

实验报告

计算机网络 Computer Networking

报告标题: 协议分析——ARP 协议

学号: 19240212

姓名: 华博文

日期: 2025 年 9 月 24 日

一 实验目的

本实验旨在掌握 Wireshark 的启动、网卡选择、数据包捕获与筛选操作, 熟悉其主窗口三部分结构及功能, 同时理解 HTTP 协议的工作流程并能分析 HTTP 数据包在数据链路层、网络层、传输层及应用层的关键字段信息; 掌握 ARP 协议实现 IP 地址到 MAC 地址映射的核心功能、ARP 报文的结构(字段组成、分层特点)及“请求——响应”机制, 学会使用 C++ 语言按照 ARP 协议格式解析指定十六进制数据包并提取规范输出 ARP 报文关键字段, 最终加深对计算机网络分层模型(数据链路层、网络层、传输层、应用层)的理解, 验证“协议封装与解封装”的实际过程.

二 实验内容简要描述

1. Wireshark 与 HTTP 协议分析: 通过终端启动 Wireshark, 选择目标网卡(如 eth0), 启动抓包后用 Chrome 浏览器访问南京师范大学官网, 页面加载完成后停止抓包, 使用 http 筛选器筛选 HTTP 报文, 分析报文的分层协议信息(源 / 目的 MAC、源 / 目的 IP、TCP 端口、HTTP 请求 / 响应内容等);
2. ARP 协议分析: 在 Wireshark 中使用 arp 筛选器显示 ARP 报文, 分析 ARP 报文的分层结构(仅数据链路层与 ARP 协议层), 并提取 ARP 请求 / 响应报文的硬件类型、协议类型、操作码、发送方 / 目标 IP 与 MAC 等字段;
3. ARP 数据包解析: 基于给定 ARP 十六进制数据, 用 C++ 编写解析代码, 按指定格式输出 9 个关键字段(硬件类型、协议类型、硬件地址长度等).

三 实验步骤与结果分析

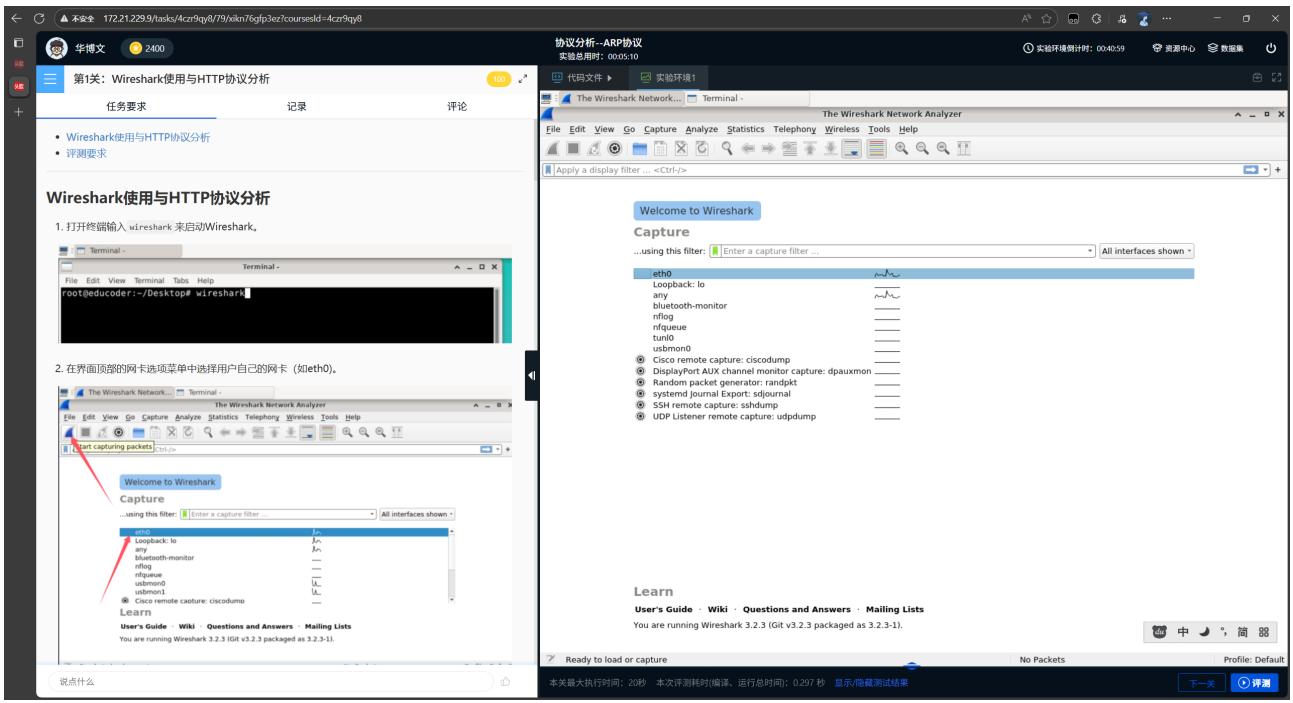
1 Wireshark 使用与 HTTP 协议分析

1.1 启动 Wireshark

打开 Linux 终端, 输入命令 `wireshark` 并回车, 启动 Wireshark 网络分析工具.

1.2 选择捕获网卡

在 Wireshark 顶部的网卡列表中, 选择当前设备使用的网卡 `eth0`(而非回环网卡 `lo`), 确保能捕获外网通信数据.

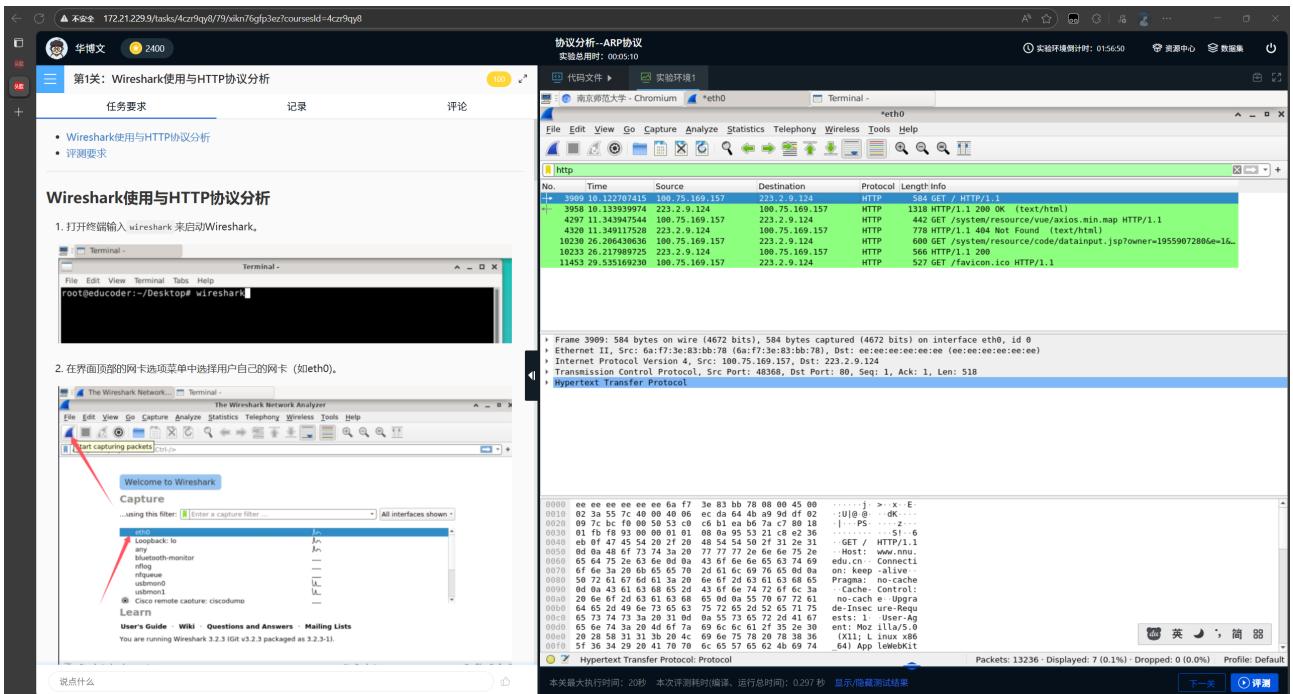


1.3 启动浏览器与数据包捕获

点击桌面 Chrome 浏览器图标打开浏览器，随后点击 Wireshark 工具栏中的“捕获”按钮（红色圆形图标），开始数据包捕获，界面底部显示 Capturing from eth0。在 Chrome 地址栏输入 `http://www.nnu.edu.cn/`，回车后等待页面完整加载。

1.4 停止捕获与筛选 HTTP 报文

页面加载完成后，点击 Wireshark 的“停止捕获”按钮（灰色方形图标），结束抓包。在 Wireshark 顶部的 Apply a display filter 输入框中输入 `http` 并回车，报文列表仅显示 HTTP 协议相关报文。



1.5 分析 Wireshark 主窗口结构

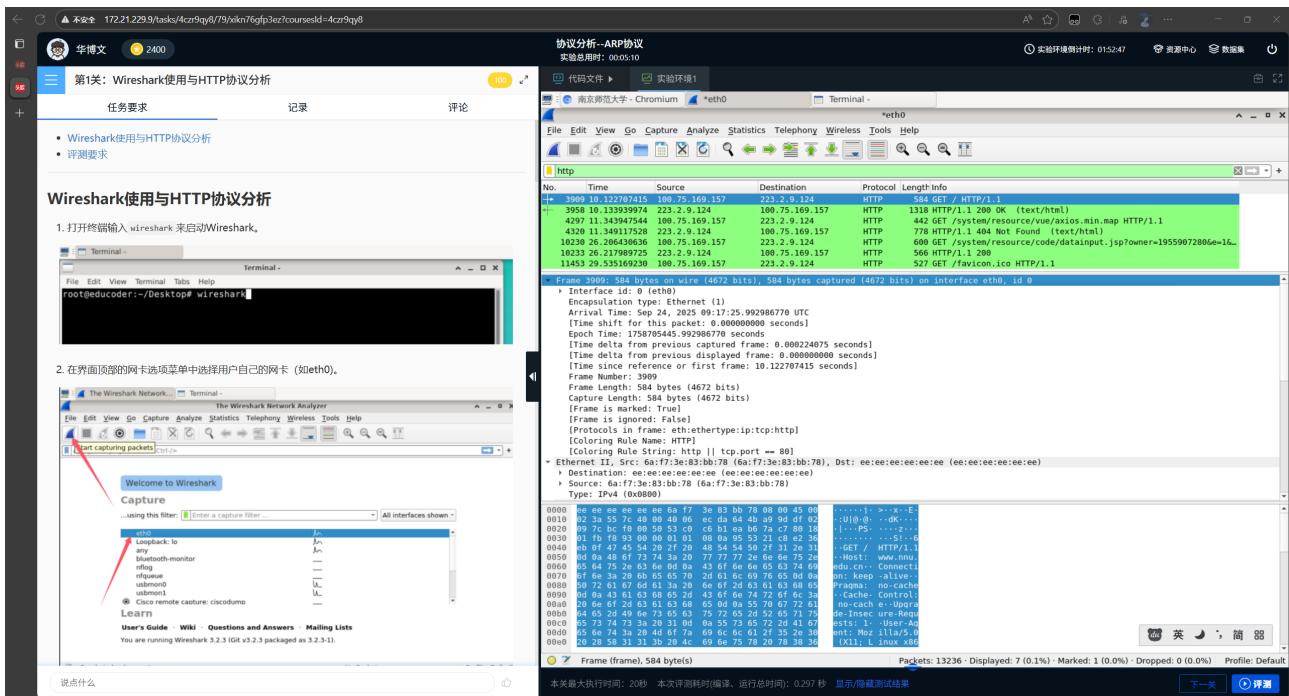
Wireshark 主窗口分为三部分：

- 数据包列表 (顶部): 显示报文序号 (No.)、捕获时间 (Time)、源 IP (Source)、目的 IP (Destination)、协议 (Protocol)、长度 (Length)、简要信息 (Info). 例如 No.3909 报文的 Source 为 100.75.169.157, Destination 为 223.2.9.124, Info 为 GET / HTTP/1.1.
- 数据包详细信息 (中部): 按分层协议展开单个报文内容, 从数据链路层 (Ethernet II) 到应用层 (HTTP).
- 数据包原始信息 (底部): 以十六进制 (左侧) 和 ASCII 码 (右侧) 显示报文原始字节, 如 47 45 54 对应 ASCII 的 GET.

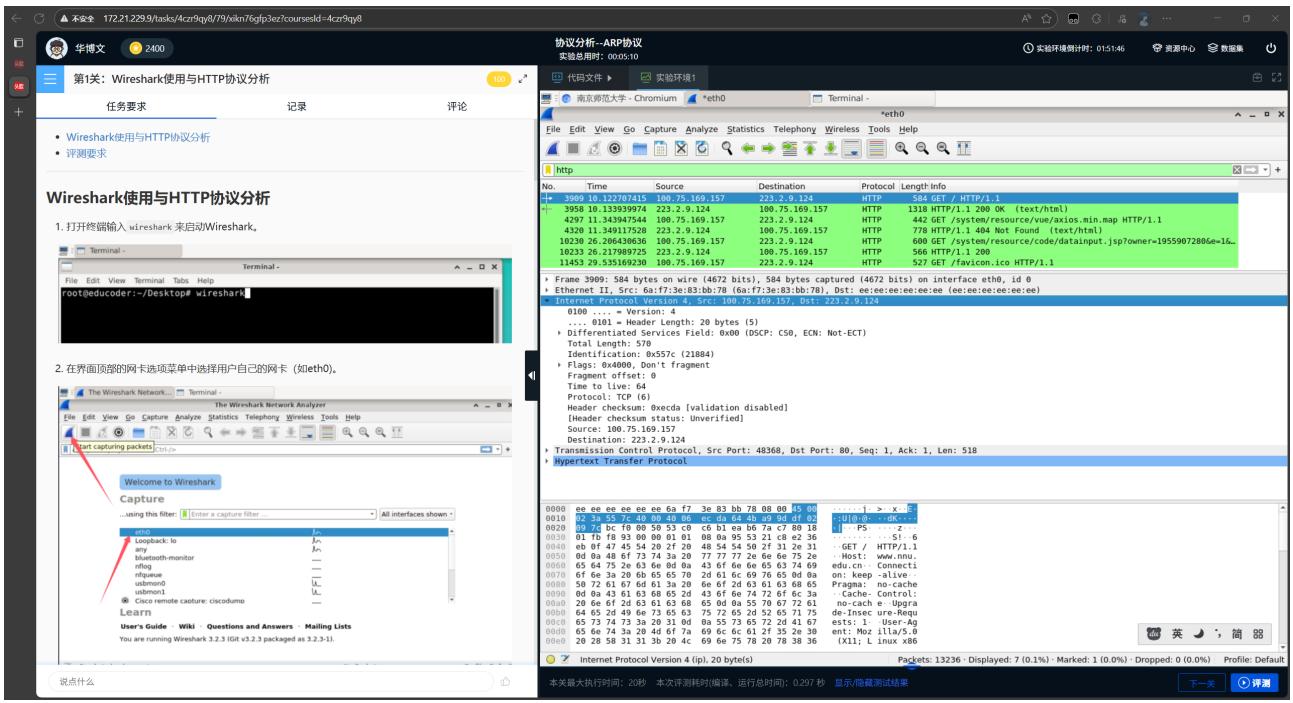
1.6 HTTP 报文分层字段分析

以 No.3909 报文为例：

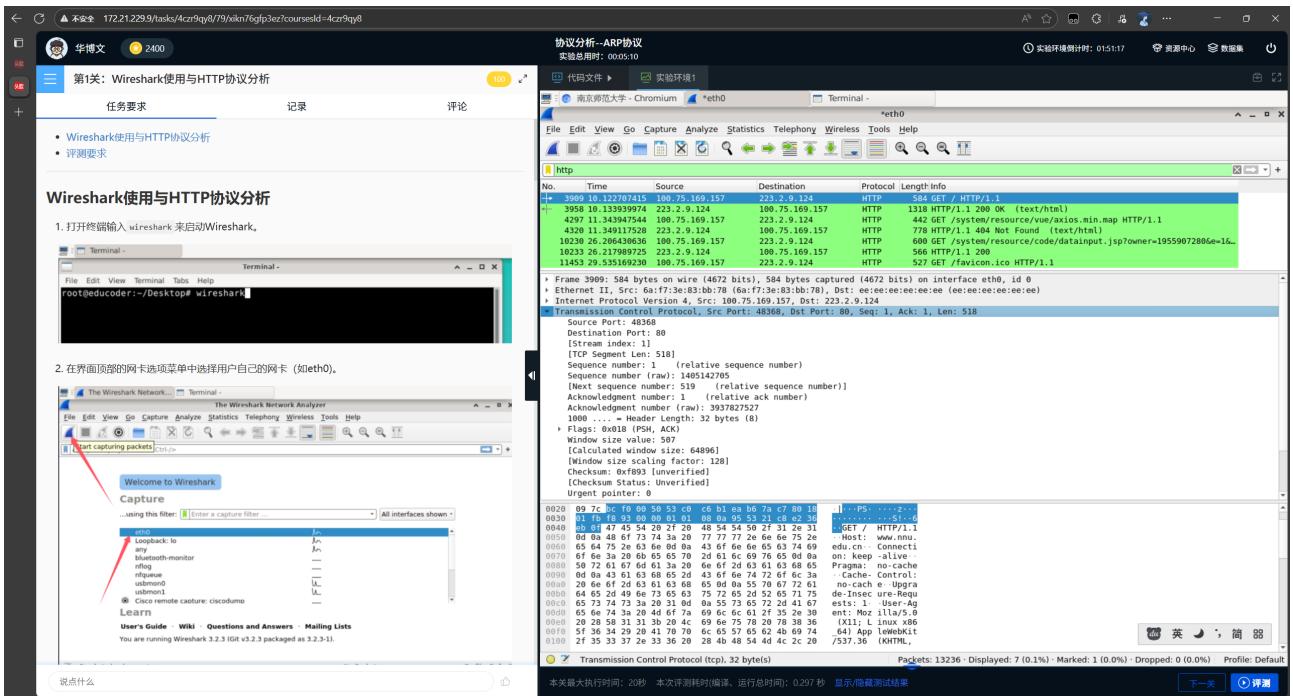
- 数据链路层 (Ethernet II): 展开 Ethernet II 字段, 可见源 MAC 地址 (Src) 为 6a:f7:3e:83:bb:78, 目的 MAC 地址 (Dst) 为 ee:ee:ee:ee:ee:ee (广播地址), 类型 (Type) 为 IPv4 (0x0800, 标识上层为 IP 协议).



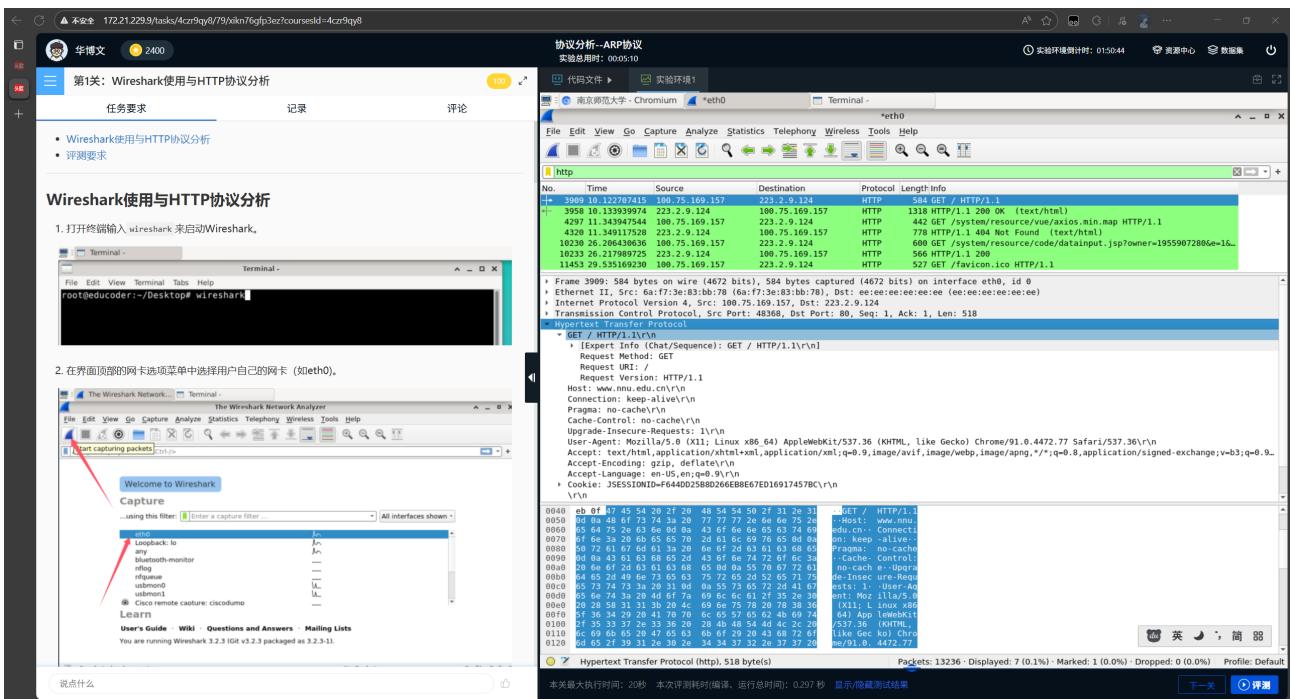
- 网络层 (IPv4): 展开 Internet Protocol Version 4 字段, IP 版本 (Version) 为 4, 总长度 (Total Length) 为 570 bytes, 源 IP (Src) 为 100.75.169.157, 目的 IP (Dst) 为 223.2.9.124, 生存时间 (Time to live) 为 64 (最多经过 64 个路由器), 上层协议 (Protocol) 为 TCP (6) (标识上层为 TCP 协议), IPv4 头部的长度 (Header Length) 为 20 bytes.



- 传输层 (TCP): 展开 Transmission Control Protocol 字段, 源端口 (Src Port) 为 48368 (客户端随机端口), 目的端口 (Dst Port) 为 80 (HTTP 服务默认端口), 序列号 (Seq) 为 1, 确认号 (Ack) 为 1, 数据长度 (Len) 为 518 bytes, 标志 (Flags) 为 0x018 (PSH, ACK) (推送数据并确认), TCP 段头部的长度 (Header Length) 为 32 bytes.



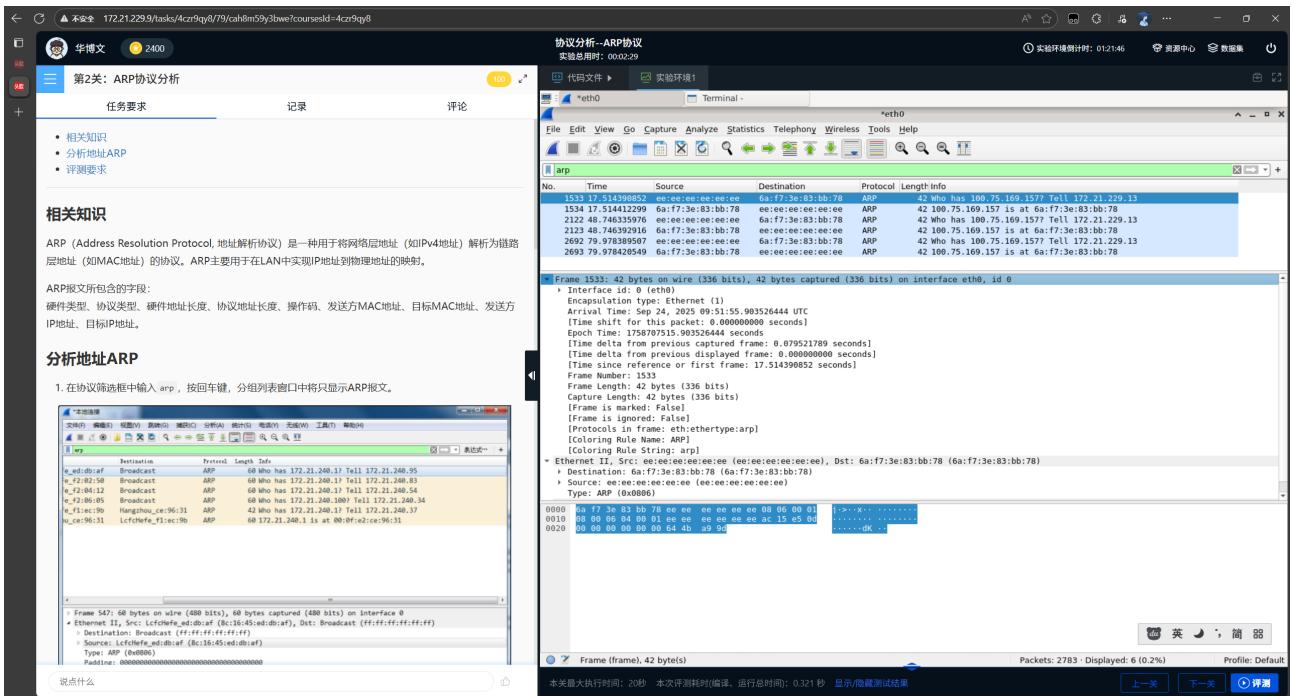
- 应用层 (HTTP): 展开 Hypertext Transfer Protocol 字段, 请求方法为 GET / HTTP/1.1 (获取网站根目录资源), 包含请求头 (Accept: 接受的资源类型、Accept-Encoding: 支持的压缩格式、Accept-Language: 语言偏好等), 使用的 HTTP 协议版本 (Request Version) 为 HTTP/1.1.



2 ARP 协议分析

2.1 筛选 ARP 报文

在 Wireshark 筛选框中输入 arp 并回车, 报文列表仅显示 ARP 协议报文, 包含 ARP 请求 (who has ...? Tell ...) 和 ARP 响应 (... is at ...) 两类。



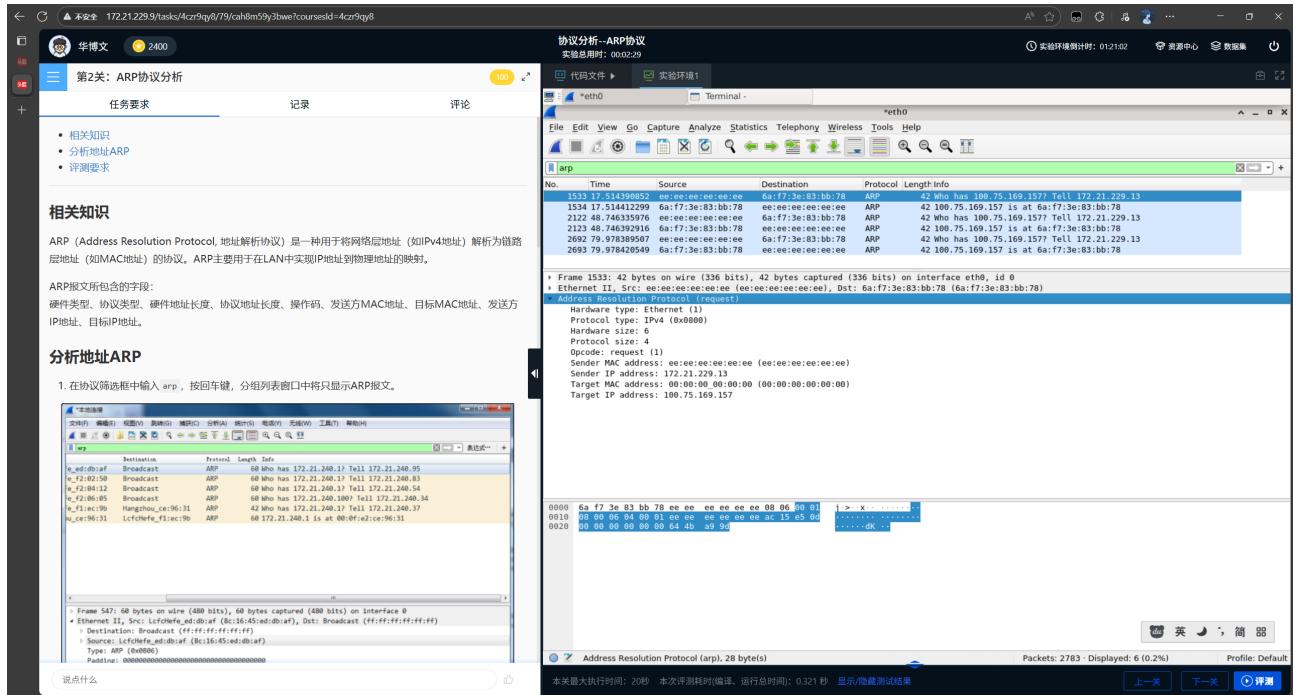
2.2 ARP 报文分层结构分析

查看任意 ARP 报文的详细信息, 发现仅包含 Ethernet II (数据链路层) 和 Address Resolution Protocol (ARP 协议层), 无传输层和应用层——因 ARP 工作在网络层与数据

链路层之间，负责 IP 与 MAC 的映射，不属于传统四层模型。

2.3 ARP 协议字段分析

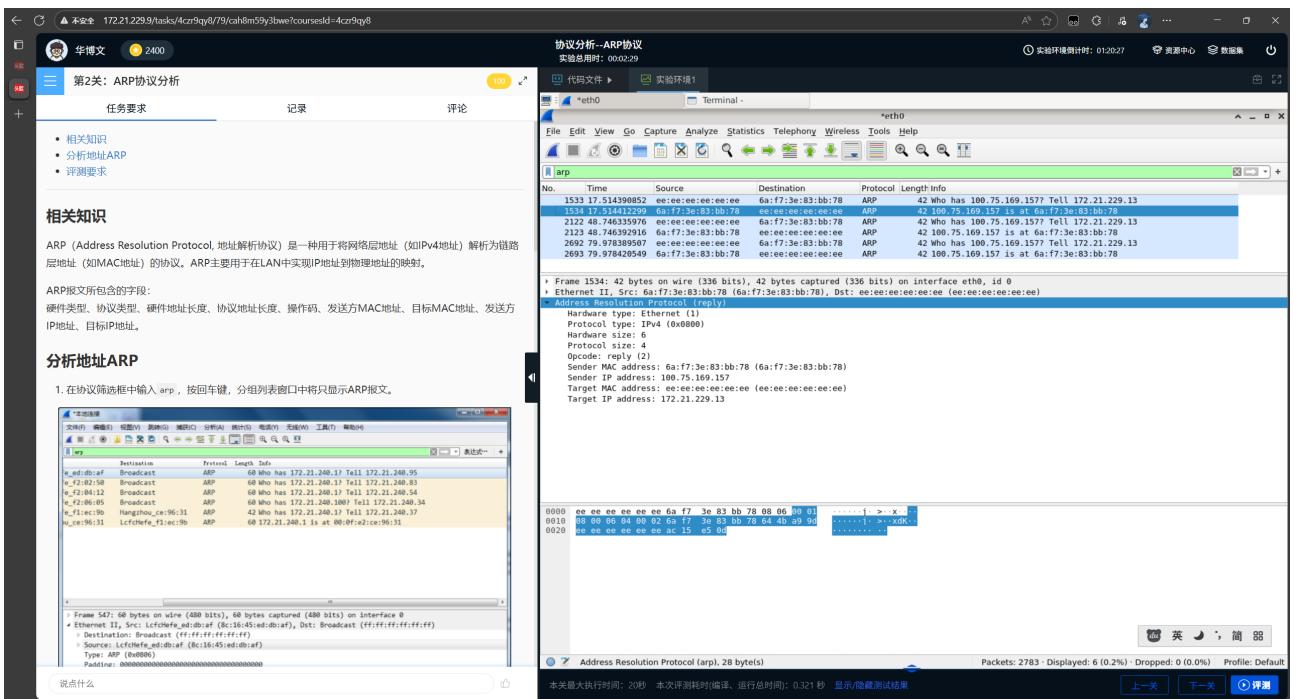
以 No.1533 ARP 请求报文为例，展开 Address Resolution Protocol (request) 字段，各关键字段如下：



- 硬件类型 (Hardware type): Ethernet (1) (表示底层为以太网);
- 协议类型 (Protocol type): IPv4 (0x0800) (表示映射的上层协议为 IPv4);
- 硬件地址长度 (Hardware size): 6 bytes (MAC 地址长度固定为 6 字节);
- 协议地址长度 (Protocol size): 4 bytes (IPv4 地址长度固定为 4 字节);
- 操作码 (Opcode): request (1) (标识为 ARP 请求);
- 发送方 MAC (Sender MAC address): ee:ee:ee:ee:ee:ee (请求方 MAC);
- 发送方 IP (Sender IP address): 172.21.229.13 (请求方 IP);
- 目标 MAC (Target MAC address): 00:00:00:00:00:00 (请求时未知目标 MAC，填充为全 0);
- 目标 IP (Target IP address): 100.75.169.157 (需解析 MAC 的目标 IP)。

2.4 ARP 响应报文对比

ARP 响应报文的操作码 (Opcode) 为 reply (2)，目标 MAC 地址更新为 6a:f7:3e:83:bb:78 (已知目标 MAC)，其他字段与请求报文一一对应，实现“IP→MAC”的映射告知。



3 ARP 数据包解析

3.1 明确解析需求

给定 ARP 十六进制数据: 0001 0800 0604 0001 eeee eeee eeee ac15 e50b 0000 0000 0000 6463 ae47, 需解析为 9 个字段, 输出格式为“Hardware Type: ...”、“Protocol Type: ...”等. 报文总长 28 字节, 分为两部分:

- 前 8 字节是 ARP 报头, 包含硬件类型、上层协议类型、MAC 地址长度、IP 地址长度、操作类型, 用来规定网络类型与报文功能;
- 后 20 字节是地址信息段, 包含源 MAC 地址、源 IP 地址、目的 MAC 地址、目的 IP 地址, 用于明确通信双方的 IP 与 MAC 映射关系.

对该十六进制数据按各字段的字节长度拆分后, 即可依次解析出这 9 个字段的具体内容.

ARP 报文, 28 字节								
ARP 报头, 8 字节					20 字节			
2 字节 硬件类型	2 字节 上层协议类型	1 字节 MAC 地址长度	1 字节 IP 地址长度	2 字节 操作类型	6 字节 源 MAC 地址	4 字节 源 IP 地址	6 字节 目的 MAC 地址	4 字节 目的 IP 地址
值为 1 表示以太网地址.	映射 IP 地址时值为 0x0800.	数值为 6.	数值为 4.	值为 1 时请求, 值为 2 时应答.			若为请求报文, 该字段值全为 0.	

3.2 编写 C++ 解析代码

将十六进制字符串转为字节数组, 按 ARP 协议字段的固定偏移和长度提取数据, 再转换为人类可读格式(如 MAC 转冒号分隔、IP 转点分十进制). 代码需补充部分如下:

```
void parseARPPacket(const uint8_t *arpData, const std::string
→ &outputFile)
{
```

```

...
uint16_t hardwareType = ntohs(*(const uint16_t *)&arpData[0]);
uint16_t protocolType = ntohs(*(const uint16_t *)&arpData[2]);
uint8_t hardwareLen = arpData[4];
uint8_t protocolLen = arpData[5];
uint16_t operation = ntohs(*(const uint16_t *)&arpData[6]);

char senderMacStr[18];
sprintf(senderMacStr, "%02x:%02x:%02x:%02x:%02x:%02x",
    ↵ arpData[8], arpData[9], arpData[10], arpData[11],
    ↵ arpData[12], arpData[13]);

char senderIpStr[16];
sprintf(senderIpStr, "%d.%d.%d.%d", arpData[14], arpData[15],
    ↵ arpData[16], arpData[17]);

char targetMacStr[18];
sprintf(targetMacStr, "%02x:%02x:%02x:%02x:%02x:%02x",
    ↵ arpData[18], arpData[19], arpData[20], arpData[21],
    ↵ arpData[22], arpData[23]);

char targetIpStr[16];
sprintf(targetIpStr, "%d.%d.%d.%d", arpData[24], arpData[25],
    ↵ arpData[26], arpData[27]);

outFile << "Hardware Type: " << hardwareType << std::endl;
outFile << "Protocol Type: 0x" << std::hex << std::setw(4) <<
    ↵ std::setfill('0') << protocolType << std::dec << std::endl;
outFile << "Hardware Address Length: " <<
    ↵ static_cast<int>(hardwareLen) << std::endl;
outFile << "Protocol Address Length: " <<
    ↵ static_cast<int>(protocolLen) << std::endl;
outFile << "Operation: " << operation << std::endl;
outFile << "Sender MAC: " << senderMacStr << std::endl;
outFile << "Sender IP: " << senderIpStr << std::endl;
outFile << "Target MAC: " << targetMacStr << std::endl;
outFile << "Target IP: " << targetIpStr << std::endl;

outFile.close();
}

```

3.3 编译与运行代码

在 Linux 终端中执行编译命令: g++ arp_parse.cpp -o arp_parse (无报错则生成可执行文件); 执行运行命令: ./arp_parse 0001...ae47 result.txt, 输出文件 result.txt 结果如下:

```
ls -al
总计 16
drwxrwxr-x 2 xv1r xv1r 4096 9月 24 18:27 .
drwxrwxr-x 4 xv1r xv1r 4096 9月 24 18:24 ..
-rw-rw-r-- 1 xv1r xv1r 4425 9月 24 18:26 arg_parse.cpp

g++ arg_parse.cpp -o arg_parse

./arg_parse 0001080006040001eeeeeeeeeeeeac15e50b0000000000006463ae47 result.txt

cat result.txt
Hardware Type: 1
Protocol Type: 0x0800
Hardware Address Length: 6
Protocol Address Length: 4
Operation: 1
Sender MAC: ee:ee:ee:ee:ee:ee
Sender IP: 172.21.229.11
Target MAC: 00:00:00:00:00:00
Target IP: 100.99.174.71
```

结果说明, 发送方 IP 由字节 0xac (172)、0x15 (21)、0xe5 (229)、0x0b (11) 转换, 目标 IP 由 0x64 (100)、0x63 (99)、0xae (174)、0x47 (71) 转换, 符合 ARP 请求报文特征——目标 MAC 为 00:00:00:00:00:00.

四 实验中遇到的问题及体会

在实验过程中, 曾遇到 ARP 报文长度不一致的问题: 筛选出的 ARP 报文长度呈现 42 字节与 60 字节两种情况. 起初并不清楚差异的成因, 经过分析后发现, ARP 报文的最小长度为 42 字节, 这是由数据链路层头部 14 字节与 ARP 报文本身 28 字节共同构成的; 而 60 字节的长度则与以太网最小帧长度相关——以太网最小帧长度为 64 字节 (包含前导码和帧间隙), 当数据部分不足 60 字节时, 会自动添加填充字节, 因此部分 ARP 报文会显示为 60 字节.

通过这次实验, 对计算机网络的认知有了多方面深化. 课本中“网络协议分层封装”的理论, 在 Wireshark 抓包过程中变得直观: HTTP 请求需经 TCP (传输层)、IP (网络层)、Ethernet (数据链路层) 依次封装, 每层都会添加头部信息, 接收方则反向解封装, 这让我切实理解了“分层解耦、各司其职”的设计思想.

此前仅知晓“IP 用于路由、MAC 用于局域网通信”, 但 ARP 抓包让我看到其“底层支撑”作用: 若没有 ARP 将目标 IP 解析为 MAC 地址, 即便 IP 地址正确, 数据也无法在局域网内传输; 而 ARP 的“广播请求、单播响应”机制, 更清晰解释了“同一网段设备能直接通信”的底层原理.

同时也意识到协议规范的核心意义: 解析 ARP 数据包时, 只要严格依照“硬件类型 2 字节、协议类型 2 字节、硬件地址长度 1 字节”的固定格式提取字段, 就能正确解析; 反之, 任何字段偏移或长度错误, 都会导致解析失败, 这印证了网络设备互通对严谨协议规范的依赖.

此外,问题排查的过程也提升了实践能力.从对 ARP 报文长度差异的疑问,到结合以太网帧长度要求、协议格式进行理论分析,再通过抓包数据对比验证,整个“发现问题——分析原因——解决问题”的流程,比单纯执行实验步骤更能加深对知识的掌握,也让我更熟悉网络实验的调试思路.