LAB4实验报告

专业	学号	姓名	开始/结束日期	Email
计算机科学与技术系	191220156	张桓	5.1~5.3	2659428453@qq.com

实验名称

Forwarding Packets

实验目的

本实验作为实验 `IPV4` 路由器的第二个阶段，在 `Lab3` 的基础上改进路由器，实现数据包的转发功能，主要实现下面两个方面：

- 接收和转发到达链路并发往其他主机的数据包。 转发过程的一部分是在转发表中执行“**最长前缀匹配**”查找。我们将仅在路由器中使用“静态”路由，而不是实现像 `RIP` 或 `OSPF` 这样的动态路由协议。
- 对不知道 `MAC` 地址的目的 `IP` 地址**发出 `ARP` 请求**。路由器通常需要向其他主机发送数据包，并需 `MAC` 地址来完成这一工作。

实验内容

Task 2: IP Forwarding Table Lookup

建立转发表

> 路由器的一项基本功能就是**建立转发表**，当接收到数据包后将包的目的 `IP` 地址与转发表**匹配**，然后将数据包从正确的接口转发出去。通过 `lab_Manual` 我们知道，**转发表的表项为：<网络地址，子网掩码，下一跳 `IP` 地址，出端口>**，如下：

network address	subnet address	next hop address	interface
192.168.1.0	255.255.255.0	0.0.0.0	router-eth0
172.16.1.0	255.255.255.0	0.0.0.0	router-eth1

实现逻辑

- 表项的来源有两个：路由器的自身端口和 `forwarding_table.txt` 文件
  - 显然可以简单按照来源分成两次 `for` 循环，一次遍历路由器所有端口，可通过 `net.interfaces()` 以及 `interfaces` 类函数来获取需要的端口信息，比如端口的 `IP` 地址，端口的子网掩码来建立表项。如果下一跳就是目标 `IP` 时，即上图的 `0.0.0.0`，我用 `None` 表示。
  - 另一次 `for` 循环则通过读取 `forwarding_table.txt` 文件，使用 `split()` 方法将字符串按照空格和回车分隔开，建立表项**附加到转发表**中即可。
- 转发表最重要的功能是 **`IP` 和子网表项的匹配**，按照 `manual` 的提示，可以通过 `IPv4Network` 类提供的 `in` 方法来简单判断。因此我在建立转发表时直接就先将原始的 `IP` 地址(`IPv4Adderss` 类型)配合子网掩码转换成了子网地址(`IPv4Network` 类型)，这样就**方便了以后的判断。**
- 如何**实现最长前缀匹配**？如果一个目的 `IP` 地址可以和转发表中多个表项匹配时，选择**匹配前缀最长**的那一项。因此我们只需要**简单排序**，将前缀长的表项放在前面，这样在从前往后遍历转发表时，**最先匹配到的表项总是前缀最长的**，这样就实现了。

> 使用 `sort()` 方法，参数 `key = lambda x:x[1]` 表示按照第1项即子网掩码来排序，参数 `reverse = True` 表示逆序排序。显然**子网掩码更大的表示前缀更长**，被排在了更前面。

核心代码

```
self.fw_table = [] #constract the forward table
    for intf in self.net.interfaces():  #from the ports in router
        netaddr = IPv4Network(str(IPv4Address(int(intf.ipaddr) &
int(intf.netmask))) + '/' + str(intf.netmask)) # get the netaddr
        self.fw_table.append([netaddr, intf.netmask, None, intf.name])
    for line in open("forwarding_table.txt"): #from the txt file
        item = line.split()
        item[0] = IPv4Network(item[0] + '/' + item[1])
        item[1] = IPv4Address(item[1])
        item[2] = IPv4Address(item[2])
        self.fw_table.append(item)
     self.fw_table.sort(key = lambda x:x[1], reverse = True) # the longest prefix
in head
```

实验结果

为了查看建立的转发表是否正确，在初始化转发表之后将其打印，输出如下：

```
[IPv4Network('172.16.42.0/30'), IPv4Address('255.255.255.252'), None, 'router-eth2']
[IPv4Network('192.168.1.0/24'), IPv4Address('255.255.255.0'), None, 'router-eth0']
[IPv4Network('172.16.128.0/18'), IPv4Address('255.255.192.0'), IPv4Address('10.10.0.254'), 'router-eth1']
[IPv4Network('172.16.64.0/18'), IPv4Address('255.255.192.0'), IPv4Address('10.10.1.254'), 'router-eth1']
[IPv4Network('10.10.0.0/16'), IPv4Address('255.255.0.0'), None, 'router-eth1']
[IPv4Network('172.16.0.0/16'), IPv4Address('255.255.0.0'), IPv4Address('192.168.1.2'), 'router-eth0']
[IPv4Network('10.100.0.0/16'), IPv4Address('255.255.0.0'), IPv4Address('172.16.42.2'), 'router-eth2']
```

可以看到子网地址成功变成`IPv4Network`类型，更重要的是表项的前缀长度成功按照从大到小排序，符合预期。

匹配目的IP地址

实现逻辑

本实验对于接收到的`IP`数据包，查看其目的`IP`地址，不能为路由器某个端口地址，因此必须满足`ipv4.dst not in self.myIPs`。

> 其中`self.myIPs = [intf.ipaddr for intf in self.net.interfaces()]`，表示路由器所有端口`IP`地址的列表。

并且必须在转发表中可以找到表项使目的`IP`属于对应的子网，否则**直接丢弃**。

> 判断目的`IP`地址是否属于某个表项对应的子网时，利用`IPv4Network`类的属性直接查看，即`ipv4.dst in item[0]`。其中`item[0]`是预先处理好的`IPv4Network`类型，**详见上文**。

核心代码

```
1  elif(packet.has_header(IPv4)): #a IP packet
2      ipv4 = packet.get_header(IPv4)
3      ipv4.ttl -= 1
4          if ipv4.dst not in self.myIPs: # the dest is not in router
5              for item in self.fw_table:
6                  if ipv4.dst in item[0]: # the dst addr is in item's netaddr
7                  ...
8                  break #NOTE!!!
```

- 上述代码第7行的省略号是转发的具体工作，将在下面的 `Task3` 中展开
- 代码第8行 `break` 是实现**最长前缀匹配的关键**，因为从前向后遍历转发表，最先匹配到的一定是前缀最长的，匹配到之后应该立即跳出循环，否则有可能继续匹配到后续前缀较短的表项产生错误。

Task 3: Forwarding the Packet and ARP

实现等待回应表及其更新

> 路由器为了得知下一跳 `IP` 的 `MAC` 地址时需要发出 `ARP` 请求包，我们需要实现如果一个请求超过 `1s` 没有被回应则再次发送请求，**最多5次**后还没有回应的话则**丢弃**想要发到此地的所有包。

实现逻辑

- 这里用**字典**来实现等待回应表 `wait_Queue` ，字典的表项为 `<nextHop : [time, send_counts, out_intf, arp_request, packet_Queue]>` ，每次对一个**新的** `nextHop` 发出 `ARP` 请求，就需要在 `wait_Queue` 中建立对应的**新的表项。**
  - `nextHop` 是需要请求 `MAC` 的目的 `IP` 地址，是从接收到的 `IP` 包头中得到的。
  - `time` 为发送请求的最新时间，通过 `time.time()` 获取， `send_counts` 为对某个 `nextHop` 发送请求的次数，**第一次发送**对某个 `nextHop` 的请求时 `send_counts` 设置为1。
  - `out_intf` 和 `arp_request` 是 `ARP` 请求包发出的端口以及 `ARP` 请求包本身，之所以放到表项中是为了方便检测到表项**超时后再次发出请求。**
  - `packet_Queue` 为数据包**队列。**当多个数据包发往同一个 `nextHop` 时，需要维护一个队列来保持这些**数据包的次序关系。**当该 `nextHop` 被回应后，按照 `FIFO` **的次序**依次将这些包发出。
- 实现超时重发
  - 遍历等待回应表中的表项，用当前时间 `time.time()` 减去 `nextHop` 的表项的 `time` ，大于1则重发
  - 每次重新发送某个 `ARP` 请求后，对应 `nextHop` 的表项中的 `send_counts` 加1，若已经为5则 `pop` 掉该 `nextHop` 表项

核心代码

```
self.wait_Queue = {}
#the item is <nextHop : [time, send_counts, out_inft, arp_request, [packet]] >
...
for nextHop in list(self.wait_Queue):
    if(time.time() - self.wait_Queue[nextHop][0] > 1):
        print(f"repeat a arp_request for nextHop's IP:{nextHop} because of timeout")
        send_counts = self.wait_Queue[nextHop][1]
        if(send_counts <= 4): #repeat again
            self.net.send_packet(self.wait_Queue[nextHop][2], self.wait_Queue[nextHop][3])
            self.wait_Queue[nextHop][0] = time.time()
            self.wait_Queue[nextHop][1] = send_counts + 1
        else: #send_counts more 5
            self.wait_Queue.pop(nextHop)
```

- 上述代码实现了等待回应表的更新，为了实现**实时更新**，这部分代码放在`def start(self)`的`while(true)`中，具体位置如下，放到了`try`之前，因为如果放在`try`之后只能在**收到包之后才能更新**，很可能等待太长时间导致没有及时处理过时表项。因此**放到`try`之前**保证实时更新，即下面代码省略号的位置。

```
def start(self):
        while True:
            #TODO: update the wait_Queue
            ...
            try:
                recv = self.net.recv_packet(timeout=1.0)
            except NoPackets:
                continue
            except Shutdown:
                break
            self.handle_packet(recv)
```

对IP数据包的转发

> `Task 2`中在转发表中匹配数据包的目的`IP`地址之后，下一步就是将包**转发**出去。路由器收到一个`IP`包之后，**查询转发表**通过链路层将其转发到**下一跳**，这需要知道**下一跳**的`MAC`地址才行，这里使用到`Lab3`**实现的**`ARP`表。

实现逻辑

- 当目的`IP`地址为**路由器的端口地址**或者目的地址**不在转发表中**则丢弃。
- 查询转发表可以知道将数据包发到目的地址的**下一跳`IP`地址**以及**对应的出端口eth**。查询`ARP`表可以找到下一跳`IP`地址对应的`MAC`地址时，就可以依此构造好`Ethernet Head`直接将数据包从出端口`eth`发出即可。
- 但是当`ARP`表中**没有找到**下一跳`IP`这一项时：

    - 如果在等待回应表 `wait_Queue` 中没有等待此 `IP` 对应 `MAC` 的表项：
        - 路由器需要在 `wait_Queue` 中添加新的表项，具体操作**见上文**
        - 还需要构造 `ARP_request` 请求包，源 `MAC` 和源 `IP` 即 `eth` 的 `MAC` 和 `IP`，目的 `IP` 为下一跳 `IP`，从 `eth` 端口发出该请求包即可。
    - 如果表中已经有请求该 `IP` 对应 `MAC` 的表项，只需要将该数据包附加到**对应的数据包队列**末尾即可。
- 维护**数据包队列**
    - 这里实际上是用列表来实现队列的，通过 `append` 方法，当某个数据包的 `nextHop` 请求已经在等待回应表中，只需将其 `append` 到队尾即可。

核心代码

```
1  elif(packet.has_header(IPv4)): #a IP packet
2      packet.get_header(IPv4).ttl -= 1
3      ipv4 = packet.get_header(IPv4)
4      if ipv4.dst not in self.myIPs: # the dest is not in router
5          for item in self.fw_table:
6              if ipv4.dst in item[0]: # the dst addr is belong item's netaddr
7                  print(f"the nextHop's IP:{nextHop} has been in arp_table")
8                  nextHop = item[2] if item[2] != None else ipv4.dst
9                  out_intf = self.net.interface_by_name(item[3])
10                 if nextHop in self.arp_table.keys():
11                     eth_head = packet.get_header(Ethernet) # modify the ethernet
   head
12                     eth_head.src = out_intf.ethaddr
13                     eth_head.dst = self.arp_table[nextHop] # use the arp_table
   here
14                     next_packet = eth_head + ipv4 + packet.get_header(ICMP)
15                     self.net.send_packet(out_intf, next_packet)
16                 else: #nextHop is not in arp_table, make ARP request for
   nextHop's IP
17                     if  nextHop not in self.wait_Queue:
18                         print(f"the nextHop's IP:{nextHop} is not in waitQueue")
19                         arp_request = create_ip_arp_request(out_intf.ethaddr,
   out_intf.ipaddr, nextHop)
20                         self.net.send_packet(out_intf, arp_request)
21                         self.wait_Queue[nextHop] = [time.time(), 1, out_intf,
   arp_request, [packet]]
22                         print(f"add a nextHop's IP: {nextHop} in waitQueue")
23                     else: #nextHop is in wait queue, add the packet in the pkt
   queue
24                         print(f"the nextHop's IP:{nextHop} has been in
   waitQueue")
25                         self.wait_Queue[nextHop][4].append(packet)
```

- 有个细节在于转发该包时 `IP` 头的 `ttl` 应该减1，我放在了查找转发表之前(第3行)。
- 第8行中设置 `nextHop` 时，如果转发表中为 `None` 则表示下一跳为路由器端口对应的子网，这时将 `nextHop` 赋值成 `ipv4.dst` 表示下一跳将**直接到达目的地**，其余情况需要按照转发表设置。

对ARP回应包的处理

实现逻辑

> 通过上面的分析可以知道，路由器收到一个 `IP` 数据包后，路由器发出对其 `nextHop` 的 `IP` 对应 `MAC` 的 `ARP` 请求包。由于可能有多个 `IP` 数据包发向同一个 `nextHop` ，我们维护了一个**数据包队列**。需要注意的是收到 `ARP` **回应包**后这些包必须**按照到达路由器的顺序**依次转发，并且更新 `ARP` 表。

- 如上文，通过列表实现数据包队列，添加时用 `append()` 方法，在依次转发时通过 `for` 循环**从头开始遍历**依次转发，也就实现了所谓的 `FIFO` 。
- 构建新的转发包需要将之前以太网头未知的**目的** `MAC` 填好，即 `eth_head.dst = next_mac` ，同时将以太网头的源地址填成**出端口的地址**。
- `nextHop` 对应数据包队列**全都依次发完后**，将 `nextHop` 从等待回应表删除即可。

核心代码

```
elif arp.operation == ArpOperation.Reply:
    next_ip = arp.senderprotoaddr
    next_mac = arp.senderhwaddr
    self.arp_table[next_ip] = next_mac
    if next_ip in self.wait_Queue.keys(): #if this ip is waited
    for wait_pkt in self.wait_Queue[next_ip][4]: # release the packets which wait ip's(nextHop) MAC
        eth_head = wait_pkt.get_header(Ethernet)
        eth_head.src =  self.wait_Queue[next_ip][2].ethaddr
        eth_head.dst = next_mac
        next_packet = eth_head + wait_pkt.get_header(IPv4) + wait_pkt.get_header(ICMP)
        self.net.send_packet(self.wait_Queue[next_ip][2],  next_packet)
    self.wait_Queue.pop(next_ip) #delete this ip
```

实验结果

在虚拟环境下通过 `testcase/myrouter2_testscenario.srpy` **所有测试用例**，结果如下：

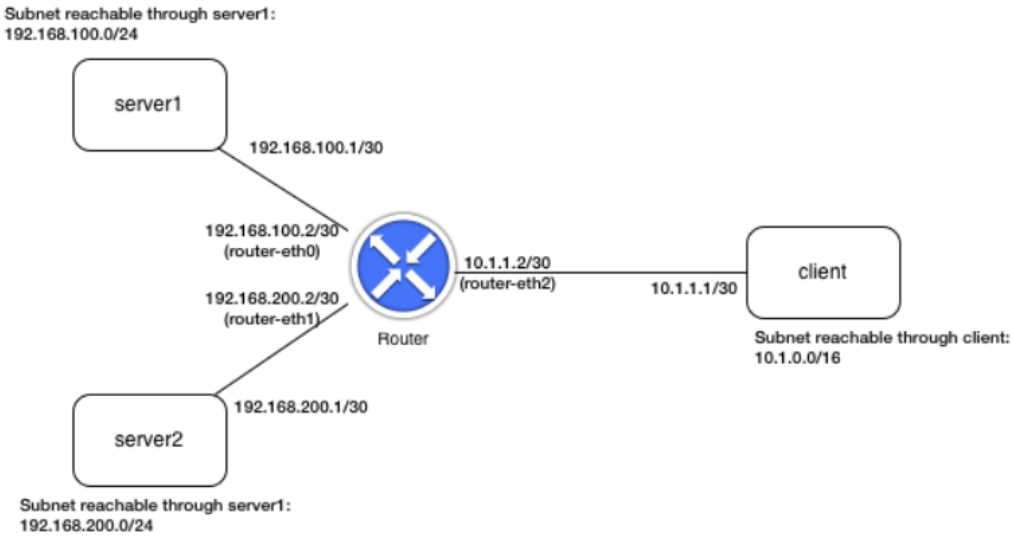
```
19  An IP packet from 192.168.1.239 for 10.200.1.1 should arrive
    on router-eth0.  No forwarding table entry should match.
20  An IP packet from 192.168.1.239 for 10.10.50.250 should
    arrive on router-eth0.
21  Router should send an ARP request for 10.10.50.250 on
    router-eth1
22  Router should try to receive a packet (ARP response), but
    then timeout
23  Router should send an ARP request for 10.10.50.250 on
    router-eth1
24  Router should try to receive a packet (ARP response), but
    then timeout
25  Router should send an ARP request for 10.10.50.250 on
    router-eth1
26  Router should try to receive a packet (ARP response), but
    then timeout
27  Router should send an ARP request for 10.10.50.250 on
    router-eth1
28  Router should try to receive a packet (ARP response), but
    then timeout
29  Router should send an ARP request for 10.10.50.250 on
    router-eth1
30  Router should try to receive a packet (ARP response), but
    then timeout
31  Router should try to receive a packet (ARP response), but
    then timeout


All tests passed!
```

在mininet上部署测试

测试方法

> `mininet` 上部署的网络拓扑如下：我选择用 `server1` 来 `ping client`



在 `router` 上的 `xterm` 上运行 `myrouter2.py`，然后 `router wireshark &` 监控 `router-eth0` 和 `riuter-eth2` 两个端口，构造流量 `server1 ping -c2 10.1.1.1`，得到如下信息：

为了方便观察，我在 `myrouter2.py` 中添加了一些输出语句，用于**打印路由器状态和动作信息**，如下图：

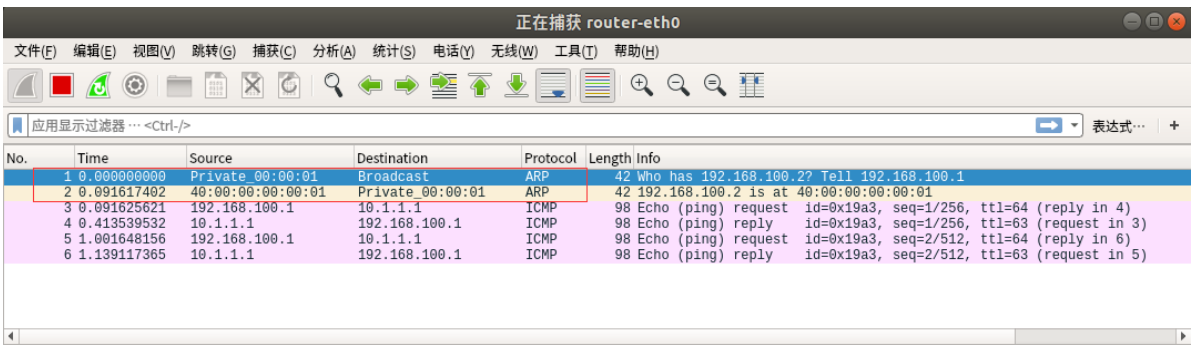
```
16:29:30 2021/05/03     INFO received packet Ethernet 10:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.1
00.2 on router-eth0
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|    192.168.100.1     |  10:00:00:00:00:01   |
+--------------------------------------------+
================================================
16:29:30 2021/05/03     INFO received packet Ethernet 10:00:00:00:00:01->40:00:0
0:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 7177 1 (56
data bytes) on router-eth0
the nextHop's IP:10.1.1.1 is not in waitQueue
add a nextHop's IP: 10.1.1.1 in waitQueue
16:29:30 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 ARP | Arp 30:00:00:00:00:01:10.1.1.1 40:00:00:00:00:03:10.1.1.2 on ro
uter-eth2
receive a arp_reply for the nextHop's IP:10.1.1.1
the waitQueue's keys: dict_keys([IPv4Address('10.1.1.1')])
arp_reply reply the waited nextHop's IP:10.1.1.1
16:29:30 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 7177 1 (56 da
ta bytes) on router-eth2
the nextHop's IP:192.168.100.1 has been in arp_table
16:29:31 2021/05/03     INFO received packet Ethernet 10:00:00:00:00:01->40:00:0
0:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 7177 2 (56
data bytes) on router-eth0
the nextHop's IP:10.1.1.1 has been in arp_table
16:29:31 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 7177 2 (56 da
ta bytes) on router-eth2
the nextHop's IP:192.168.100.1 has been in arp_table
16:29:35 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 ARP | Arp 30:00:00:00:00:01:10.1.1.1 00:00:00:00:00:00:10.1.1.2 on ro
uter-eth2
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|    192.168.100.1     |  10:00:00:00:00:01   |
+--------------------------------------------+
|       10.1.1.1       |  30:00:00:00:00:01   |
+--------------------------------------------+
================================================
```

分析完整过程

- 首先路由器收到来自 `server1` 的 `ARP request` 包，询问 `router-eth0` 的 `MAC` 地址，这时候 `router` 的 `ARP` 表被更新，即：

```
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|    192.168.100.1     |  10:00:00:00:00:01   |
+--------------------------------------------+
================================================
```

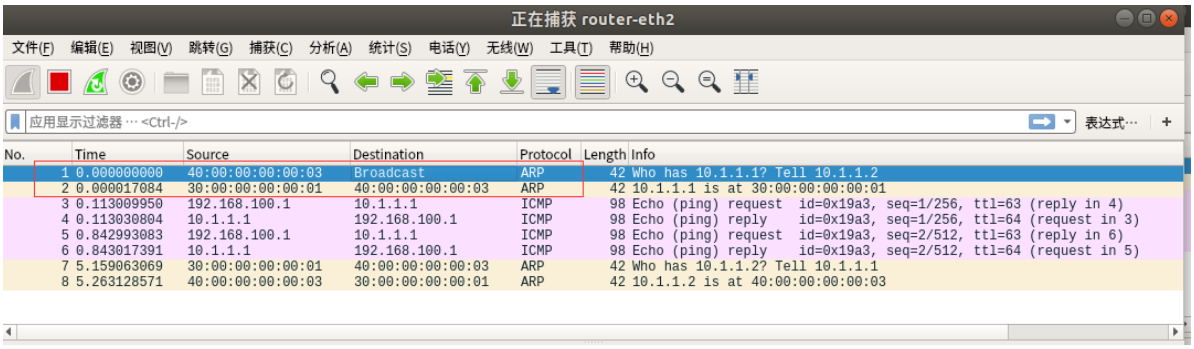
  之后 `router` 应该通过 `router-eth0` 发出 `ARP` 回应包，通过 `wireshark` 可以看到捕获到了这两个 `ARP` 包：



- 然后 `server1` 向网关 `router` 的 `router-eth0` 端口发出目的地址为 `10.1.1.1` 的 `echo request` 包，对应上图第3项。之后 `router` 在转发表中查询到最长前缀匹配表项，填好 `nextHop` 为 `10.1.1.1`，转发表匹配如下图：

```
[IPv4Network('192.168.100.0/30'), IPv4Address('255.255.255.252'), None, 'router-
eth0']
[IPv4Network('10.1.1.0/30'), IPv4Address('255.255.255.252'), None, 'router-eth2'
]
[IPv4Network('192.168.200.0/30'), IPv4Address('255.255.255.252'), None, 'router-
eth1']
[IPv4Network('192.168.100.0/24'), IPv4Address('255.255.255.0'), IPv4Address('192
.168.100.1'), 'router-eth0']
[IPv4Network('192.168.200.0/24'), IPv4Address('255.255.255.0'), IPv4Address('192
.168.200.1'), 'router-eth1']
[IPv4Network('10.1.0.0/16'), IPv4Address('255.255.0.0'), IPv4Address('10.1.1.1')
, 'router-eth2']
```

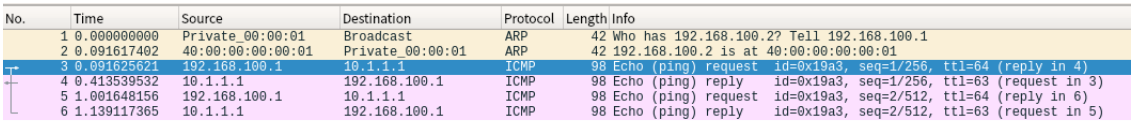
由于 `router` 在当前 `ARP` 表中**找不到** `10.1.1.1` 的表项，因此需要从 `router-eth2` 端口发出 `ARP request` 包，当前的 `wait_Queue` 中**也没有**该表项，因此还需要将 `10.1.1.1` 放入等待队列中，之后在此端口收到了 `client` 的 `ARP reply`，如下图：

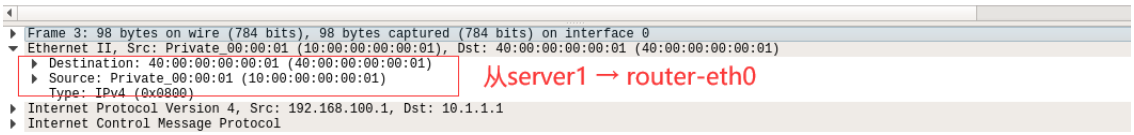


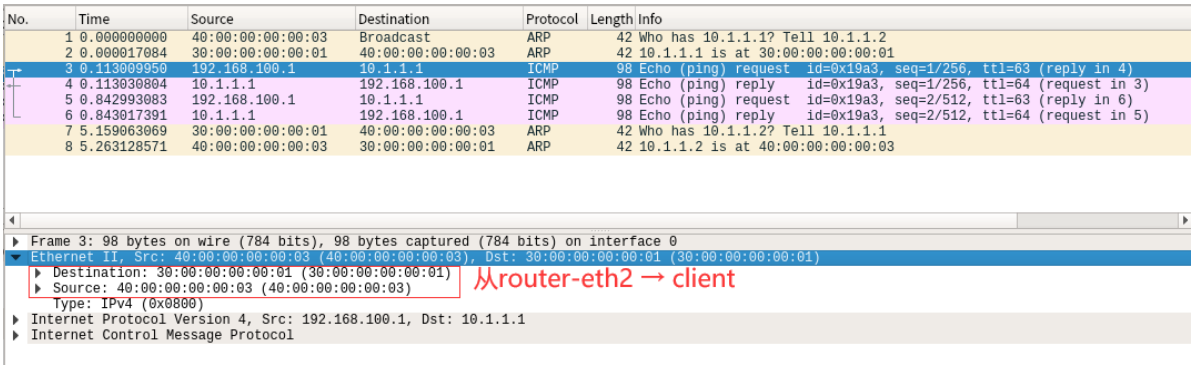
对应的 `router` 输出信息如下：记录了 `router` 发出询问 `10.1.1.1` 的 `ARP` 请求包到收到 `ARP` 回应全过程，涉及到了等待队列的维护，见下图的彩色输出：

```
16:29:30 2021/05/03     INFO received packet Ethernet 10:00:00:00:00:01->40:00:0
0:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 7177 1 (56
data bytes) on router-eth0
the nextHop's IP:10.1.1.1 is not in waitQueue
add a nextHop's IP: 10.1.1.1 in waitQueue
16:29:30 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 ARP | Arp 30:00:00:00:00:01:10.1.1.1 40:00:00:00:00:03:10.1.1.2 on ro
uter-eth2
receive a arp_reply for the nextHop's IP:10.1.1.1
the waitQueue's keys: dict_keys([IPv4Address('10.1.1.1')])
arp_reply reply the waited nextHop's IP:10.1.1.1
```

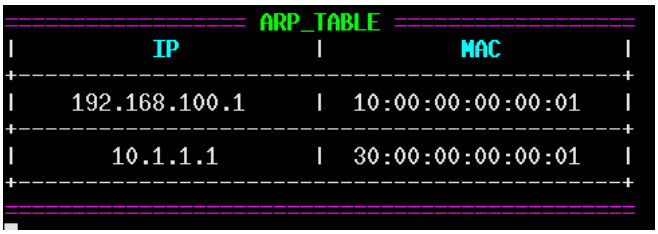
- `router` 收到对 `10.1.1.1` 的回应后，知道了其 `MAC` 地址，然后将 `echo request` 转发到 `10.1.1.1`，这时 `echo request` 的以太网头中的源地址和目的地址会发生改变，原来是 `server1 → router-eth0`，现在是 `router-eth2 → client`，二者对比如下图：







并且在收到对 `10.1.1.1` 的回应后， `ARP` 表也会更新成下图的样子：



- `router` 从 `router-eth2` 端口转发出 `echo request` 后同样会在该端口收到 `echo reply` (源为 `10.1.1.1` 目的为 `192.168.100.1`)。 `router` 需要将其转发给 `client` ，查询转发表匹配项如下，填好 `nextHop` 为 `192.168.100.1` 。

```
[IPv4Network('192.168.100.0/30'), IPv4Address('255.255.255.252'), None, 'router-eth0']
[IPv4Network('10.1.1.0/30'), IPv4Address('255.255.255.252'), None, 'router-eth2']
[IPv4Network('192.168.200.0/30'), IPv4Address('255.255.255.252'), None, 'router-eth1']
[IPv4Network('192.168.100.0/24'), IPv4Address('255.255.255.0'), IPv4Address('192.168.100.1'), 'router-eth0']
[IPv4Network('192.168.200.0/24'), IPv4Address('255.255.255.0'), IPv4Address('192
```

这时的 `nextHop` **已经在** `ARP` **表中**了，无需再次询问，**直接转发**即可，输出信息如下：

```
15:31:25 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 6563 1 (56 data bytes) on router-eth2
the nextHop's IP:192.168.100.1 has been in arp_table
15:31:26 2021/05/03     INFO received packet Ethernet 10:00:00:00:00:01->40:00:00:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 6563 2 (56 data bytes) on router-eth0
the nextHop's IP:10.1.1.1 has been in arp_table
15:31:26 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 6563 2 (56 data bytes) on router-eth2
the nextHop's IP:192.168.100.1 has been in arp_table
15:31:30 2021/05/03     INFO received packet Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 ARP | Arp 30:00:00:00:00:01:10.1.1.1 00:00:00:00:00:00:10.1.1.2 on router-eth2
```

上图记录了 `router` 第一次收到 `client` 的 `echo reply` 后，直接查询 `ARP` 表转发给 `192.168.100.1` 。

- 由于我的命令是 `ping -c2` 要发两次。第二次时 `router` 还是在 `router-eth0` 收到 `echo request` ，而这时的 `ARP` 表记录了 `10.1.1.1` 和 `192.168.100.1` 的信息，因此直接查 `ARP` 表转发，之后在 `router-eth2` 收到 `echo reply` ，同样直接查 `ARP` 表转发，上图中有详细记录。

通过上面的分析，结合 `wireshark` 抓包文件和输出信息，可以看到 `router` 的行为**完全符合预期**，测试正确。

> 上面分析过程中的抓包文件已经放在 `/report` 文件中，命名为 `lab_4_eth0` 和 `lab_4_eth2`

总结感想

本次实验相比上次内容更多更难，编写过程中出错的地方比较多，在编写时多输出一些状态信息，比如表项内容等等，方便我找出了很多 `bug` 。比如这次我使用 `IPv4Network` 的 `in` 方法来判断某个 `IP` 是否属于某个子网，但是编写时出现了匹配不上的情况，原来是因为在读取 `txt` 文件配置转发表时，我的 `nextHop` 是字符串类型的，导致有些表项匹配不上，而那些按照路由器端口配置的表项却可以正常匹配。这个 `bug` 是通过完整打印转发表的信息后才发现的(**包括每一项的类型**)。这之后的所有路由器的每个**关键动作**我都设置了输出，之后的 `debug` 还是非常顺利的，也为在 `mininet` 上验证提供了参考。

再有就是帮助我理清了路由器的完整转发逻辑，对三种包： `ARP` 请求包、 `ARP` 相应包、 `IP` 包的处理转发，等待回应队列的构建等加深了印象。

还有对于 `python` 的使用参考一些文章:

IPv4/IPv6 manipulation library

How to sort a list of objects based on an attribute of the objects?

Python四种逐行读取文件内容的方法

最后感谢助教的耐心批改😁😁😁