

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机网络基础 |
| 实验名称： | Lab3 the network interface and IP router |
| 姓 名： | 祖敬涵 |
| 学 院： | 计算机学院 |
| 系： | 计算机科学与技术 |
| 专 业： | 信息安全 |
| 学 号： | 3220102091 |
| 指导教师： |  |

2024年 11月 13日

**浙江大学实验报告**

实验名称： Lab3 the network interface and IP router 实验类型： 设计实验

同组学生： 实验地点：

# 实验目的：

* 学习掌握网络接口的工作原理
* 学习掌握ARP地址解析协议相关知识
* 学习掌握IP路由的工作原理

# 实验内容

* 实现network interface，为每个下一跳地址查找（和缓存）以太网地址，即实现地址解析协议ARP。
* 实现简易路由器，对于给定的数据包，确认发送接口以及下一跳的IP地址。

# 主要仪器设备

* 联网的PC机
* Linux虚拟机

# 操作方法与实验步骤

**（1）地址解析协议：首先是实现网络接口，即完成ARP地址解析协议的代码编写。文件路径 sponge/libsponge/network\_interface.cc。**

1. 首先你需要阅读以下内容：

* [NetworkInterface](https://cs144.github.io/doc/lab5/class_network_interface.html)对象的公共接口。
* 维基百科对[ARP](https://en.wikipedia.org/wiki/Address_Resolution_Protocol)的总结以及原始的ARP规范（[RFC 826](https://www.rfc-editor.org/rfc/rfc826)）。
* [EthernetFrame](https://cs144.github.io/doc/lab5/class_ethernet_frame.html)和[EthernetHeader](https://cs144.github.io/doc/lab5/struct_ethernet_header.html)对象的文档和实现。
* [IPv4Datagram](https://cs144.github.io/doc/lab5/class_i_pv4_datagram.html)和[IPv4Header](https://cs144.github.io/doc/lab5/struct_i_pv4_header.html)对象的文档和实现（这可以解析和序列化一个Internet数据报，当序列化时，可以作为以太网帧的有效负载）。、
* [ARPMessage](https://cs144.github.io/doc/lab5/struct_a_r_p_message.html)对象的文档和实现。（它知道如何解析和序列化ARP消息，也可以在序列化时作为以太网帧的有效负载）。

2. 实现**network\_interface.cc**文件中的三个方法：自行在.hh文件里添加额外的成员变量。

1. NetworkInterface::send\_datagram()：当调用方希望向下一跳发送出IP数据包时，将调用此方法将此数据报转换为以太网帧并发送。

* 如果目标以太网地址已经知道，立即发送。创建一个类型为 **EthernetHeader::TYPE\_IPv4**的以太网帧，将序列化的数据报设置为负载，并设置源地址和目的地址。
* 如果目的以太网地址未知，则广播请求下一跳的以太网地址，并将IP数据报排队，以便在收到ARP应答后发送。

1. NetworkInterface::recv\_frame()：当以太网帧从网络到达时调用此方法，代码应该忽略任何不发送到该网络接口的帧。

* 如果入站帧是**IPv4**，将有效负载解析为**InternetDatagram（调用parse()）**，如果成功（即，**parse()**方法返回**ParseResult::NoError**），将结果**InternetDatagram**返回给调用者。
* 如果入站帧是ARP，将负载解析为**ARPMessage**。若成功，记住发送者的IP地址和以太网地址之间的映射关系30秒。另外，如果它是一个请求我们的IP地址的ARP请求，发送一个适当的ARP回复。

1. NetworkInterface::tick():此方法在到达时间点时调用，终止已过期的IP到Ethernet的映射。

3. 完成代码编写后，你可以运行**ctest -V -R “^arp”**命令来进行测试。

**（2）实现简易路由: 文件路径sponge/libsponge/root.cc。**

1. 首先你需要阅读下面的内容：[Router](https://cs144.github.io/doc/lab6/class_router.html)类的文档。

2. 实现**router.cc**文件中的两个主要方法：

1. Router::add\_route ()：这个方法将路由添加到路由表中，你需要在Router类中添加一个数据结构作为私有成员来存储这些信息，保存路由以备以后使用，即需要修改.hh文件。
2. Router::route\_one\_datagram：此方法需要将一个数据报从适当的接口路由到下一跳，它需要实现IP路由器的“最长前缀匹配”逻辑，以找到最好的路由。
3. 提示：

* 路由器在路由表中查找数据报目的地址匹配的路由，即目的地址的最长**prefix\_length**与**route\_prefix**的最长**prefix\_length**相同。
* 在匹配的路由中，路由器选择**prefix\_length**最长的路由。
* 如果没有匹配的路由，则丢弃该数据报。
* 路由器减少数据报的TTL（存活时间）。如果TTL已经为零，或者在减少之后达到零，路由器应该丢弃数据报。
* 否则，路由器将修改后的数据报从接口发送到适当的下一跳（**interface(interface\_num).send\_datagram()**）。

**（3）完成network interface和router的编写后，运行make check\_lab1命令来自动测试本次实验所需实现代码的正确性。**

**（4）温馨提示**

1. 你可能需要将IP地址从32位整数转换为一个地址对象，或是从地址对象转换为32位整数。遇到 这种情况你可以使用**Address::from\_ipv4\_numeric**方法以及**Address::ipv4\_numeric**方法解决。

2. 当你在开发代码的时候，可能会遇到无法解决的问题，下面给出解决的办法。

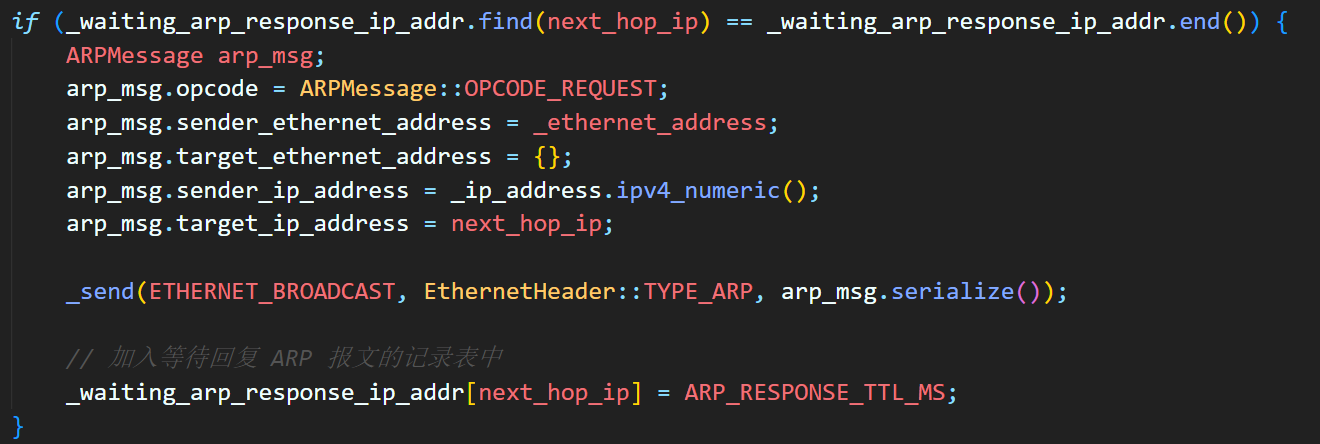
* 运行**cmake .. -DCMAKE\_BUILD\_TYPE=RelASan**命令配置build目录，使编译器能够检测内存错误和未定义的行为并给你很好的诊断。
* 你还可以使用valgrind工具。
* 你也可以运行**cmake .. -DCMAKE\_BUILD\_TYPE=Debug**命令配置并使用GNU调试器（**gdb**）。
* 你可以运行**make clean**和**cmake .. -DCMAKE\_BUILD\_TYPE=Release**命令重置构建系统。
* 如果你不知道如何修复遇到的问你题，你可以运行**rm -rf build**命令删除build目录，创建一个新的build目录并重新运行**cmake ..**命令。

# 实验数据记录和处理

* 实现network interface，即实现ARP协议的关键代码截图

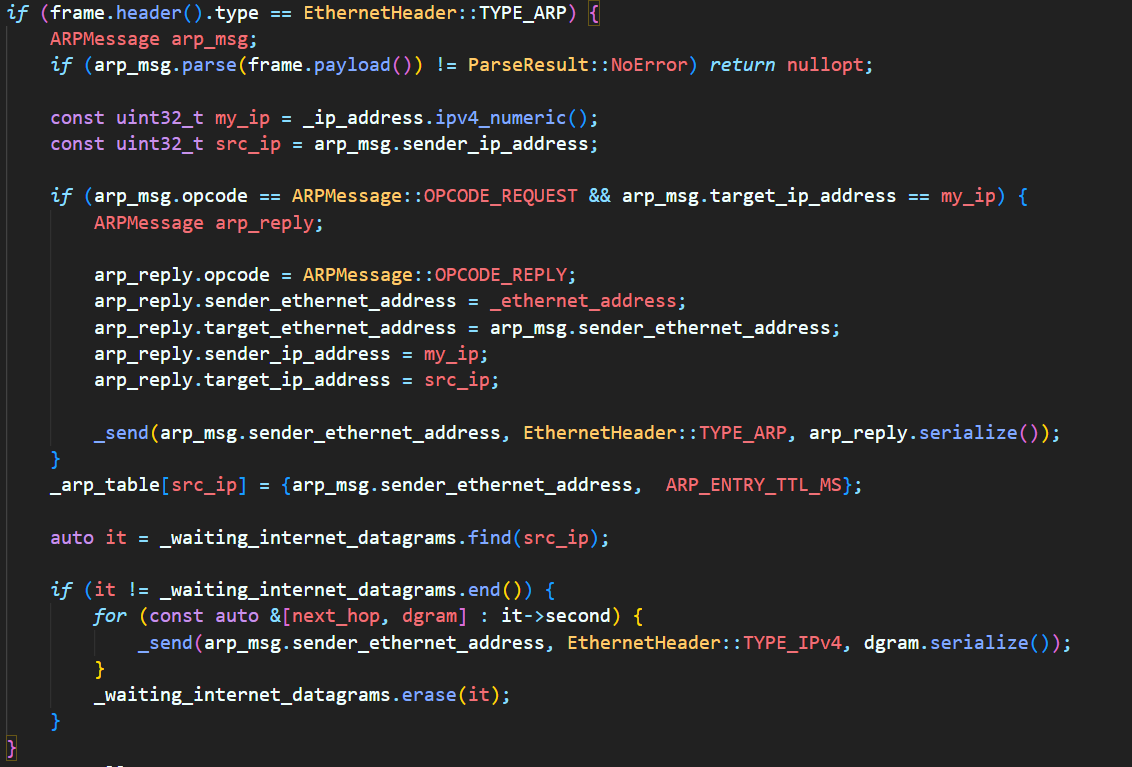
**1. ARP 请求发送：**

**在 `send\_datagram` 方法中，如果目标 IP 地址的 ARP 表项没有命中，会发送一个 ARP 请求：**



**2. ARP 响应处理：**

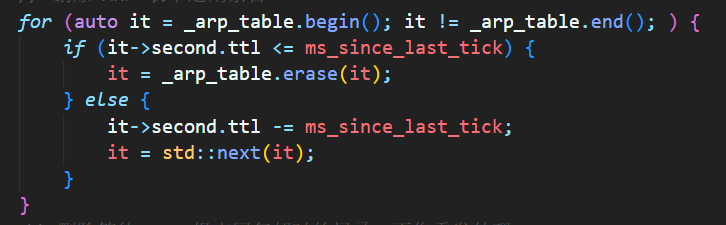
**在 `recv\_frame` 方法中，如果接收到的是 ARP 数据包，会进行解析，学习 IP-MAC 映射，并可能会根据 ARP 请求发送 ARP 回复：**



**3. ARP 表维护：**

**- 在 `recv\_frame` 中，解析 ARP 请求或响应并更新 ARP 表：**

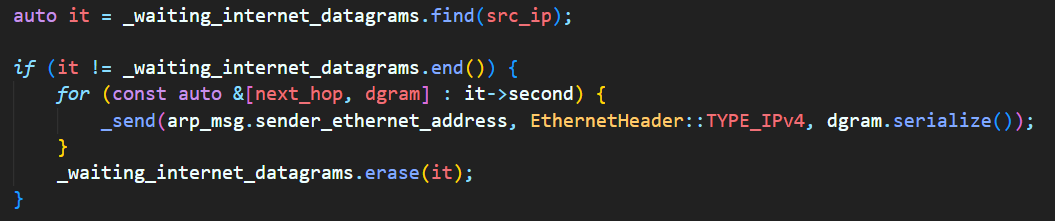
 **- 在 `tick` 方法中，会定期清理过期的 ARP 表项：**



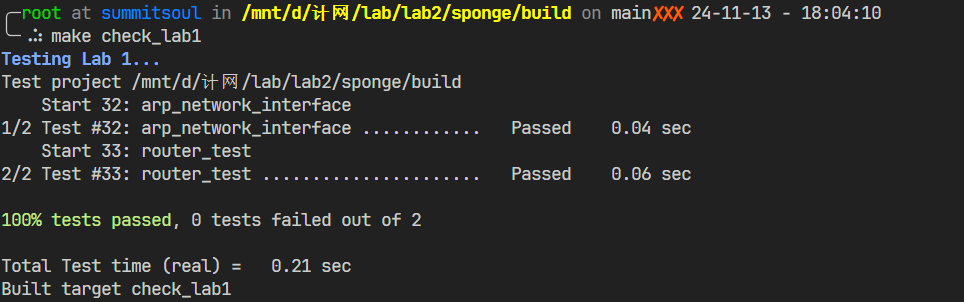
**4. 等待 ARP 响应并重发数据：**

**- 如果 ARP 请求后没有收到响应，`\_waiting\_arp\_response\_ip\_addr` 用于记录哪些 IP 地址等待 ARP 响应。**

**- 一旦收到 ARP 回复并解析成功，会检查是否有待发送的数据报，若有则继续发送：**



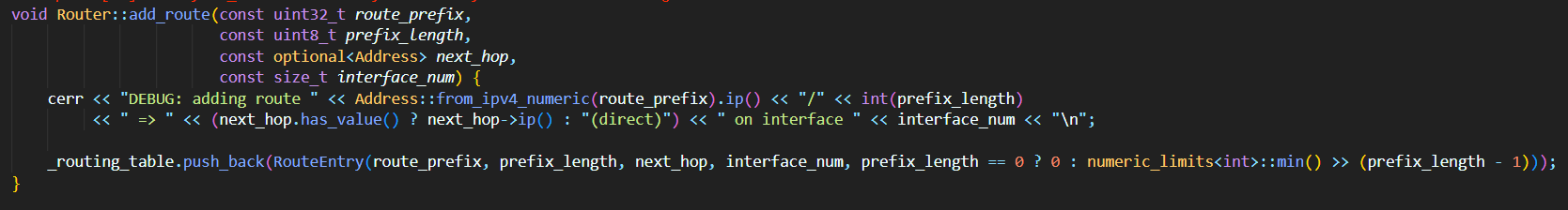
* 测试ARP协议的运行截图



* 实现简易路由的关键代码截图

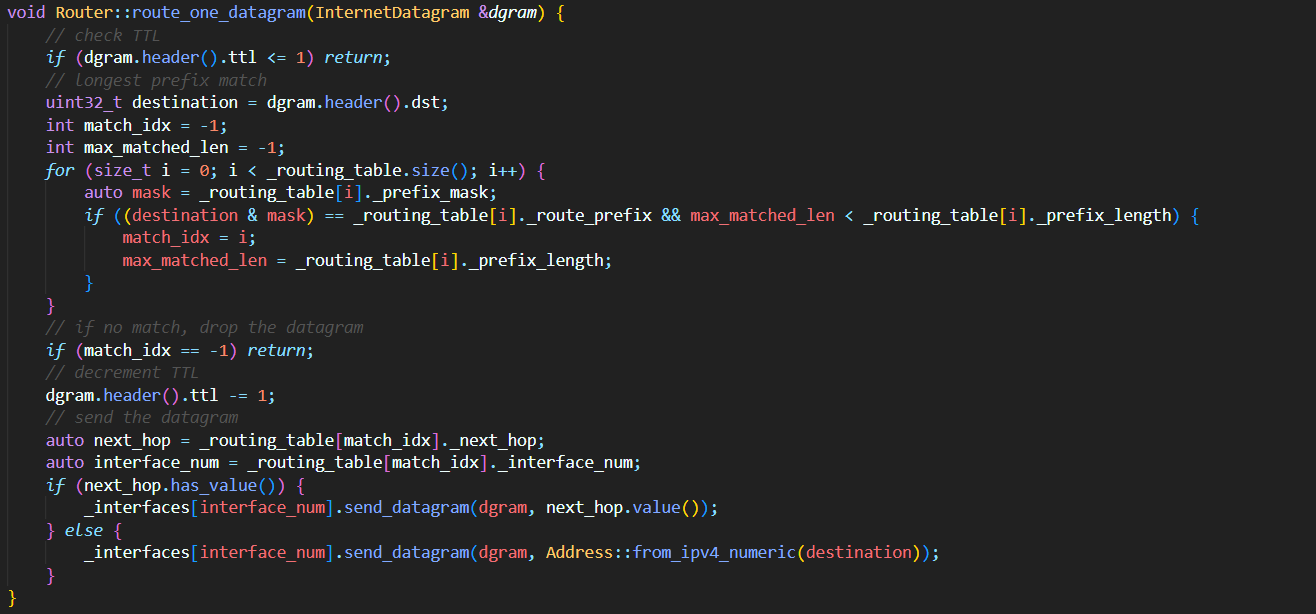
**1. 添加路由条目**

`Router::add\_route` 方法用于向路由表中添加一个新的路由条目。

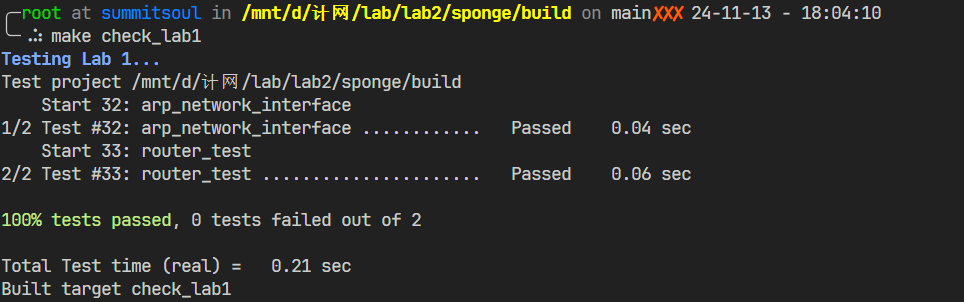


**2. 路由数据报**

`Router::route\_one\_datagram` 方法负责将传入的单个数据报送到适当的接口。



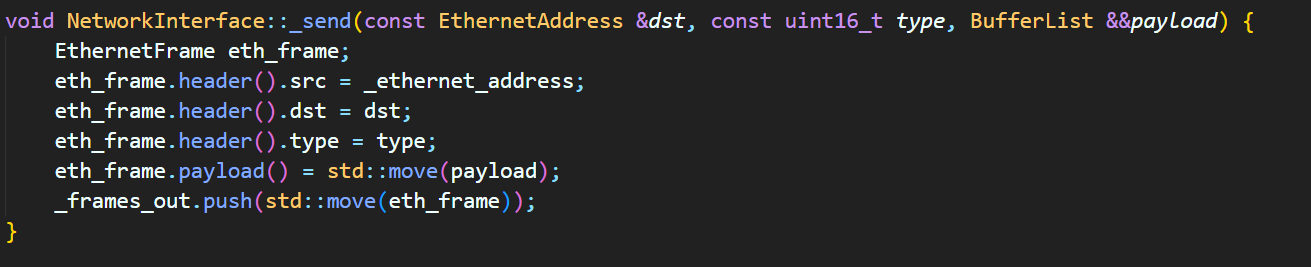
* 运行make check\_lab1命令的测试结果展示



# 实验数据记录和处理

根据你编写的程序运行效果，分别解答以下问题（看完请删除本句）：

* 通过代码，请描述network interface是如何发送一个以太网帧的？



1、创建以太网帧

2、设置帧头信息

3、设置负载

4、推送到输出队列

* 虽然在此次实验不需要考虑这种情况，但是当network interface发送一个ARP请求后如果没收到一个应答该怎么解决？请思考。
* 请描述一下你为了记录路由表所建立的数据结构？为什么？ `RouteEntry` 是路由表中的每一条路由的基本单位，它包含了路由条目的关键信息：
* class RouteEntry {
* public:
* const uint32\_t \_route\_prefix;
* const uint8\_t \_prefix\_length;
* const std::optional<Address> \_next\_hop;
* const size\_t \_interface\_num;
* const uint32\_t \_prefix\_mask;
* RouteEntry(uint32\_t *a*, uint8\_t *b*, std::optional<Address> *c*, size\_t *d*, uint32\_t *e*):\_route\_prefix(a), \_prefix\_length(b), \_next\_hop(c), \_interface\_num(d), \_prefix\_mask(e){}
* };

- `\_route\_prefix`：

这个字段保存了路由条目的路由前缀，是一个 32 位的 IPv4 地址（用 `uint32\_t` 表示）。这个前缀代表了路由目的地址的高位部分，用于与数据报的目标地址进行匹配。

- `\_prefix\_length`：

这是一个 8 位的字段，表示路由前缀的长度。它指定了路由前缀的高位部分有多少个比特用于匹配目标 IP 地址。

- `\_next\_hop`：

这是一个可选的字段，表示该路由条目的下一跳 IP 地址。如果目标地址直接连接到路由器（例如，目标地址就在同一网络中），该字段可能为空。

- `\_interface\_num`：

这个字段表示该路由条目使用的网络接口的编号。每个路由条目都需要知道通过哪个网络接口将数据包发送出去。

- `\_prefix\_mask`：

这是一个计算得到的掩码，用于将路由前缀与目标地址进行比较。它是一个 32 位的整数，在路由查找时通过按位与操作与目标地址进行匹配。

# 实验数据记录和处理