Computer Networks Lab5

黄嘉祺 221220108

1 Program Structure and Design

1.1 The Address Resolution Protocol

- 实验要求我们补充实现 `NetworkInterface` 类型来支持ARP协议，在网络层中发送数据帧
- 具体来说，需要在 `src/network_interface.cc` 中实现 `send_datagram` 、 `recv_frame` 、 `tick` 三个成员函数
  - `send_datagram` 函数用于将数据帧发送到网络层
  - `recv_frame` 函数用于接收数据帧
  - `tick` 函数用于处理ARP协议的超时请求
- 为了便于处理，额外在 `src/network_interface.cc` 中补充实现了 `create_frame` 和 `create_arp_request` 两个函数用于创建数据帧和ARP请求，并在 `src/network_interface.h` 中声明了这两个函数并定义了一些相关的成员变量和类型

```
EthernetFrame NetworkInterface::create_frame( const EthernetAddress& src_ethernet_address,
                                              const EthernetAddress& dest_ethernet_address,
                                              const EthernetType type,
                                              std::vector<std::string> data )
{
  EthernetFrame frame;
  frame.header.src = src_ethernet_address;
  frame.header.dst = dest_ethernet_address;
  frame.header.type = type;
  frame.payload = std::move( data );
  return frame;
}

ARPMessage NetworkInterface::create_arp_request( const uint16_t opcode,
                                                 const EthernetAddress sender_ethernet_address,
                                                 const uint32_t sender_ip_address,
                                                 const EthernetAddress target_ethernet_address,
                                                 const uint32_t target_ip_address )
{
  ARPMessage arp;
  arp.opcode = opcode;
  arp.sender_ethernet_address = sender_ethernet_address;
  arp.sender_ip_address = sender_ip_address;
  arp.target_ethernet_address = target_ethernet_address;
  arp.target_ip_address = target_ip_address;
  return arp;
}
```

- `send_datagram` 的实现思路是：
  - 首先，从 next_hop 地址对象中提取出IPv4地址的数值表示。然后，在ARP表中查找该IP地址对应的以太网地址。

- 如果在ARP表中找到了对应的以太网地址，则创建一个以太网帧并发送。这个帧包含了源以太网地址、目的以太网地址、以太网类型（IPv4）以及序列化后的数据报。
- 如果在ARP表中没有找到对应的以太网地址，则将数据报加入等待队列。如果该IP地址的等待队列不存在，则创建一个新的队列。接着，检查是否已经发送过ARP请求。如果已经发送过，则直接返回。
- 如果没有发送过ARP请求，则创建一个ARP请求并发送。这个请求包含了源以太网地址、源IP地址、目标以太网地址（空）和目标IP地址。发送ARP请求后，将该IP地址记录在已发送ARP请求的集合中，并设置请求的过期时间。

```
void NetworkInterface::send_datagram( const InternetDatagram& dgram, const Address& next_hop )
{
  // Your code here.
  // (void)dgram;
  // (void)next_hop;
  auto next_hop_ip_address = next_hop.ipv4_numeric();
  auto it = arp_table_.find( next_hop_ip_address );
  if ( it != arp_table_.end() ) {
    transmit( create_frame(
      ethernet_address_, it->second.ethernet_address, EthernetHeader::TYPE_IPv4, serialize( dgram ) ) );
  } else {
    auto wait_it = waiting_datagrams_.find( next_hop_ip_address );
    if ( wait_it == waiting_datagrams_.end() ) {
      waiting_datagrams_[next_hop_ip_address] = queue<InternetDatagram>();
    }
    waiting_datagrams_[next_hop_ip_address].push( dgram );
    if ( arp_requests_sent_.contains( next_hop_ip_address ) ) {
      return;
    }
    auto arp = create_arp_request(
      ARPMessage::OPCODE_REQUEST, ethernet_address_, ip_address_.ipv4_numeric(), {}, next_hop_ip_address );
    transmit( create_frame( ethernet_address_, ETHERNET_BROADCAST, EthernetHeader::TYPE_ARP, serialize( arp ) ) );
    arp_requests_sent_.insert( next_hop_ip_address );
    arp_request_expire_time_[next_hop_ip_address] = timer_ + 5000;
  }
}
```

- `recv_frame` 的实现思路是：
  - 首先，检查帧的目的地址是否是本机的以太网地址或广播地址。如果都不是，则直接返回，不处理该帧。
  - 接着，根据帧的类型进行不同的处理。如果帧的类型是IPv4，则尝试解析帧的有效载荷为一个互联网数据报。如果解析成功，则将数据报加入接收队列。
  - 如果帧的类型是ARP，则尝试解析帧的有效载荷为一个ARP消息。如果解析失败，则直接返回。解析成功后，将发送方的IP地址和以太网地址添加到ARP表中，并设置该条目的过期时间。
  - 如果ARP消息是一个ARP请求且目标IP地址是本机的IP地址，则创建一个ARP回复消息并发送给请求方。
  - 最后，检查是否有等待发送到发送方IP地址的数据报。如果有，则将这些数据报逐个发送出去，并从等待队列中移除该IP地址的条目。

```
void NetworkInterface::recv_frame( const EthernetFrame& frame )
{
  // Your code here.
  // (void)frame;
  if ( frame.header.dst != ethernet_address_ && frame.header.dst != ETHERNET_BROADCAST ) {
    return;
  }
  if ( frame.header.type == EthernetHeader::TYPE_IPv4 ) {
    InternetDatagram dgram;
    auto res = parse( dgram, frame.payload );
    if ( res ) {
      datagrams_received_.push( dgram );
    }
```

```
  } else if ( frame.header.type == EthernetHeader::TYPE_ARP ) {
    ARPMessage arpmsg;
    auto res = parse( arpmsg, frame.payload );
    if ( !res ) {
      return;
    }
    arp_table_[arpmsg.sender_ip_address] = { arpmsg.sender_ethernet_address, timer_ + 30000 };
    if ( arpmsg.opcode == ARPMessage::OPCODE_REQUEST && arpmsg.target_ip_address == ip_address_.ipv4_numeric() ) {
      auto arp = create_arp_request( ARPMessage::OPCODE_REPLY,
                                     ethernet_address_,
                                     ip_address_.ipv4_numeric(),
                                     arpmsg.sender_ethernet_address,
                                     arpmsg.sender_ip_address );
      transmit( create_frame(
        ethernet_address_, arpmsg.sender_ethernet_address, EthernetHeader::TYPE_ARP, serialize( arp ) ) );
    }
    if ( waiting_datagrams_.contains( arpmsg.sender_ip_address ) ) {
      auto& q = waiting_datagrams_[arpmsg.sender_ip_address];
      while ( !q.empty() ) {
        auto datagram = q.front();
        transmit( create_frame(
          ethernet_address_, arpmsg.sender_ethernet_address, EthernetHeader::TYPE_IPv4, serialize( datagram ) ) );
        q.pop();
      }
      waiting_datagrams_.erase( arpmsg.sender_ip_address );
    }
  }
}
```

- `tick` 的实现思路是：
  - 首先，增加内部计时器的值，增加的量是自上次调用以来经过的毫秒数。
  - 接着，遍历ARP表中的所有条目，检查每个条目的过期时间。如果某个条目的过期时间小于或等于当前时间，则将其从ARP表中删除。否则，继续检查下一个条目。
  - 然后，遍历ARP请求的过期时间表，检查每个请求的过期时间。如果某个请求的过期时间小于或等于当前时间，则将其从已发送ARP请求的集合中删除，并从过期时间表中删除该条目。否则，继续检查下一个条目。

```
void NetworkInterface::tick( const size_t ms_since_last_tick )
{
  // Your code here.
  // (void)ms_since_last_tick;
  timer_ += ms_since_last_tick;
  for ( auto it = arp_table_.begin(); it != arp_table_.end(); ) {
    if ( it->second.expire_time <= timer_ ) {
      it = arp_table_.erase( it );
    } else {
      ++it;
    }
  }
  for ( auto it = arp_request_expire_time_.begin(); it != arp_request_expire_time_.end(); ) {
    if ( it->second <= timer_ ) {
      arp_requests_sent_.erase( it->first );
      it = arp_request_expire_time_.erase( it );
    } else {
      ++it;
    }
  }
}
```

2 Implementation Challanges

- 在实现 `send_datagram` 函数的时候，需要设定发送的arp请求的超时时间，在设定超时时间的时候记错了，导致在测试时在不该发送arp请求的时候发送了arp请求，后来发现问题并修改了代码，解决了问题

3 Remaining Bugs

目前我的代码已经通过了全部的测试，暂时未发现明显的bug

4 Experimental Results and Performance

- net_interface: Passed all 2 tests