check 4

2. The Address Resolution Protocol

```
1 void NetworkInterface::send datagram(const InternetDatagram &dgram, const
2 Address &next hop);
```

向下一站发送数据，将数据报转为以太网帧并发送。

- 已知mac地址就直接发送
- 如果未知，先用ARP请求，收到响应后在发送（不可短时间内连续发送，不然会导致网络特别阻塞）

```
1 optional<InternetDatagram> NetworkInterface::recv frame(const EthernetFrame
2 &frame);
```

当以太网帧到达时调用，忽略任何不是以网络接口为目的地的帧。

- 如果入栈桢是ipv4，将负载解析为InternetDatagram，解析成功将InterDatagram返回给调用者
- 如果入栈桢是ARP，将负载解析为ARPMessage，如果成功，则记住ip地址与mac地址的映射30秒；如果是ARP请求ip，返回一个ARP相应

```
1 std::optional<EthernetFrame> maybe send();
```

网络接口发送以太网帧

```
1 void NetworkInterface::tick(const size t ms since last tick);
```

记录时间流逝，一些ip与mac映射过期

network_interface.hh

```
private:
  // Ethernet (known as hardware, network-access, or link-layer) address of the interface
  EthernetAddress ethernet_address_;

  // IP (known as Internet-layer or network-layer) address of the interface
  Address ip_address_;
  //next_ip and data
  std::unordered_map<size_t,std::vector<InternetDatagram>> ip_grams;
  //the data of ready send
  std::deque<EthernetFrame> ethernet_frame;
  //ip map mac,time
  std::unordered_map<size_t,std::pair<EthernetAddress,size_t>>  ip_map_mac;
  //arp time
  std::unordered_map<size_t,size_t> arp_time;
  //map max time 30s
  const size_t MAP_TTL;
  //arp max time 5s
  const size_t ARP_TTL;
```

network_interface.cc

```
// ethernet_address: Ethernet (what ARP calls "hardware") address of the interface
// ip_address: IP (what ARP calls "protocol") address of the interface
NetworkInterface::NetworkInterface( const EthernetAddress& ethernet_address, const Address& ip_address )
  : ethernet_address_( ethernet_address ),
    ip_address_( ip_address ),
    ip_grams(),
    ethernet_frame(),
    ip_map_mac(),
    arp_time(),
    MAP_TTL(30000),
    ARP_TTL(5000)
{
  cerr << "DEBUG: Network interface has Ethernet address " << to_string( ethernet_address_ ) << " and IP address "
       << ip_address.ip() << "\n";
}

// dgram: the IPv4 datagram to be sent
// 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 destination)

```

```cpp
// Note: the Address type can be converted to a uint32_t (raw 32-bit IP address) by using the
// Address::ipv4_numeric() method.
void NetworkInterface::send_datagram( const InternetDatagram& dgram, const Address& next_hop )
{
  if(ip_map_mac.contains(next_hop.ipv4_numeric())){
    EthernetFrame ethe_frame_;
    //find ip
    ethe_frame_.header.type=EthernetHeader::TYPE_IPv4;
    //source address &&dst address
    ethe_frame_.header.src=ethernet_address_;
    ethe_frame_.header.dst=ip_map_mac[next_hop.ipv4_numeric()].first;
    //serialize data
    ethe_frame_.payload=serialize(dgram);
    //push&&ready send
    ethernet_frame.push_back(ethe_frame_);
  }else{
    //not find the next ip,start a arp
    if(!arp_time.contains(next_hop.ipv4_numeric())){
      ARPMessage arp_msg;
      //arp request
      arp_msg.opcode=ARPMessage::OPCODE_REQUEST;
      arp_msg.sender_ethernet_address=ethernet_address_;
      arp_msg.sender_ip_address=ip_address_.ipv4_numeric();
      //target ip
      arp_msg.target_ip_address=next_hop.ipv4_numeric();
      //mac frame
      EthernetFrame ethe_frame;
      ethe_frame.header.type=EthernetHeader::TYPE_ARP;
      ethe_frame.header.src=ethernet_address_;
      ethe_frame.header.dst=ETHERNET_BROADCAST;
      //serualizer
      ethe_frame.payload=serialize(arp_msg);
      //save next_ip and data
      ip_grams[next_hop.ipv4_numeric()].push_back(dgram);
      arp_time.emplace(next_hop.ipv4_numeric(),0);
      //push
      ethernet_frame.push_back(ethe_frame);
    }
  }

  (void)dgram;
  (void)next_hop;
}

// frame: the incoming Ethernet frame
```

```cpp
optional<InternetDatagram> NetworkInterface::recv_frame( const EthernetFrame& frame )
{


 if(frame.header.dst!=ethernet_address_&&frame.header.dst!=ETHERNET_BROADCAST){
    return nullopt;
  }
  //ip data
  if (frame.header.type==EthernetHeader::TYPE_IPv4)
  {
    InternetDatagram ip_gram;
    if(parse(ip_gram,frame.payload))
      return ip_gram;
    return nullopt;
    //arp data
  }else if(frame.header.type==EthernetHeader::TYPE_ARP){
    ARPMessage arp_gram;
    if(parse(arp_gram,frame.payload)){
      ip_map_mac.insert({arp_gram.sender_ip_address,{arp_gram.sender_ethernet_address,0}});
      if(arp_gram.opcode==ARPMessage::OPCODE_REQUEST){
        if(arp_gram.target_ip_address==ip_address_.ipv4_numeric()){
          //product arp reply
          ARPMessage reply_grame;
          reply_grame.opcode=ARPMessage::OPCODE_REPLY;
          reply_grame.sender_ethernet_address=ethernet_address_;
          reply_grame.sender_ip_address=ip_address_.ipv4_numeric();

 reply_grame.target_ethernet_address=arp_gram.sender_ethernet_address;
          reply_grame.target_ip_address=arp_gram.sender_ip_address;

          //produce mac frame

          EthernetFrame reply_eth_frame;
          reply_eth_frame.header.type=EthernetHeader::TYPE_ARP;
          reply_eth_frame.header.src=reply_grame.sender_ethernet_address;
          reply_eth_frame.header.dst=reply_grame.target_ethernet_address;
          reply_eth_frame.payload=serialize(reply_grame);

          ethernet_frame.push_back(reply_eth_frame);
        }
      }else if(arp_gram.opcode==ARPMessage::OPCODE_REPLY){
        ip_map_mac.insert({arp_gram.sender_ip_address,{arp_gram.sender_ethernet_address,0}});

        auto &ip_gram=ip_grams[arp_gram.sender_ip_address];
```

```
108
109         for(auto &dgram:ip_gram){
110
    send_datagram(dgram,Address::from_ipv4_numeric(arp_gram.sender_ip_address));
111         }
112         ip_grams.erase(arp_gram.sender_ip_address);
113       }
114     }
115   }
116   (void)frame;
117   return nullopt;
118 }
119
120 // ms_since_last_tick: the number of milliseconds since the last call to this
    method
121 void NetworkInterface::tick( const size_t ms_since_last_tick )
122 {
123   for(auto it=ip_map_mac.begin();it!=ip_map_mac.end();)
124   {
125     it->second.second+=ms_since_last_tick;
126     //over time,reset
127     if(it->second.second>=MAP_TTL){
128       it=ip_map_mac.erase(it);
129     }else it++;
130   }
131
132   for(auto it=arp_time.begin();it!=arp_time.end();){
133     it->second+=ms_since_last_tick;
134     //over time,reset
135     if(it->second>=ARP_TTL){
136       it=arp_time.erase(it);
137     }else it++;
138   }
139   (void)ms_since_last_tick;
140 }
141
142 optional<EthernetFrame> NetworkInterface::maybe_send()
143 {
144   if(ethernet_frame.empty()){
145     return nullopt;
146   }
147   auto frame=ethernet_frame.front();
148
149   ethernet_frame.pop_front();
150   return frame;
151 }
152
```

```
Test project /home/sgt/cs/minnow/build
    Start  1: compile with bug-checkers
1/2 Test  #1: compile with bug-checkers ........   Passed    6.92 sec
    Start 35: net_interface
2/2 Test #35: net_interface ....................   Passed    0.03 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   6.95 sec
Built target check4
```