

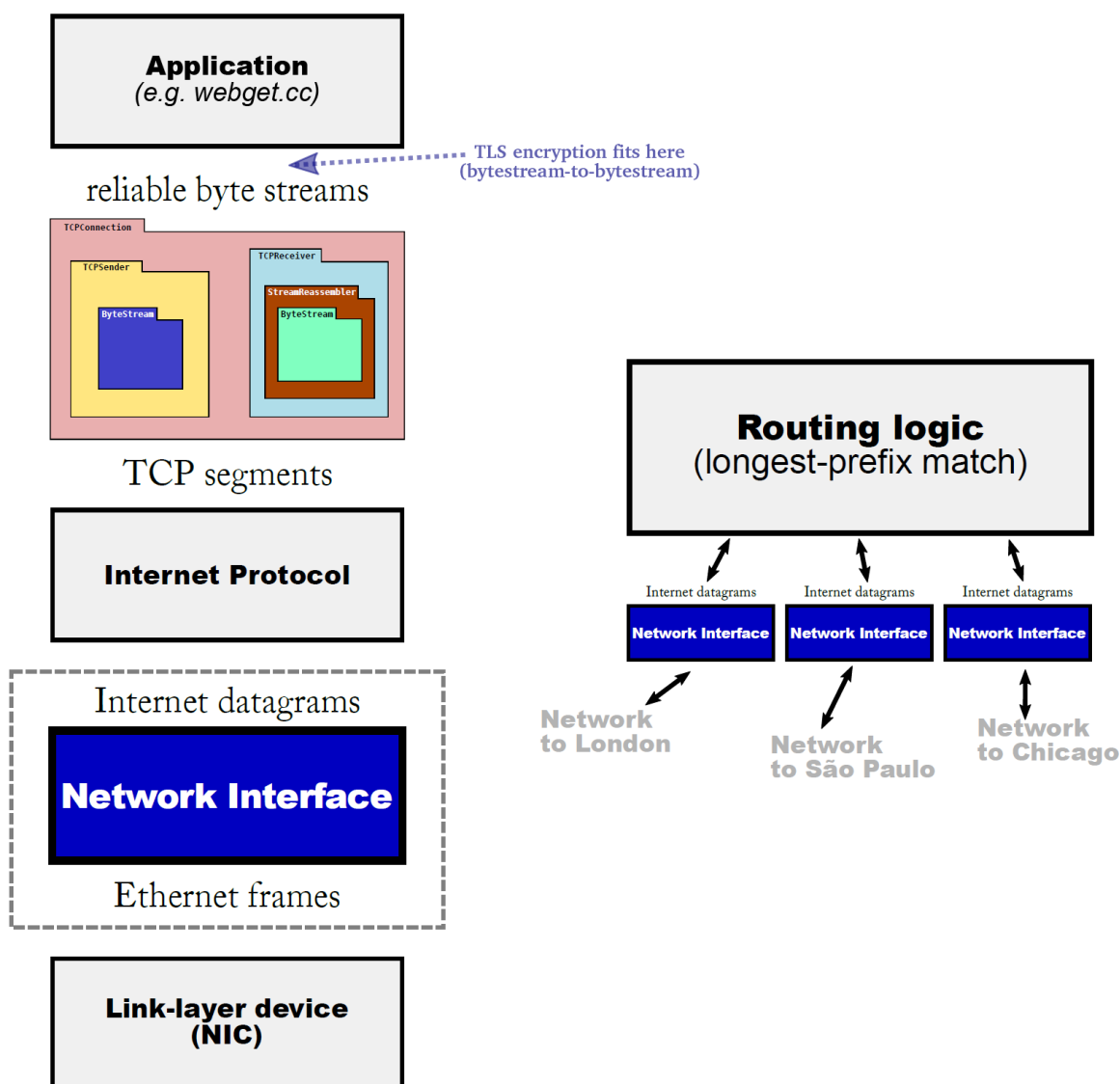
Lab 6 Writeup

My name: 李易

My Student ID: 191830079

This lab took me about 14 hours to do. I did attend the lab session.

Program Structure and Design of the NetworkInterface:



这次我们完成的是一个简单的网络层。主要实现部分涉及ARP协议，我们需要利用ARP协议找到下一条IP对应的以太网地址。整体难度大概和lab2 类似吧（设计Stream Reassembler和Bytestream）。不过这次lab开始前需要阅读一些相关的背景知识和接口，了解ARP协议内容和lab里提供的EthernetFrame, Internet Datagram等结构接口。

相关数据结构：

我们需要维护两个结构，一个是用于联系IP地址和MAC地址的cache。我们在收到一个链路层frame时，如果是ARP消息，就可以将其IP和MAC地址缓存下来。为了检测其是否超时，cache里还应该包含已经建立缓存的时间time_caching。

另一个map结构是IP地址到其对应等待队列的mapping。对于一个IP对应的等待队列(Queuing_List)，我们需要做的有：

在需要发送datagram却不知道目的MAC地址时在map结构中建立对应mapping并将datagram加入队列中

超过5s则发送一个ARP请求

收到ARP消息时如果我们得到了对应的目的MAC地址，就将该IP地址对应mapping清除，将等待队列中datagram全部发出。

设计完数据结构之后还需要了解一下对应Frame和datagram的接口，可以写出两个函数主要的流程，对应注释的地方就是需要实现的细节部分：

```
void NetworkInterface::send_datagram(const InternetDatagram &dgram, const Address
&next_hop) {
    // 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();
    optional<EthernetAddress> next_hop_MAC = get_EthernetAddr(next_hop_ip);
    if(next_hop_MAC.has_value())
    {
        //send datagrams
    }
    else
    {
        //queue datagrams
    }
}

optional<InternetDatagram> NetworkInterface::recv_frame(const EthernetFrame
&frame) {
    optional<InternetDatagram> res = nullopt;
    //check: ignore any frames not destined for the network interface
    if(frame.header().type == TYPE_ARP)
    {
        ARPMessage arp;
        // create a mapping for source ip & MAC
        // now clear correspondent queuing list with known dst MAC addr
        if(arp.opcode == ARPMessage::OPCODE_REQUEST && arp.target_ip_address ==
_ip_address.ipv4_numeric())
        {
            //send an arp reply
        }
    }
    else if(frame.header().type == TYPE_IPv4)
    {
        InternetDatagram datagram;
        res = datagram;
    }
}
```

Implementation Challenges:

本次试验难度适中，没有特别难发现的bug。然后具体实现细节的过程比较类似于完成业务代码，不像lab5那种设计fsm非常烧脑的感觉。主要的几个bug：

1. 利用迭代器的过程。map主要使用到的也就erase, find就行。但在清除cache中超时的(>30s)的IP-MAC缓存时, 需要注意如果用for循环的话会出现bug。原因是用 `_cache.erase(iter)` 后iter实际上已经不可用, 此时再 `iter++` 就会出错, 正确的方式:

```
while(iter_cache != _cache.end())
{
    iter_cache->second.time_caching += ms_since_last_tick;
    if(iter_cache->second.time_caching >= 30000)
        _cache.erase(iter_cache++);
    else
        iter_cache++;
}
```

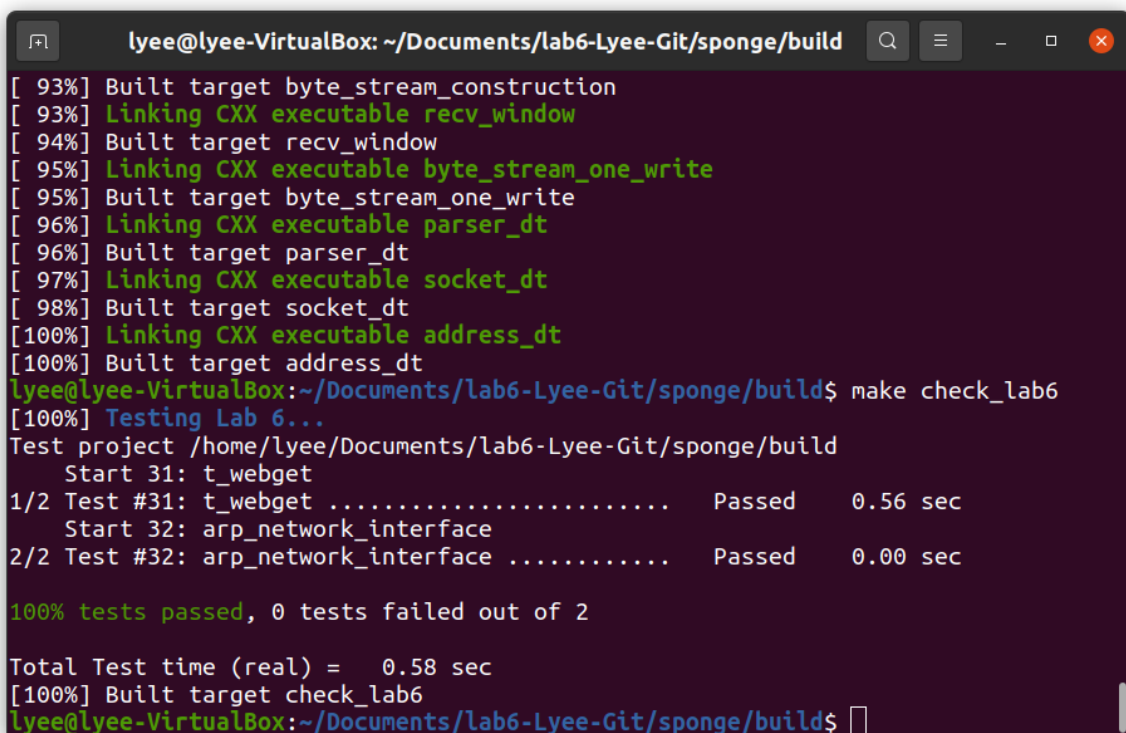
在iter被erase前将其+1指向下一元素, 可以避免出错。

2. 还有一些容易出错的地方是与ARP细节相关的。比如讲义特别提到:

Except: You don't want to flood the network with ARP requests. If the network interface already sent an ARP request about the same IP address in the last five seconds, don't send a second request—just wait for a reply to the first one. Again, queue the datagram until you learn the destination Ethernet address.

超过5s发送一个ARP request, 但这只是在send_datagram()被调用的情况下。其他时候不需要我们发送ARP request (即使已经超过5秒), 我之前在tick中也遍历检查每个IP-queueing_list mapping, 超过5s就发送request, 就出现了错误。

Remaining Bugs:



```
lyee@lyee-VirtualBox: ~/Documents/lab6-Lyee-Git/sponge/build
[ 93%] Built target byte_stream_construction
[ 93%] Linking CXX executable recv_window
[ 94%] Built target recv_window
[ 95%] Linking CXX executable byte_stream_one_write
[ 95%] Built target byte_stream_one_write
[ 96%] Linking CXX executable parser_dt
[ 96%] Built target parser_dt
[ 97%] Linking CXX executable socket_dt
[ 98%] Built target socket_dt
[100%] Linking CXX executable address_dt
[100%] Built target address_dt
lyee@lyee-VirtualBox:~/Documents/lab6-Lyee-Git/sponge/build$ make check_lab6
[100%] Testing Lab 6...
Test project /home/lyee/Documents/lab6-Lyee-Git/sponge/build
  Start 31: t_webget
1/2 Test #31: t_webget ..... Passed    0.56 sec
  Start 32: arp_network_interface
2/2 Test #32: arp_network_interface ..... Passed    0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =  0.58 sec
[100%] Built target check_lab6
lyee@lyee-VirtualBox:~/Documents/lab6-Lyee-Git/sponge/build$
```

