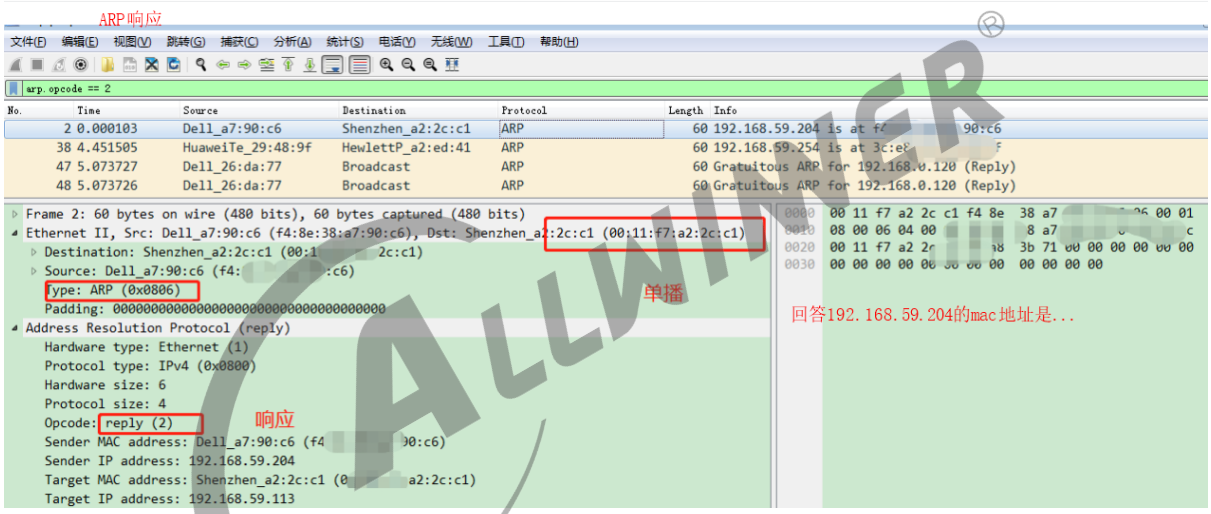


图 6-9: ARP 回复

图 6-10: ARP 回复地址段说明

长度：8 位/字节，MAC 地址 48 位，即 6 字节，IP 地址 32 位，即 4 字节。

6.4 DNS 流程

域名解析 DNS 原理

当某一个应用进程需要把主机名解析为 IP 地址时，该应用进程就调用解析程序 (resolver)，并成为 DNS 的一个客户，把待解析的域名放在 DNS 请求报文中，以 UDP 用户数据报方式发给本地域名服务器 (使用 UDP 是为了减少开销)。本地域名服务器在查找域名后，把对应的 IP 地址放在回答报文中返回。应用进程获得目的主机的 IP 地址后即可进行通信。若本地域名服务器不能回答该请求，则此域名服务器就暂时成为 DNS 中的另一个客户，并向其他域名服务器发出查询请求。这种过程直至找到能够回答该请求的域名服务器为止。DNS 的基本功能就是负责将服务器域名转换为服务器的 IP 地址。

事务 ID(Transaction ID)	标志(Flags)
问题计数(Questions)	回答资源记录数(Answer RRs)
权威名称服务器计数(Authority RRs)	附加资源记录数(Additional RRs)
查询问题区域(Queries)	
回答问题区域(Answers)	
权威名称服务器区域(Authoritative nameservers)	
附加信息区域(Additional records)	

图 6-11: DNS 头部

抓包示例

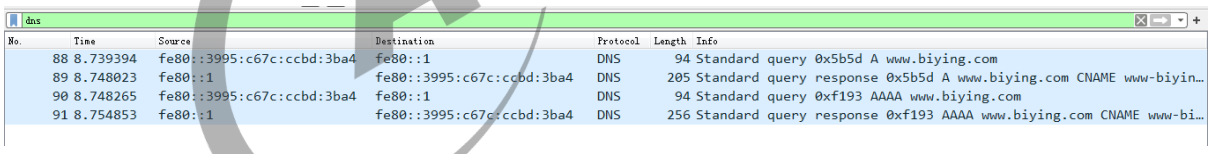


图 6-12: DNS 流程 1

可以看到，访问 www.biying.com 网址，产生了四条数据 (将抓包报文使用了 DNS 过滤后的结果)，两次 DNS 请求 (请求与响应配对存在)。

第一次 DNS 请求查询 www.biying.com 域名对应的 IP4 地址 (A 代表 IP4)。

第二次 DNS 请求查询 www.biying.com 域名对应的 IP6 地址 (AAAA 代表 IP6)。

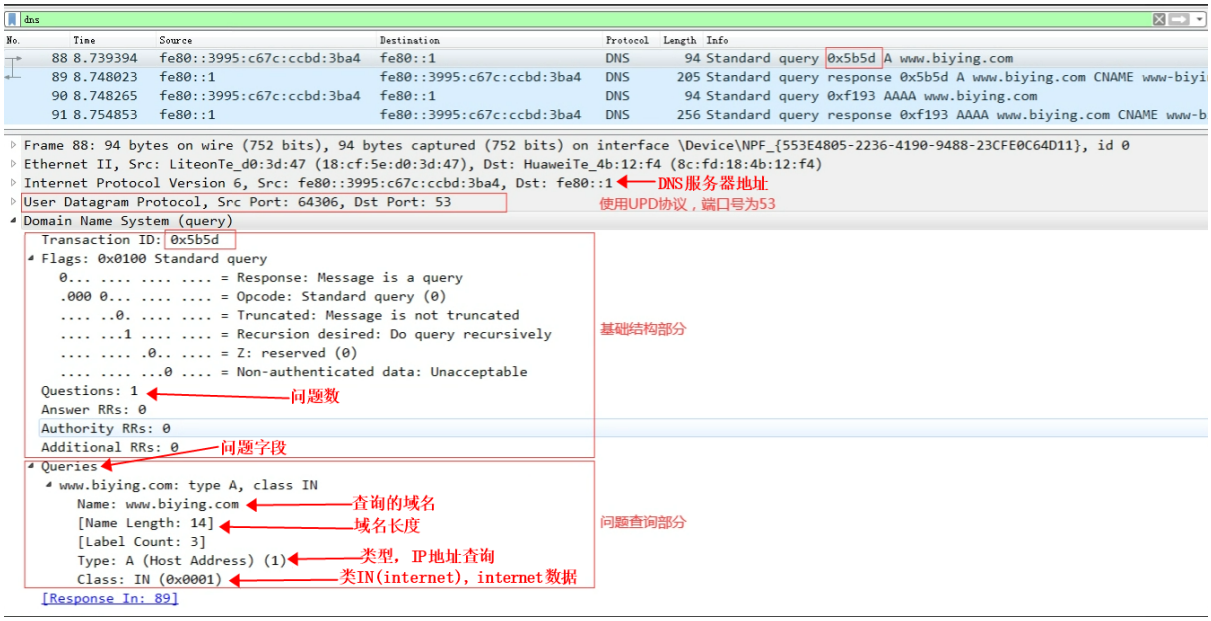


图 6-13: DNS 请求报文

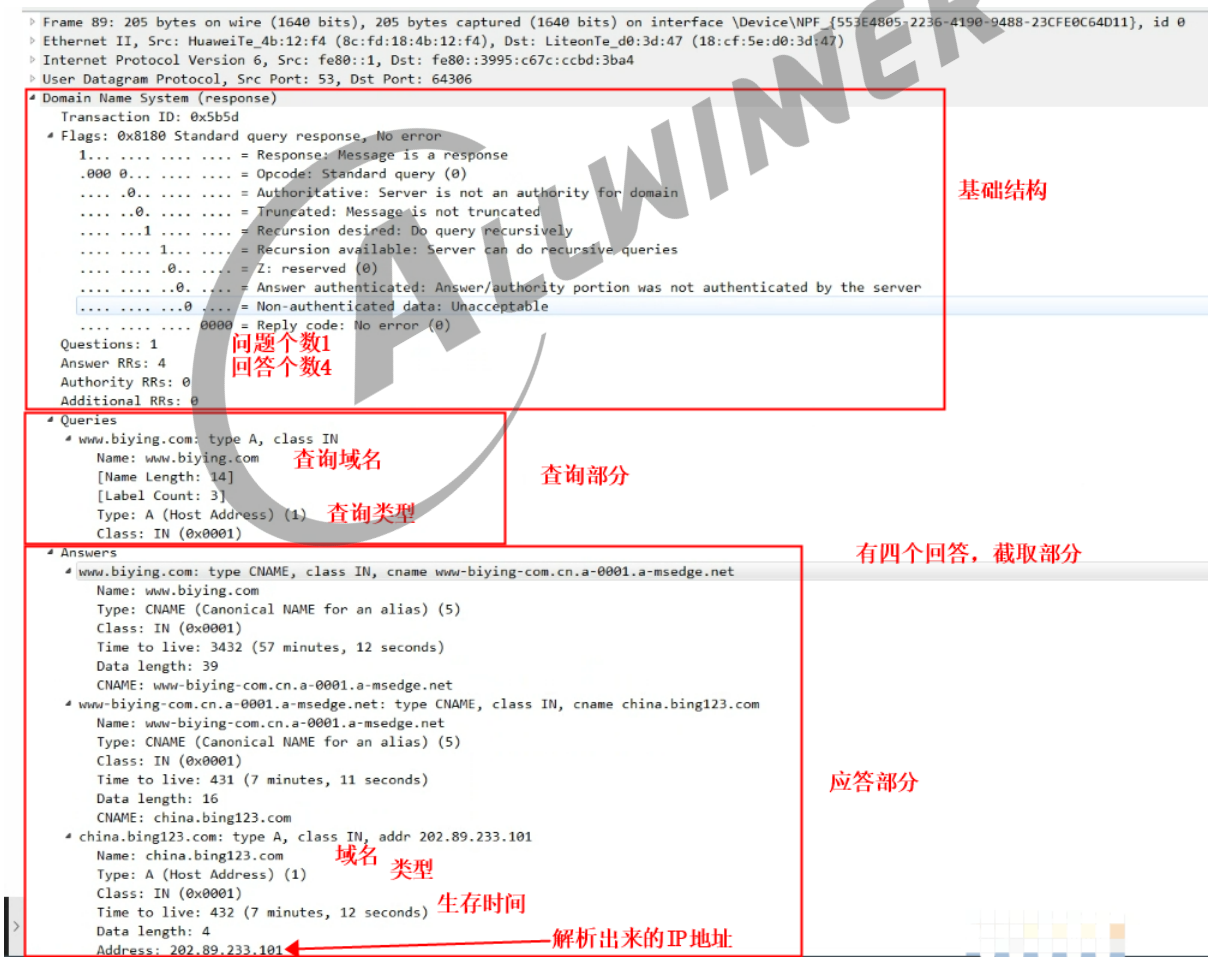


图 6-14: DNS 响应报文

6.5 DHCP 流程

动态主机配置 DHCP 原理

DHCP(Dynamic Host Configuration Protocol，动态主机配置协议)，前身是 BOOTP 协议，是一个局域网的网络协议，使用 UDP 协议工作，统一使用两个 IANA 分配的端口：67(服务器端)，68(客户端)。属于应用层协议，主要作用是集中的管理、分配 IP 地址，使 client 动态的获得 IP 地址、Gateway 地址、DNS 服务器地址等信息，并能够提升地址的使用率。

DHCP 的基本功能就是自动给内网机器分配 IP 地址。

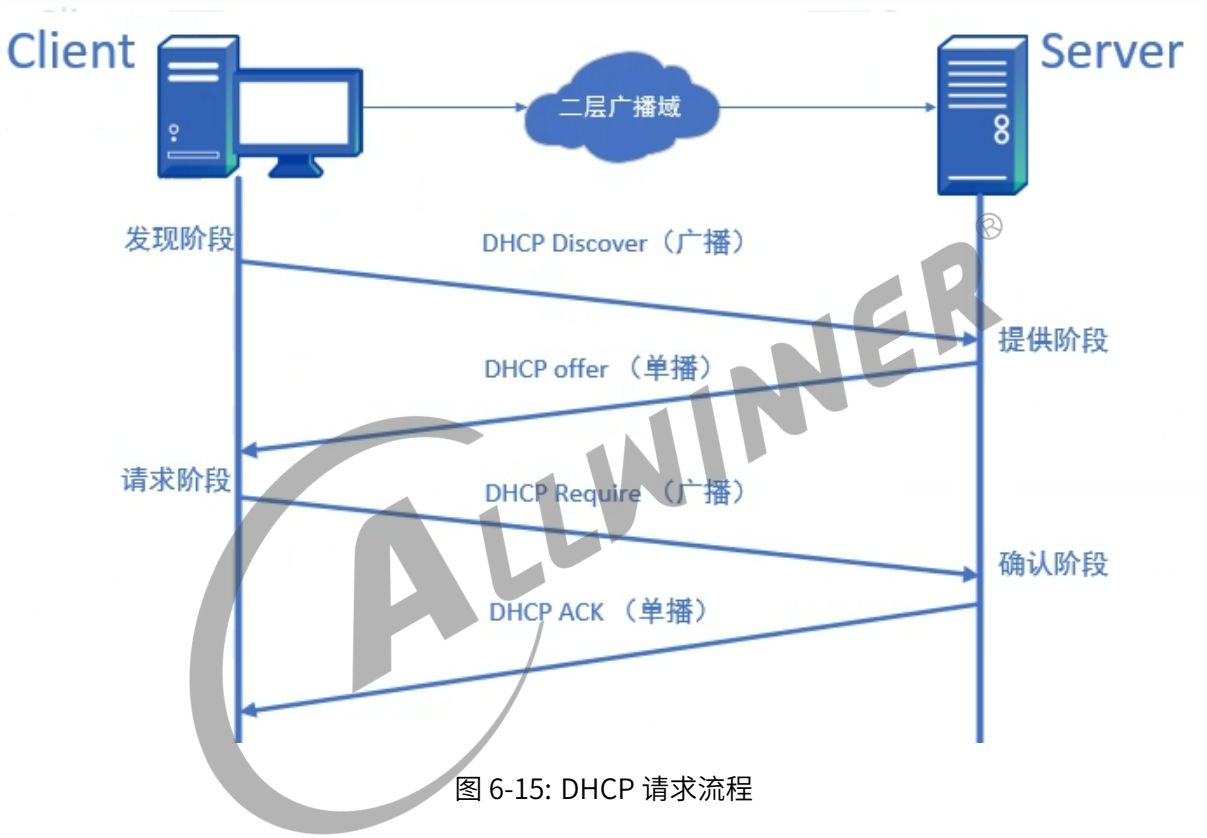


图 6-15: DHCP 请求流程

报文类型描述

DHCP Discover: DHCP 客户端请求地址时，并不知道 DHCP 服务器的位置，因此 DHCP 客户端会在本地网络内以广播方式发送请求报文，这个报文成为 Discover 报文，目的是发现网络中的 DHCP 服务器，所有收到 Discover 报文的 DHCP 服务器都会发送回应报文，DHCP 客户端据此就可以知道网络中存在的 DHCP 服务器的位置。

DHCP Offer: DHCP 服务器收到 Discover 报文后，就会在所配置的地址池中查找一个合适的 ip 地址，加上相应的租约期限和其他配置信息 (网关，DNS 服务器等)，构造一个 Offer 报文，发送给客户，告知用户本服务器可以为其提供 IP 地址。(只是告诉 client 可以提供，是预分配，还需要 client 通过 ARP 检测该 IP 是否重复)。

DHCP Request: DHCP 客户端会收到很多 Offer，所以必须在这些回应中选择一个。Client 通常选

择第一个回应 Offer 报文的服务器作为自己的目标服务器，并回应一个广播 Request 报文，通告选择的服务器。DHCP 客户端成功获取 IP 地址后，在地址使用租期过去 1/2 时，会向 DHCP 服务器发送单播 Request 报文续延租期，如果没有收到 DHCP ACK 报文，在租期过去 3/4 时，发送广播 Request 报文续延租期。

DHCP ACK: DHCP 服务器收到 Request 报文后，根据 Request 报文中携带的用户 MAC 来查找有没有相应的续约记录，如果有则发送 ACK 报文作为回应，通知用户可以使用分配的 ip 地址。

DHCP NAK: 如果 DHCP 服务器收到 Request 报文后，没有发现相应的租约记录或者由于某些原因无法正常分配 ip 地址，则发送 ACK 报文作为回应，通知用户无法分配合适的 ip 地址。

DHCP Release: 当用户不在需要使用分配 ip 地址时，就会向 DHCP 服务器发送 Release 报文，告知服务器用户不再需要分配 ip 地址，DHCP 服务器会释放被绑定的租约。

DHCP Decline: DHCP 客户端收到 DHCP 服务器回应的 ACK 报文后，通过地址冲突检测发现服务器分配的地址冲突或者由于其他原因导致不能使用，则发送 Decline 报文，通知服务器所分配的 ip 地址不可用。

DHCP Inform: DHCP 客户端如果需要从 DHCP 服务器端获取更为详细的配置信息，则发送 Inform 报文向服务器进行请求，服务器收到该报文后，将根据租约进行查找，找到相应的配置信息后，发送 ACK 报文回应 DHCP 客户端 (极少用到)。

DHCP 过程主要为 DHCP Discover->DHCP Offer->DHCP Request->DHCP Ack 四个过程。

还有一个 DHCP Release 释放过程

注：当一个拥有了 IP 的客户端在租约内重新启动，需要进行一次简单的 DORA 过程重新认领新的 IP，只需要完成请求和确认后两部就可以了。

Table：DHCP 消息类型

类型号	消息类型	描述
1	发现	客户端定位可用的服务器
2	提供	服务器响应发现包
3	请求	客户端请求服务器提供的参数
4	拒绝	客户端向服务器指明无效的参数
5	ACK	服务器向客户端发送所请求的配置参数
6	NAK	客户端向服务器拒绝其配置参数的请求
7	释放	客户端向服务器通过取消配置参数来取消租约
8	通知	当客户端已有 IP 时，向服务器请求配置参数

抓包示例

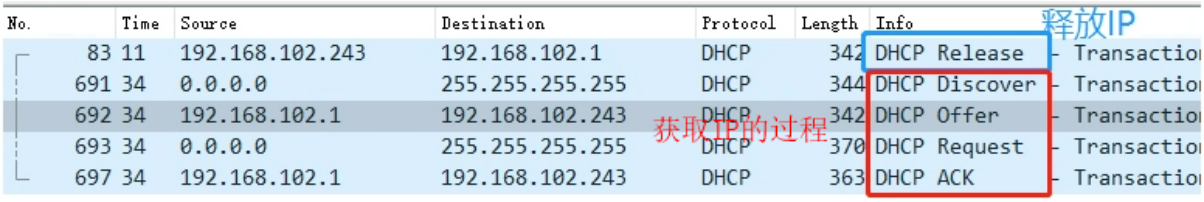


图 6-16: DHCP 流程

动态主机配置协议 DHCP				
偏移位	0-15		16-31	
0	操作代码	硬件类型	硬件长度	跳数
32	事务 ID			
64	消耗时间		标记	
96	客户 IP			
128	你的 IP			
160	服务器 IP			
196	网关 IP			
228+	客户端 MAC 地址(16 字节)			
	服务器主机名(64 字节)			
	启动文件(128 字节)			
	选项			

图 6-17: DHCP 头部

操作代码 Opcode: DHCP 请求后者 DHCP 回复。

硬件类型 Hardware Type: 10MB 以太网、IEEE802、ATM 等。

硬件长度 Hardware Length: 硬件地址长度。

跳数 Hops: 中继代理用来帮助寻找 DHCP 服务器。

事务 ID: 用来匹配请求和响应的一个随机数。

消耗时间 Seconds Elasped: 客户端首次向服务器发出请求后的时间。

标记 Flags: 客户端能够接受的流量类型 (单播、广播、以及其他)。

你的 IP: 服务器为客户端提供的 IP。

DHCP Discover 发现阶段：

用户发送 DHCP Discover 消息请求地址，将自己的 mac 地址封装在 DHCP 的报文里进行广播寻找 DHCP Server，并表示自己需要获得一个 IP 地址。获取 IP 时，PC 会发送 DHCP Discover 广播报文。由于此时客户端 IP 被释放，源 ip 为：0.0.0.0；特别注意的是，PC 会随机出一个 Transaction

ID，如果之后收到的 OFFer 报文中的 Transaction ID 与 PC 模拟出的不同，PC 会将该 Offer 报文直接丢弃。

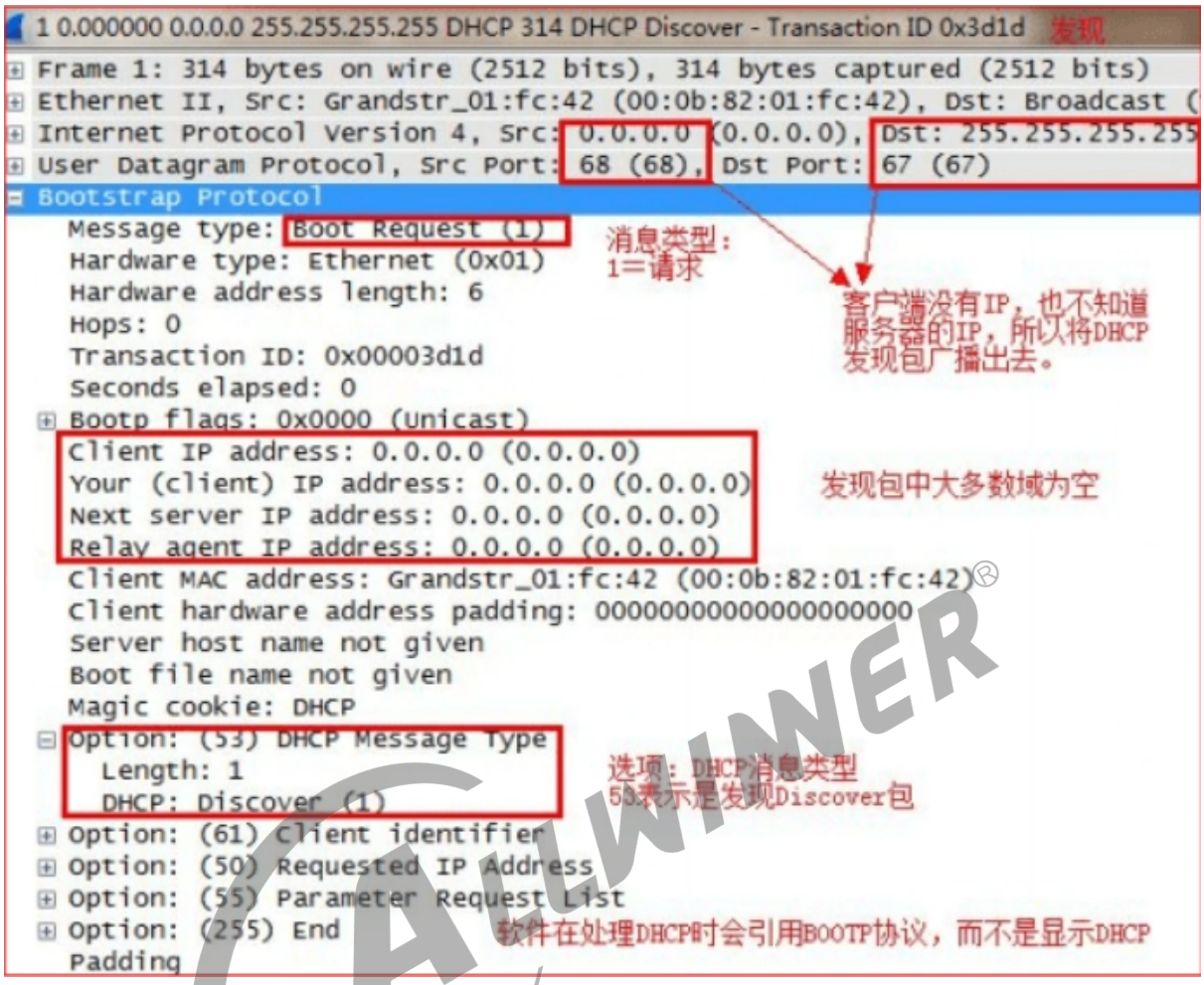


图 6-18: DHCP-Discover

DHCP Offer 提供阶段：

响应所受到的 DHCP Discover 消息，把准备提供的 IP 地址携带在 DHCP offer 消息中，并将此消息发送给客户端。

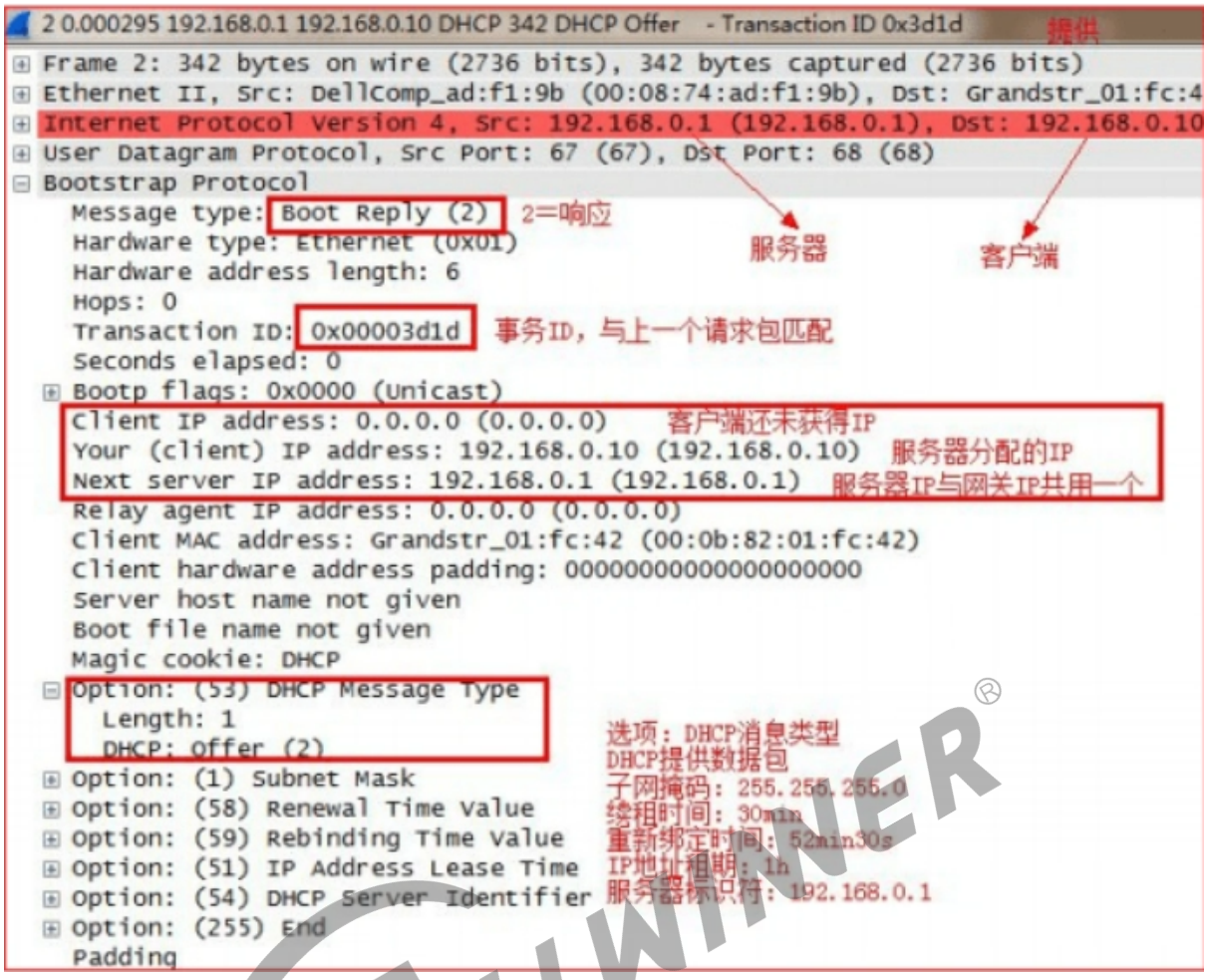


图 6-19: DHCP-Offer

DHCP Request 请求阶段：

在收到的所有 offer 中选择接受第一个到达的 Offer，向相应的 DHCP Server 发送 DHCP Request 消息，表示愿意接受该 offer。

图 6-20: DHCP-Request

DHCP ACK 确认阶段：

确认在提供阶段所提供的 IP 地址是否可以分配给客户端。如果可以，则发送 DHCP ACK 消息。表示 IP 地址分配成功；如果不可以，则发送 DHCP ACK 消息。表示 IP 地址分配失效。如果收到了 DHCP ACK 消息，就可以开始使用所分配的 IP 地址：如果收到了 DHCP NAK 消息，就表示 IP 地址获取失败，需要重新进入到发现阶段。

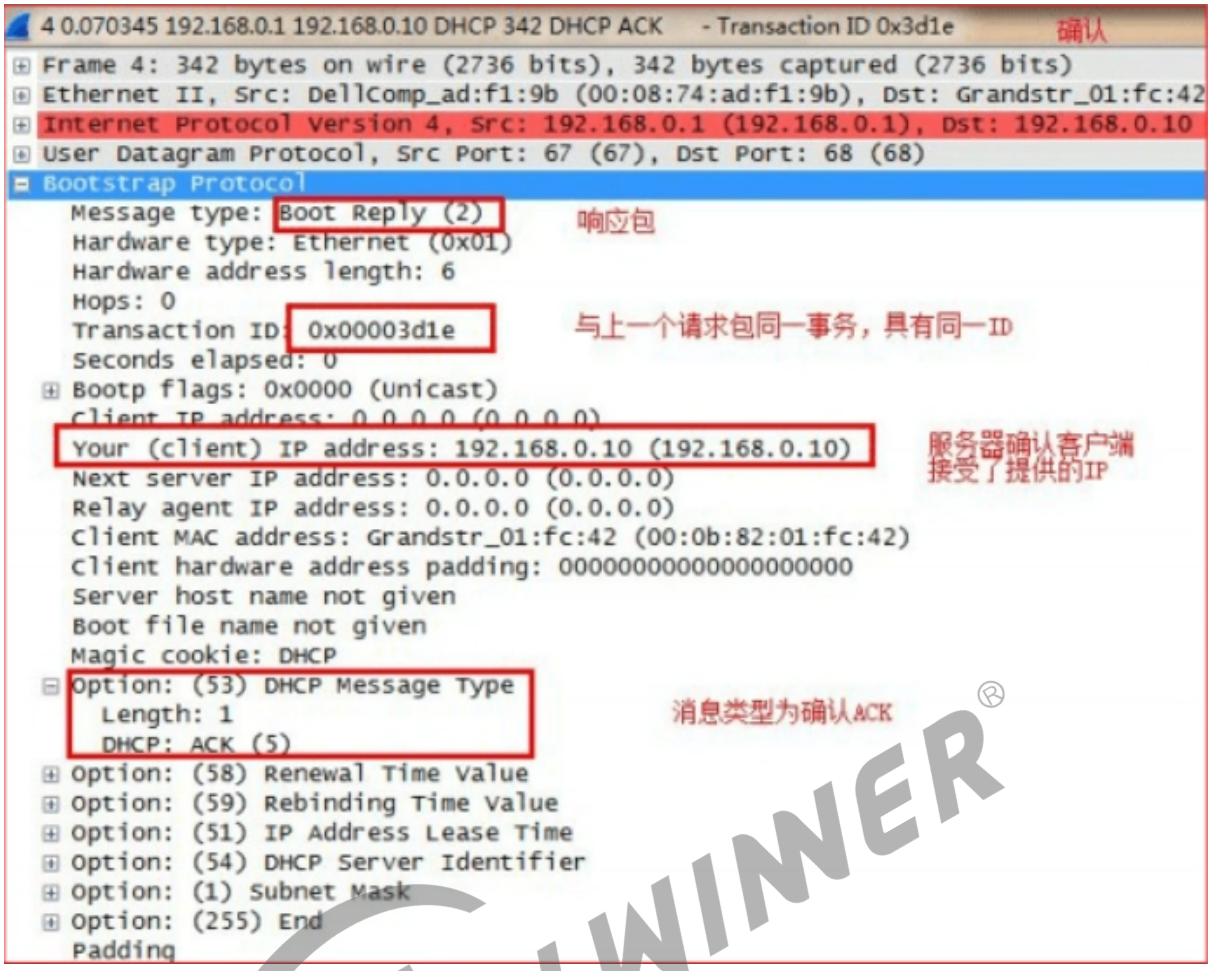


图 6-21: DHCP-ACK

6.6 ICMP 流程

网络控制消息协议 ICMP 原理

ICMP，全称为 Internet Control Message Protocol，即为因特网控制报文协议。

ICMP 既不是网络层协议，也不是传输层协议，但是通常 ICMP 被认为是 IP 层的一部分，所以一般把他归为网络层。

一个新搭建好的网络，往往需要先进行一个简单的测试，来验证网络是否畅通；但是 IP 协议并不提供可靠传输。如果丢包了，IP 协议并不能通知传输层是否丢包以及丢包的原因。

ICMP 的主要功能就是确认 IP 包是否成功到达目标地址以及通知在发送过程中 IP 包被丢弃的原因。

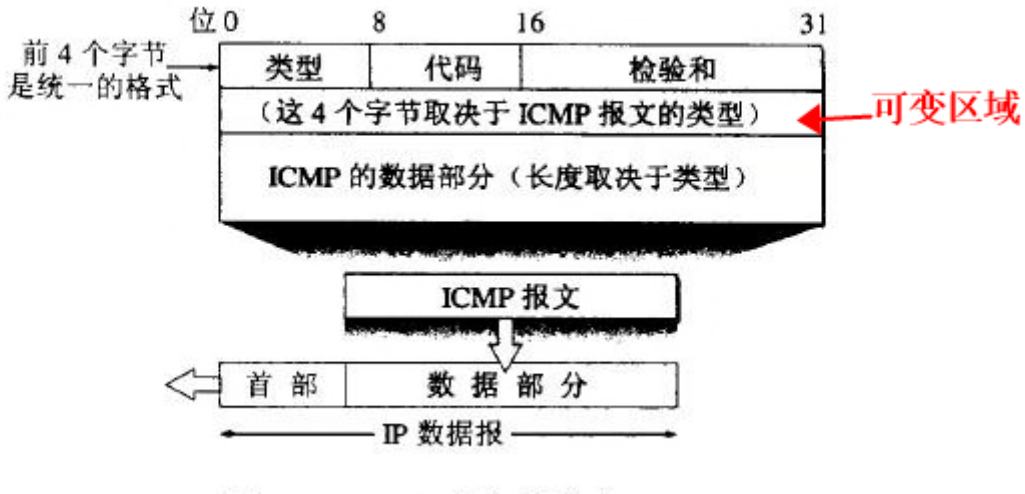


图 6-22: ICMP 报文

ICMP 报文的种类分为两种：ICMP 差错报告报文和 ICMP 询问报文。

报文类型	类型值	描述	说明
询问报文	0/8	Echo Request/Reply	Echo 请求/回复
差错报告报文	1/2/7	Unassigned	未赋值
差错报告报文	3	Destination Unreachable	目标不可达
差错报告报文	4	Source Quench(Deprecated)	报源抑制(弃用)
差错报告报文	5	Redirect	重定向
差错报告报文	6	Alternate Host Address （Deprecated）	修改主机地址(弃用)
询问报文	9	Router Advertisement/Solicitation	路由器公告/请求
差错报告报文	11	Time Exceeded	超时
差错报告报文	12	Parameter Problem	参数问题
询问报文	13/14	Timestamp/Timestamp Reply	时间戳/时间戳应答

图 6-23: ICMP 报文类型

No.	Time	Source	Destination	Protocol	Length	Info
46	7.69306900	192.168.100.155	220.181.57.217	ICMP	74	Echo (ping) request id=0x0001, seq=9/2304, ttl=128 (reply in 47)
47	7.71148200	220.181.57.217	192.168.100.155	ICMP	74	Echo (ping) reply id=0x0001, seq=9/2304, ttl=51 (request in 46)
54	8.69062500	192.168.100.155	220.181.57.217	ICMP	74	Echo (ping) request id=0x0001, seq=10/2560, ttl=128 (reply in 55)
55	8.70889700	220.181.57.217	192.168.100.155	ICMP	74	Echo (ping) reply id=0x0001, seq=10/2560, ttl=51 (request in 54)
62	9.69066300	192.168.100.155	220.181.57.217	ICMP	74	Echo (ping) request id=0x0001, seq=11/2816, ttl=128 (reply in 63)
63	9.70903400	220.181.57.217	192.168.100.155	ICMP	74	Echo (ping) reply id=0x0001, seq=11/2816, ttl=51 (request in 62)
73	10.6917340	192.168.100.155	220.181.57.217	ICMP	74	Echo (ping) request id=0x0001, seq=12/3072, ttl=128 (reply in 75)
75	10.7101140	220.181.57.217	192.168.100.155	ICMP	74	Echo (ping) reply id=0x0001, seq=12/3072, ttl=51 (request in 73)

图 6-24: PING 流程

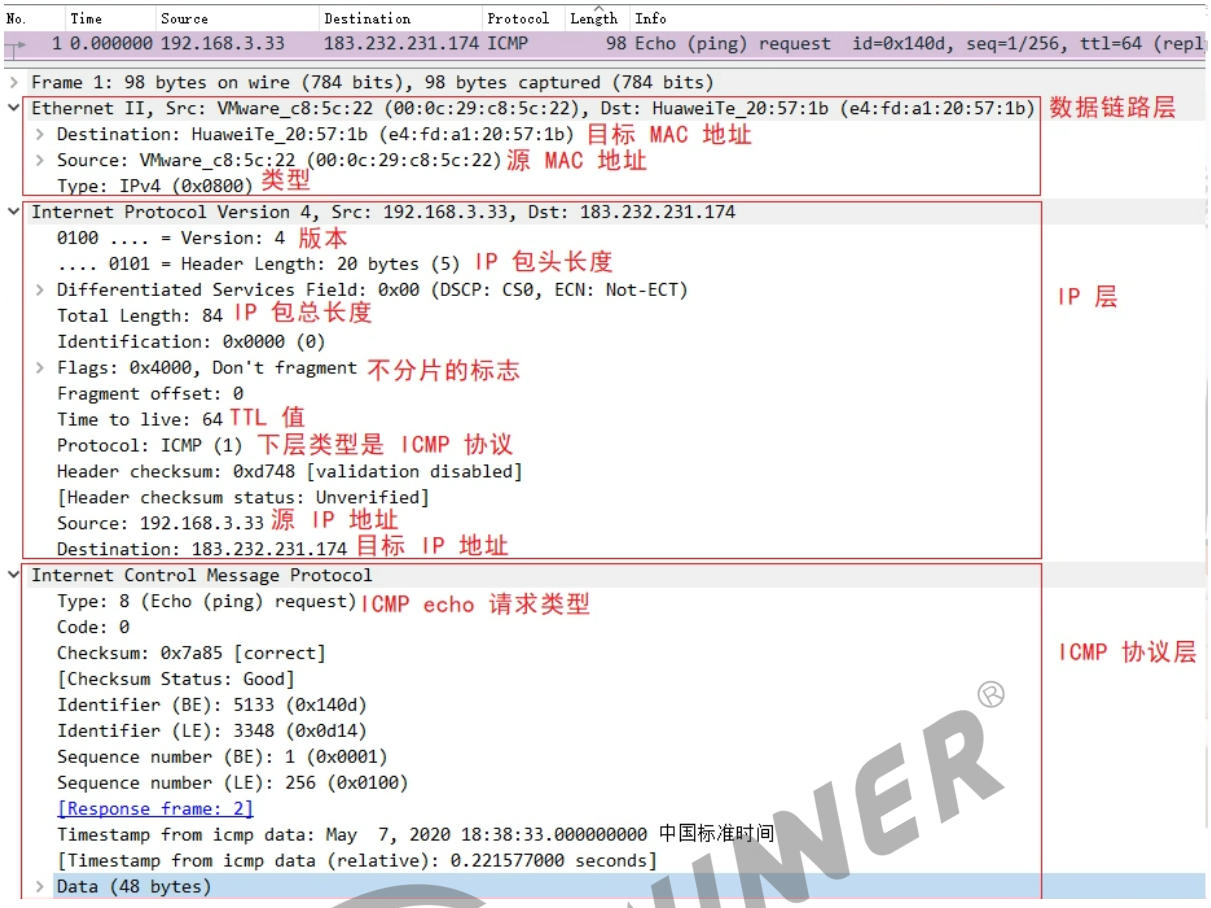


图 6-25: ICMP-Request 帧

6.7 TCP 流程

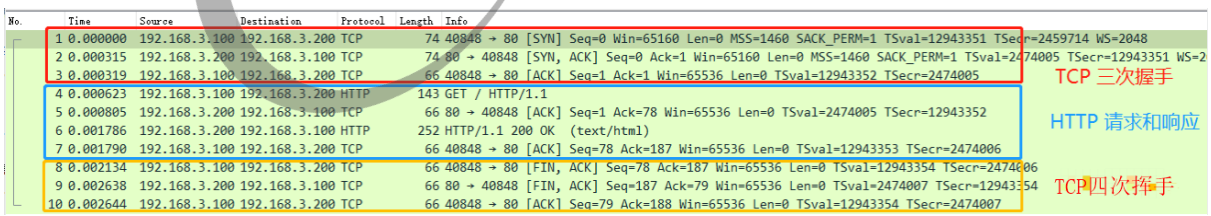


图 6-26: TCP 抓包分析

由抓包文件可以明显的将 TCP 的通信分为三个流程，TCP 三次握手，HTTP 请求响应，TCP 四次挥手。

6.7.1 TCP 三次握手

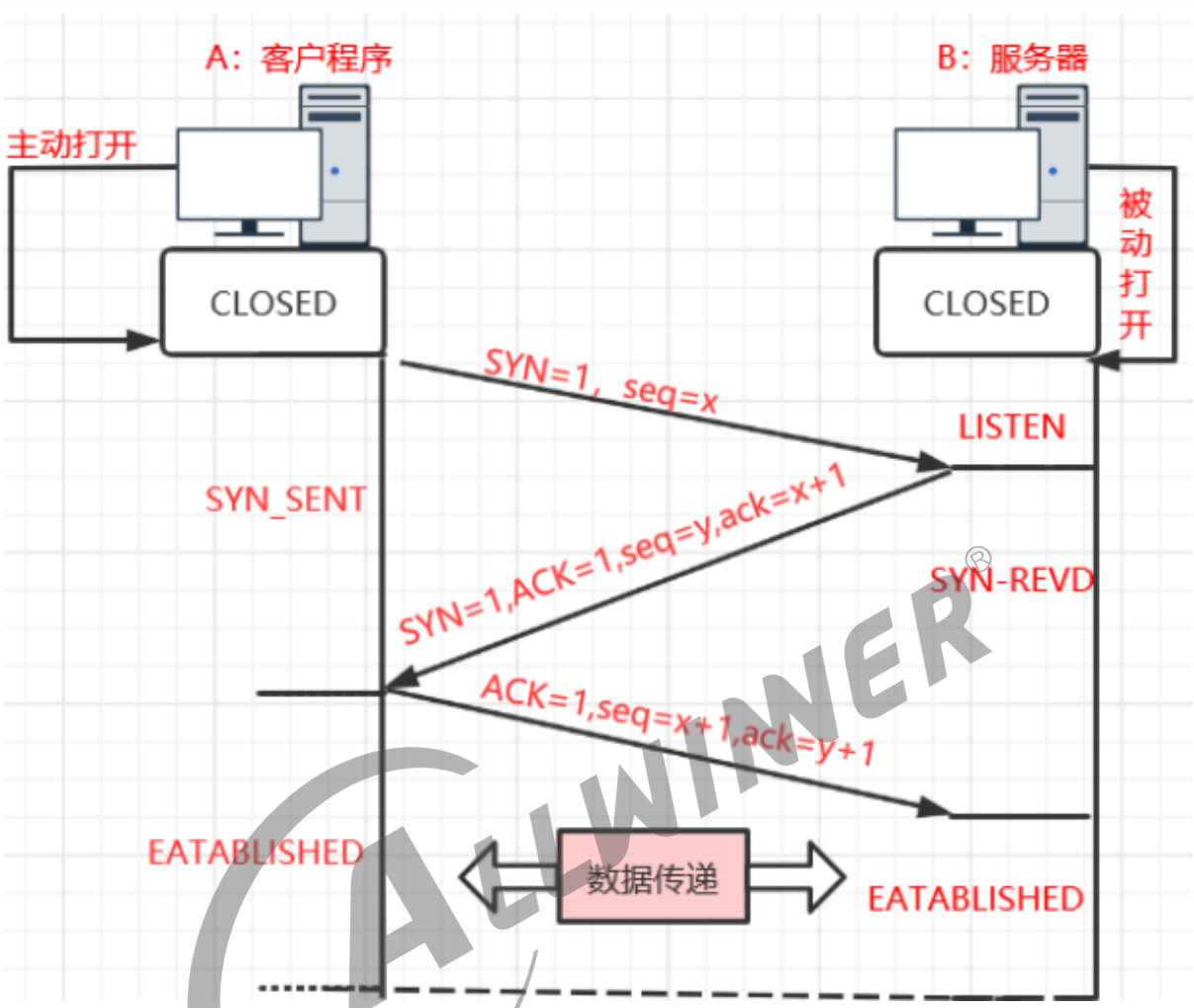


图 6-27: TCP 三次握手

第一次握手：

客户端执行 connect 函数，向服务器发送连接请求报文段，这时 TCP 报文首部的同步位 SYN=1。这时客户端进入 SYN_SENT(同步已发送) 状态，等待服务器的确认。

第二次握手：

服务器收到请求报文段后，如果同意建立连接，则向 A 发送确认，在确认报文段中 SYN=1，ACK=1。这时服务器就进入了 SYN_RCVD(同步收到) 状态客户端进入 ESTABLISHED 状态。

第三次握手：

客户端在收到服务器的确认过后，还要给服务器给出确认，确认报文段将 ACK 置 1。TCP 连接已经建立服务器都进入 ESTABLISHED 状态。完成三次握手，随后客户端与服务器之间就可以传输数据了。

三次握手的结果就是服务器知道了客户端发送数据的序列号，客户端也知道了服务器发送数据的序列号，并且双方都知道彼此知道了。

6.7.2 TCP 四次挥手

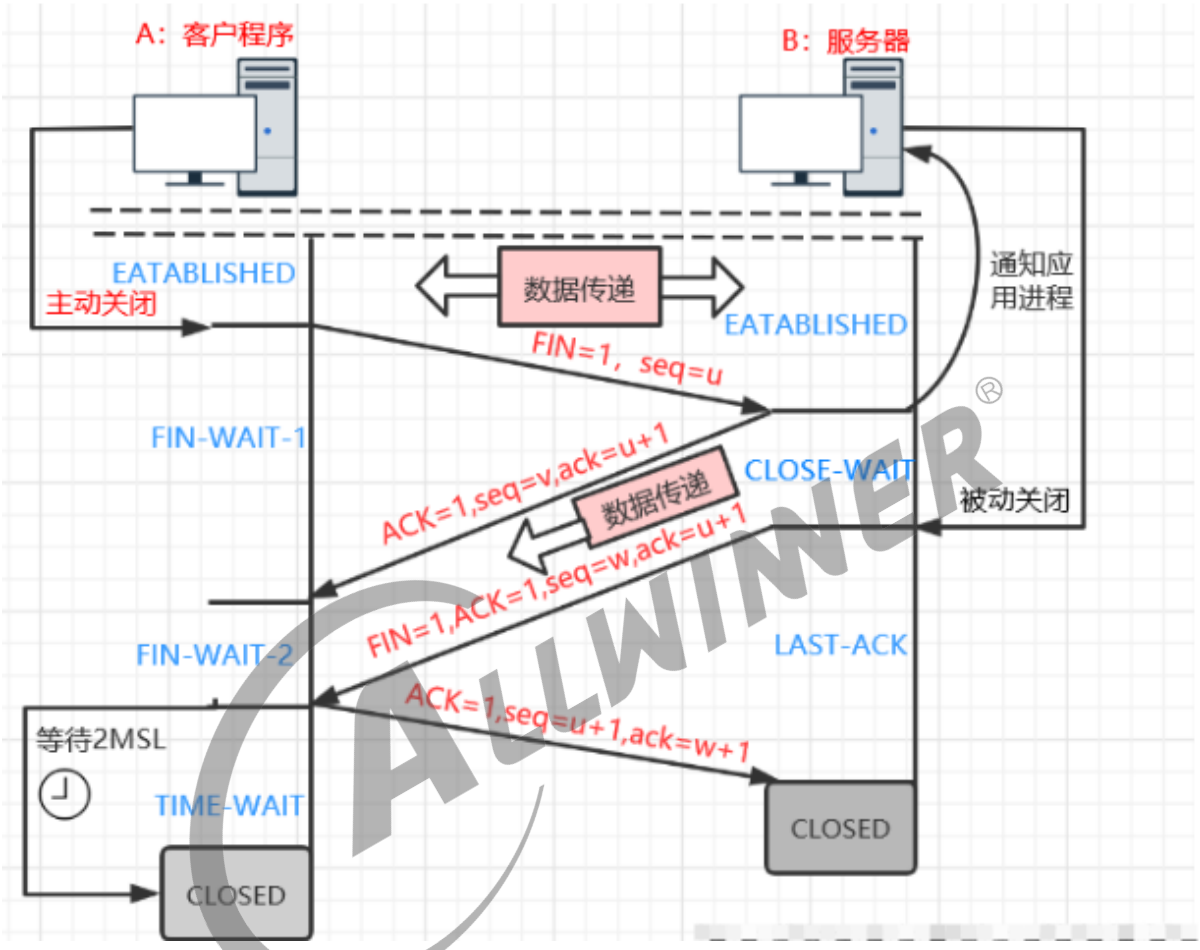


图 6-28: TCP 四次挥手

第一次挥手：

客户端进程先向 TCP 发出链接释放报文段，并停止再发送数据，主动关闭 TCP 连接。连接释放报文段首部 FIN=1。C 进入 FIN_WAIT_1 状态。

第二次挥手：

服务器收到连接释放报文段后即发出确认，把确认报文段首部 ACK 置 1。然后 S 进入 CLOSE_WAIT(等待关闭) 状态。C 收到 S 的确认后，就进入 FIN_WAIT_2(终止等待 2) 状态，等待服务器发出的连接释放报文段。

此时的 TCP 连接处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端仍要接收。也就是说，从服务器到客户端这个方向的连接并没有关闭，这个状态可能会持续

一些时间。

第三次挥手：

若 S 已经没有要向 C 发送的数据，其应用进程就通知 TCP 释放连接，这时 S 发出的连接释放报文段必须使 FIN=1。这时 S 就进入 LAST_ACK(最后确认) 状态，等待 C 的确认。

第四次挥手：

C 在收到 S 的连接释放报文段后，必须对此发出确认。在确认报文段中把 ACK 置 1。然后客户端进入到 TIME_WAIT(时间等待) 状态。

注意！！！

现在 TCP 连接还没有释放掉，必须经过时间等待计时器设置的 2MSL 后，A 才进入 CLOSE 状态。MSL 叫做最长报文段寿命，即报文段存活最大时间服务器收到客户端的确认报文段过后就会进入 close 状态。服务器在撤销相应的传输控制块 TCB 后，就结束了这次的 TCP 连接，B 结束 TCP 连接的时间要比客户端早一些。

6.7.3 Wireshark 分析 TCP 时常见问题

1)TCP out-of-order segment

TCP 存在问题。

Wireshark 判断 TCP out-of-order 是基于 TCP 包中 SEQ number 并非期望收到的下一个 SEQ number，则判断为 out-of-order。因此，出现 TCP out-of-order 时，很大可能是 TCP 存在乱序或丢包，导致接收端的 seq number 不连续。

如图 6-29，第 4 包数据，在客户端已经收到服务端的 SYN ACK 后，服务端再次发送了 SYN ACK，wireshark 将此包标记为 out-of-order。

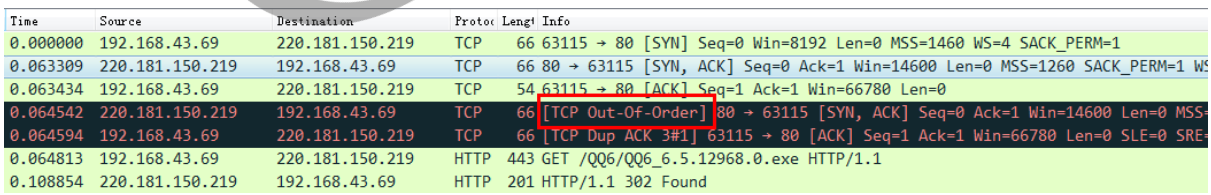


图 6-29: TCP 包乱序或丢包 1

如图 6-30，第 7 包数据，本应收到 seq number 为 1366882 的 TCP 包，但却收到了 1044834 的包，这包数据应该是晚到了，因此 wireshark 标记为 out-of-order。

Time	Source	Destination	Prot	Length	Seq num	Next seq num	Info
182.550…	74.125.218.151	176.90.47.210	TCP	1476	1362658	1364066	[TCP segment of a reassembled PDU]
182.550…	176.90.47.210	74.125.218.151	TCP	88	705		[TCP Dup ACK 3284#224] 57082 → 80 [ACK] Se
182.589…	74.125.218.151	176.90.47.210	TCP	1476	1364066	1365474	[TCP segment of a reassembled PDU]
182.590…	176.90.47.210	74.125.218.151	TCP	88	705		[TCP Dup ACK 3284#225] 57082 → 80 [ACK] Se
182.639…	74.125.218.151	176.90.47.210	TCP	1476	1365474	1366882	[TCP segment of a reassembled PDU]
182.639…	176.90.47.210	74.125.218.151	TCP	88	705		[TCP Dup ACK 3284#226] 57082 → 80 [ACK] Se
182.729…	74.125.218.151	176.90.47.210	TCP	1476	1044834	1046242	[TCP Out-Of-Order] [TCP segment of a reass
182.730…	176.90.47.210	74.125.218.151	TCP	80	705		57082 → 80 [ACK] Seq=705 Ack=1222242 Win=2

图 6-30: TCP 包乱序或丢包 2

如果抓包中出现大量的 out-of-order 包，则说明网络存在较大的 TCP 乱序或丢包。

2)TCP Previous segment not captured

前一个 TCP 分段没有抓到。

在 TCP 连接建立的时候，SYN 包里面会把彼此 TCP 最大的报文段长度，即 MSS 标志，一般都是 1460。如果发送的包比最大的报文段长度长的话就要分片了，被分片出来的包，就会被标记了“TCP segment of a reassembled PDU”，这些包分片存在同样的 ack number，且每个分片的 seq number 不同。

Protocol	Length	Seq num	Next seq num	Ack num	Info
TCP	68	1256		53505	36296 → 80 [ACK] Seq=1256 Ack=53505
TCP	68	1		1	[TCP Spurious Retransmission] 60965
TCP	1476	53505	54913	1256	[TCP segment of a reassembled PDU]
TCP	1476	54913	56321	1256	[TCP segment of a reassembled PDU]
TCP	1476	56321	57729	1256	[TCP segment of a reassembled PDU]
TCP	1476	57729	59137	1256	[TCP segment of a reassembled PDU]
TCP	1476	59137	60545	1256	[TCP segment of a reassembled PDU]
TCP	1476	60545	61953	1256	[TCP segment of a reassembled PDU]

图 6-31: TCP 分段 1

这些分片正常应该是连续接收的，即前一个分片指示的 next seq number 即为下一个收到的分片的 seq number。假如收到的下一个分片的 seq number 与上一个比不连续的话，wireshark 就会将该分片标记为 TCP Previous segment not captured。如图 6-32，ack number 为 705 的 TCP 包被分为多个分片发送，其中有一个长度为 1408 的分片没有被抓到。

Protocol	Length	Seq num	Next seq num	Ack num	Info
TCP	1476	1039202	1040610	705	[TCP segment of a reassembled PDU]
TCP	68	705		10406…	57082 → 80 [ACK] Seq=705 Ack=1040610 Win=41
TCP	1476	1040610	1042018	705	[TCP segment of a reassembled PDU]
TCP	1476	1042018	1043426	705	[TCP segment of a reassembled PDU]
TCP	68	705		10434…	57082 → 80 [ACK] Seq=705 Ack=1043426 Win=41
TCP	1476	1043426	1044834	705	[TCP segment of a reassembled PDU]
TCP	1476	1046242	1047650	705	[TCP Previous segment not captured] [TCP se
TCP	80	705		10448…	57082 → 80 [ACK] Seq=705 Ack=1044834 Win=41
TCP	1476	1047650	1049058	705	[TCP segment of a reassembled PDU]

图 6-32: TCP 分段 2

需要注意的是，前一个分片丢失，有可能是网络中确实丢失了，或者晚到了，也有可能是 wireshark 本身并没有抓到。

3)TCP Spurious Retransmission

TCP 虚假重传。

当抓到 2 次同一包数据时，wireshark 判断网络发生了重传，同时，wireshark 抓到初传包的反馈 ack，因此 wireshark 判断初传包实际并没有丢失，因此称为虚假重传。基于 wireshark 的判断机制，如果抓包点在客户端的话，虚假重传一般为下行包，因为这时，客户端在收到服务端的下行包后发送反馈 ack，并被 wireshark 抓到，但很有可能服务端未收到此反馈 ack，RTO 超时，触发服务端重传。如图 6-33，红框内出现了 2 次虚假重传，其中绿色的两条为同一包数据，粉色的两条为同一包数据。

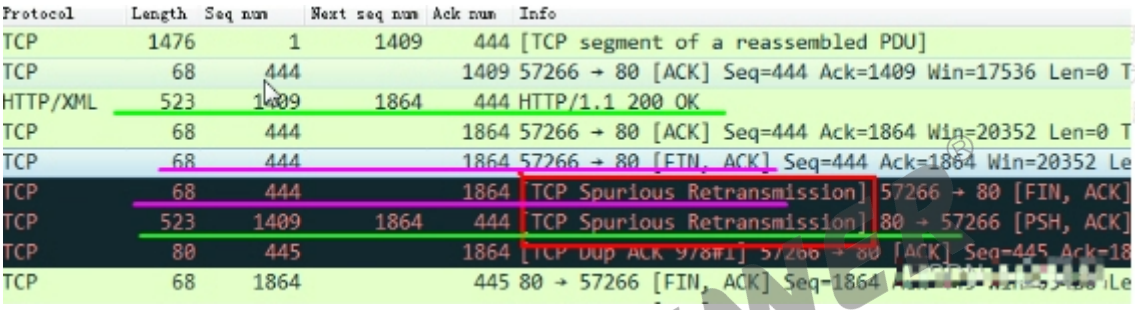


图 6-33: TCP 虚假重传

4)TCP Retransmission

TCP 重传。

当抓到 2 次同一包数据时，wireshark 判断发生了重传，同时 wireshark 没有抓到初传包的反馈 ack，因此，wireshark 判定重传有效，标记为 TCP Retransmission。基于软件的判定机制，如果抓包点在客户端的话，TCP 重传一般为上行包，因为这时，客户端并没有收到服务端的反馈 ack，无从知晓服务端是否收到了初传包，RTO 超时后触发客户端重传。此时存在 2 种情况，即

1) 服务端收到了初传包，只是下发的反馈 ack 丢包，客户端没有收到。

2) 服务端没有收到初传包，因此也就没有下发反馈 ack。

对于第一种情况，如果抓包点在服务端的话，wireshark 很有可能就会把来自客户端的重传包标记为 TCP Spurious Retransmission。

如图 6-34，红线的 TCP 包为重传包，wireshark 为该包添加了重传原因，是由于 TRO 超时导致，以及初传包序号 45，并给出了当前的 RTO 超时时间。

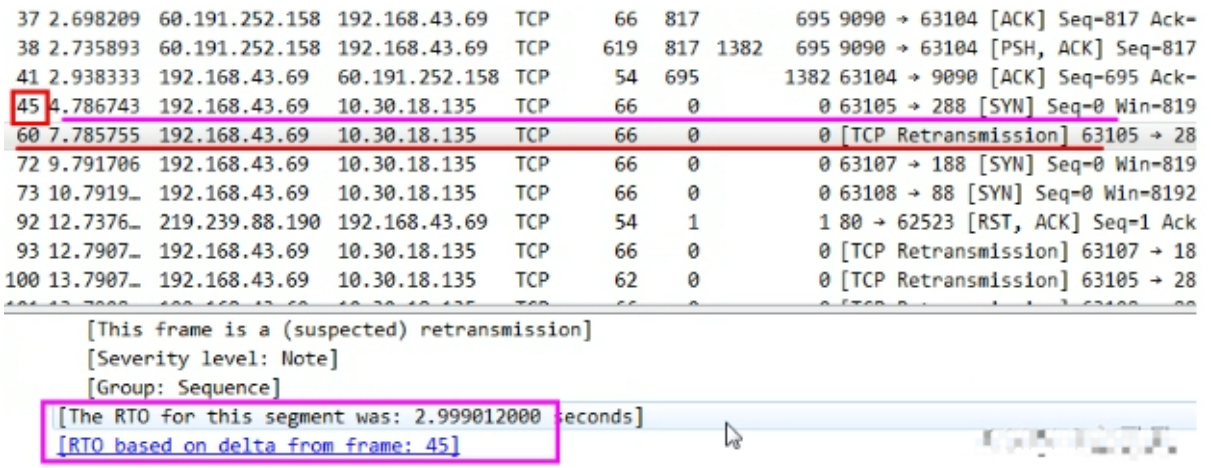


图 6-34: TCP 重传

5)TCP fast Retransmission

TCP 快速重传。

TCP 协议设定了快速重传机制以避免过多的慢启动对传输速率的影响。快速重传通过接收到 3 个或 3 个以上重复的 ack 反馈触发。快速重传不需要等待 RTO 超时。如图 6-35。

325 包，客户端向服务端反馈 ack=133251，说明下一个期望收到服务端 seq=133251 的包；

326 包，服务端向客户端发送了 seq=135771 的数据包，与客户端的期望不符，因此客户端在 327 包重传了 ack=133251 的包，再次申明期望收到 seq=133251 的包。Wireshark 将重复 ack 标记为 TCP Dup ACK，# 后边指明为第几次重传。

328 包，服务端向客户端发送了 seq=137031 的数据包，仍然与客户端期望不符，客户端在 329 包再次重传 ack=133251 的包。

330 包，服务端收到 3 次重复 ack，触发快速重传，重传了 seq=133251 的 TCP 分片。

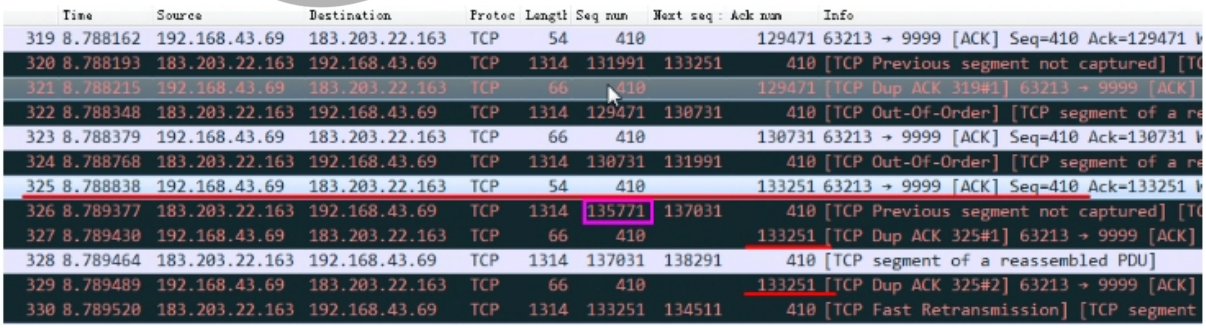


图 6-35: TCP 快速重传

6)TCP Dup ACK

重复 ack。

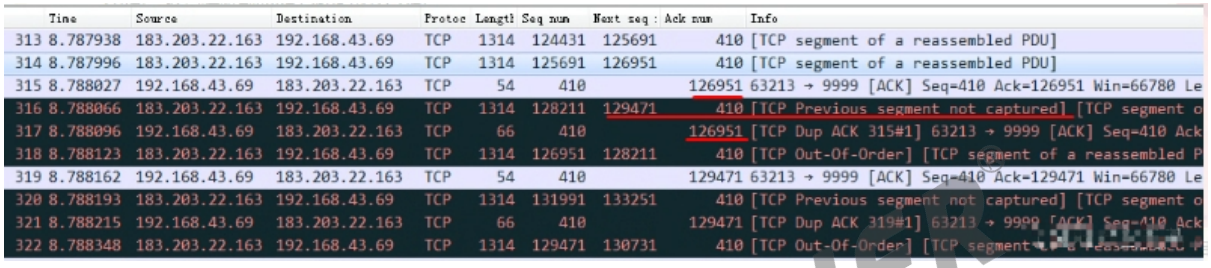
当网络中存在乱序或者丢包时，将会导致接收端接收到的 seq number 不连续。此时接收端会向发送端回复重复 ack，ack 值为期望收到的下一个 seq number。重复 ack 数大于等于 3 次将会触发快速重传。如图 6-36。

315 包，客户端向服务端发送 ack=126951 的反馈，期望下一包收到 seq=126951 的 TCP 包。但下一包接收到的却是 seq=128211 的 TCP 包，由于 seq 不连续，wireshark 将该报标记为 TCP Previous segment not captured。

317 包，客户端向服务端重复发送 ack=126951 的包，第一次重发，# 后边带 1。

318 包，客户端收到 seq=126951 的 TCP 包。

319 包，截止到 seq=129471 的所有 TCP 包全部收到，因此客户端回复了 ack=129471 的反馈。



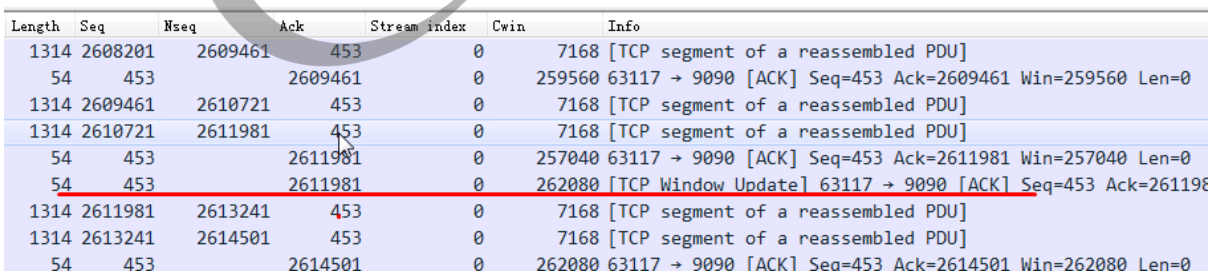
	Time	Source	Destination	Protoc	Length	Seq num	Next seq	Ack num	Info
313	8.787938	183.203.22.163	192.168.43.69	TCP	1314	124431	125691	410	[TCP segment of a reassembled PDU]
314	8.787996	183.203.22.163	192.168.43.69	TCP	1314	125691	126951	410	[TCP segment of a reassembled PDU]
315	8.788027	192.168.43.69	183.203.22.163	TCP	54	410		126951	63213 → 9999 [ACK] Seq=410 Ack=126951 Win=66780 Le
316	8.788066	183.203.22.163	192.168.43.69	TCP	1314	128211	129471	410	[TCP Previous segment not captured] [TCP segment o
317	8.788096	192.168.43.69	183.203.22.163	TCP	66	410		126951	[TCP Dup ACK 315#1] 63213 → 9999 [ACK] Seq=410 Ack
318	8.788123	183.203.22.163	192.168.43.69	TCP	1314	126951	128211	410	[TCP Out-Of-Order] [TCP segment of a reassembled P
319	8.788162	192.168.43.69	183.203.22.163	TCP	54	410		129471	63213 → 9999 [ACK] Seq=410 Ack=129471 Win=66780 Le
320	8.788193	183.203.22.163	192.168.43.69	TCP	1314	131991	133251	410	[TCP Previous segment not captured] [TCP segment o
321	8.788215	192.168.43.69	183.203.22.163	TCP	66	410		129471	[TCP Dup ACK 319#1] 63213 → 9999 [ACK] Seq=410 Ack
322	8.788348	183.203.22.163	192.168.43.69	TCP	1314	129471	130731	410	[TCP Out-Of-Order] [TCP segment

图 6-36: TCP 重复确认

7)TCP window update

TCP 窗口更新。

当接收方的 TCP window 发生突变时，接收方通过 TCP window update 消息告知对方当前的接收窗口大小。如图 6-37，TCP window Update 同时携带了反馈 ack，ack 序列号与前一个反馈 ack 序列号相同，但这并不是重复 ack。



Length	Seq	Nseq	Ack	Stream index	Cwin	Info
1314	2608201	2609461	453	0	7168	[TCP segment of a reassembled PDU]
54	453		2609461	0	259560	63117 → 9090 [ACK] Seq=453 Ack=2609461 Win=259560 Len=0
1314	2609461	2610721	453	0	7168	[TCP segment of a reassembled PDU]
1314	2610721	2611981	453	0	7168	[TCP segment of a reassembled PDU]
54	453		2611981	0	257040	63117 → 9090 [ACK] Seq=453 Ack=2611981 Win=257040 Len=0
54	453		2611981	0	262080	[TCP Window Update] 63117 → 9090 [ACK] Seq=453 Ack=261198
1314	2611981	2613241	453	0	7168	[TCP segment of a reassembled PDU]
1314	2613241	2614501	453	0	7168	[TCP segment of a reassembled PDU]
54	453		2614501	0	262080	63117 → 9090 [ACK] Seq=453 Ack=2614501 Win=262080 Len=0

图 6-37: TCP 窗口更新

8)TCP acked unseen segment

反馈 ACK 指向了一个未知的 TCP 片段。

这个意思是说 ACK 反馈的是一个 wireshark 上不存在的 TCP 包。很可能是 wireshark 漏抓了这个包，但却抓到了对端反馈的该报的 ack 包。如图 6-38，标记为 ack unseen segment 的包反馈的

ack=2721，看着像是反馈的 seq=1361 的包，但其实这个 ack 还反馈了 seq=1 的包，由于 seq=1 的包没有抓到，因此 wireshark 将反馈 ack 标记为 ack unseen segment。从下面的图还可知，由于对端已经反馈了 ack=2721，说明发端发送的 seq=1 的包，对端也收到了，只不过 wireshark 可能漏抓了而已。

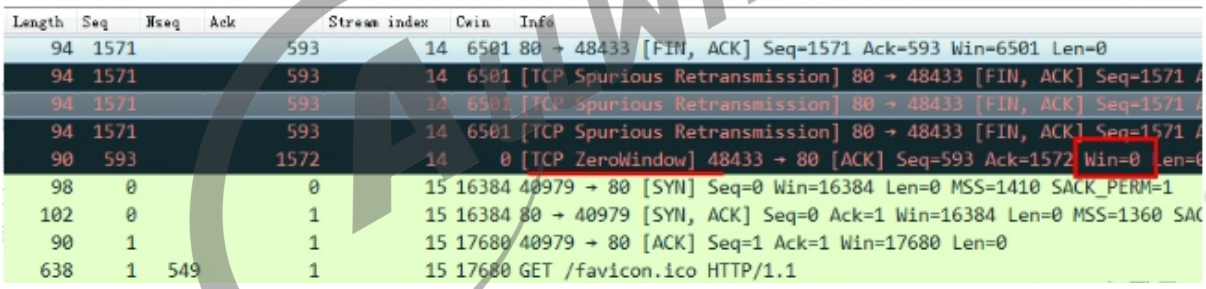
Protoc	Length	Seq	Nseq	Ack	Stream index	Cwin	Info
GTP…	148	475	529	228	0	65308	Response: 125 Data connection already open; Tran
GTP…	117	213	240	479	1	17202	Request: STOR BOhf201011261700.cnw
GTP…	98	0		0	3	16384	1080 → 2367 [SYN] Seq=0 Win=16384 Len=0 MSS=1460
GTP…	94	479		240	1	65296	21 → 1078 [ACK] Seq=479 Ack=240 Win=65296 Len=0
GTP…	1450	1361	2721	1	2	17680	[TCP Previous segment not captured] FTP Data: 13
GTP…	90	228		529	0	17152	1077 → 21 [ACK] Seq=228 Ack=529 Win=17152 Len=0
GTP…	94	1		2721	2	65535	[TCP ACKed unseen segment] 2366 → 1079 [ACK] Seq
GTP…	1450	2721	4081	1	2	17680	FTP Data: 1360 bytes
GTP…	958	4081	4949	1	2	17680	FTP Data: 868 bytes

图 6-38: TCP 确认错误

9)TCP ZeroWindow

TCP 滑动窗口为 0。

当发送端发包速率大于接收端的接收速率时，会造成接收端 TCP window 越来越小，当接收端在反馈 ack 时携带的 window size=0 时，wireshark 标记 TCP Zero window。此时发送端将暂停发送数据，直到收到接收端 window size!=0 的标志。



Length	Seq	Nseq	Ack	Stream index	Cwin	Info
94	1571		593	14	6501	80 → 48433 [FIN, ACK] Seq=1571 Ack=593 Win=6501 Len=0
94	1571		593	14	6501	[TCP Spurious Retransmission] 80 → 48433 [FIN, ACK] Seq=1571
94	1571		593	14	6501	[TCP Spurious Retransmission] 80 → 48433 [FIN, ACK] Seq=1571
94	1571		593	14	6501	[TCP Spurious Retransmission] 80 → 48433 [FIN, ACK] Seq=1571
90	593		1572	14	0	[TCP ZeroWindow] 48433 → 80 [ACK] Seq=593 Ack=1572 Win=0 Len=0
98	0		0	15	16384	40979 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1410 SACK_PERM=1
102	0		1	15	16384	80 → 40979 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1360 SAC
90	1		1	15	17680	40979 → 80 [ACK] Seq=1 Ack=1 Win=17680 Len=0
638	1	549	1	15	17680	GET /favicon.ico HTTP/1.1

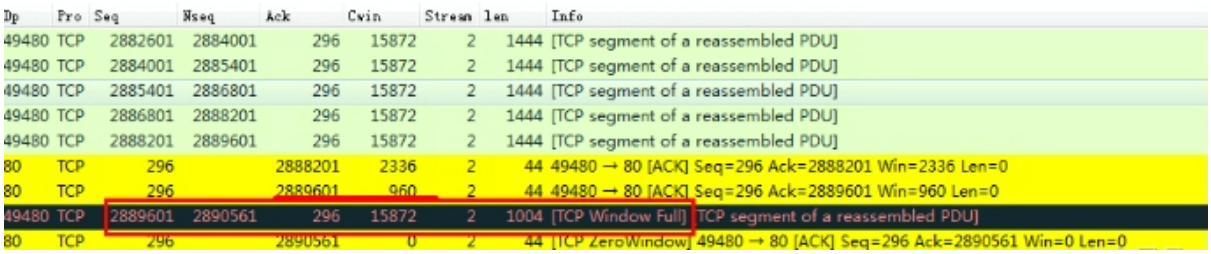
图 6-39: TCP 滑动窗口为 0

10)TCP window full

TCP window 满。

是指的发送端发送的数据已经达到的接受窗口的上限。发送端暂停发送，等待新的接收窗口的通告。

如图 6-40，客户端向服务端发送的 ack 反馈，期望下一包收到的 seq=288961，但接收窗口仅有 960，服务端在收到 ack 后发送了 960 字节的数据，TCP window full。注意，len=1004 是整个 IP 包的长度，需要减去 IP 头 44 字节，即 960 字节。



Dp	Pro	Seq	Nseq	Ack	Cwin	Stream	Len	Info
49480	TCP	2882601	2884001	296	15872	2	1444	[TCP segment of a reassembled PDU]
49480	TCP	2884001	2885401	296	15872	2	1444	[TCP segment of a reassembled PDU]
49480	TCP	2885401	2886801	296	15872	2	1444	[TCP segment of a reassembled PDU]
49480	TCP	2886801	2888201	296	15872	2	1444	[TCP segment of a reassembled PDU]
49480	TCP	2888201	2889601	296	15872	2	1444	[TCP segment of a reassembled PDU]
80	TCP	296		2888201	2336	2	44	49480 → 80 [ACK] Seq=296 Ack=2888201 Win=2336 Len=0
80	TCP	296		2889601	960	2	44	49480 → 80 [ACK] Seq=296 Ack=2889601 Win=960 Len=0
49480	TCP	2889601	2890561	296	15872	2	1004	[TCP Window Full] [TCP segment of a reassembled PDU]
80	TCP	296		2890561	0	2	44	[TCP ZeroWindow] 49480 → 80 [ACK] Seq=296 Ack=2890561 Win=0 Len=0

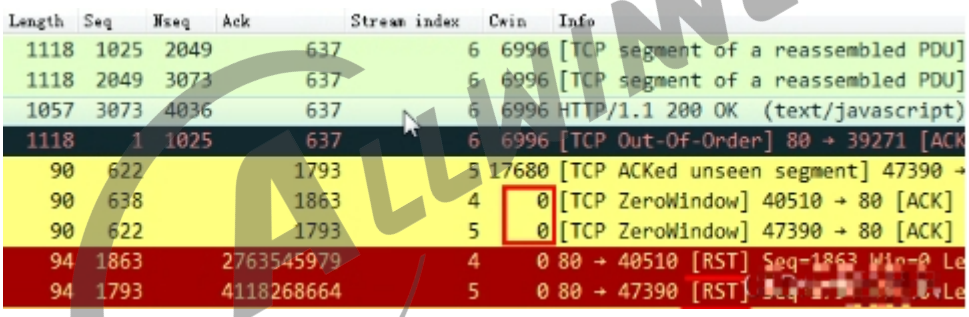
图 6-40: TCP 窗口满

11)TCP RST

TCP 重置。

是 TCP 协议结束异常连接的一种方式，通过 flog 中的 reset=1 标记。当 TCP 连接无法通过正常的 4 次挥手结束时，一方可以通过发送携带 reset 标志的 TCP 包结束 TCP 连接。

如图 6-41，发送方通过 2 个 TCP 流发送数据，截图中，接收方首先向发送方反馈了 TCP window=0，让发送方暂缓发送数据，之后紧接着发送了 TCP RST 标记，释放了 TCP 连接。猜测可能接收方程序突然崩溃了，导致缓存区数据没法清空，之后接收方系统发送了 TCP reset 释放 TCP 连接。



Length	Seq	Nseq	Ack	Stream index	Cwin	Info
1118	1025	2049	637	6	6996	[TCP segment of a reassembled PDU]
1118	2049	3073	637	6	6996	[TCP segment of a reassembled PDU]
1057	3073	4036	637	6	6996	HTTP/1.1 200 OK (text/javascript)
1118	1	1025	637	6	6996	[TCP Out-Of-Order] 80 → 39271 [ACK
90	622		1793	5	17680	[TCP ACKed unseen segment] 47390 →
90	638		1863	4	0	[TCP ZeroWindow] 40510 → 80 [ACK]
90	622		1793	5	0	[TCP ZeroWindow] 47390 → 80 [ACK]
94	1863	2763545979		4	0	80 → 40510 [RST] Seq=1863 Win=0 Le
94	1793	4118268664		5	0	80 → 47390 [RST]

图 6-41: TCP 连接重置

6.8 HTTPS 流程

整个完整的 HTTPS 请求的大体过程如下：

1)TCP 三次握手。

2) 使用 TLSv1.2 进行 SSL 握手。

3) 使用握手协商好的密钥对 HTTP 进行加密传输。

4)TCP 四次挥手。

抓包分析一下详细流程：

1)TCP 的三次握手 (SYN，SYNC ACK，ACK)

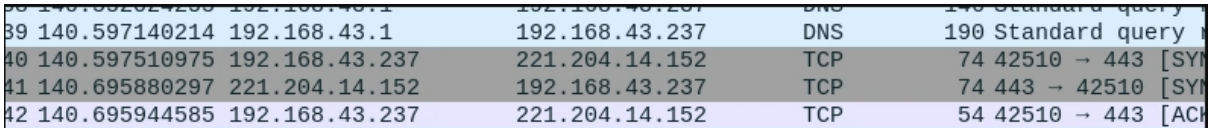


图 6-42: HTTPS 流程 1-TCP 三次握手

前两个报文 DNS 解析，后面是 TCP 建立连接的过程，本例中客户端首先发起连接，客户端向服务器发送 SYN 报文，服务器接收到之后回复 SYN ACK 确认，之后客户端再向服务器发送 ACK 确认。通过三次交互，一个 TCP 连接就建立起来了。

需要注意的是，如果按照 HTTP 协议的规定，建立 TCP 连接之后，就可以正式开始通信了，客户端可以构建 HTTP 请求，来请求服务器上的页面，服务器也会按照要求进行响应。但是这个过程是完全不进行加密的，并没有安全性可言。

这部分属于 TCP 的范畴，下面我们才真正开始讨论 TLS 1.2。

2)Client Hello

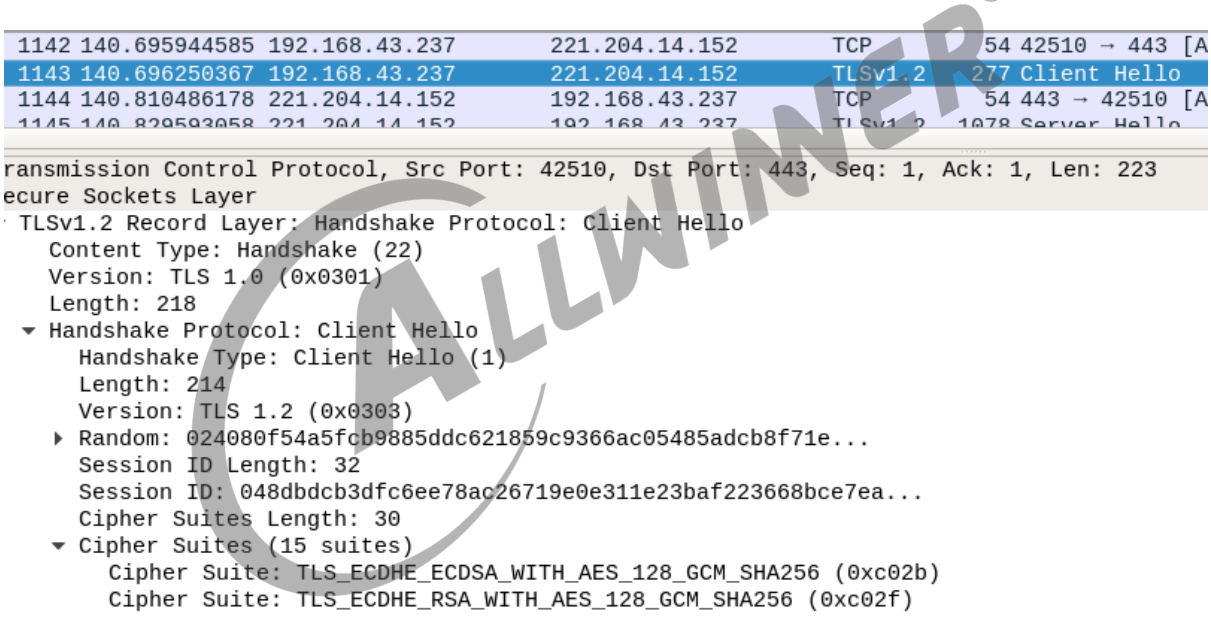


图 6-43: HTTPS 流程 2-client-hello

客户端发送的，属于 TLS 握手协议 (TLS handshake) 的一部分，其内容包括客户端的一个 Unix 时间戳 (GMT Unix Time)、一些随机的字节 (Random Bytes)，还包括了客户端接受的算法类型 (Cipher Suites)。

这是最基本的部分，同时还有其他的扩展参数，这里就不做介绍了。

Client Hello 的目的就类似于 SYN，客户端对服务器公布自己支持的算法等等，同时也附带请求服务器证书的意思。这里不验证客户端证书，所以 Client Hello 就只有这些内容。

3)Server Hello

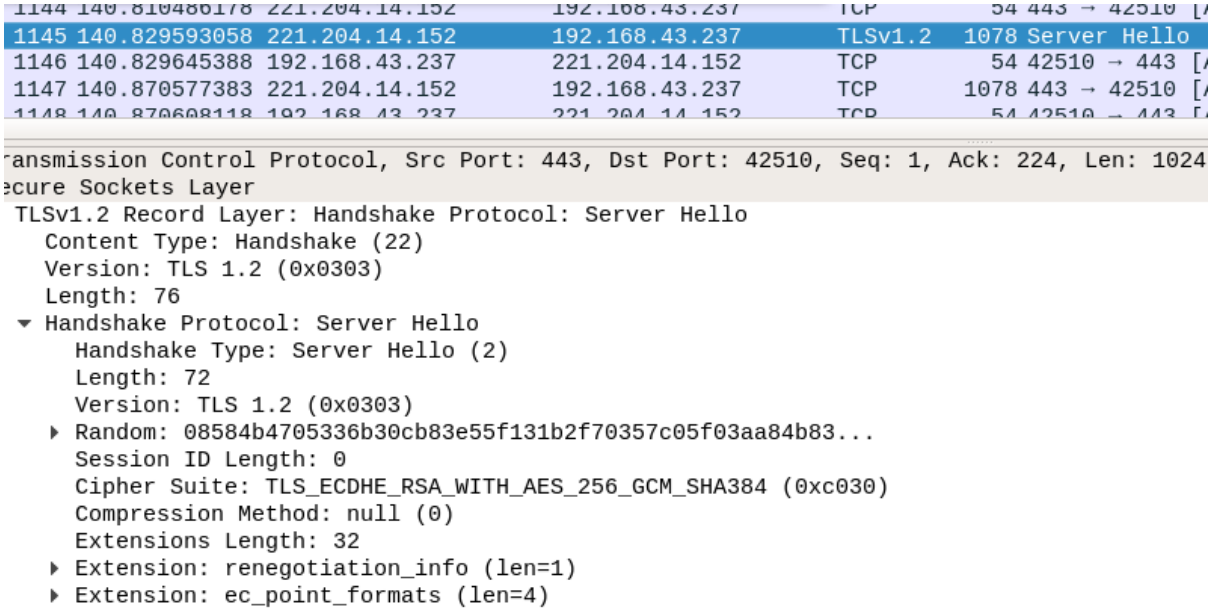


图 6-44: HTTPS 流程 3-server-hello

服务器发送的，同样属于 TLS handshake，内容包括服务器的 GMT Unix Time 以及 Random Bytes，和上面类似就不再解释。这时候服务器会将自己支持的算法类型发送给客户端，以便于客户端进行验证。

Server Hello 目的就是服务器对客户端公布自己的算法，以便于后续操作。

可以看出 S 选择了 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 加密组件，解释如下：

密钥交换算法，用于决定客户端与服务器之间在握手的过程中如何认证，用到的算法包括 RSA，Diffie-Hellman，ECDH，PSK 等，这里选择了 ECDHE。

加密算法，用于加密消息流，该名称后通常会带有两个数字，分别表示密钥的长度和初始向量的长度，比如 DES 56/56、RC2 56/128、RC4 128/128、AES 128/128、AES 256/256。这里选择了 AES。

报文认证信息码 (MAC) 算法，用于创建报文摘要，确保消息的完整性 (没有被篡改)，算法包括 MD5，SHA 等。这里选择了 SHA384。

PRF(伪随机数函数)，用于生成 “master secret”。

S 还发送了 32 字节随机数 s。

4)Certificate

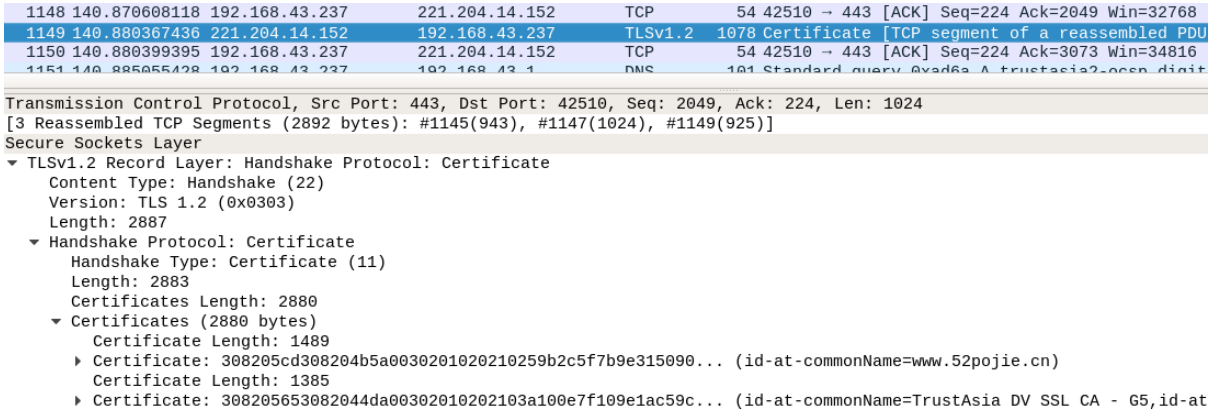


图 6-45: HTTPS 流程 4-certificate1

第一个 cert 是 52pojie 网站的证书，第二个 cert 是颁发者 trustasia 机构的证书。

图 6-46: HTTPS 流程 4-certificate2

这里可以获得证书的详细信息。

服务器或者客户端发送的，依旧属于 TLS handshake，它一般和 Server Hello 在同一个 TCP 报文中传送。显然的，这里服务器将自己的数字证书发送给客户端，客户端就会进行证书验证，如果不通过的话，客户端会中断握手的过程。

如果也要求客户端提供证书的话，Client Hello 后面也会紧跟着该报文，形式完全一样，就不再解释了。

5)Server Key Exchange 和 Server hello done

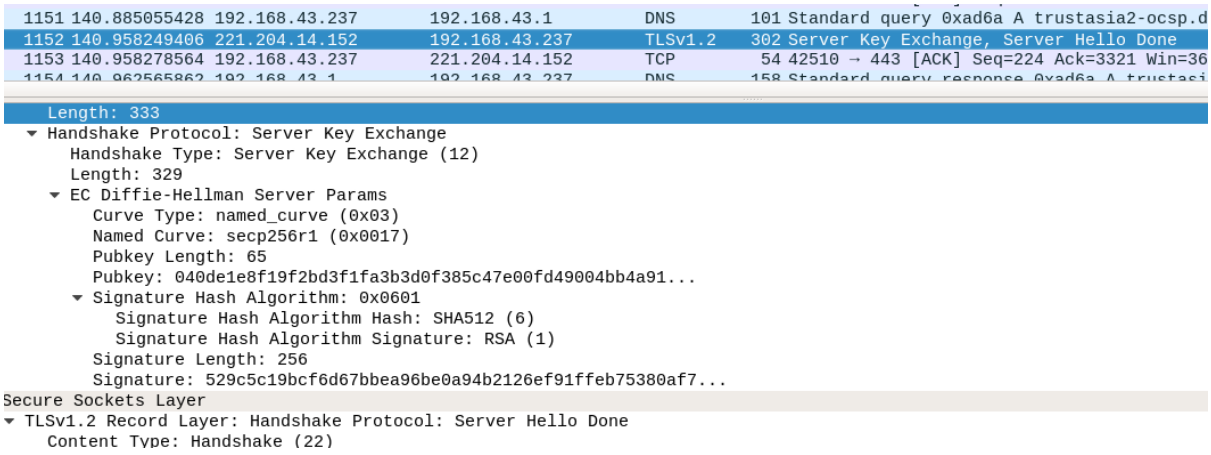


图 6-47: HTTPS 流程 5-ServerKeyExchange 和 Serverhellodone

可以看出使用 ECDH 密钥交换算法，指定椭圆曲线 secp256r1，还有发送了 DH 算法协商的公钥给 C。

服务器发送的，属于 TLS handshake，一般也和 Server Hello 与 Certificate 在一个 TCP 报文中。服务器将自己的公钥参数传输给了客户端，因为使用的是 ECDHE，这里传输的内容有：椭圆曲线域参数，以及公钥的值。这个就是密钥交换协议的一部分，最终协商出密钥来。

服务器发送的，属于 TLS handshake，一般也和 Server Hello、Certificate 和 Server Key Exchange 在一个 TCP 报文中，介于 Server Hello 和 Server Hello 之间的是服务器想要给客户端的东西。所以这里面客户端没有发送 Client Hello Done。

6)Client Key Exchange 和 Change Cipher Spec

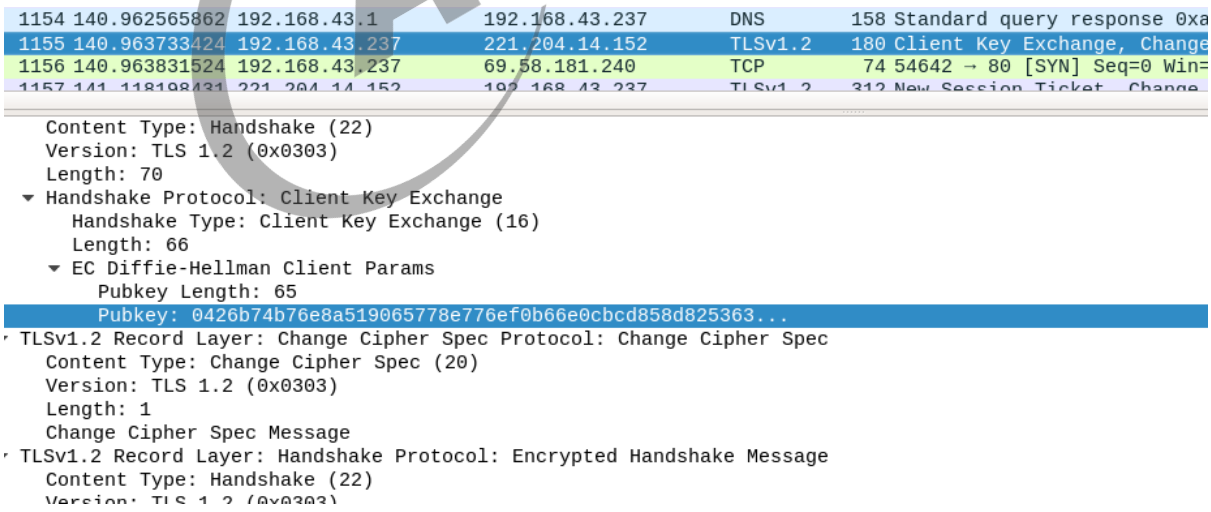


图 6-48: HTTPS 流程 6-ClientKeyExchange 和 ChangeCipherSpec

这里 C 发送了 DH 算法协商的公钥给 S，以及加密了握手消息给 S 进行验证。

客户端发送的，属于 TLS handshake，客户端收到了服务器的证书进行验证，如果验证通过了，就会继续密钥交换的过程，向服务器发送自己的公钥参数。待服务器收到之后进行数学计算，就

可以协商出密钥了，这里客户端实际上已经计算出了密钥。

7)server change cipher spec

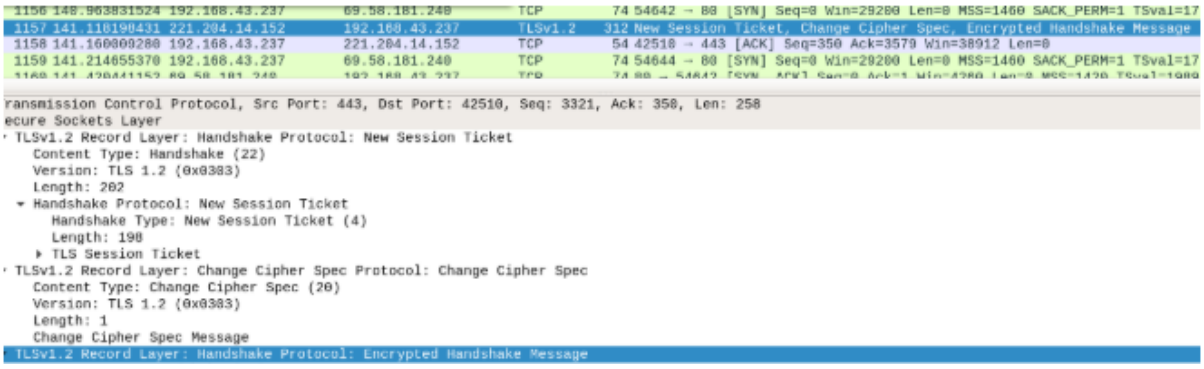


图 6-49: HTTPS 流程 7-serverchangecipherspec

服务端使用 Ticket 方式存储 session 状态，在 Server Change Cipher Spec 之前就需要发送 New Session Ticket 消息，这部分就不细说了。这里 S 加密握手消息给 C 进行验证。

客户端或者服务器发送的，属于 TLS handshake，紧跟着 Key Exchange 发送，代表自己生成了新的密钥，通知对方以后将更换密钥，使用新的密钥进行通信。

8)Encrypted Handshake Message

客户端或者服务器发送的，属于 TLS handshake，也是紧跟着 Key Exchange 发送。这里是进行一下测试，一方用自己的刚刚生成的密钥加密一段固定的消息发送给对方，如果密钥协商正确无误的话，对方应该可以解密。

这段加密内容的明文一般是协议中规定好的，双方都知道。

9)New Session Ticket

服务器发送的，属于 TLS handshake，服务器给客户端一个会话，用处就是在一段时间之内 (超时时间到来之前)，双方都以刚刚交换的密钥进行通信。从这以后，加密通信正式开始。

10)Application Data

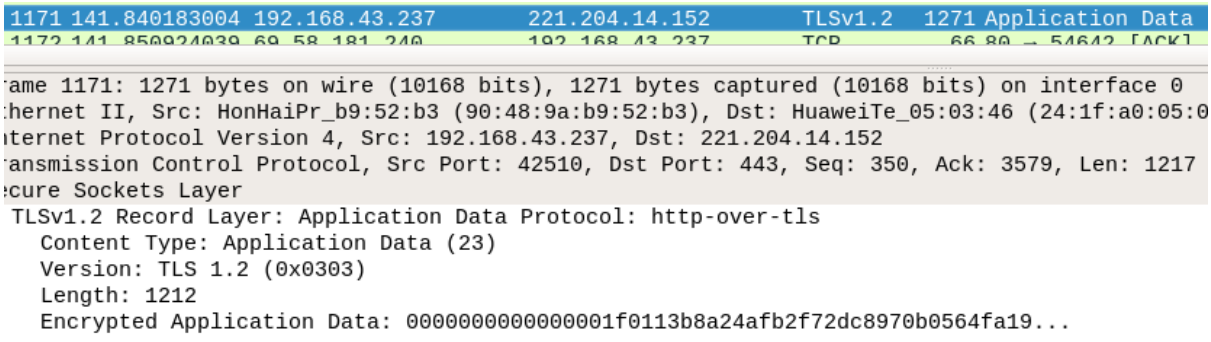


图 6-50: HTTPS 流程 10-Application-Data

应用层的数据，是加密的，使用密钥交换协议协商出来的密钥加密。因为我们的 Wireshark 不知道服务器的私钥，所以这段通信是安全的，我们在 Wireshark 中也无法解密这段信息。

11)Encrypted Alert

客户端或服务器发送的，意味着加密通信因为某些原因需要中断，警告对方不要再发送敏感的数据。

著作权声明

商标声明

免责声明