哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2025 春季

课程名称：计算机网络

实验名称：协议栈设计与实现

学生班级：

学生学号：

学生姓名：

评阅教师：

报告成绩：

实验与创新实践教育中心制

2025 年 3 月

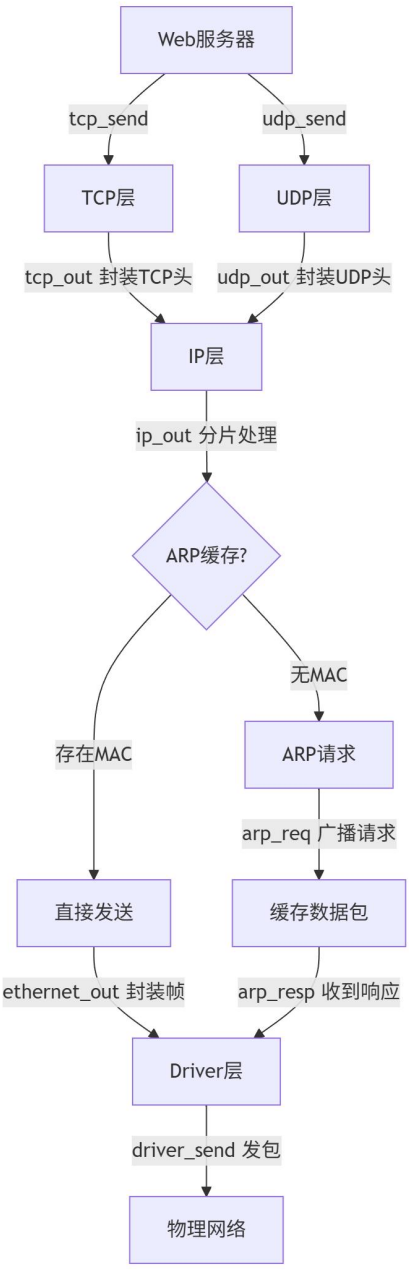
一、 协议实现详述

（注意不要完全照搬实验指导书上的内容，请根据你自己的设计方案来填写图文并茂地描述实验实现的所有功能和详细的设计方案及实验过程中的特色部分。）

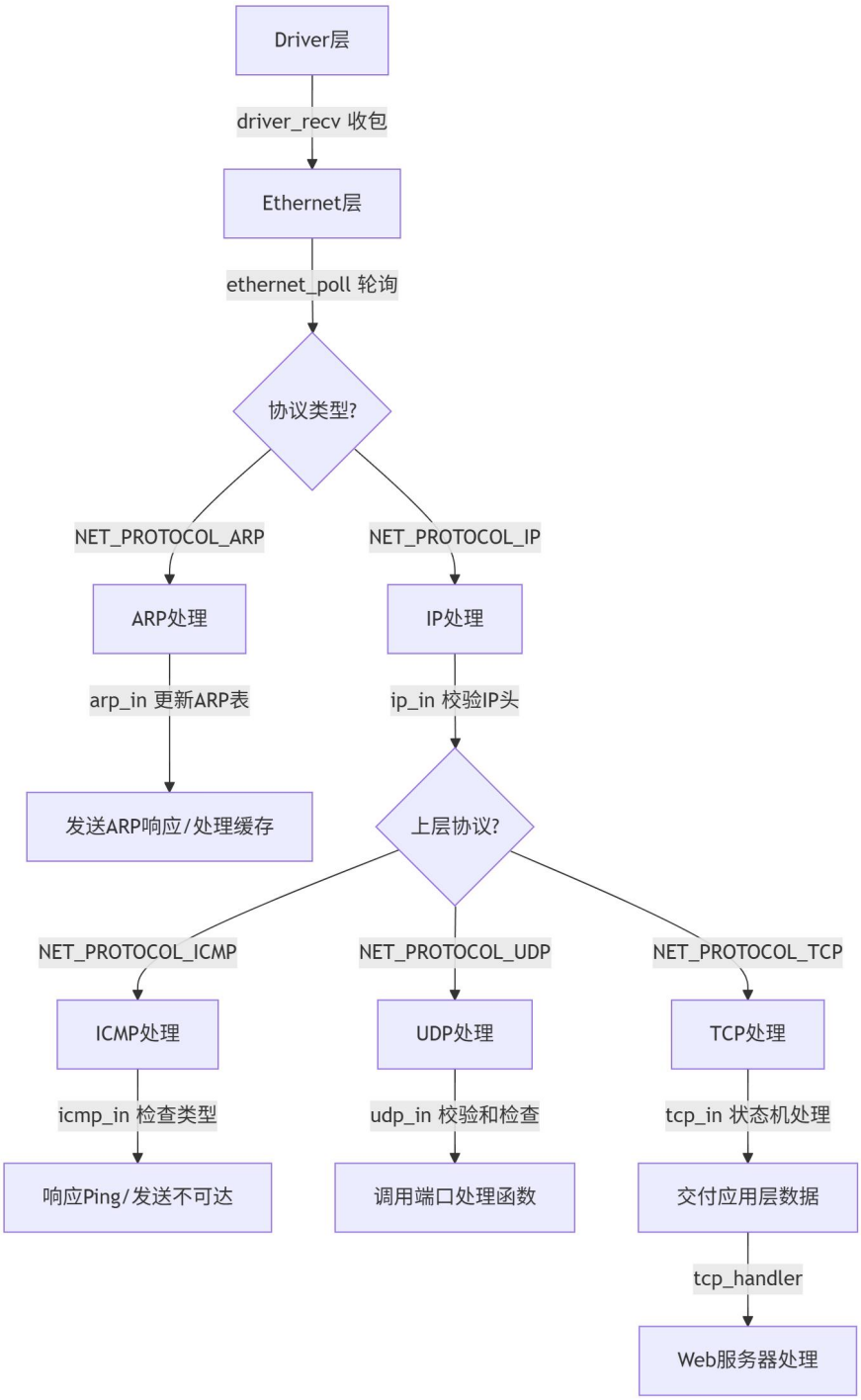
1. 请给出协议栈实验的整体流程图

（绘制协议栈实验的整体流程图，涵盖协议栈接收和发送主要步骤，包括 Eth 接收 / 发送、ARP 处理、IP 接收 / 发送、ICMP 处理、UDP 接收 / 发送、TCP 接收 / 发送以及 web 服务器请求处理等步骤，并标注主要函数调用关系。）

发送数据：

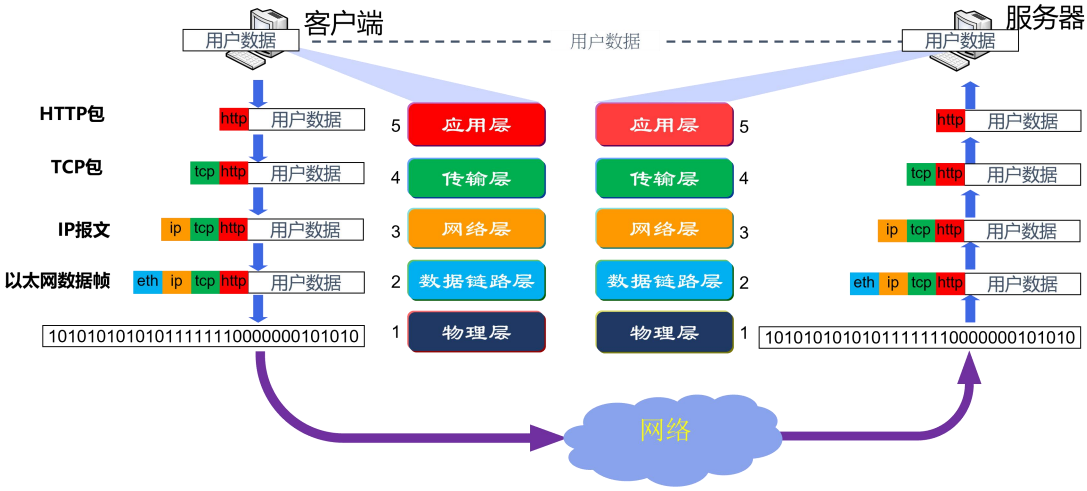


接收并处理数据：


Driver层
driver_recv 收包
Ethernet层
ethernet_poll 轮询
协议类型?
NET_PROTOCOL_ARP
NET_PROTOCOL_IP
ARP处理
IP处理
arp_in 更新ARP表
ip_in 校验IP头
发送ARP响应/处理缓存
上层协议?
NET_PROTOCOL_ICMP
NET_PROTOCOL_UDP
NET_PROTOCOL_TCP
ICMP处理
UDP处理
TCP处理
icmp_in 检查类型
udp_in 校验和检查
tcp_in 状态机处理
响应Ping/发送不可达
调用端口处理函数
交付应用层数据
tcp_handler
Web服务器处理

2. Eth 协议详细设计

（描述以太网（Eth）协议的数据封装与解封装过程等。）



图中左边从上到下为数据封装的过程，数据链路层（以太网）接收来自网络层的数据包，添加以太网帧头（源/目的 MAC + Type）和帧尾（FCS），形成一个完整的帧传给物理层。MAC 地址用于局域网内部的物理寻址（下一跳），Type 标识上层协议（通常是 IP）。逻辑寻址（IP）由网络层负责。

图中右边从下到上为数据解封装的过程，数据链路层（以太网）从物理层接收比特流组装成帧，进行 FCS 校验（错误检测）和目的 MAC 地址匹配。校验失败或 MAC 不匹配则直接丢弃帧。匹配且校验通过后，剥离帧头和帧尾，根据 Type 字段将数据包交给相应的网络层协议处理。不负责逻辑寻址/路由（由网络层负责），只负责本地物理传输和错误检。

3. ARP 协议详细设计

（描述 ARP 请求/响应处理逻辑、ARP 表项的更新机制等。）

ARP 请求/响应处理逻辑

ARP 请求处理

当接收到一个 ARP 请求时，arp_in 函数会被调用：

1. 数据包长度检查：首先检查数据包长度是否小于 ARP 头部长度，若是则丢弃。
2. 报头检查：检查硬件类型、协议类型、MAC 地址长度和 IP 地址长度是否正确。
3. 操作类型检查：获取操作类型并检查是否为 ARP_REQUEST 或 ARP_REPLY。
4. 更新 ARP 表项：调用 map_set 函数将发送方的 IP 和 MAC 地址映射关系存储到 ARP 表中。
5. 查看缓存情况：检查 ARP 缓存中是否有待发送的数据包。若有，则将缓存的数据包发送给以太网层，并从缓存中删除。
6. 处理 ARP 请求：若接收到的报文为 ARP_REQUEST，且目标 IP 为本机 IP，则发送 ARP 响应。

ARP 响应处理

当接收到一个 ARP 响应时，arp_in 函数同样会被调用：
1. 数据包长度检查：首先检查数据包长度是否小于 ARP 头部长度，若是则丢弃。
2. 报头检查：检查硬件类型、协议类型、MAC 地址长度和 IP 地址长度是否正确。
3. 操作类型检查：获取操作类型并检查是否为 ARP_REQUEST 或 ARP_REPLY。
4. 更新 ARP 表项：调用 map_set 函数将发送方的 IP 和 MAC 地址映射关系存储到 ARP 表中。
5. 查看缓存情况：检查 ARP 缓存中是否有待发送的数据包。若有，则将缓存的数据包发送给以太网层，并从缓存中删除。

ARP 表项的更新机制

ARP 表项的更新通过 map_set 函数实现，该函数将 IP 和 MAC 地址的映射关系存储到 ARP 表中。ARP 表项有一个超时机制，由 ARP_TIMEOUT_SEC 定义，超时后表项会被自动删除。

ARP 缓存机制

ARP 缓存用于存储等待 ARP 响应的数据包。当 ARP 表中找不到目标 IP 对应的 MAC 地址时，会将数据包缓存到 ARP 缓存中，并发送 ARP 请求。收到 ARP 响应后，将缓存的数据包发送出去，并从缓存中删除。

4. IP 协议详细设计

（描述 IP 数据包的封装与解封装、IP 数据包的分片、校验和计算等。）

IP 数据包的封装与解封装

封装

在 ip_out 函数中，IP 数据包的封装过程如下：

1. 检查数据包长度：首先检查从上层传递下来的数据包长度是否大于 IP 协议最大负载包长（MTU - IP 头部长度）。
2. 直接发送：如果数据包长度小于等于最大负载包长，则直接调用 ip_fragment_out 函数发送数据包。
3. 分片发送：如果数据包长度大于最大负载包长，则需要分片发送。调用 buf_init 函数初始化一个分片缓冲区，将数据复制到分片缓冲区，然后调用 ip_fragment_out 函数发送分片。

解封装

在 ip_in 函数中，IP 数据包的解封装过程如下：

1. 检查数据包长度：首先检查数据包长度是否小于 IP 头部长度，若是则丢弃。
2. 报头检查：检查 IP 头部的版本号是否为 IPv4，总长度字段是否小于或等于收到的数据包长度。
3. 校验头部校验和：计算并检查 IP 头部的校验和，如果不一致，说明数据包在传输过程中可能出现损坏，将其丢弃。
4. 对比目的 IP 地址：检查目的 IP 地址是否为本机的 IP 地址，如果不是，说明该数据包并非发送给本机，将其丢弃。
5. 去除填充字段：如果接收到的数据包的长度大于 IP 头部的总长度字段，说明该数据包存在填充字段，调用 buf_remove_padding 函数去除填充字段。
6. 去掉 IP 报头：调用 buf_remove_header 函数去掉 IP 报头。
7. 向上层传递数据包：调用 net_in 函数将数据包传递给上层协议处理。

IP 数据包的分片

在 ip_out 函数中，IP 数据包的分片过程如下：

1. 检查数据包长度：首先检查从上层传递下来的数据包长度是否大于 IP 协议最大负载包长。
2. 直接发送：如果数据包长度小于等于最大负载包长，则直接发送。
3. 分片发送：如果数据包长度大于最大负载包长，则需要分片发送。计算当前分片的大小，初始化一个分片缓冲区，将数据复制到分片缓冲区，然后调用 ip_fragment_out 函数发送分片。

校验和计算

在 ip_fragment_out 函数中，IP 数据包的校验和计算过程如下：

1. 增加头部缓存空间：调用 buf_add_header 函数增加头部缓存空间。
2. 填写头部字段：填写 IP 头部字段，包括版本号、首部长度、区分服务、总长度、数据包标识、分片标志和偏移量、生存时间、上层协议类型、源 IP 地址和目标 IP 地址。
3. 计算并填写校验和：先将 IP 头部的首部校验和字段填为 0，然后调用 checksum16 函数计算校验和，最后将计算结果填入首部校验和字段。

5. ICMP 协议详细设计

（解释如何处理 ICMP 请求和响应，以及如何利用 ICMP 报文进行网络故障诊断等。）

ICMP 请求的处理
ICMP 请求的处理主要在 icmp_in 函数中实现：
1. 首先检查接收到的数据包长度是否足够包含 ICMP 头部。如果数据包长度不足，则丢弃数据包。
2. 然后查看 ICMP 类型，如果 ICMP 类型为回显请求（ICMP_TYPE_ECHO_REQUEST），则调用 icmp_resp 函数发送回显应答。

ICMP 响应的发送
ICMP 响应的发送主要在 icmp_resp 函数中实现：
1. 初始化发送缓冲区，并在缓冲区中构造 ICMP 报头，设置响应报头的字段，包括类型、代码、标识符和序列号。
2. 复制请求报文的数据部分到响应报文的数据部分。
3. 计算整个 ICMP 报文的校验和。
4. 调用 ip_out 函数发送 ICMP 响应报文。

ICMP 不可达消息的发送
ICMP 不可达消息的发送主要在 icmp_unreachable 函数中实现：
1. 初始化发送缓冲区，并构造 ICMP 报头，设置报头的类型和代码字段。
2. 填写 ICMP 数据部分，包括原始 IP 头部和原始 IP 数据报的前 8 字节数据。
3. 计算 ICMP 报文的校验和。
4. 调用 ip_out 函数发送 ICMP 不可达消息。

ICMP 协议的初始化
ICMP 协议的初始化在 icmp_init 函数中实现，主要通过调用 net_add_protocol 函数将 ICMP 协议添加到网络协议列表中，以便接收到 ICMP 数据包时能够正确处理。

网络故障诊断
ICMP 协议在网络故障诊断中起着重要作用。例如，通过发送 ICMP 回显请求（即 ping 命令），可以检测网络连通性。如果目标主机收到请求并返回响应，则说明网络连通；否则，可能存在网络故障。

6. UDP 协议详细设计

（描述 UDP 数据包的封装与解封装、UDP 校验和计算等。）

UDP 数据包解封装（udp_in 函数）

UDP 数据包的解封装过程包含以下几个关键步骤：

1. 数据包完整性检查
首先检查接收到的数据包长度是否满足 UDP 协议的基本要求：
- 验证数据包长度是否小于 UDP 首部长度（8 字节）
- 检查实际接收长度是否与 UDP 首部中声明的长度一致

2. 校验和验证
- 保存原始校验和字段
- 将校验和字段置零
- 重新计算校验和并与原始值比较
- 如果校验和不匹配则丢弃数据包

3. 端口处理与分发
- 提取目标端口号
- 在 UDP 处理程序表中查找对应的处理函数
- 如果未找到处理函数，则发送 ICMP 端口不可达消息

4. 数据传递
- 移除 UDP 首部
- 调用相应的处理函数处理载荷数据

UDP 数据包封装（udp_out 函数）

UDP 数据包的封装过程包括：

1. 添加 UDP 首部
为数据包添加 8 字节的 UDP 首部空间

2. 填充首部字段
- 设置源端口号和目标端口号
- 设置 UDP 数据包总长度

3. 校验和计算
- 初始化校验和字段为 0
- 调用传输层校验和计算函数
- 填入计算得到的校验和

4. 数据包发送
通过 IP 层发送 UDP 数据包

UDP 校验和计算机制

UDP 校验和计算采用伪首部机制，具体实现在 transport_checksum 函数中：

1. 伪首部构造
创建包含以下字段的伪首部结构：
- 源 IP 地址（4 字节）
- 目标 IP 地址（4 字节）
- 填充字节（1 字节，置 0）
- 协议号（1 字节）
- UDP 数据包长度（2 字节）

2. 校验和计算过程
- 将伪首部添加到 UDP 数据包前
- 如果数据长度为奇数，添加填充字节
- 使用 16 位校验和算法计算整个数据块
- 移除伪首部和填充，恢复原始数据包结构

3. 16 位校验和算法
checksum16 函数实现标准的 Internet 校验和算法：
- 将数据按 16 位分组求和
- 处理进位，将高 16 位加到低 16 位
- 对结果取反得到最终校验和

UDP 端口管理

系统提供完整的 UDP 端口管理功能：

端口注册（udp_open）
将端口号与对应的处理函数绑定，存储在 UDP 处理程序表中

端口关闭（udp_close）
从 UDP 处理程序表中移除指定端口的绑定关系

数据发送接口（udp_send）
提供便捷的 UDP 数据发送接口，封装了缓冲区管理和数据包发送过程

这套 UDP 实现提供了完整的数据包处理能力，包括严格的协议验证、高效的端口管理和标准的校验和计算，确保了网络通信的可靠性和正确性。

7. TCP 协议详细设计

（描述 TCP 连接的建立与关闭过程（三次握手、四次挥手）等。解释如何处理 TCP 数据包的确认、连接状态等问题。）

TCP 连接建立过程（三次握手）

TCP 连接建立采用经典的三次握手机制，在 tcp_in 函数中实现了服务器端的处理逻辑：

第一次握手（SYN）
当服务器处于 TCP_STATE_LISTEN 状态时，接收到客户端的 SYN 报文：
- 验证接收到的报文是否包含 SYN 标志位
- 生成服务器端的初始序列号（ISN）
- 设置确认号为客户端序列号加 1
- 准备发送 SYN+ACK 回复
- 状态转换为 TCP_STATE_SYN_RECEIVED

第二次握手（SYN+ACK）
服务器发送 SYN+ACK 报文给客户端，包含自己的序列号和对客户端 SYN 的确认。

第三次握手（ACK）
当服务器处于 TCP_STATE_SYN_RECEIVED 状态时，接收到客户端的 ACK 报文：
- 验证报文包含 ACK 标志位
- 状态转换为 TCP_STATE_ESTABLISHED
- 连接建立完成

TCP 连接关闭过程（四次挥手）

TCP 连接关闭采用四次挥手机制，实现了被动关闭的处理：

接收 FIN 报文
当连接处于 TCP_STATE_ESTABLISHED 状态时，如果接收到 FIN 报文：
- 设置回复标志为 ACK+FIN
- 状态转换为 TCP_STATE_LAST_ACK
- 进入被动关闭流程

最终确认
当连接处于 TCP_STATE_LAST_ACK 状态时，接收到对方的 ACK 确认：
- 验证报文包含 ACK 标志位
- 调用 tcp_close_connection 关闭连接
- 从连接表中删除连接记录

TCP 数据包确认机制

序列号管理
TCP 实现了完整的序列号管理机制：
- 使用 tcp_generate_initial_seq()生成随机初始序列号
- 通过 bytes_in_flight()函数计算序列空间长度，考虑 SYN 和 FIN 标志位
- 发送数据后更新序列号：tcp_conn->seq += bytes_in_flight(len, 0)

确认号处理
- 接收数据时更新确认号：tcp_conn->ack += bytes_in_flight(data_len, recv_flags)
- 对于乱序数据包，发送重复 ACK 并丢弃数据包
- 实现了顺序检查：if (remote_seq != tcp_conn->ack)

数据交付机制
当接收到有效数据时：
- 查找对应端口的处理函数
- 如果未找到处理函数，发送 ICMP 端口不可达消息
- 找到处理函数则移除 TCP 头部并调用应用层处理函数

TCP 连接状态管理

连接标识与查找
TCP 连接使用三元组（远程 IP、远程端口、本地端口）作为唯一标识：
- generate_tcp_key()函数生成连接键值
- tcp_get_connection()函数查找或创建连接
- 支持动态创建新连接

状态转换
实现了关键的 TCP 状态转换：
- TCP_STATE_LISTEN → TCP_STATE_SYN_RECEIVED（收到 SYN）
- TCP_STATE_SYN_RECEIVED → TCP_STATE_ESTABLISHED（收到 ACK）
- TCP_STATE_ESTABLISHED → TCP_STATE_LAST_ACK（收到 FIN）
- TCP_STATE_LAST_ACK → 连接关闭（收到 ACK）

连接重置
- 收到 RST 报文时立即关闭连接
- tcp_rst()函数用于重置连接状态
- 支持异常情况下的连接清理

数据发送与接收

发送机制
tcp_send()函数实现数据发送：
- 检查数据长度合法性
- 创建发送缓冲区并复制数据
- 调用 tcp_out()发送带 ACK 的数据包
- 更新序列号并标记已发送 ACK

接收处理
数据接收时进行完整性检查：
- 验证 TCP 头部长度
- 校验和验证

- 序列号顺序检查
- 根据连接状态进行相应处理

这套 TCP 实现提供了完整的连接管理功能，包括标准的三次握手建立连接、四次挥手关闭连接、可靠的数据传输确认机制，以及健壮的状态管理，确保了 TCP 协议的可靠性和正确性。

8. web 服务器详细设计

（描述 web 服务器的请求处理流程和响应等。）

HTTP 请求处理流程

请求解析阶段

当 TCP 连接接收到数据时，http_request_handler 函数负责解析 HTTP 请求：

方法验证：提取 HTTP 请求的方法字段，目前仅支持 GET 请求，其他方法将被直接忽略。

URL 提取：通过字符串解析提取请求的 URL 路径，跳过 HTTP 方法和空格，直到遇到下一个空格为止，构建完整的资源路径。

响应生成阶段

解析完成后调用 http_respond 函数生成 HTTP 响应，该函数是整个响应处理的核心。

HTTP 响应处理机制

文件路径处理

默认页面处理：当请求路径为"/"时，自动重定向到 index.html 作为默认主页。

路径构建：将 HTTP_RESOURCE_DIR 与请求路径拼接，形成完整的文件系统路径。

文件访问：以二进制读取模式打开目标文件，支持各种文件类型的处理。

404 错误响应

当请求的文件不存在时，服务器发送标准的 HTTP 404 响应：

状态行：发送"HTTP/1.1 404 Not Found"状态行。

响应头：包含连接保持信息、内容类型（text/html）和内容长度等必要头部字段。

响应体：发送预定义的 HTML 错误页面，提供用户友好的错误信息。

成功响应处理

当文件存在时，服务器生成 HTTP 200 响应：

状态行：发送"HTTP/1.1 200 OK"表示请求成功。

内容类型识别：通过 http_get_mime_type 函数根据文件扩展名确定 MIME 类型，支持 HTML、CSS、JPEG 等常见格式。

内容长度计算：通过文件指针操作获取文件大小，确保客户端能正确接收完整内容。

分块传输：采用缓冲区循环读取文件内容，分块发送给客户端，避免大文件导致的内存问题。

MIME 类型处理

服务器实现了基础的 MIME 类型识别机制：
HTML 文件：识别.html 和.htm 扩展名，返回"text/html; charset=utf-8"。
CSS 样式表：识别.css 扩展名，返回"text/css"。
JPEG 图片：识别.jpg 和.jpeg 扩展名，返回"image/jpeg"。
默认类型：对于未识别的文件类型，返回"application/octet-stream"作为二进制数据处理。

连接管理
持久连接支持
服务器在所有响应中都包含"Connection: Keep-Alive"头部，支持 HTTP 持久连接，提高传输效率。
TCP 连接复用
通过 TCP 连接对象管理多个 HTTP 请求，避免频繁的连接建立和断开开销。

响应发送机制

分段发送
HTTP 响应通过多次 tcp_send 调用分段发送，包括状态行、各种头部字段和响应体内容。

缓冲区管理
使用固定大小的响应缓冲区进行内容格式化和传输，在内存使用和性能之间取得平衡。

错误处理
当文件操作失败时，服务器能够优雅地处理错误情况，发送适当的 HTTP 错误响应而不是崩溃。

二、 实验结果截图及分析

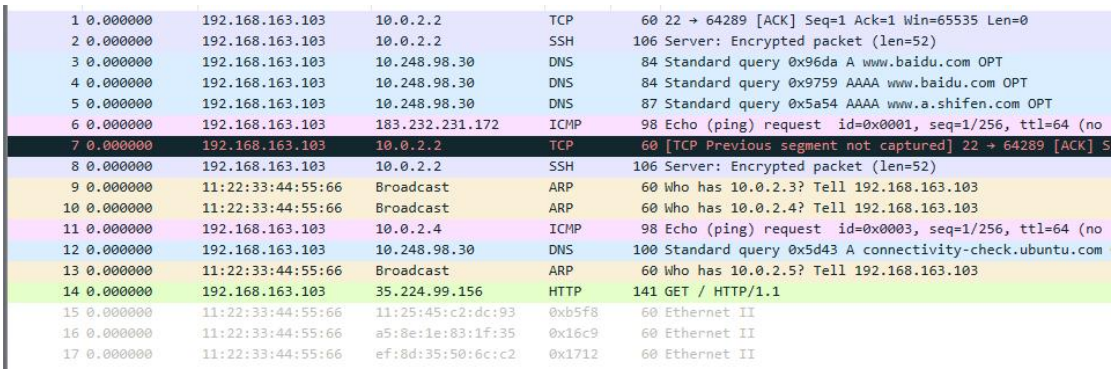
（请展示并详细分析实验结果的截图。可以利用 log 文件、通过 Wireshark 打开的 pcap 文件或 Wireshark 实时捕获的网络报文。）

1. Eth 协议实验结果及分析

（展示以太网帧捕获截图，分析帧的结构和内容是否符合预期。检查目的 MAC 地址、源 MAC 地址、协议类型字段以及数据部分）

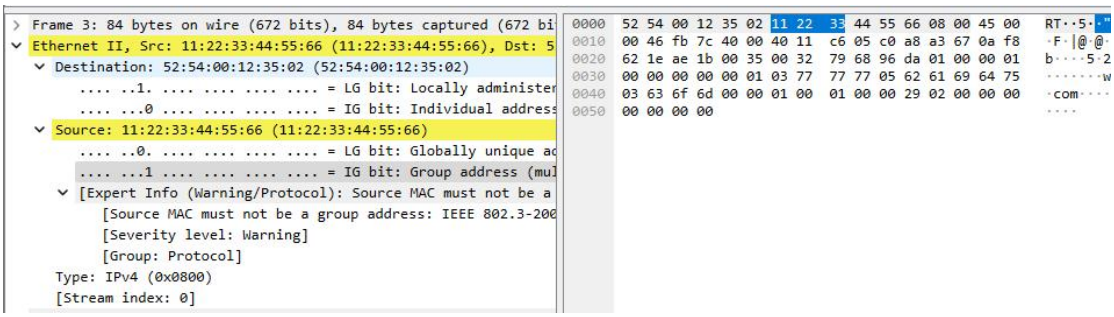
ethernet_out：

out.pcap 文件如下：



1	0.000000	192.168.163.103	10.0.2.2	TCP	60 22 → 64289 [ACK] Seq=1 Ack=1 Win=65535 Len=0
2	0.000000	192.168.163.103	10.0.2.2	SSH	106 Server: Encrypted packet (len=52)
3	0.000000	192.168.163.103	10.248.98.30	DNS	84 Standard query 0x96da A www.baidu.com OPT
4	0.000000	192.168.163.103	10.248.98.30	DNS	84 Standard query 0x9759 AAAA www.baidu.com OPT
5	0.000000	192.168.163.103	10.248.98.30	DNS	87 Standard query 0x5a54 AAAA www.a.shifen.com OPT
6	0.000000	192.168.163.103	183.232.231.172	ICMP	98 Echo (ping) request id=0x0001, seq=1/256, ttl=64 (no
7	0.000000	192.168.163.103	10.0.2.2	TCP	60 [TCP Previous segment not captured] 22 → 64289 [ACK] S
8	0.000000	192.168.163.103	10.0.2.2	SSH	106 Server: Encrypted packet (len=52)
9	0.000000	11:22:33:44:55:66	Broadcast	ARP	60 Who has 10.0.2.3? Tell 192.168.163.103
10	0.000000	11:22:33:44:55:66	Broadcast	ARP	60 Who has 10.0.2.4? Tell 192.168.163.103
11	0.000000	192.168.163.103	10.0.2.4	ICMP	98 Echo (ping) request id=0x0003, seq=1/256, ttl=64 (no
12	0.000000	192.168.163.103	10.248.98.30	DNS	100 Standard query 0x5d43 A connectivity-check.ubuntu.com
13	0.000000	11:22:33:44:55:66	Broadcast	ARP	60 Who has 10.0.2.5? Tell 192.168.163.103
14	0.000000	192.168.163.103	35.224.99.156	HTTP	141 GET / HTTP/1.1
15	0.000000	11:22:33:44:55:66	11:25:45:c2:dc:93	0xb5f8	60 Ethernet II
16	0.000000	11:22:33:44:55:66	a5:8e:1e:83:1f:35	0x16c9	60 Ethernet II
17	0.000000	11:22:33:44:55:66	ef:8d:35:50:6c:c2	0x1712	60 Ethernet II

任意查看一个帧结构如下：

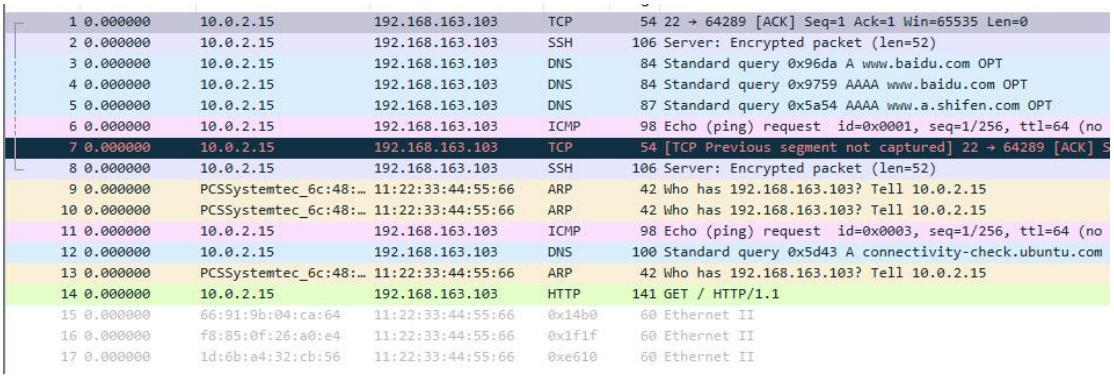


```
Frame 3: 84 bytes on wire (672 bits), 84 bytes captured (672 bi
Ethernet II, Src: 11:22:33:44:55:66 (11:22:33:44:55:66), Dst: 5
  Destination: 52:54:00:12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally administer
      .... ...0 .... .... .... .... = IG bit: Individual address
  Source: 11:22:33:44:55:66 (11:22:33:44:55:66)
      .... ..0. .... .... .... .... = LG bit: Globally unique ad
      .... ...1 .... .... .... .... = IG bit: Group address (mul
    [Expert Info (Warning/Protocol): Source MAC must not be a
      [Source MAC must not be a group address: IEEE 802.3-200
      [Severity level: Warning]
      [Group: Protocol]
  Type: IPv4 (0x0800)
  [Stream index: 0]
```

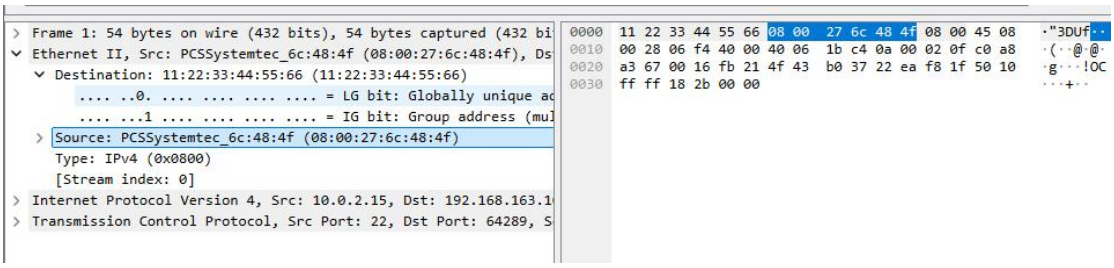
```
0000  52 54 00 12 35 02 11 22  33 44 55 66 08 00 45 00   RT··5·"3 DUf··E·
0010  00 46 fb 7c 40 00 40 11  c6 05 c0 a8 a3 67 0a f8   ·F·|@·@· ·····g··
0020  62 1e ae 1b 00 35 00 32  79 68 96 da 01 00 00 01   b····5·2 yh······
0030  00 00 00 00 00 01 03 77  77 77 05 62 61 69 64 75   ·······w ww·baidu
0040  03 63 6f 6d 00 00 01 00  01 00 00 29 02 00 00 00   ·com···· ···)····
0050  00 00 00 00                                       ····
```

可以看到源 MAC 地址是本机，协议字段正确填写的报文都被正确发送。

ethernet_in：



1	0.000000	10.0.2.15	192.168.163.103	TCP	54 22 → 64289 [ACK] Seq=1 Ack=1 Win=65535 Len=0
2	0.000000	10.0.2.15	192.168.163.103	SSH	106 Server: Encrypted packet (len=52)
3	0.000000	10.0.2.15	192.168.163.103	DNS	84 Standard query 0x96da A www.baidu.com OPT
4	0.000000	10.0.2.15	192.168.163.103	DNS	84 Standard query 0x9759 AAAA www.baidu.com OPT
5	0.000000	10.0.2.15	192.168.163.103	DNS	87 Standard query 0x5a54 AAAA www.a.shifen.com OPT
6	0.000000	10.0.2.15	192.168.163.103	ICMP	98 Echo (ping) request id=0x0001, seq=1/256, ttl=64 (no
7	0.000000	10.0.2.15	192.168.163.103	TCP	54 [TCP Previous segment not captured] 22 → 64289 [ACK] S
8	0.000000	10.0.2.15	192.168.163.103	SSH	106 Server: Encrypted packet (len=52)
9	0.000000	PCSSystemtec_6c:48:…	11:22:33:44:55:66	ARP	42 Who has 192.168.163.103? Tell 10.0.2.15
10	0.000000	PCSSystemtec_6c:48:…	11:22:33:44:55:66	ARP	42 Who has 192.168.163.103? Tell 10.0.2.15
11	0.000000	10.0.2.15	192.168.163.103	ICMP	98 Echo (ping) request id=0x0003, seq=1/256, ttl=64 (no
12	0.000000	10.0.2.15	192.168.163.103	DNS	100 Standard query 0x5d43 A connectivity-check.ubuntu.com
13	0.000000	PCSSystemtec_6c:48:…	11:22:33:44:55:66	ARP	42 Who has 192.168.163.103? Tell 10.0.2.15
14	0.000000	10.0.2.15	192.168.163.103	HTTP	141 GET / HTTP/1.1
15	0.000000	66:91:9b:04:ca:64	11:22:33:44:55:66	0x14b0	60 Ethernet II
16	0.000000	f8:85:0f:26:a0:e4	11:22:33:44:55:66	0x1f1f	60 Ethernet II
17	0.000000	1d:6b:a4:32:cb:56	11:22:33:44:55:66	0xe610	60 Ethernet II



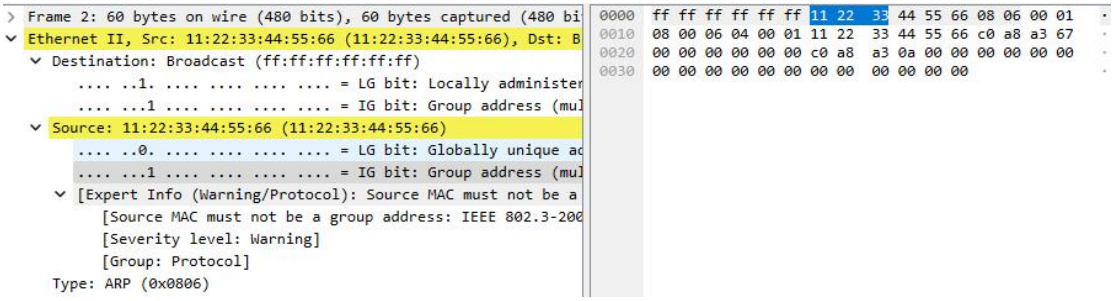
查看每个数据包结构，可以看到目的 MAC 地址都是本机，被正常接收和处理。

2. ARP 协议实验结果及分析

（展示 ARP 请求和响应包的捕获截图，分析其请求和响应过程是否正常。检查 ARP 请求中的目标 IP 地址和发送方 MAC 地址，ARP 响应中的目标 MAC 地址。）

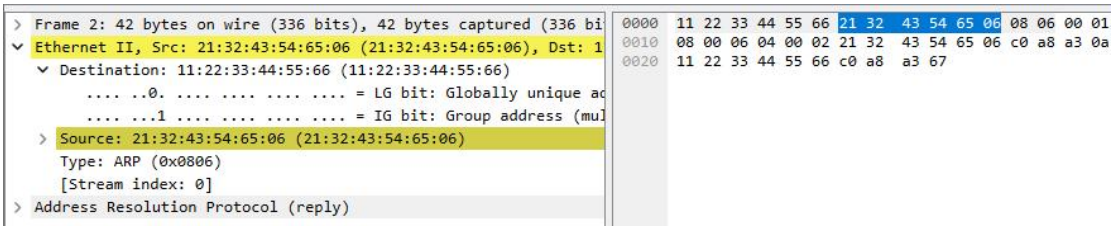
下图中左边为 in.pcap 文件，右边为 out.pcap 文件。

可以看到一开始本机想要发送 DNS 请求给 192.168.163.10，但是此 arp table 中没有 mac 地址，因此先发送了一个 ARP 广播请求其 mac 地址。



从图中可以看到请求的源 mac 地址为本机，目的 mac 为广播地址。

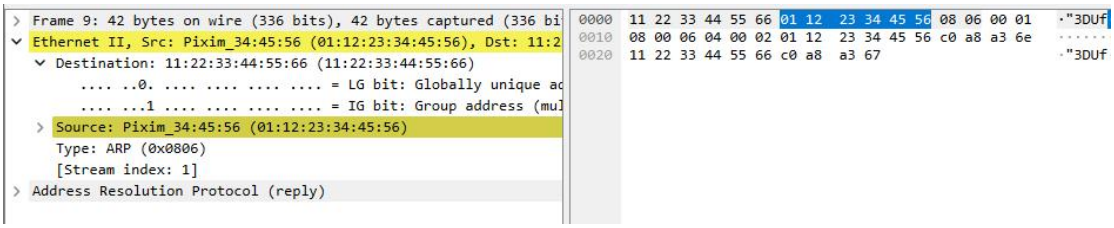
而 in 文件中的第二个包就是对这个 ARP 请求的响应。



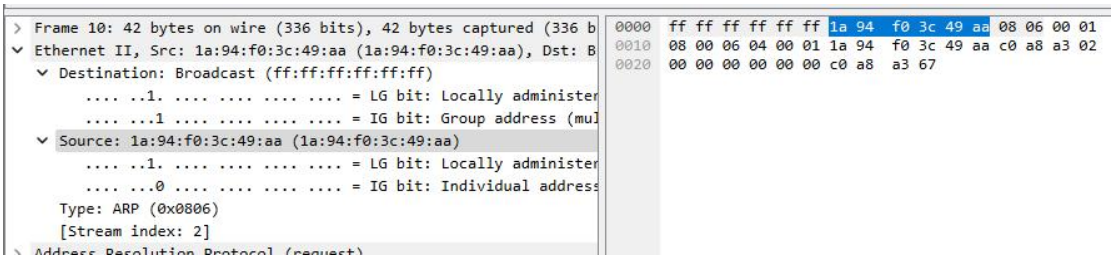
源 mac 地址即 192.168.163.10 对应的 mac 地址，目的地址为本机。

in 中的第九条数据包是收到的 ARP 响应请求，告诉其 192.168.163.110 的 MAC 地址。

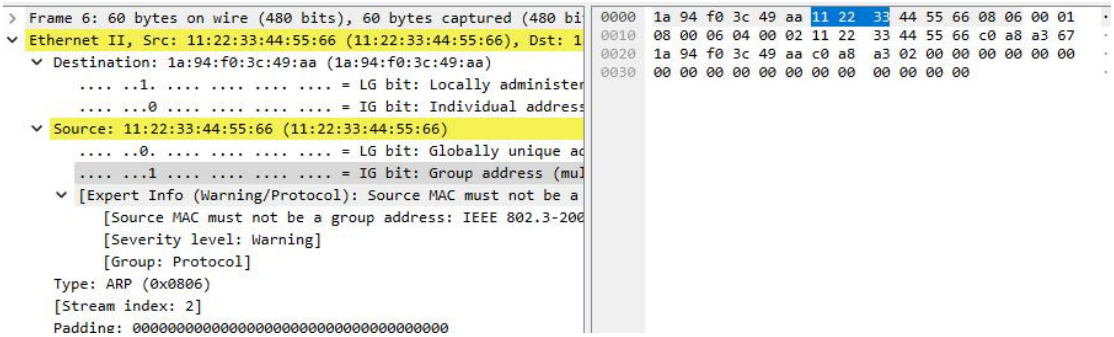
该请求可能是 192.168.163.110 主机启动时发送的 announcement。



in 的第十条数据包如下：



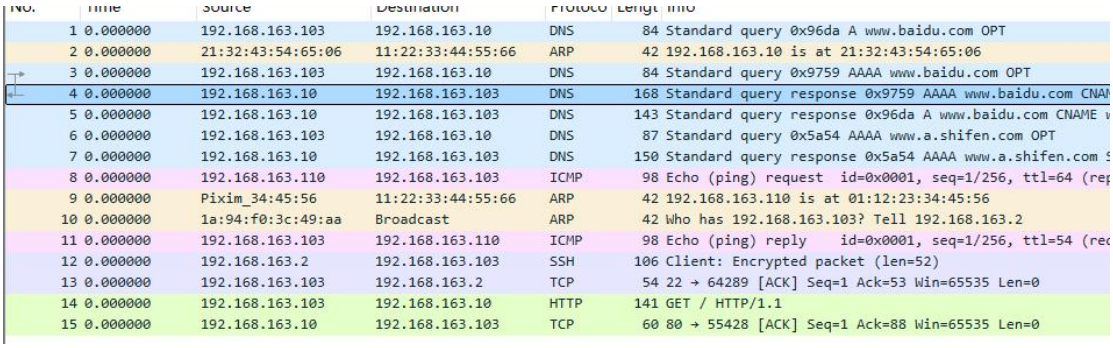
本机接收到了其他主机发送的 ARP 请求，请求的地址正好为本机，于是应该要发送 ARP 响应报文。而 out 文件中的第 6 条数据包就是对这个请求的响应。其内容如下：



3. IP 协议实验结果及分析

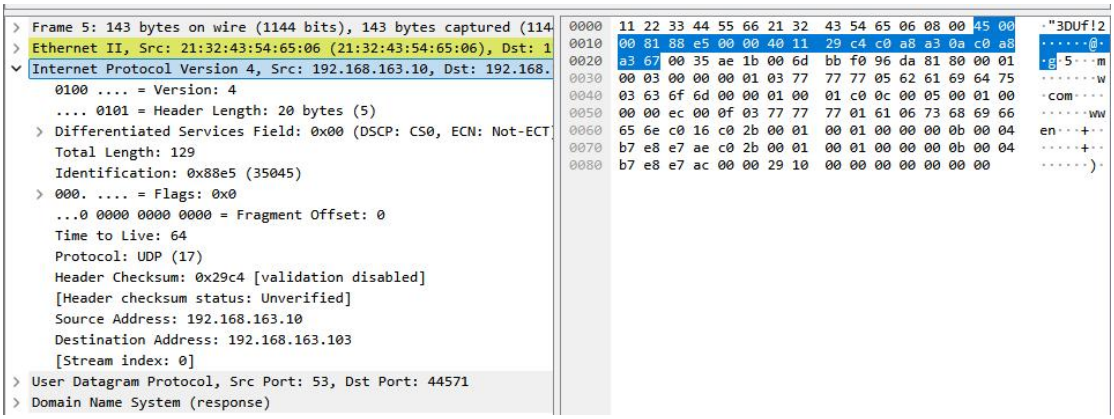
（展示 IP 数据包（包括分片）的捕获截图，分析 IP 头部字段的正确性。检查版本号、首部长度、总长度、标识、标志位、片偏移、TTL、协议类型等字段，同时分析分片机制是否准确。）

分析 in.pcap 文件，其内容如下：



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.163.103	192.168.163.10	DNS	84	Standard query 0x96da A www.baidu.com OPT
2	0.000000	21:32:43:54:65:06	11:22:33:44:55:66	ARP	42	192.168.163.10 is at 21:32:43:54:65:06
3	0.000000	192.168.163.103	192.168.163.10	DNS	84	Standard query 0x9759 AAAA www.baidu.com OPT
4	0.000000	192.168.163.10	192.168.163.103	DNS	168	Standard query response 0x9759 AAAA www.baidu.com CNAM
5	0.000000	192.168.163.10	192.168.163.103	DNS	143	Standard query response 0x96da A www.baidu.com CNAME w
6	0.000000	192.168.163.103	192.168.163.10	DNS	87	Standard query 0x5a54 AAAA www.a.shifen.com OPT
7	0.000000	192.168.163.10	192.168.163.103	DNS	150	Standard query response 0x5a54 AAAA www.a.shifen.com S
8	0.000000	192.168.163.110	192.168.163.103	ICMP	98	Echo (ping) request id=0x0001, seq=1/256, ttl=64 (rep
9	0.000000	Pixim_34:45:56	11:22:33:44:55:66	ARP	42	192.168.163.110 is at 01:12:23:34:45:56
10	0.000000	1a:94:f0:3c:49:aa	Broadcast	ARP	42	Who has 192.168.163.103? Tell 192.168.163.2
11	0.000000	192.168.163.103	192.168.163.110	ICMP	98	Echo (ping) reply id=0x0001, seq=1/256, ttl=54 (rec
12	0.000000	192.168.163.2	192.168.163.103	SSH	106	Client: Encrypted packet (len=52)
13	0.000000	192.168.163.103	192.168.163.2	TCP	54	22 → 64289 [ACK] Seq=1 Ack=53 Win=65535 Len=0
14	0.000000	192.168.163.103	192.168.163.10	HTTP	141	GET / HTTP/1.1
15	0.000000	192.168.163.10	192.168.163.103	TCP	60	80 → 55428 [ACK] Seq=1 Ack=88 Win=65535 Len=0

第五条数据是本机收到的 IP 数据报，其内容如下：



```
Frame 5: 143 bytes on wire (1144 bits), 143 bytes captured (114
Ethernet II, Src: 21:32:43:54:65:06 (21:32:43:54:65:06), Dst: 1
Internet Protocol Version 4, Src: 192.168.163.10, Dst: 192.168.
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 129
    Identification: 0x88e5 (35045)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x29c4 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.163.10
    Destination Address: 192.168.163.103
    [Stream index: 0]
> User Datagram Protocol, Src Port: 53, Dst Port: 44571
> Domain Name System (response)
```

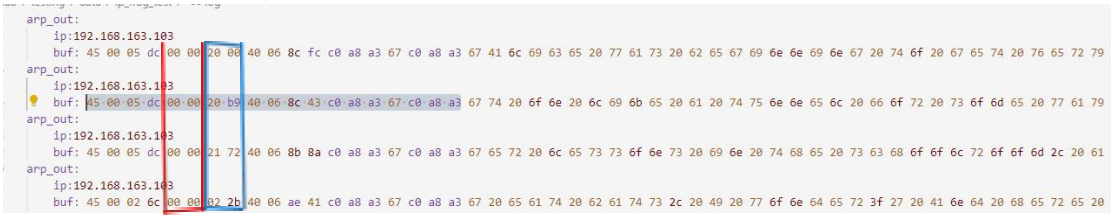
```
0000  11 22 33 44 55 66 21 32  43 54 65 06 08 00 45 00   ·"3DUf!2 CTe···E·
0010  00 81 88 e5 00 00 40 11  29 c4 c0 a8 a3 0a c0 a8   ····@· )·····
0020  a3 67 00 35 ae 1b 00 6d  bb f0 96 da 81 80 00 01   ·g·5···m ········
0030  00 03 00 00 00 01 03 77  77 77 05 62 61 69 64 75   ·······w ww·baidu
0040  03 63 6f 6d 00 00 01 00  01 c0 0c 00 05 00 01 00   ·com···· ········
0050  00 00 ec 00 0f 03 77 77  77 01 61 06 73 68 69 66   ·····ww ·a·shif
0060  65 6e c0 16 c0 2b 00 01  00 01 00 00 00 0b 00 04   en···+·· ········
0070  b7 e8 e7 ae c0 2b 00 01  00 01 00 00 00 0b 00 04   ·····+·· ········
0080  b7 e8 e7 ac 00 00 29 10  00 00 00 00 00 00 00      ·····)·· ·····
```

其中蓝色部分为 IP 头部字段，版本号为 4，首部长度为 5，即 20 字节，正确。总长度字段为 0x0081，即 129，标识位为 0x88e5，即 35045。片便宜为 0，MF 位为 0，说明该数据未分片。TTL 字段为 0x40，即 64。协议类型字段为 0x11，即 UDP 协议。源 IP 地址为 192.168.163.10，目的 IP 地址为 192.168.163.103，即本机，符合预期。

查看 log 文件中收到的数据，与之吻合，故正确。

```
Round 05 ----------------------------
udp_in:
    src_ip:192.168.163.10
    buf: 00 35 ae 1b 00 6d bb f0 96 da 81 80 00 01 00 03 00 00 00 01 03 77 77 77 05 62 61 69 64 75 03 63 6f 6d 00 00 01 00 01 c0 0c 00 05 00 01 00 00 00 ec 00 0f
```

对于 IP 分片发送，log 文件如下：



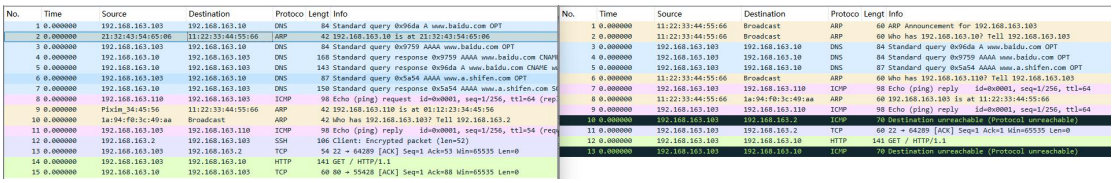
```
arp_out:
    ip:192.168.163.103
    buf: 45 00 05 dc 00 00 20 00 40 06 8c fc c0 a8 a3 67 c0 a8 a3 67 41 6c 69 63 65 20 77 61 73 20 62 65 67 69 6e 6e 69 6e 67 20 74 6f 20 67 65 74 20 76 65 72 79
arp_out:
    ip:192.168.163.103
    buf: 45 00 05 dc 00 00 20 b9 40 06 8c 43 c0 a8 a3 67 c0 a8 a3 67 74 20 6f 6e 20 6c 69 6b 65 20 61 20 74 75 6e 6e 65 6c 20 66 6f 72 20 73 6f 6d 65 20 77 61 79
arp_out:
    ip:192.168.163.103
    buf: 45 00 05 dc 00 00 21 72 40 06 8b 8a c0 a8 a3 67 c0 a8 a3 67 65 72 20 6c 65 73 73 6f 6e 73 20 69 6e 20 74 68 65 20 73 63 68 6f 6f 6c 72 6f 6f 6d 2c 20 61
arp_out:
    ip:192.168.163.103
    buf: 45 00 02 6c 00 00 02 2b 40 06 ae 41 c0 a8 a3 67 c0 a8 a3 67 20 65 61 74 20 62 61 74 73 2c 20 49 20 77 6f 6e 64 65 72 3f 27 20 41 6e 64 20 68 65 72 65 20
```

可以看到四个数据包的首部中，标识位都为 0x0000，因此可以确定是同一个包的数据分片。前三个数据包中的 MF 位均为 1，最后一个为 0.第一个包的片偏移为 0，第二个包位 0xb9（185），第三个包为 0x172（380），第四个包为 0x22b（555），均符合预期，因此可以认为分片机制是准确的。

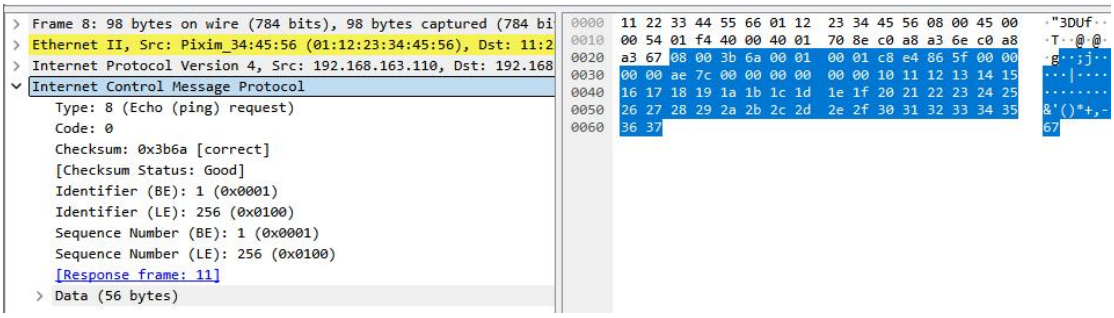
4. ICMP 协议实验结果及分析

（展示 ICMP 报文的捕获截图，分析其报文内容（包括差错报文和查询报文）。检查 ICMP 类型、代码、校验和等字段，以及报文携带的信息。）

下图中左边为 in.pcap 文件，右边为 out.pcap 文件。

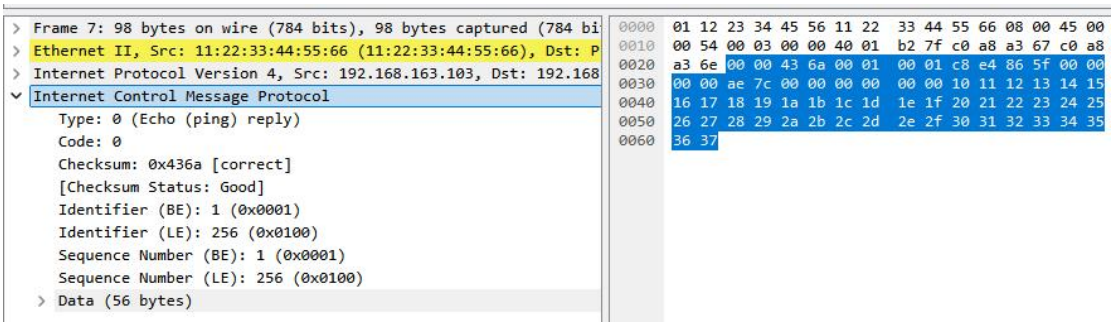


in 文件中的第 8 条数据是 ICMP 请求报文，其内容如下：



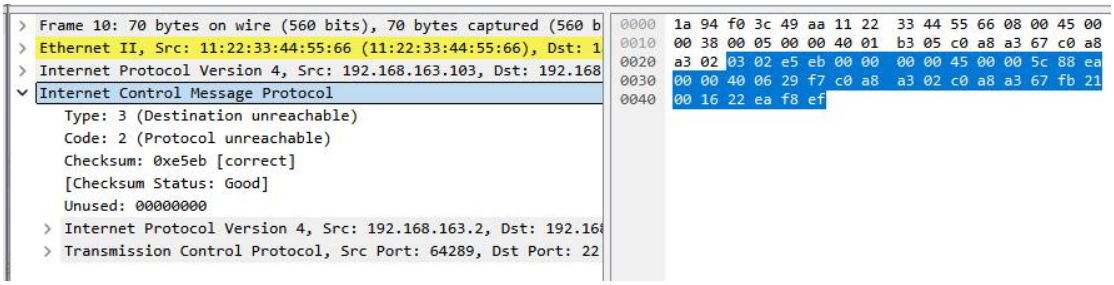
其类型字段为 0x08，代码字段为 0x00，校验和字段为 0x3b6a，标识符为 0x0001，序列号为 0x0001，之后为数据部分。

out 文件中的第七条数据为对上一天请求报文的 ICMP 响应报文，其内容如下：



其类型字段为 0x00，代码字段为 0x00，校验和字段为 0x436a，标识符为 0x0001，序列号为 0x0001，之后为数据部分，可以看到，数据部分和 ICMP 请求报文相同。

in 文件中的第 12 条数据为一个 SSH 协议的请求报文，不支持该请求，因此将会返回差错报文，差错代码为 unreachable。对应 out 文件中的第 10 条数据，如下图所示：



可以看到 type 字段为 0x03，代码字段为 0x02.代表这是一个差错报文，错误类型为 ubreachable。

5. UDP 协议实验结果及分析

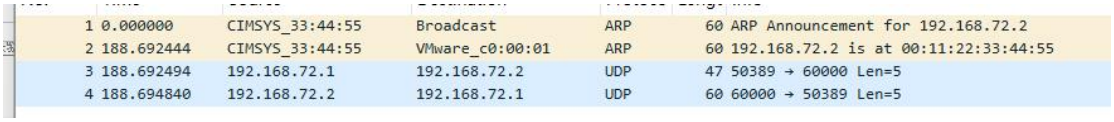
（展示 UDP 数据包的捕获截图，解析 UDP 头部和载荷内容，分析是否达到预期。检查源端口号、目的端口号、长度、校验和等字段，以及载荷数据。）

终端内容如下：

```
Using interface \Device\NPF_{BEA768AC-A4E0-4ABF-AFFE-DD433D4360C1}, my ip is 19      ..2.
recv udp packet from 192.      .:50389 len=5
HITSZ
```

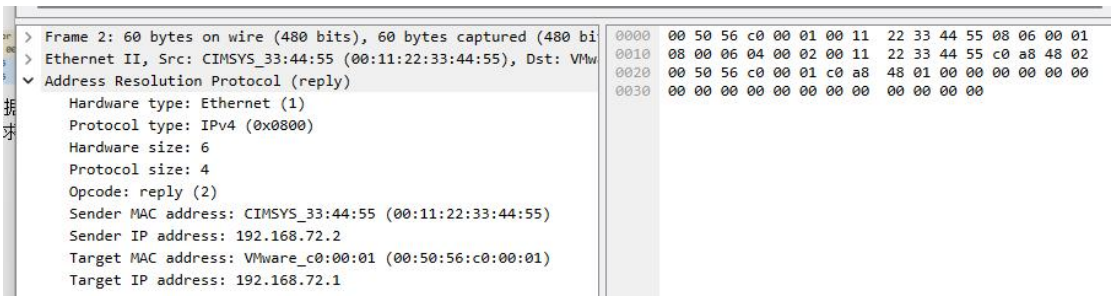
测试工具也正常收到响应：

wireshark 中捕获到的报文如下：



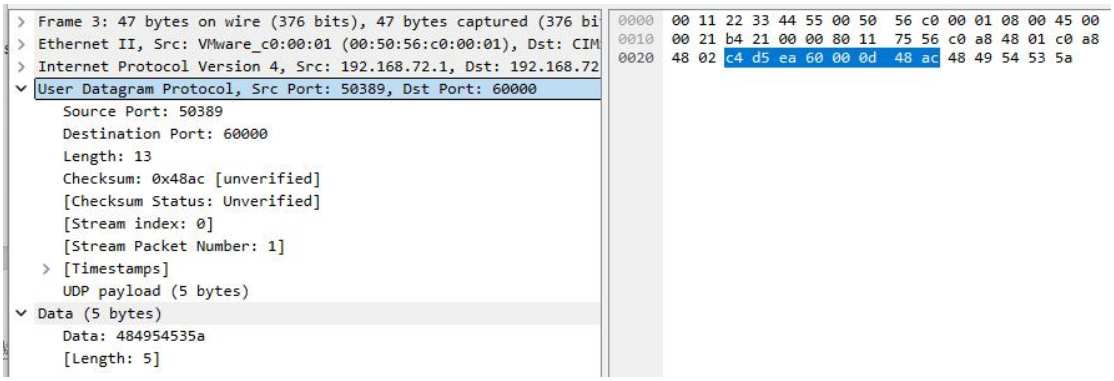
1	0.000000	CIMSYS_33:44:55	Broadcast	ARP	60 ARP Announcement for 192.168.72.2
2	188.692444	CIMSYS_33:44:55	VMware_c0:00:01	ARP	60 192.168.72.2 is at 00:11:22:33:44:55
3	188.692494	192.168.72.1	192.168.72.2	UDP	47 50389 → 60000 Len=5
4	188.694840	192.168.72.2	192.168.72.1	UDP	60 60000 → 50389 Len=5

第一个是 server 启动时发送的 ARP announcement。然后要向 server 发送数据，因为此时 arp table 中没有 server 对应的<ip, mac>项，因此要先广播发送一个 ARP 请求，收到 ARP 响应。第二条数据就是收到的 ARP 响应。其内容如下：



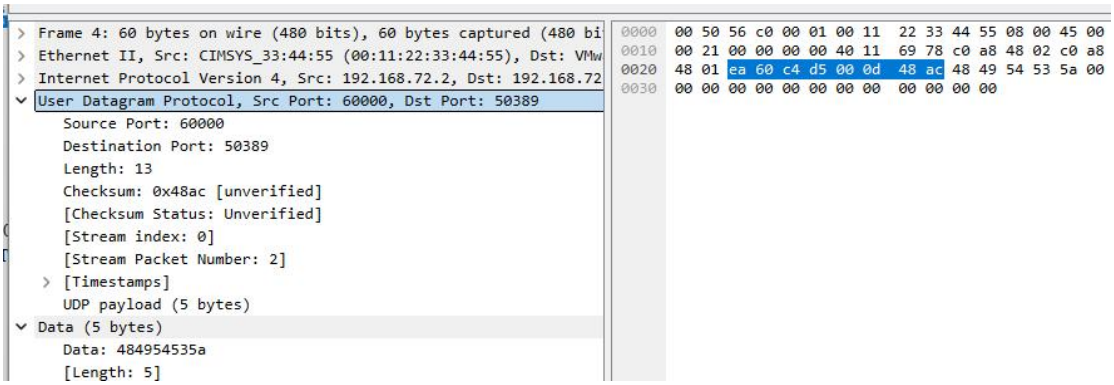
可以看到源 IP 地址为自定义的网卡 IP 地址，目的地址为本机 IP 地址。

得到 server 的 mac 地址后，就可以发送数据了，因此第三条数据为从本机到 server 的 udp 数据报，其内容如下：



可以看到 udp 首部中的源端口号为 0xc4d5（50389），目的端口号为 0xea60（60000），长度为 0x000d（13），校验和为 0x48ac，数据字段为 0x484954535a（HITSZ），均符合预期。

第四条数据为 server 回应本机的 udp 请求，其内容如下：



其源端口号为 0xea60（60000），目的端口号为 0xc4d5（50389），长度为 0x000d（13），校验和为 0x48ac，负载数据为 0x484954535a（HITSZ），均符合预期。

6. TCP 协议实验结果及分析

（展示 TCP 数据包的捕获截图，分析 TCP 连接的建立、数据传输和关闭过程。检查 TCP 头部的源端口号、目的端口号、序列号、确认号、标志位等字段，以及连接的状态转换。）

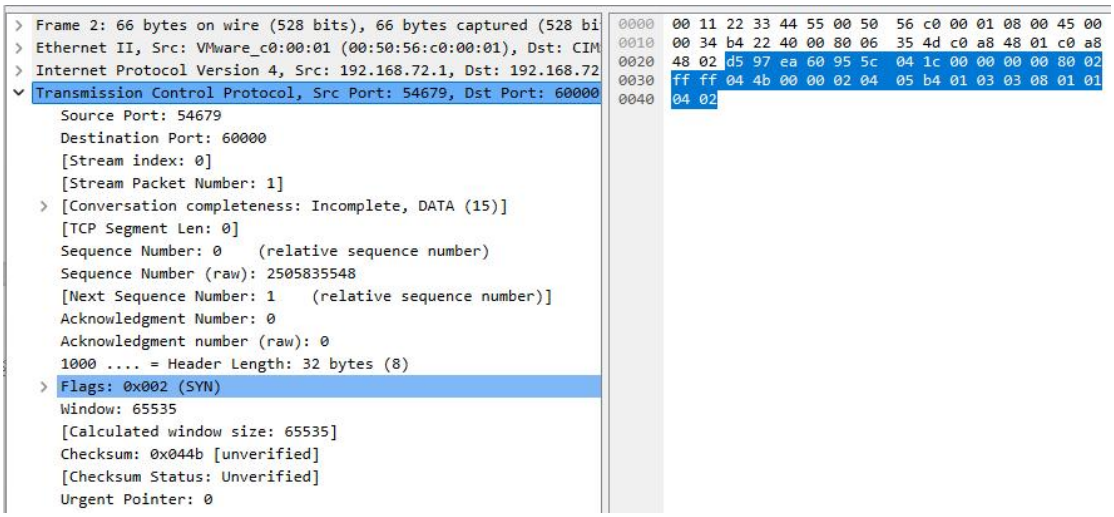
服务器终端输出如下：

```
Using interface \Device\NPF_{BEA768AC-A4E0-4ABF-AFFE-DD433D4360C1}, my ip is 192.168.72.2.
HITSZ
```

捕获到的包如下图所示：

1	0.000000	CIMSYS_33:44:55	Broadcast	ARP	60 ARP Announcement for 192.168.72.2
2	75.350393	192.168.72.1	192.168.72.2	TCP	66 54679 → 60000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=
3	75.352224	CIMSYS_33:44:55	Broadcast	ARP	60 Who has 192.168.72.1? Tell 192.168.72.2
4	75.352248	VMware_c0:00:01	CIMSYS_33:44:55	ARP	42 192.168.72.1 is at 00:50:56:c0:00:01
5	75.353152	192.168.72.2	192.168.72.1	TCP	60 60000 → 54679 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
6	75.353367	192.168.72.1	192.168.72.2	TCP	54 54679 → 60000 [ACK] Seq=1 Ack=1 Win=65535 Len=0
7	94.764881	192.168.72.1	192.168.72.2	TCP	59 54679 → 60000 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=5
8	94.765619	192.168.72.2	192.168.72.1	TCP	60 60000 → 54679 [ACK] Seq=1 Ack=6 Win=65535 Len=5
9	94.808716	192.168.72.1	192.168.72.2	TCP	54 54679 → 60000 [ACK] Seq=6 Ack=6 Win=65530 Len=0

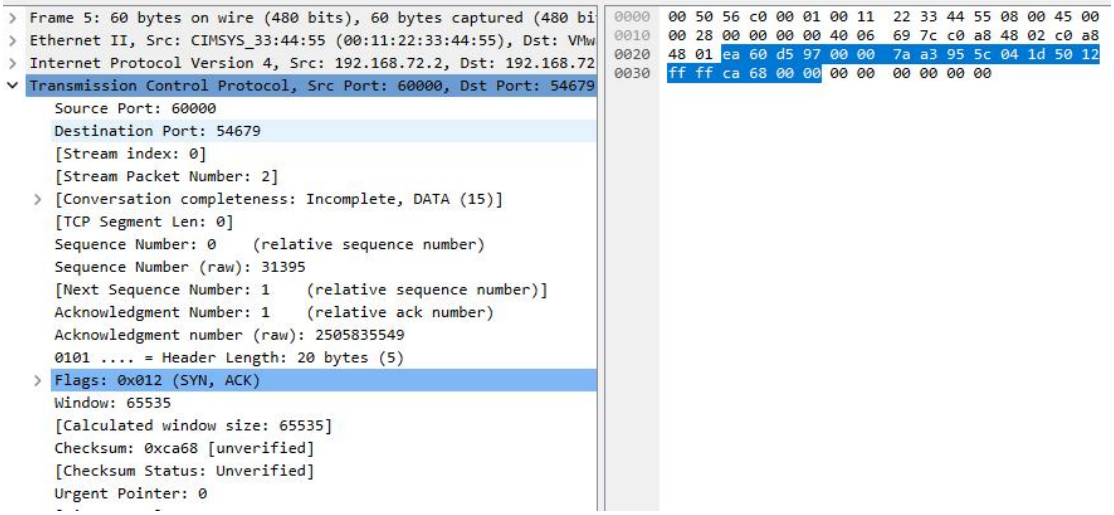
第一个包为服务器启动时广播的 ARP announcement。第二个包为本机向服务器发送的建立 TCP 请求的 TCP 数据报，是“第一次握手”，其内容如下图所示：



可以看到，TCP 数据报首部的源端口号为 0xd579（54679），目的端口号为 0xea60（60000），序列号为 0x955c041c（2505835548），确认号为 0x00000000，首部长度为 0x8（4*8 = 32 字节），保留位均为 0，标志位为 0x02（SYN=1，其余=0），校验和为 0x044b，紧急指针为 0x0000，后面的位为选项字段，没有携带数据。均符合预期。

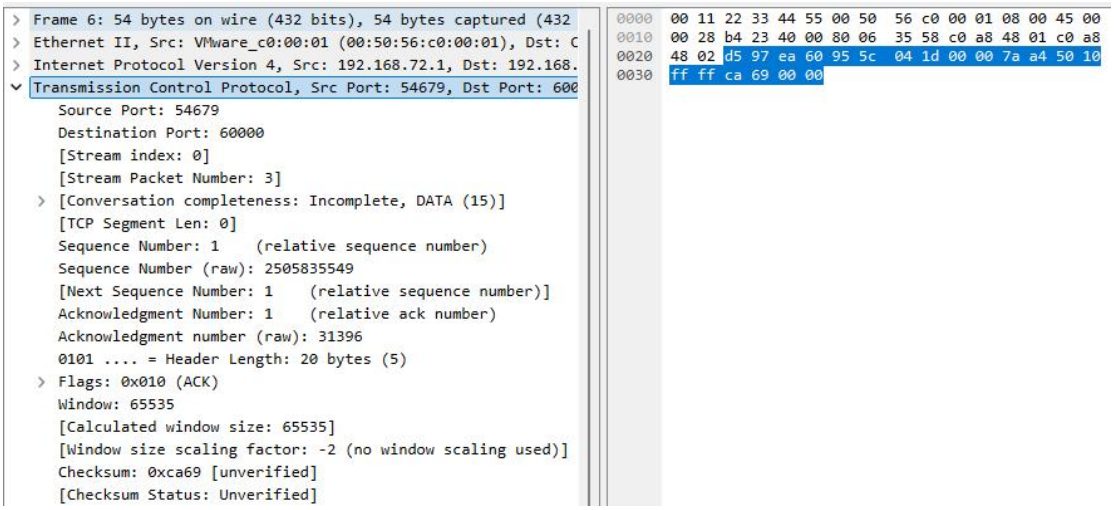
第三条数据为服务器广播的 ARP 请求报文，因为服务器要向本机发送数据，但是 arp table 中没有对应的 mac 地址。第四条数据时本机响应服务器的 ARP 响应报文，告知服务器本机的 mac 地址。然后服务器就可以向本机发送数据了。

第五条数据为服务器回应本机 TCP 建立连接请求的 TCP 响应请求，也就是“第二次握手”，其内容如下：



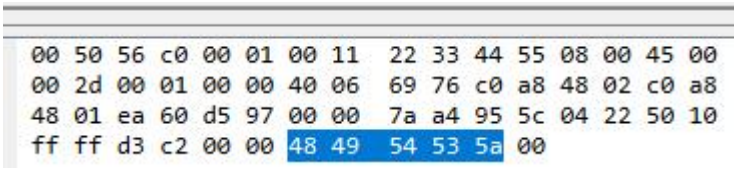
可以看到，TCP 数据报首部的源端口号为 0xea60（60000），目的端口号为 0xd579（54679），序列号为 0x00007aa3（31395），确认号为 0x955c041d（2505835549），首部长度为 0x5（4*5 = 20 字节），保留位均为 0，标志位为 0x12（ACK=1，SYN=1，其余=0），校验和为 0xca68，紧急指针为 0x0000。均符合预期。

第六条数据为“第三次握手”，其内容如下图所示：



序列号为 0x955c041d（2505835549），确认号为 0x00007aa4（31396），首部长度为 0x5（4*5 = 20 字节），标志位为 0x10（ACK=1，其余=0）。均符合预期。

第八条数据和第九条数据为后续的数据传输，分析与之前类似，不再赘述。唯一不同的是数据包的数据负载部分为 0x484954535a（HITSZ），如下图所示，符合预期。

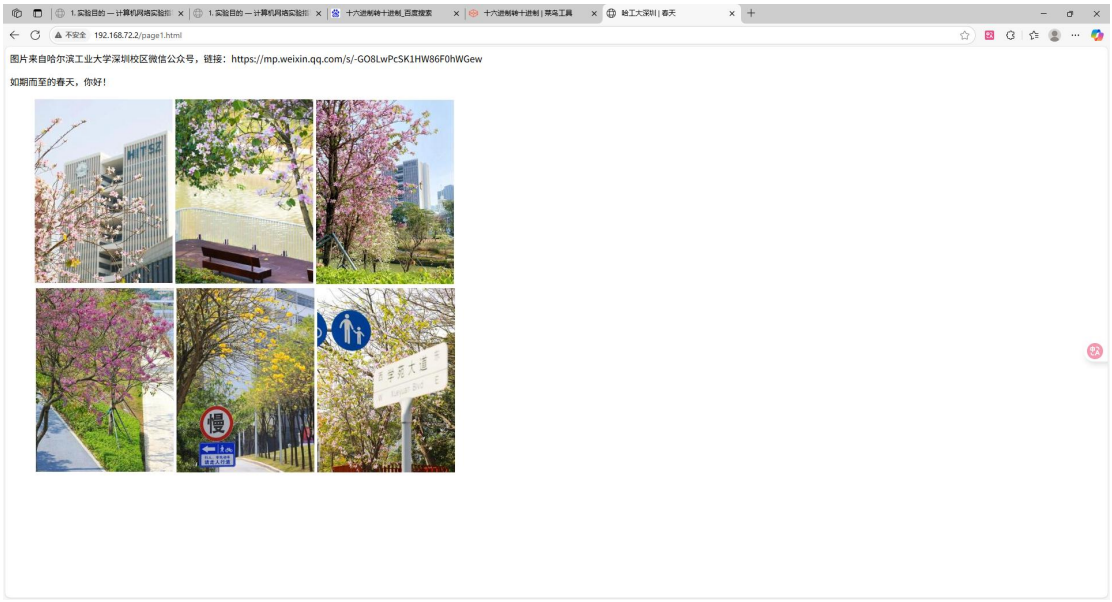


7. web 服务器实验结果及分析

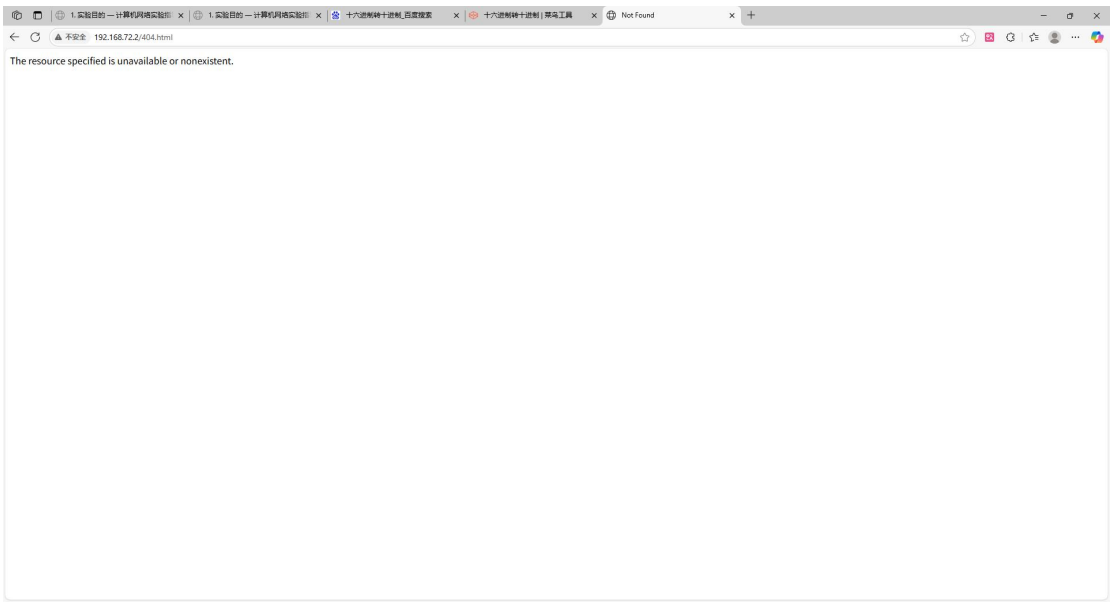
（展示 web 服务器的请求和响应过程截图，分析 HTTP 请求和响应的格式、内容。检查请求方法、请求 URL、请求头、响应状态码、响应头、响应体等部分。）

访问服务器 IP 地址根目录如下所示：

链接 1：



链接 2：

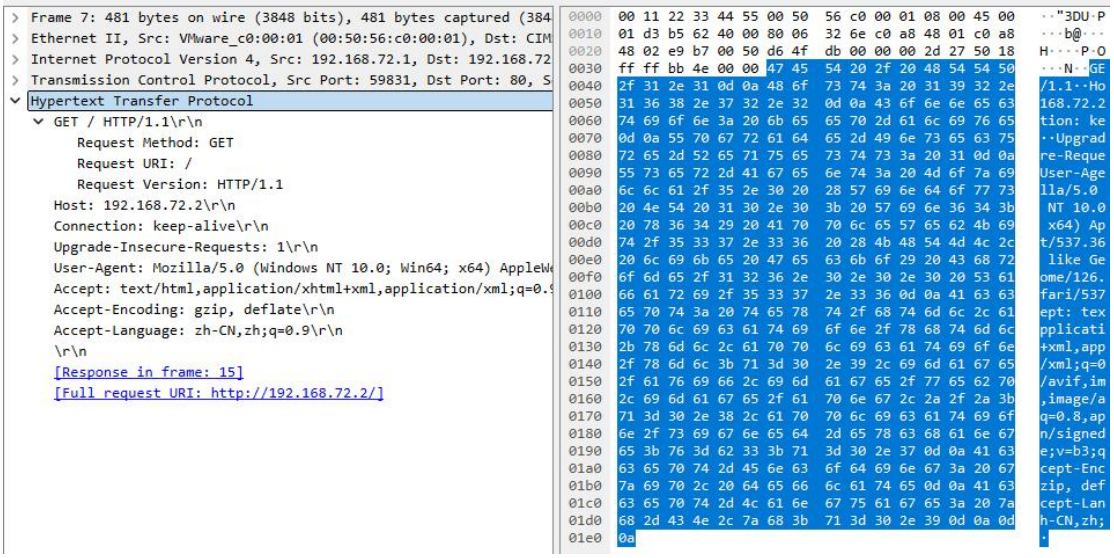


浏览器第一次访问服务器根目录时捕获的包如下：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.72.1	192.168.72.2	TCP	66	59831 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=
2	0.000213	192.168.72.1	192.168.72.2	TCP	66	59832 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=
3	0.002160	CIMSYS_33:44:55	Broadcast	ARP	60	Who has 192.168.72.1? Tell 192.168.72.2
4	0.002197	VMware_c0:00:01	CIMSYS_33:44:55	ARP	42	192.168.72.1 is at 00:50:56:c0:00:01
5	0.005340	192.168.72.2	192.168.72.1	TCP	60	80 → 59831 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
6	0.005577	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
7	0.005839	192.168.72.1	192.168.72.2	HTTP	481	GET / HTTP/1.1
8	0.006241	192.168.72.2	192.168.72.1	TCP	71	80 → 59831 [ACK] Seq=1 Ack=428 Win=65535 Len=17 [TC
9	0.006300	192.168.72.2	192.168.72.1	TCP	78	80 → 59831 [ACK] Seq=18 Ack=428 Win=65535 Len=24 [T
10	0.006331	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=428 Ack=42 Win=65494 Len=0
11	0.006349	192.168.72.2	192.168.72.1	TCP	94	80 → 59831 [ACK] Seq=42 Ack=428 Win=65535 Len=40 [T
12	0.006397	192.168.72.2	192.168.72.1	TCP	75	80 → 59831 [ACK] Seq=82 Ack=428 Win=65535 Len=21 [T
13	0.006428	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=428 Ack=103 Win=65433 Len=0
14	0.006442	192.168.72.2	192.168.72.1	TCP	60	80 → 59831 [ACK] Seq=103 Ack=428 Win=65535 Len=2 [T
15	0.006519	192.168.72.2	192.168.72.1	HTTP	556	HTTP/1.1 200 OK (text/html)
16	0.006555	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=428 Ack=607 Win=65535 Len=0
17	0.264440	192.168.72.1	192.168.72.2	HTTP	422	GET /favicon.ico HTTP/1.1
18	0.264814	192.168.72.2	192.168.72.1	TCP	78	80 → 59831 [ACK] Seq=607 Ack=796 Win=65535 Len=24 [
19	0.264900	192.168.72.2	192.168.72.1	TCP	78	80 → 59831 [ACK] Seq=631 Ack=796 Win=65535 Len=24 [
20	0.264945	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=796 Ack=655 Win=65487 Len=0
21	0.264962	192.168.72.2	192.168.72.1	TCP	79	80 → 59831 [ACK] Seq=655 Ack=796 Win=65535 Len=25 [
22	0.264997	192.168.72.2	192.168.72.1	TCP	75	80 → 59831 [ACK] Seq=680 Ack=796 Win=65535 Len=21 [
23	0.265028	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=796 Ack=701 Win=65441 Len=0
24	0.265041	192.168.72.2	192.168.72.1	TCP	60	80 → 59831 [ACK] Seq=701 Ack=796 Win=65535 Len=2 [T
25	0.265070	192.168.72.2	192.168.72.1	HTTP	158	HTTP/1.1 404 Not Found (text/html)
26	0.265099	192.168.72.1	192.168.72.2	TCP	54	59831 → 80 [ACK] Seq=796 Ack=807 Win=65335 Len=0

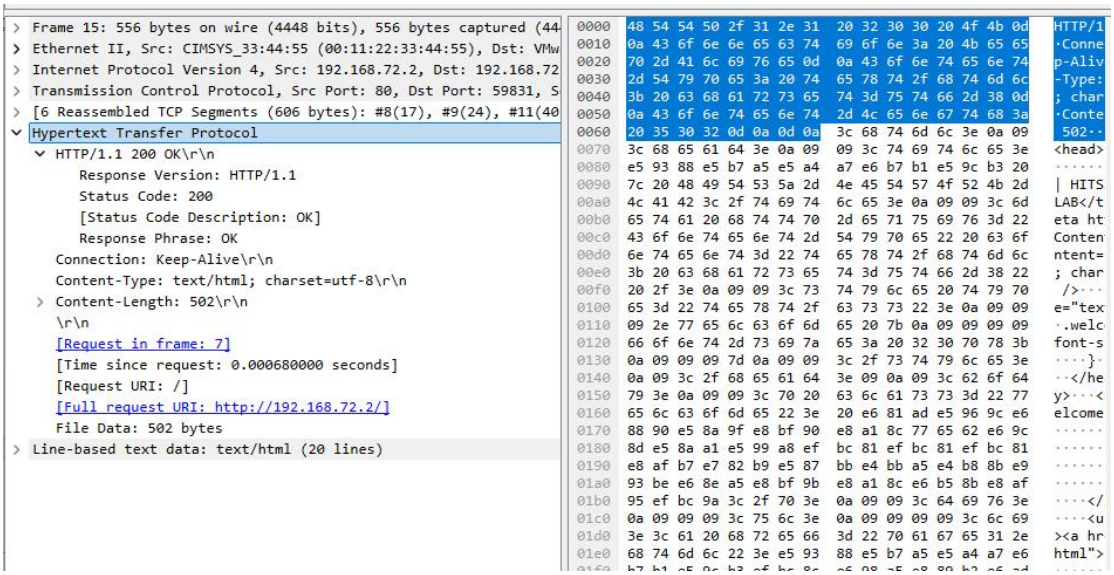
从第 3、4 条数据可以看到，服务器正确发送了 ARP 请求，并且本机回应了。第 2、5、6 条数据时 TCP 连接建立过程中的三次握手，从图中知其 flags 符合预期。

第七条数据为浏览器向服务器发送的 HTTP 请求，请求的对象应该是 index.html 文件。该数据包的内容如下：



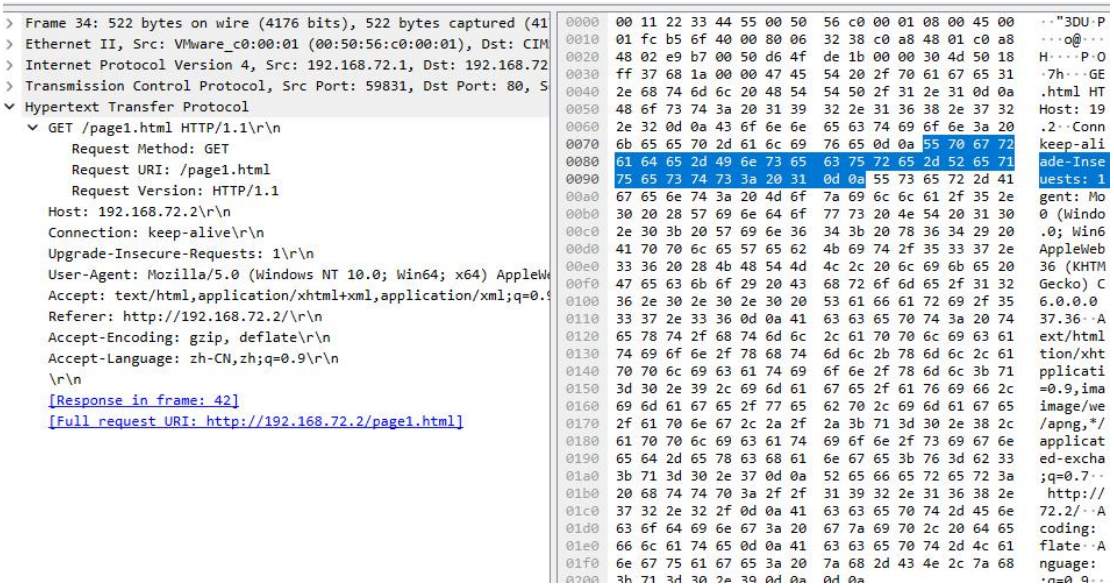
请求行为 `GET / HTTP/1.1\r\n` ，请求的 URI 为 / ，说明请求的确实是根目录，即 html.index 文件。协议类型为 HTTP/1.1，也是符合预期的。请求的方法为 GET，这是因为浏览器默认的请求方法就是 GET，也是符合预期的。

第十五条数据为服务器响应的 HTTP 响应消息。其主要内容如下：



从状态行中可以看出，协议类型为 HTTP/1.1，响应的状态码为 200，content-Type，content-Length 均符合预期。响应的数据负载类型为 text/html，也是符合预期的。从右边负载数据来看，确实是是 HTML 文件。

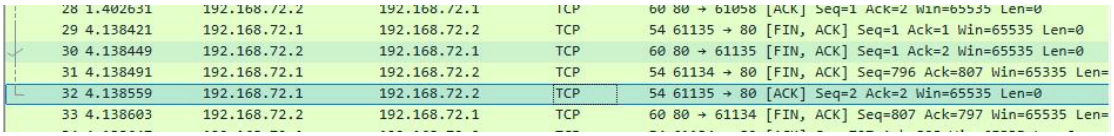
第 34 条数据是在浏览器中点击链接 1 后发送的 HTTP 请求，其主要内容如下：



与之前主要的不同就是请求的 URI 不同，为/page1.html。HTTP 响应消息与之前类似，不再赘述。

访问链接 2 时，由于链接不存在，因此应该会返状态码 404 的 HTTP 消息，第 321 条数据就是对访问链接 2 的请求的响应。可以看到状态码为 404 not found，符合预期。

关闭浏览器一段时间后，在 wireshark 中观察到 FIN 消息。

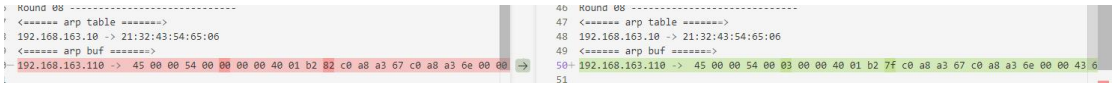


三、 实验中遇到的问题及解决方法

（详细描述在设计或测试过程中遇到的问题，包括错误描述、排查过程以及最终的解决方案。）

1. icmp_test 不通过

log 文件对比如下：

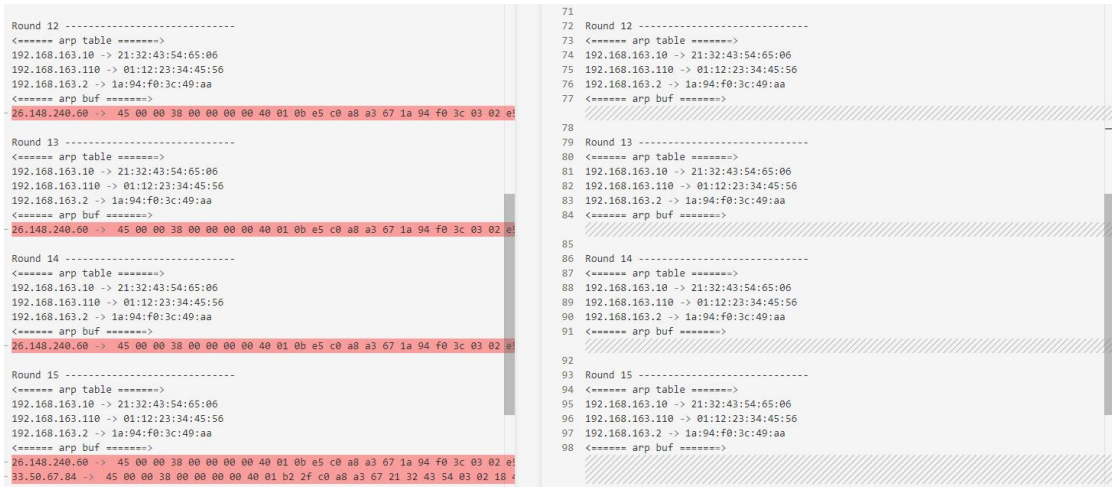


此处两个字节内容不同，根据 IP 数据报的首部结构可知，不同的字段为标识和校验和。而校验和的不同是标识导致的，因此问题出在标识。据此可以定位到 ip 数据包发送函数 ip_out，在发送数据包时，标识参数的传递有误，仅在分片时增加了 ip 标识。相关代码修改后如下：

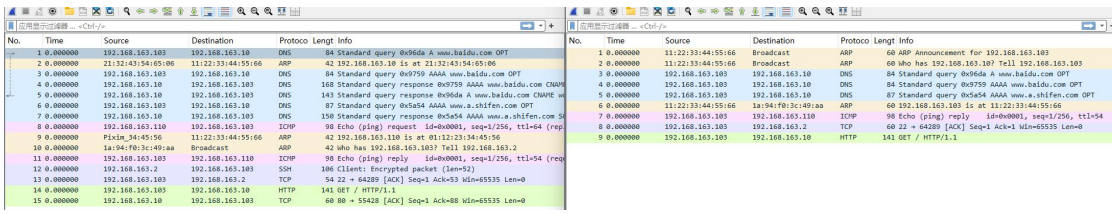
```
// Step1 检查从上层传递下来的数据报包长是否大于 IP 协议最大负载包长
static int packet_id = 0; // 数据包ID，每个数据包递增
int current_id = packet_id++;
if (buf->len <= max_payload) {
    // 直接发送
    ip_fragment_out(buf, ip, protocol, current_id, 0, 0);
    return;
}
```

之后该错误消失。

另一个错误是在 arp buf 中多了很多项。如下图所示。



分析 in.pcap 文件和 out.pcap 文件，如下所示（左边为 in，右边为 out）



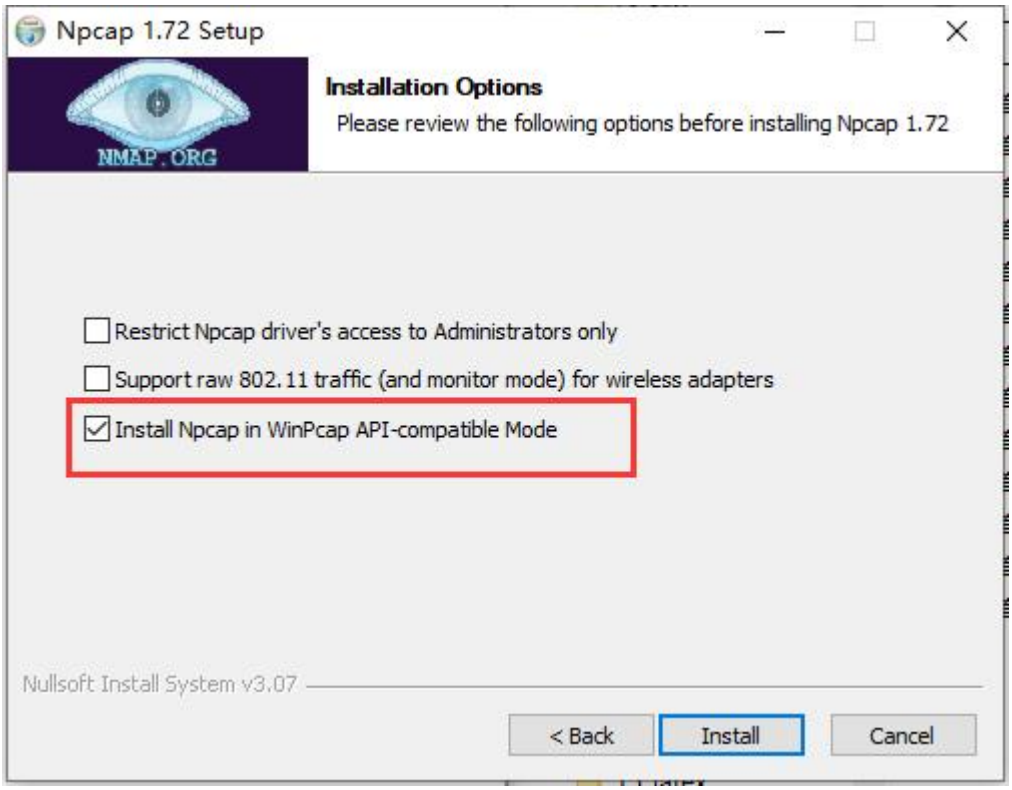
发现当本机收到 ssh 请求时，本应该发送 ARP 差错报文，但是实际上却发送了 ARP 请求报文。再结合 arp buf 中的信息，发现 ip 是错误的，因此推测在调用 icmp_unreachable 函数时 ip 参数传入错误了。经过逐步排查后，发现这个困扰了我很久的问题竟然只是由于一个微小的、粗心的错误造成的。错误代码如下：

```
// Step7 向上层传递数据包
int result = net_in(buf, ip_hdr->protocol, ip_hdr->src_ip);

// 若遇到不能识别的协议类型（通过net_in的返回结果判断）
if (result < 0) {
    // 在发出 ICMP 协议报文前，需重新加入 IP 报头，以确保报文格式的完
    buf_add_header(buf, sizeof(ip_hdr_t));

    // 调用 icmp_unreachable() 函数返回 ICMP 协议不可达信息
    icmp_unreachable(buf, src_mac, ICMP_CODE_PROTOCOL_UNREACH);
}
```

在 ip_in 中，当 net_in 发现不支持该协议，会调用 icmp_unreachable 函数，而在这个函数中我错误地将 ip 参数填写成了 mac，这才导致了这个十分难找的 bug。将这个参数修改为 ip 后通过测试。

2.下载 wireshark 后，无法正常编译

第一次做 net-lab 实验时，在按照实验指导书的常见错误中修改之后，可以正常编译编译。但后来我下载 wireshark 时，提示我之前安装的 npcap 版本过低，需要卸载后重新安装。但是重装后原本可以通过的测试却无法通过了，测试的错误代码不是 Failed，而是另外一个错误代码。排查后发现是我在重装是忘记了勾选下面第三个选项。



我再次重装之后，发现编译报错，有乱码，于是我推测是文件路径中有中文，修改后可正常编译。奇怪的是第一次实验时我正常编译了，但是第二次却不行，不知道是不是由于 npcap 版本变化带来的错误。

四、 实验收获和建议

（总结配置实验及协议栈实验过程中的实践收获，结合实操体验针对性提出实验流程优化及环境完善建议，为后续实验教学与研究的迭代改进提供参考依据。）

通过 VLAN 与配置实验我了解了 VLAN 的作用，掌握了 VLAN 的配置方法，熟悉了 Trunk 和 Hybrid 接口的配置和使用，把这些课堂上比较难懂的理论知识进行了简单的实践，对他们有了更加深刻和清晰的认知，虽然还比较粗浅，但相比实验之前确实感觉更加了解了。

通过 RIP 路由协议配置实验，我了解了如何手动配置路由，特亲身体会到了动态路由的方便，了解到了水平分割、触发更新、和毒性逆转机制在实际路由中到底是怎么作用的。虽然这部分内容理论比较简单易懂，课堂上已经基本清楚，但是通过实践让我印象更加深刻了。

通过 IPv6 网络过渡配置实验，我掌握了 IPv6 的基础配置方法，理解了 IPv6 数据报文的路由过程，知道了什么是 IPv6 Over IPv4 隧道技术，理解了其原理。这个实验也让我终于弄懂了一个困惑我很久的问题——既然说现在 IPv4 不够用了，要换到 IPv6，那为什么还在使用的基本上都是 IPv4。

在协议栈实验中，我从底向上实现了网络数据包的封装和解封装，对网络模型的每一层做了什么工作，数据如何在不同网络层次见正确、高效地传输有了更加深刻的了解。此外，我还初步掌握了 wireshark 工具的使用，知道如何抓取网络数据包了。通过最后小型 web 服务器的实验，我对一直感兴趣的 Tomcat 和 Nginx 服务器的原理有了初步的猜测和了解。

建议：

1. 测试文件可以再严格一点，现在的测试文件有很多问题都不能检测出来。
2. 特别提示一下下载 pcap 的版本选择比较新的版本，否则可能会和 wireshark 的冲突，并且导致一些不必要的问题。