Computer Networks: lab04

Router2 (forwading table & arp reply)

任课教师：田臣　　助教：方毓楚，郑浩等

学院	计算机科学与技术系	专业	计算机科学与技术
学号	201220154	姓名	王紫萁
Email	201220154@smail.nju.edu.cn	开始/完成时间	2022. 4.10/2022.4.11

1. 实验目的

在lab3的基础上给路由器加上转发表和和发送ARP请求的功能

2. 实验原理

路由器中的几个抽象数据类型

- 转发表：通常为以下格式：
  - `network address`：到来的数据报的目的地址所在子网
  - `subnet address`：对应前者的子网掩码
  - `next hop`：下一条地址。当下一跳为0.0.0.0时说明该子网下的所有终端都和该路由器"直接"相连（不考虑链路层交换机的话），于是下一跳就是数据包中的目的IP；否则说明下一条需要经过其他路由器，`next hop`：中标记了该路由器的端口IP，于是下一跳就是该端口。
  - `interface`：转发这一子网的数据包需要发送到哪个端口
  - `prefix`：掩码长度（可选）

network address	subnet address	next hop	interface	prefix
192.168.1.0	255.255.255.0	0.0.0.0	router-eth0	24
10.10.0.0	255.255.0.0	0.0.0.0	router-eth1	16
172.16.42.0	255.255.255.252	0.0.0.0	router-eth2	30
172.16.0.0	255.255.0.0	192.168.1.2	router-eth0	16
172.16.128.0	255.255.192.0	10.10.0.254	router-eth1	18
172.16.64.0	255.255.192.0	10.10.1.254	router-eth1	18
10.100.0.0	255.255.0.0	172.16.42.2	router-eth2	16

- ARP缓存
    - 由于我们只知道下一跳（可能是数据包的直接目的地址或者是转发表中记录的下一跳地址）的IP地址，而转发数据报需要明确下一跳的MAC地址，因此ARP中就负责记录这种IP-MAC映射关系
- 端口等待队列

  在端口处的一个很重要的处理结构，发送给这一端口的所有的数据报都需要经过该**队列组**（而不是单一队列）处理，可以类比为一个拥有多个车道的单向红绿灯路口
    - 当某一个车道（所有在这一车道上的车拥有同一目的IP地址，或者是直接目的IP或者是下一跳IP）允许通行时（说明此时已知这一目的IP的MAC地址）车道就会畅通无阻
    - 否则（目的MAC在ARP缓存中的映射表项为空）所有到达这一车道的车辆都需要排队等待并由路由器在这条大道上派出一个使者（ARP Request）询问这些车辆实际开往的物理地址（MAC）。
        - 如果该端口接收到了回复，就看这一回复中的目的物理地址应该分配给哪一车道（根据该回复的源IP地址（就是所有在该车道上等待的车辆的目的IP）判断）；
        - 如果在某一指定的时间段后仍没有收到回复重复发送若干次直到若干次后仍没有收到就果断放弃，将所有车辆销毁（清空该车道，实际上不应该销毁而是应该让这些车辆带着ICMP回到源目的地）

为了写出逻辑清晰的路由器，我们需要了解路由器处理三种不同类型的数据报的机制：

2.1. ARP Request（Lab3）

lab3已经实现了对ARP请求的应答，即对于ARP目的IP是路由器的一个端口时，我们需要将自己（该端口）的MAC地址原路返回，使路由器与该终端设备在子网的意义下建立关联。

2.2. ARP Reply

ARP 回复的报文很大概率是回答之前路由器因为某种原因（在端口等待队列小节中我们做了类比说明）发送的ARP请求，于是就知道到达该端口的下一个目的地的MAC地址从而得以转发数据报

2.3. IPv4 datagram

IPv4的报文中通常包含了网络建立联系后的正式数据内容，路由器需要根据目的IP查找转发表，如果成功找到就更新端口等待队列的MAC地址值并将该包加入队列由队列处理发包，否则发送ARP Request请求该端口的MAC地址

3. 转发表

我们重点关注建表的过程而不会刻意讲解具体写法

3.1. 转发表项（FWEntry）

成员如下：

```
class FWEntry(object):
    def __init__(self, networkAddr: IPv4Address,
                 subnetAddr: IPv4Address,
                 nextHopAddr: IPv4Address,
```

```
                  intfName: str):
        self.networkAddr = networkAddr
        self.subnetAddr = subnetAddr
        self.nextHopAddr = nextHopAddr
        self.intfName = intfName
        self.prefixLen = self._get_prefix_len()

    def _get_prefix_len(self):
        netaddr = IPv4Network(str(self.networkAddr) + '/' +
str(self.subnetAddr))
        return netaddr.prefixlen

```

值得注意的是子网地址必须是有效前缀和全零后缀的形式而不是子网中某一具体的IP地址，因此在转发表建立时需要提前处理

```
networkAddr = IPv4Address(int(networkAddr) & int(subnetAddr))
```

将具体的IP地址转换成子网的IP地址

3.2. 转发表建立（FWTable）

首先建立端口的转发表项（next），随后根据farwarding_table.txt中的列表推导式提取四个重要表项的信息。

```
# set forwarding entry of interfaces
networkAddr = intf.ipaddr
subnetAddr = intf.netmask
nextHopAddr = IPv4Address('0.0.0.0')
networkAddr = IPv4Address(int(networkAddr) & int(subnetAddr))
interface = intf.name
self._table.append(FWEntry(networkAddr, subnetAddr, nextHopAddr,
interface))
```

从文件扩展表的方法和从端口构建的无异因此不再赘述，本次实验根据 `testscenario` 生成文件扩展得到的转发表在第二节中展示

3.3. 转发表查找

满足最大长度匹配原则查找，需要遍历一遍转发表，对IP地址和表项中的子网掩码做按位与得到的子网地址和表项中子网地址比较并给出，并选择表项中 `prefix length` 最大的一项返回，如果找不到就返回 `None`

4. 端口等待队列

4.1. 端口队列（IPQueue）

4.1.1. 结构声明

```
class IPQueue:
    def __init__(self,
                 capacity:int,
                 srcMAC:EthAddr,
                 srcIP:IPv4Address,
                 dstIP:IPv4Address,
                 interface:Interface,
                 net: switchyard.llnetbase.LLNetBase):
        self._net = net          # the net which router is in
        self._queue = Queue()    # main queue
        self._lastSendTime = 0.0    # last arp request sending time
        self._timeLimit = 1.0    # time limit of arp waiting
        self._cntTrial = 0       # count of sent arp request
        self._trialLimit = 5     # count limit of sending request
        self._capacity = capacity
        self._srcMAC = srcMAC    # source MAC of interface
        self._srcIP = srcIP
        self._dstMAC = None      # destination MAC of interface
        self._dstIP = dstIP
        self._interface = interface
```

值得注意的是，`self._dstMAC`指定了当前队列的状态，是阻塞（None）还是畅通（有效MAC地址）

4.1.2. 首次发送ARP请求

```
    def send_arp_request(self):
        # first find there's no IP-MAC entry in the ARP cache
        if not self._queue.empty():
            self._dstMAC = None      # set queue state to BLOCKED
            self._cntTrial = 0
            self._lastSendTime = time()
            srcMAC = self._srcMAC
            srcIP = self._srcIP
            self._cntTrial += 1      # first trial of sending
            arpRequest = create_ip_arp_request(srchw=srcMAC,
                                               srcip=srcIP,
                                               targetip=self._dstIP)
            #forward arp request
            self._net.send_packet(self._interface, arpRequest)
```

首次发送ARP请求的时机就是在ARP缓存表中找不到对应表项。经过队列不空的检查后进行一系列状态设定后发送请求

4.1.3. 更新ARP请求

```
    def update_request(self):
        # check if time is on , if so ,resend the arp request
        if self._queue.empty() and self._dstMAC is None or self._dstIP is
None:
            pass    #send iff there's packets waiting and state is BLOCKED
        elif self._dstMAC is None:
            now = time()
            if now - self._lastSendTime > self._timeLimit:
                # count of trials hasn't expired yet
                if (self._cntTrial < self._trialLimit):
                    self._cntTrial += 1
                    self._lastSendTime = now    # update last sending time
                    arpRequest = create_ip_arp_request(srchw=self._srcMAC,
                                                       srcip=self._srcIP,

targetip=self._dstIP)
                    self._net.send_packet(self._interface, arpRequest)
                else:
                    # drop all packets
                    while not self._queue.empty():
                        self._queue.get()
```

在主循环中每次都检查端口上的所有目的IP是否有目的MAC与之对应，没有的话检查是否超时，并将在尝试次数未满的情况下发送ARP请求。

4.1.4. 发送等待在队列中的所有数据报

```
    def clear(self):
        if self._dstMAC is not None:    # destination MAC available
            print(f'forward dst mac: {self._dstMAC}')
            while not self._queue.empty():
                packet = self._queue.get()  #send all packet on to the
interface
                headList = packet.headers()
                for headName in headList:
                    header = packet.get_header_by_name(headName)
                    if isinstance(header, Ethernet):#change dstMAC and
srcMAC
                        header.dst = self._dstMAC
                        header.src = self._srcMAC
                    elif isinstance(header, IPv4):  #decrease ttl before
forwarding
                        header.ttl -= 1
                self._net.send_packet(self._interface, packet)
```

4.2. 端口队列组（INTFQueue）

是该端口上的所有端口队列的字典集合，拥有与端口队列一致的接口，只不过需要路由器传入目的IP查找正确的队列，单独有一个成员函数 `add_queue(dstIP)` 用来对一个新到来的目的IP（包括从转发表的下一跳初始化构建时）添加端口队列，该函数初始化一个新的端口队列并加入字典。

5. 路由器主要逻辑

5.1. ARP Request

实现方法见实验三

5.2. ARP Reply

经过下述三步处理：

- 向 `ARP cache` 中添加 `reply` 发送者的MAC-IP映射表项

  `ARPCache.add_entry(dstIP=srcIP, dstMAC=srcMAC)`

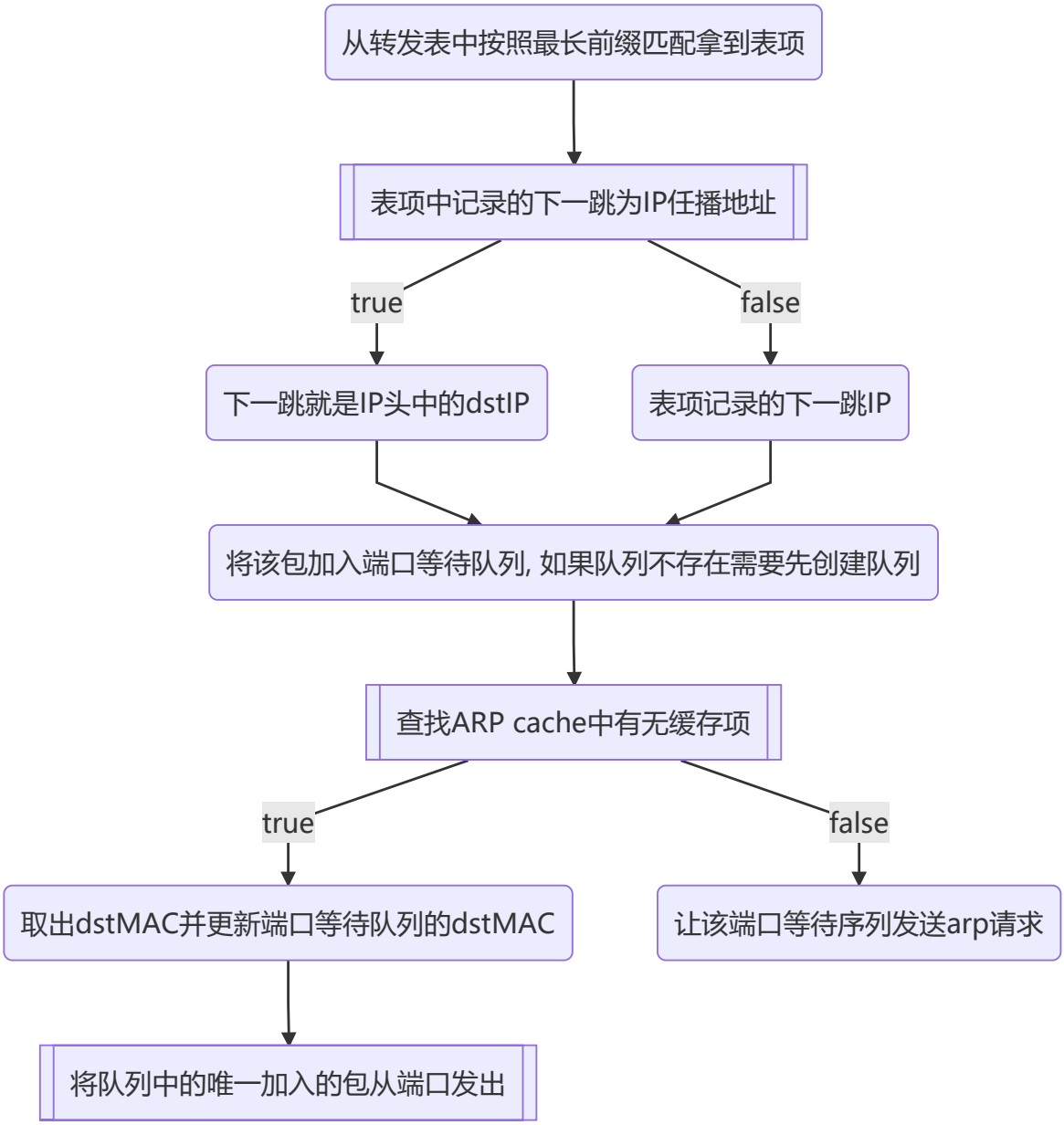
- 更新到来端口（`intfName`）的端口队列组的目的MAC：
  `INTFQueues[intfName].update_dstMAC(dstIP=srcIP, dstMAC=srcMAC)`

- 发送等待在该队列中的所有数据报

  `INTFQueues[intfName].clear(dstIP=srcIP)`

5.3. IPv4 Packet

我们只负责处理目的IP可在转发表中找到的包，找不到的包暂时丢弃，发送给路由器自己（某一端口）的包也不做处理



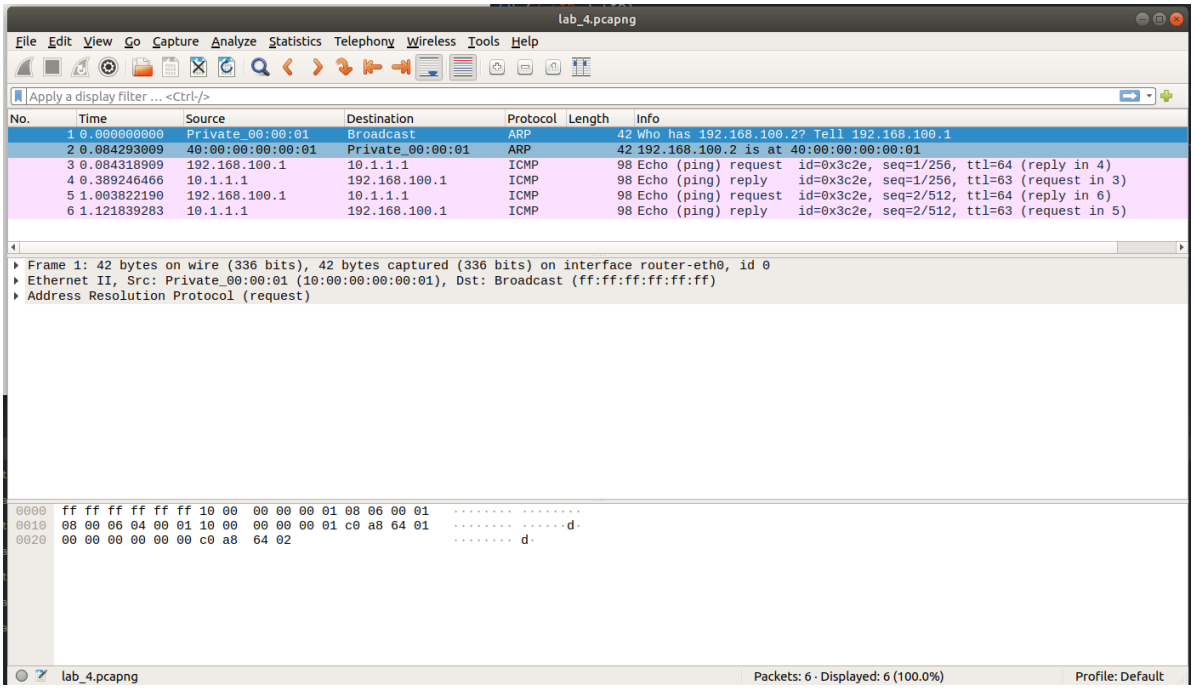
6. 测试结果

- 6.1. test scenario测试

```
    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!
```

6.2. mininet测试



从 `server1 ping` 两个 `ICMP` 报文给到 `client`，结果发现 `ARP` 被成功处理且 `ttl` 也相应减1

7. 实验收获

掌握了路由器的路由原理，并灵活掌握了简单静态路由器的内部结构和转发表结构以及转发数据报和发送ARP请求的时机和方法。