

浙江大学

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： 蔡佳伟

学 院： 计算机学院

系：

专 业： 软件工程

学 号： 3220104519

指导教师： 高艺

2024 年 11 月 5 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: 曹西 304

一、 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

二、 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。

三、 主要仪器设备

- 联网的 PC 机
- Linux 虚拟机

四、 操作方法与实验步骤

- 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。
 - 首先你需要阅读以下内容:
 1. [NetworkInterface](#) 对象的公共接口。
 2. 维基百科对 [ARP](#) 的总结以及原始的 ARP 规范 ([RFC 826](#))。
 3. [EthernetFrame](#) 和 [EthernetHeader](#) 对象的文档和实现。
 4. [IPv4Datagram](#) 和 [IPv4Header](#) 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)。
 5. [ARPMessgae](#) 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)。
 - 实现 `network_interface.cc` 文件中的三个主要方法:

```
1. void NetworkInterface::send_datagram(const InternetDatagram &dgram,
                                         const Address &next_hop)
```

 - 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。
 1. 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。
 2. 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。

```
2. optional<InternetDatagram> NetworkInterface::recv_frame(const EthernetFrame &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。

1. 如果入站帧是 **IPv4**，将有效负载解析为 **InternetDatagram**，如果成功（意味着 **parse()** 方法返回 **ParseResult::NoError**），将结果 **InternetDatagram** 返回给调用者。

2. 如果入站帧是 **ARP**，将负载解析为 **ARPMessage**，如果成功，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。

```
3. void NetworkInterface::tick(const size_t ms_since_last_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。
- 完成代码编写后，你可以运行 **ctest -V -R “^arp”** 命令来进行测试。

● 实现简易路由：同样的，为了实现一个简易的路由器，你需要实现提供的 **router.hh** 文件中的方法。

- 首先你需要阅读下面的内容：[Router](#) 类的文档。
- 实现 **router.cc** 文件中的两个主要方法：

```
void add_route(const uint32_t route_prefix,
               const uint8_t prefix_length,
               const optional<Address> next_hop,
               const size_t interface_num);
```

- 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。

```
void route_one_datagram(InternetDatagram &dgram);
```

- 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。

提示：

1. 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。
2. 在匹配的路由中，路由器选择 **prefix_length** 最长的路由。
3. 如果没有匹配的路由，则丢弃该数据报。
4. 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。
5. 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()**）。

● 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。

● 需要注意的是，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_ip4_numeric**

方法以及 `Address::ipv4_numeric` 方法解决。

- 温馨提示：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。

五、实验数据记录和处理

以下实验记录均需结合屏幕截图（截取源代码或运行结果），进行文字标注（看完请删除本句）。

- 实现 network interface，即实现 ARP 协议的关键代码截图

network_interface.cc:

```
libsponge > C:\network_interface.cc X C:\router.cc M C:\router.hh M
1 #include "network_interface.hh"
2
3 #include "arp_message.hh"
4 #include "ethernet_frame.hh"
5
6 #include <iostream>
7
8 // Dummy implementation of a network interface
9 // Translates from (IP datagram, next hop address) to link-layer frame, and from link-layer frame to IP datagram
10
11 // For Lab 5, please replace with a real implementation that passes the
12 // automated checks run by "make check_lab5".
13
14 // You will need to add private members to the class declaration in "network_interface.hh"
15
16 template <typename... Targs>
17 void DUMMY_CODE(Targs &&...) /* unused */ {}
18
19 using namespace std;
20
21 //! \param[in] ethernet_address Ethernet (what ARP calls "hardware") address of the interface
22 //! \param[in] ip_address IP (what ARP calls "protocol") address of the interface
23 NetworkInterface::NetworkInterface(const EthernetAddress &ethernet_address, const Address &ip_address)
24 : _ethernet_address(ethernet_address), _ip_address(ip_address) {
25     cerr << "DEBUG: Network interface has Ethernet address " << to_string(_ethernet_address) << " and IP address "
26          << ip_address.ip() << "\n";
27 }
28
29 //! \param[in] dgram the IPv4 datagram to be sent
30 //! \param[in] next_hop the IP address of the interface to send it to (typically a router or default gateway, but may also be another host if directly connected to the same network as the
31 //! (Note: the Address type can be converted to a uint32_t (raw 32-bit IP address) with the Address::ipv4_numeric() method.)
32 void NetworkInterface::send_datagram(const InternetDatagram &dgram, const Address &next_hop) {
33     // convert IP address of next hop to raw 32-bit representation (used in ARP header)
34     const uint32_t next_hop_ip = next_hop.ipv4_numeric();
35     auto it = _addr_cache.find(next_hop_ip); // 查找是否有缓存的以太网地址
36     if(it != _addr_cache.end()) { // 如果已经缓存，直接构造以太网帧并发送
37         _frames_out.emplace_back(make_frame((!it).second.first, EthernetHeader::TYPE_IPv4, dgram.serialize()));
38     }
39     else{
```

```

40     auto it1= _addr_request_time.find(next_hop_ip); // 没有缓存，检查是否需要发送ARP请求
41     if(it1== _addr_request_time.end() || _current_time-(*it1).second>3000){
42         ARPMessage msg;
43         msg.sender_ethernet_address= ethernet_address;
44         msg.sender_ip_address= ip_address.ipv4.numeric();
45         msg.target_ip_address=next_hop_ip;
46         msg.opcode=ARPMessage::OPCODE_REQUEST;
47         frames_out.emplace( make_frame(ETHERNET_BROADCAST,EthernetHeader::TYPE_ARP,BufferList(msg.serialize())));
48         if(it1== _addr_request_time.end()){ // 记录请求的时间
49             _addr_request_time.emplace(next_hop_ip, _current_time);
50         }
51         else{
52             (*it1).second= _current_time;
53         }
54         _waiting_dgrams.emplace_back(make_pair(next_hop_ip,dgram)); // 将待发送的IP数据报缓存起来，等 ARP 回复
55     }
56 }
57
58 //DUMMY_CODE(dgram, next_hop, next_hop_ip);
59 }
60
61 //! \param[in] frame the incoming Ethernet frame
62 optional<InternetDatagram> NetworkInterface::recv_frame(const EthernetFrame &frame) {
63     optional<InternetDatagram> res=nullopt;
64     const auto &header=frame.header();
65     if(header.dst== ethernet_address || header.dst==ETHERNET_BROADCAST){ // 帧的目标地址是本地地址或广播地址
66         if(header.type==EthernetHeader::TYPE_IPv4){ // IPv4类型的帧
67             InternetDatagram dgram;
68             if(dgram.parse(Buffer(frame.payload()))==ParseResult::NoError){
69                 res.emplace(dgram); // 成功解析，返回IP数据报
70             }
71         }
72         else if(header.type==EthernetHeader::TYPE_ARP){ // ARP类型的帧
73             ARPMessage msg;
74             if(msg.parse(Buffer(frame.payload()))==ParseResult::NoError){
75                 auto it= _addr_cache.find(msg.sender_ip_address); // 更新ARP缓存
76                 if(it!= _addr_cache.end()){
77                     (*it).second.first=msg.sender_ethernet_address;
78                     (*it).second.second= _current_time;
79                 }
80                 else{
81                     _addr_cache.emplace(msg.sender_ip_address,make_pair(msg.sender_ethernet_address, _current_time));
82                 }
83                 auto it1= _addr_request_time.find(msg.sender_ip_address); // 删除发送请求的IP地址缓存
84                 if(it1== _addr_request_time.end()){
85                     _addr_request_time.erase(it1);
86                 }
87                 try_send_waiting(msg.sender_ip_address); // 尝试发送等待中的数据报
88                 if(msg.opcode==ARPMessage::OPCODE_REQUEST && msg.target_ip_address==_ip_address.ipv4.numeric()){
89                     // 如果是ARP请求且目标IP地址匹配本机地址，发送ARP回复
90                     ARPMessage reply_msg;
91                     reply_msg.sender_ethernet_address= ethernet_address;
92                     reply_msg.sender_ip_address= ip_address.ipv4.numeric();
93                     reply_msg.target_ethernet_address=msg.sender_ethernet_address;
94                     reply_msg.target_ip_address=msg.sender_ip_address;
95                     reply_msg.opcode=ARPMessage::OPCODE_REPLY;
96                     _frames_out.emplace( make_frame(msg.sender_ethernet_address,EthernetHeader::TYPE_ARP,BufferList(reply_msg.serialize())));
97                 }
98             }
99         }
100     }
101     //DUMMY_CODE(frame);
102     return res; // 返回解析结果
103 }
104
105 //! \param[in] ms_since_last_tick the number of milliseconds since the last call to this method
106 void NetworkInterface::tick(const size_t ms_since_last_tick) {
107     _current_time += ms_since_last_tick;
108     // 更新时间并移除过期的缓存
109     for(auto it= _addr_cache.begin(); it!= _addr_cache.end(); ){
110         if( _current_time-(*it).second.second>30000){ // 超过30秒的缓存改为过期
111             it= _addr_cache.erase(it);
112         }
113         else{
114             ++it;
115         }
116     }
117     // DUMMY_CODE(ms_since_last_tick);
118 }
119
120 void NetworkInterface::try_send_waiting(uint32_t new_ip){
121     for(auto it= _waiting_dgrams.begin(); it!= _waiting_dgrams.end(); ){
122         if((*it).first==new_ip){
123             frames_out.emplace( make_frame(_addr_cache[new_ip].first,EthernetHeader::TYPE_IPv4,(*it).second.serialize()));
124             it= _waiting_dgrams.erase(it); // 移除已发送的等待数据报
125         }
126         else{
127             ++it;
128         }
129     }
130 }
131
132 EthernetFrame NetworkInterface::make_frame(const EthernetAddress &dst, uint16_t type, const BufferList &payload){
133     EthernetFrame frame; // 构造以太网帧，封装目标地址、类型和有效负载
134     frame.header().src= ethernet_address;
135     frame.header().dst=dst;
136     frame.header().type=type;
137     frame.payload().append(payload);
138     return frame;
139 }

```

network_interface.hh:

```

33 class NetworkInterface {
34 private:
35     //! Ethernet (known as hardware, network-access-layer, or link-layer) address of the interface
36     EthernetAddress _ethernet_address;
37
38     //! IP (known as internet-layer or network-layer) address of the interface
39     Address _ip_address;
40
41     //! outbound queue of Ethernet frames that the NetworkInterface wants sent
42     std::queue<EthernetFrame> _frames_out{};
43
44     size_t _current_time{0};
45     std::map<uint32_t, std::pair<EthernetAddress, size_t>> _addr_cache{};
46     std::map<uint32_t, size_t> _addr_request_time{};
47     std::deque<std::pair<uint32_t, InternetDatagram>> _waiting_dgrams{};
48     void _remove_expired_cache();
49     void _try_send_waiting(uint32_t new_ip);
50     EthernetFrame _make_frame(const EthernetAddress &dst, uint16_t type, const BufferList &payload);
51 }

```

只添加了类内 private 变量和函数。

- 测试 ARP 协议的运行截图

```
cjwjiwang@cjwjiwang-virtual-machine:~/lab2/sponge/build$ ctest -V -R "^arp"
UpdateCTestConfiguration from :/home/cjwjiwang/lab2/sponge/build/DartConfigurat
ion.tcl
UpdateCTestConfiguration from :/home/cjwjiwang/lab2/sponge/build/DartConfigurat
ion.tcl
Test project /home/cjwjiwang/lab2/sponge/build
Constructing a list of tests
Done constructing a list of tests
Updating test list for fixtures
Added 0 tests to meet fixture requirements
Checking test dependency graph...
Checking test dependency graph end
test 32
  Start 32: arp_network_interface

32: Test command: /home/cjwjiwang/lab2/sponge/build/tests/net_interface
32: Test timeout computed to be: 10000000
32: DEBUG: Network interface has Ethernet address 56:de:04:5f:37:11 and IP addre
ss 4.3.2.1
32: DEBUG: Network interface has Ethernet address 22:0b:e7:d0:b5:db and IP addre
ss 5.5.5.5
32: DEBUG: Network interface has Ethernet address 8e:3c:e4:ad:62:7b and IP addre
ss 5.5.5.5
32: DEBUG: Network interface has Ethernet address 8e:3c:e4:ad:62:7b and IP addre
ss 5.5.5.5
32: DEBUG: Network interface has Ethernet address 0a:ce:a1:b1:dd:0b and IP addre
ss 1.2.3.4
32: DEBUG: Network interface has Ethernet address a2:45:e5:d9:00:9f and IP addre
ss 4.3.2.1
32: DEBUG: Network interface has Ethernet address 32:6c:cf:86:7c:49 and IP addre
ss 10.0.0.1
1/1 Test #32: arp_network_interface ..... Passed    0.00 sec

The following tests passed:
  arp_network_interface

100% tests passed, 0 tests failed out of 1

Total Test time (real) = 0.01 sec
```

- 实现简易路由的关键代码截图

router.cc:

```
lib sponge > @ router.cc > @ route_one_datagram(InternetDatagram&)
25 void Router::add_route(const uint32_t route_prefix,
26                        const uint32_t prefix_length,
27                        const optional<Address> next_hop,
28                        const size_t interface_num) {
29     cerr << "DEBUG: adding route " << Address::from_ipv4_numeric(route_prefix).ip() << "/" << int(prefix_length)
30     << " => " << (next_hop.has_value() ? next_hop->ip() : "(direct)") << " on interface " << interface_num << "\n";
31
32     // DUMMY CODE(route_prefix, prefix_length, next_hop, interface_num);
33     // Your code here.
34
35     // 使用路由前缀和掩码，添加新路由
36     uint32_t route_round = route_prefix & get_mask(prefix_length);
37     // 查找现有路由表，若没有该路由或现有路由的前缀长度更小，则更新路由表
38     auto iter = routing_table.find(route_round);
39     if(iter == routing_table.end() || iter->second.prefix_length < prefix_length){
40         routing_table[route_round] = {prefix_length, next_hop, interface_num};
41     }
42 }
43
44 // [param[in] dgram The datagram to be routed
45 void Router::route_one_datagram(InternetDatagram& dgram) {
46     // DUMMY CODE(dgram);
47     // Your code here.
48     uint32_t dst_ip = dgram.header().dst; // 获取数据包的目标 IP 地址
49     for(uint32_t pre_len = 32; pre_len > 0; --pre_len){ // 从最长前缀开始，查找匹配的路由
50         uint32_t mask = get_mask(pre_len); // 获取对应前缀长度的掩码
51         auto iter = routing_table.find(route_round);
52         if(iter == routing_table.end() || iter->second.prefix_length < pre_len){ // 找到匹配路由，并且前缀长度相等
53             if(dgram.header().ttl < iter->second.ttl){
54                 return;
55             }
56             if(dgram.header().ttl <= 0) return; // 如果数据包的TTL小于等于0，直接丢弃
57             RouteItem item(iter->second); // 获取路由项
58             if(item.next_hop.has_value()){ // 有下一跳地址，发送到下一跳
59                 interfaces[item.interface_num].send_datagram(dgram, item.next_hop.value());
60             }
61             else{ // 没有下一跳地址，说明是直接连接的网络，发送到目标地址
62                 interfaces[item.interface_num].send_datagram(dgram, Address::from_ipv4_numeric(dgram.header().dst));
63             }
64         }
65     }
66 }
```

```

62     }
63     return;
64 }
65 }
66 }
67
68 void Router::route() {
69     // Go through all the interfaces, and route every incoming datagram to its proper outgoing interface.
70     for (auto &interface : _interfaces) {
71         auto &queue = interface.datagrams_out();
72         while (not queue.empty()) {
73             route_one_datagram(queue.front());
74             queue.pop();
75         }
76     }
77 }
78

```

router.hh:

```

class Router {
    //! The router's collection of network interfaces
    std::vector<AsyncNetworkInterface> _interfaces{};

    //! Send a single datagram from the appropriate outbound interface to the next hop,
    //! as specified by the route with the longest prefix_length that matches the
    //! datagram's destination address.
    void route_one_datagram(InternetDatagram &dgram);

    struct RouteItem{
        size_t prefix_length{0};
        std::optional<Address> next_hop{std::nullopt};
        size_t interface_num{0};
    };
    std::unordered_map<uint32_t,RouteItem> _routing_table{};
}

```

只添加了类内 private 变量和函数。

- 运行 make check_lab1 命令的测试结果展示

```

cjwjiwang@cjwjiwang-virtual-machine: ~/lab2/sponge/build
[ 95%] Linking CXX executable send_extra
[ 95%] Built target send_extra
Consolidate compiler generated dependencies of target net_interface
[ 96%] Linking CXX executable net_interface
[ 96%] Built target net_interface
[ 97%] Linking CXX executable address_dt
[ 97%] Built target address_dt
[ 97%] Linking CXX executable parser_dt
[ 98%] Built target parser_dt
[100%] Linking CXX executable socket_dt
[100%] Built target socket_dt
cjwjiwang@cjwjiwang-virtual-machine:~/lab2/sponge/build$ make check_lab1
Testing Lab 1...
Test project /home/cjwjiwang/lab2/sponge/build
  Start 32: arp_network_interface
1/2 Test #32: arp_network_interface ..... Passed    0.00 sec
  Start 33: router_test
2/2 Test #33: router_test ..... Passed    0.01 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =  0.08 sec
Built target check_lab1
cjwjiwang@cjwjiwang-virtual-machine:~/lab2/sponge/build$

```


六、实验数据记录和处理

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

当需要发送 IP 数据报时，会调用 `NetworkInterface::send_datagram` 方法。`send_datagram` 方法会尝试查找目标 IP 地址的 MAC 地址，由 `next_hop` 下一跳地址给出。`_addr_cache` 是一个缓存，存储了 IP 地址与对应的 MAC 地址（以太网地址）的映射。

```
// convert IP address of next hop to raw 32-bit representation (used in ARP header)
const uint32_t next_hop_ip = next_hop.ipv4_numeric();
auto it = _addr_cache.find(next_hop_ip); // 查找是否有缓存的以太网地址
```

如果找到了 IP 地址对应的 MAC 地址，直接创建以太网帧，并将其放入 `_frames_out` 队列准备发送。

```
if(it!=_addr_cache.end()){ // 如果已经缓存，直接构造以太网帧并发送
    _frames_out.emplace(_make_frame>(*it).second.first,EthernetHeader::TYPE_IPv4,dgram.serialize());
}
```

如果没有找到，程序会检查是否已经发送过 ARP 请求，如果还没有发送过或 ARP 请求过期，会发送一个 ARP 请求，并将请求加入缓存，等待 ARP 回复。其中 ARP 请求会为目标 IP 地址解析出 MAC 地址。

在发送 ARP 请求时，ARP 消息被封装在一个以太网帧中并广播。而 `_make_frame` 函数构造了这个以太网帧。

```
else{
    auto it1= _addr_request_time.find(next_hop_ip); // 没有缓存，检查是否需要发送ARP请求
    if(it1==_addr_request_time.end() || _current_time-(*it1).second>5000){
        ARPMessage msg;
        msg.sender_ethernet_address= ethernet_address;
        msg.sender_ip_address= ip_address.ipv4_numeric();
        msg.target_ip_address=next_hop_ip;
        msg.opcode=ARPMessage::OPCODE_REQUEST;
        _frames_out.emplace(_make_frame(ETHERNET_BROADCAST,EthernetHeader::TYPE_ARP,BufferList(msg.serialize())));
        if(it1==_addr_request_time.end()){ // 记录请求的时间
            _addr_request_time.emplace(next_hop_ip,_current_time);
        }
        else{
            (*it1).second=_current_time;
        }
        _waiting_dgrams.emplace_back(make_pair(next_hop_ip,dgram)); // 将待发送的IP数据报缓存起来，等 ARP 回复
    }
}
```

- 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。

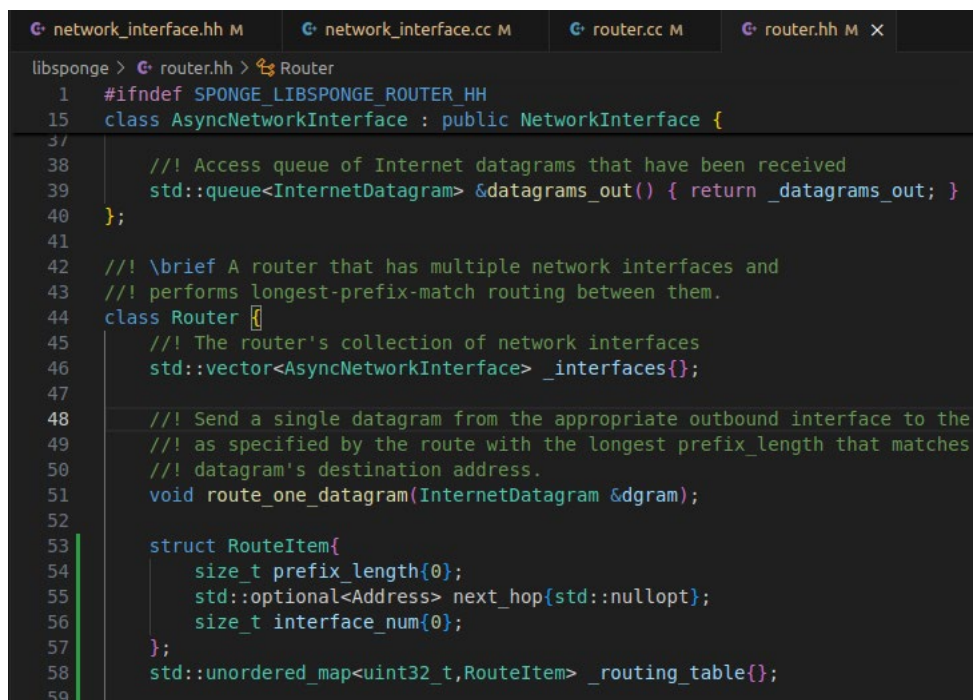
应该先检查网络问题，比如延迟过高，网络拥塞等，如果是网络问题，可以多次重试或者采用回避策略，使得重试的间隔时间更大，避免给网络造

成更大压力。

如果不是网络问题，可能是需要检查目标主机的连通性。

如果还是不行，可能可以采取 ARP 缓存管理。设备可以设置 ARP 缓存的过期时间，避免长时间未使用的过时缓存影响网络通信。如果 ARP 请求失败，主机会定期清除无效的 ARP 条目，重新尝试请求。

- 请描述一下你为了记录路由表所建立的数据结构？为什么？



```
libsponge > G router.hh > Router
1  #ifndef SPONGE_LIBSPONGE_ROUTER_HH
15 class AsyncNetworkInterface : public NetworkInterface {
37
38     //! Access queue of Internet datagrams that have been received
39     std::queue<InternetDatagram> &datagrams_out() { return _datagrams_out; }
40 };
41
42     //! \brief A router that has multiple network interfaces and
43     //! performs longest-prefix-match routing between them.
44 class Router {
45     //! The router's collection of network interfaces
46     std::vector<AsyncNetworkInterface> _interfaces{};
47
48     //! Send a single datagram from the appropriate outbound interface to the
49     //! as specified by the route with the longest prefix_length that matches
50     //! datagram's destination address.
51     void route_one_datagram(InternetDatagram &dgram);
52
53     struct RouteItem{
54         size_t prefix_length{0};
55         std::optional<Address> next_hop{std::nullopt};
56         size_t interface_num{0};
57     };
58     std::unordered_map<uint32_t,RouteItem> _routing_table{};
59 }
```

在 struct RouteItem 中,使用 prefix_length 存储前缀长度,interface_num 存储接口编号,使用 std::optional<Address> next_hop 存储目标网络,因为 next_hop 可能为空,表示该路由条目指向的目标网络直接连接到路由器。当 next_hop 存在时,路由器将数据包转发给下一跳的地址,否则转发到目的地所在的接口。

使用 std::unordered_map<uint32_t, RouteItem> _routing_table{ } 来存储路由表,因为 unorderedmap 提供了常数时间复杂度查找操作,减少了遍历整个路由表的时间,可以提高路由器的性能。

七、 实验数据记录和处理

在这次实验中,我了解了数据链路层的工作原理,内容,如何实现 TCP 协议,

如何实现地址解析 ARP 等，同时也学会了路由器的工作原理、内容。这些是非常贴近我们的生活的，我从小就知道路由器，接网线等词汇，但一直不理解具体的实现原理。这次实验也和课程的数据链路层关系很大，使我对理论课的理解更加完善。在实验过程中，也遇到了很多困难，最长的时间花费还是理解已经给出的数据结构、定义方式、何时使用等，文字的实验指导给我的帮助也非常大，照着文字内容和网上的一些解析可以完成大部分代码。