南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：田臣/李文中　　助教：

学院	**计算机科学与技术**	专业（方向）	**计算机科学与技术**
学号	**201220062**	姓名	**黄子睿**
Email	**201220062@smail.nju.edu.cn**	开始/完成日期	**3/26-3/28**

1. 实验名称：

Lab4 Respond to ARP

2. 实验目的：

利用 lab3 完成的 ARP REQUEST 处理与 IP-ETHERNT 地址映射表，进一步完善路由器的行为。包括处理并正确转发 IP 包以及利用 ARP REQUEST 查询未知 IP 对应的 ETHERNET ADDRESS。

3. 实验内容：

Task2 IP Forwarding Table LookUp

在 lab4 中，将利用 python 自带的 IPv4 库处理最长前缀匹配问题，以处理 IP 包应该向哪个接口转发的问题。

首先考察转发表中包含的元素。一个转发表表项应当包含四个信息，分别是前缀，子网掩码，下一跳地址以及对应的接口。前缀用来匹配接受到的 IP 包，子网掩码则用来处理最长前缀匹配，下一跳地址用于复杂拓扑结构的网络，接口信息就是发包的出口。

将这些内容与对应的功能抽象成一个类：*ForwardingTable*。其中的数据成员

就是一个字典，用来存储前缀匹配信息。

```
def __init__(self):
        self.dict = {}
```

转发表的构建利用成员函数 *put_entry* 实现。

```
def put_entry(self, ipaddr, netmask, nexthop, intf):
    ipaddr_num = int(IPv4Address(ipaddr))
    netmask_num = int(IPv4Address(netmask))
    prefix = ipaddr_num & netmask_num
    self.dict[IPv4Address(ipaddr)] = {'prefix':prefix, 'netmask_num':
netmask_num, 'nexthop':nexthop, 'intf':intf}
```

传入的参数 ipaddr, netmask, nexthop 都是字符串。在 ForwordingTable 类中，首先字符串 ip 地址转化为 IPv4Address 型（避免后续因为在类型不一致上面出的错），再以其为键值。再将前缀与子网掩码分别以数字 int 型，下一跳地址为字符串存入字典中，方便后续操作，intf 就是 switchyard 中的接口类。

在匹配对应的转发接口时，需要用到 *prefix_match* 函数。其原型实现为：

```
def prefix_match(self, destaddr):
        destaddr = IPv4Address(destaddr)
        destaddr_num = int(destaddr)
        # Considering the longest prefix matching rule
        matched_key = None
        for key in self.dict:
            matched = destaddr_num & self.dict[key]['netmask_num'] ==
self.dict[key]['prefix']
            if matched: # prefix matched
                if destaddr == key: # destiantion is this router, drop
 the packet at this stage
                    return None,None
                else:   # longest prefix matching rule
                    if matched_key == None or self.dict[key]['netmask_
num'] > self.dict[matched_key]['netmask_num']:
                        matched_key = key
        if matched_key != None:
            return self.dict[matched_key]['intf'], self.dict[matched_k
ey]['nexthop']
        else:
            return None,None # None means the packet is destinied at t
his router or somewhere disconnected from this router
```

对于传入的参数 destaddr，首先将其转化为 IPV4Address 类型，再记录 int 型的 destaddr_num 用来前缀匹配。每一次匹配到前缀之后，需要判断该地址是否为路由器的一个端口，如果是则不需要后续的转发；同时根据最长前缀匹配原则，每次比较子网掩码的大小（也就是长度），选取其中最长的前缀返回。如果没有匹配到，则返回 None。(根据实验的规则，返回 None 的情况 IP 包需要被丢弃。) 该函数的返回值有两个，分别是接口与下一条地址。

Task3 Forwarding the Packet and ARP

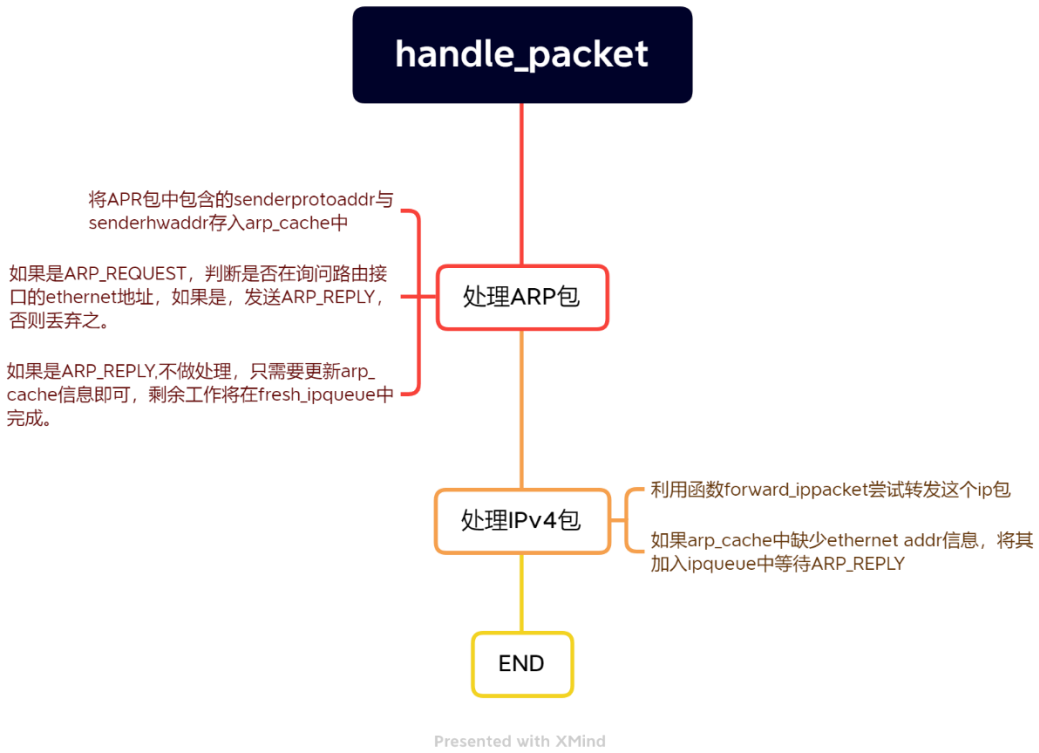
下面阐述转发 IPv4 包与 ARP 包的逻辑。

```
1.  def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
2.          timestamp, ifaceName, packet = recv
3.          log_info(f'pkt is {str(packet)}')
4.          arp = packet.get_header(Arp, None)
5.          if arp is not None:   # Handle ARP packets
6.              log_info(f"This is an ARP packet, its arp header is {arp}
    ")
7.              # self-learning
8.              # Use arpCache as a static table, so no need to refresh.
9.              # put the sender ipaddr--ethaddr into cache
10.             self.arpCache.put(arp.senderprotoaddr, arp.senderhwaddr)
11.             # Forward ARP packet
12.             if arp.operation == ArpOperation.Request:
13.                 targetIntf = self.ipaddr_in_router(arp.targetprotoadd
    r)
14.                 if targetIntf is not None:
15.                     pkt = create_ip_arp_reply(targetIntf.ethaddr, arp
    .senderhwaddr, targetIntf.ipaddr, arp.senderprotoaddr)
16.                     self.net.send_packet(self.name_in_router(ifaceNam
    e), pkt)
17.             elif arp.operation == ArpOperation.Reply:
18.                 pass
19.         else:   # Forward packets by forwarding table
20.             cnt = self.forward_ippacket(packet, 0, 0)
21.             if cnt != -1:
```

```
22.                 self.ipqueue.append({'packet':packet, 'time':time.tim
    e(), 'cnt':cnt})
```

上面是这个阶段之后的 *handle_packet* 函数原型。具体来讲首先添加了一个列表 *self.ipqueue*，用来存储等待 APR_REPLY 的所有 IP 包。具体内容有 *'packet'*，*'time'*与 *'cnt'*，分别记录 packet 本身的数据信息，最近一次发送包的时间戳，以及总共发送过多少次 ARP REQUEST。

Handle_packet 函数处理新包的转发与应答工作。函数的具体执行思路如下图：

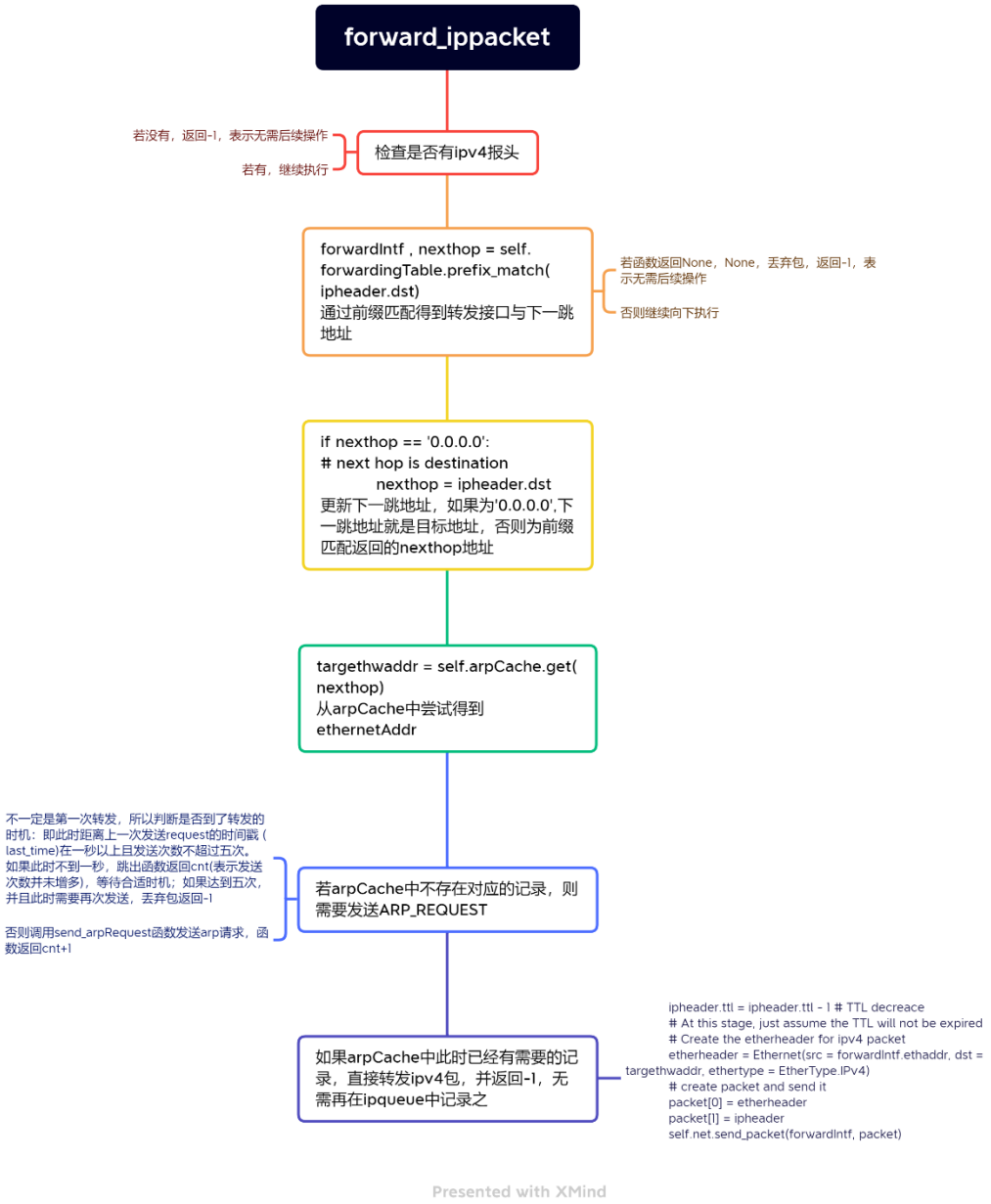


下面解释 *fresh_ipqueue* 与 *forward_ippacket* 这两个函数的实现。

```
1.
2.     def forward_ippacket(self, packet, cnt, last_time = 0):
3.         ipheader = packet.get_header(IPv4, None)
4.         if ipheader is None:
5.             return  -1 # Packet has no ip header, drop it.
6.         # prefix match to get the forwarding interface
7.         forwardIntf , nexthop = self.forwardingTable.prefix_match(iph
    eader.dst)
```

```
        if forwardIntf is None:
            log_info(f"The ipaddr {ipheader.dst} is not connected to the router or just in the router.")
            return -1
        # First check whether the targetprotoaddr is in the ARPCache
        if nexthop == '0.0.0.0': # next hop is destination
            nexthop = ipheader.dst
        targethwaddr = self.arpCache.get(nexthop)
        if targethwaddr == None:
            if time.time()-last_time < 1:
                return cnt
            elif cnt >= 5:
                return -1 # drop packet
            else: # The hwaddr not recorded in arpCache need an arp request
                # send arp request
                flag = False
                if nexthop == '0.0.0.0':    # Assume the next hop is the destination
                    flag = self.send_arpRequest(ipheader.dst)
                else:
                    flag = self.send_arpRequest(nexthop)
                if flag: # If the arp request is sended successfully
                    # put entry into ipqueue and wait for arp reply
                    # At this stage the packet has no data
                    return cnt+1
                else:
                    return -1
        else: # The targethwaddr has already been in the arp cache
            ipheader.ttl = ipheader.ttl - 1 # TTL decreace
            # At this stage, just assume the TTL will not be expired
            # Create the etherheader for ipv4 packet
            etherheader = Ethernet(src = forwardIntf.ethaddr, dst = targethwaddr, ethertype = EtherType.IPv4)
            # create packet and send it
            packet[0] = etherheader
            packet[1] = ipheader
            self.net.send_packet(forwardIntf, packet)
            return -1
```

forward_ippacket 的具体思路是：




```
1.  def send_arpRequest(self, targetprotoaddr):
2.          forwardIntf , nexthop = self.forwardingTable.prefix_match(tar
    getprotoaddr)
3.          if forwardIntf is not None:
4.              senderhwaddr = forwardIntf.ethaddr
5.              senderprotoaddr = forwardIntf.ipaddr
6.              pkt = create_ip_arp_request(senderhwaddr, senderprotoaddr
    , targetprotoaddr)
7.              self.net.send_packet(forwardIntf, pkt)
8.              log_info(f"Sending arp request succeeded from {senderprot
    oaddr} to {targetprotoaddr}")
9.              return True
10.         else:
```

```
            log_info(f"Sending arp request failed with targetprotoaddr is {targetprotoaddr}")
            return False
```

send_packet 函数的逻辑比较简单，从前缀匹配得到转发接口与下一跳地址（若为全零，则就是目标地址）之后构造 ARP REQUEST 并发送之即可。

为了不断更新 *ipqueue* 的情况，需要改写 *start* 函数中的循环体：

```
while True:
            try:
                recv = self.net.recv_packet(timeout=1.0)
            except NoPackets:
                continue
            except Shutdown:
                break
            else:
                self.handle_packet(recv)
            finally:
                self.fresh_ipqueue()
```

try-except-else-finally 保证了每次循环，不论是否有包传入路由器，路由器都会去访问并尝试更新 ipqueue 的内容。

fresh_ipqueue 的原型实现为：

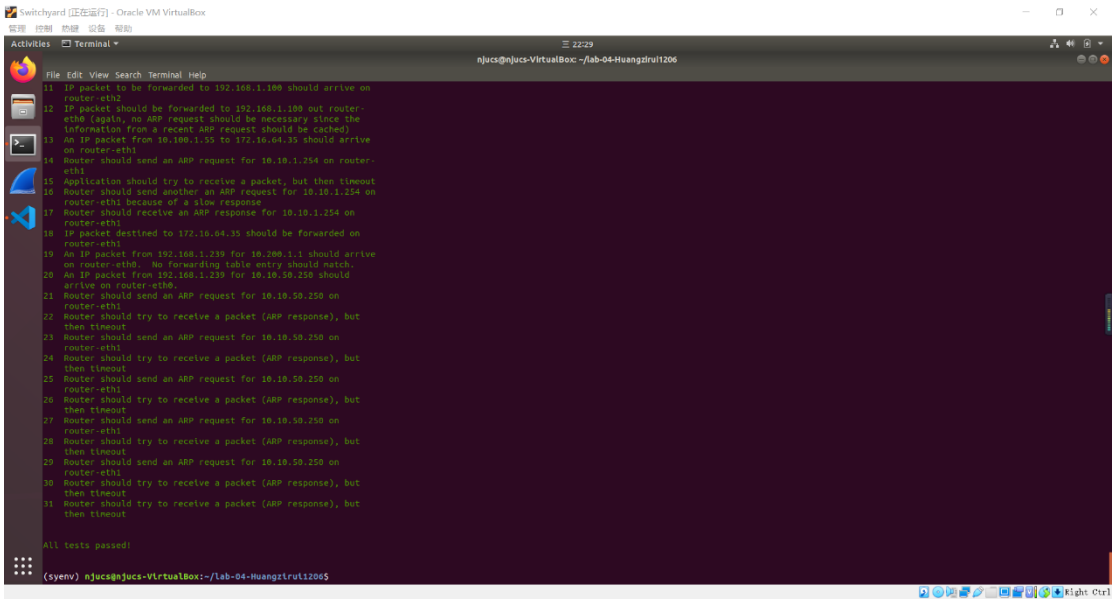
```
def fresh_ipqueue(self):
        for item in self.ipqueue:
            cnt = self.forward_ippacket(item['packet'],item['cnt'],item['time'])
            if item['cnt'] != cnt:
                item['time'] = time.time()
                item['cnt'] = cnt
        self.ipqueue = list(filter(lambda item:item['cnt'] != -1, self.ipqueue))
```

具体思路为：对 ipqueue 中等待 ARP 信息的 ip_packet，尝试对其进行 *forward_ippacket* 操作。如果返回 cnt+1，则意味着又发送了一次 ARP 请求，更新时间戳与次数；每次操作结束删去 cnt 为-1 的 ip_packet，它们不再需要后续处理。

需要注意的是，在类的__init__函数中需要初始化转发表与 arpCache:

```
for intf in self.net.interfaces():
            # Record own ip/ethernet addrs in arp table
            self.arpCache.put(intf.ipaddr, intf.ethaddr)
            # Add the interface information into forwarding table
            self.forwardingTable.put_entry(intf.ipaddr, intf.netmask, '0.0.0.0', intf)
        # read information from forwarding_table.txt
        for line in open("forwarding_table.txt","r"):
            str_list = line.split()
            self.forwardingTable.put_entry(str_list[0],str_list[1],str_list[2],self.name_in_router(str_list[3]))
```

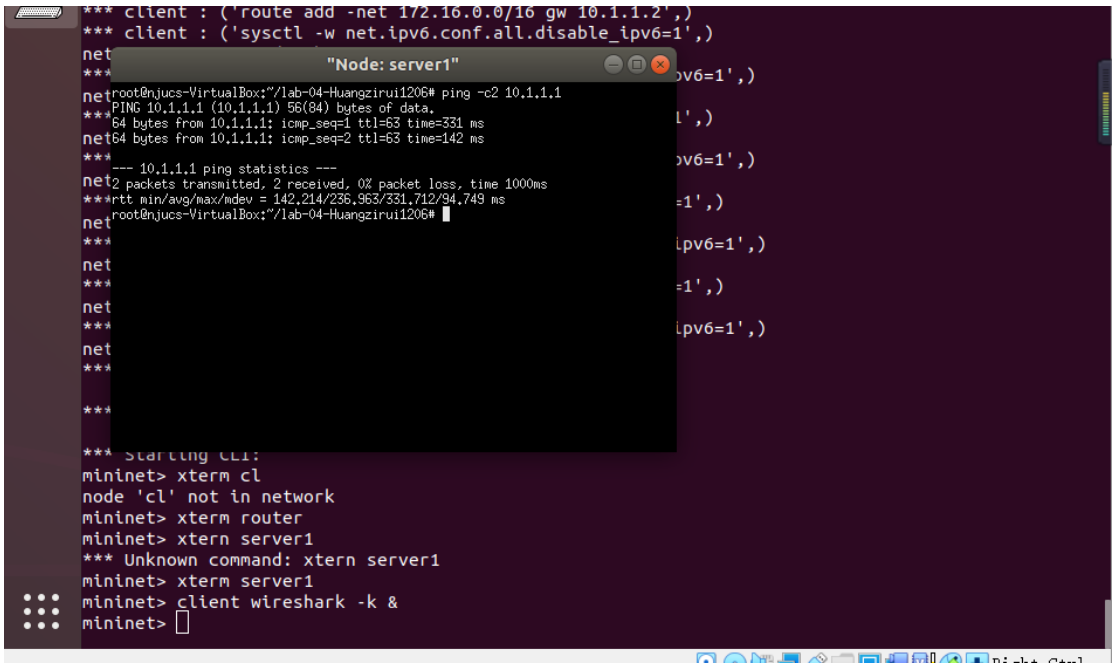
下面是通过所有 testcase 的截图。



Deploying

我在这里展示的试验过程是用 server1 ping client，观察 server1 与 client 的 wireshark。

Client 的接口 ip 地址是 10.1.1.1，在 server1 上尝试 ping client，下面是操作截图：



Client 的 wireshark 数据显示为：