南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：**李文中**

学院	计算机科学与技术系	专业（方向）	计算机科学与技术系
学号	181860109	姓名	吴润泽
Email	181860109@smail.nju.edu.cn	开始/完成日期	2020/3/25-2020/3/27

1. 实验名称： Respond to ARP

2. 实验目的

1. 理解路由器ARP协议机制
2. 掌握ARP应答表段的编写格式
3. 理解ARP缓存表的更新机制

3. 实验过程

Task 2 处理ARP请求数据包

a. 实现原理

1. 判断是否为ARP请求包以及目的ip是否在路由器接口中，若不在则抛弃；
2. 构造对应的ARP应答包发送至请求包的端口即可。

b. 代码编写

```
    def forward_packet(self, port, packet): #自定义转发数据报函数
        if packet[Ethernet].ethertype == EtherType.ARP:
            if packet[Arp].operation == ArpOperation.Request:
                self.arp_request(port, packet)#处理ARP请求包

    def arp_request(self, port, packet):#ARP请求数据报处理函数
        ...#获得目的、源的ip和mac地址
        if dst_ip in self.mydic:
            ether = Ethernet(src=self.mydic[dst_ip],
                             dst=src_mac,
                             ethertype=EtherType.ARP)
            arp = Arp(operation=ArpOperation.Reply,
                      senderhwaddr=self.mydic[dst_ip],
                      senderprotoaddr=dst_ip,
                      targethwaddr=src_mac,
                      targetprotoaddr=src_ip)
            arppacket = ether + arp#构造ARP reply包
            self.net.send_packet(port, arppacket)

    def router_main(self):
        while True:
            gotpkt = True
            try:
                timestamp, dev, pkt = self.net.recv_packet(timeout=1.0)
                self.forward_packet(dev,pkt)#收到包便进行转发处理
```

c. 实现测试

I. test scenario测试

给定测试文件测试结果

运行$swyard\ -t\ lab_3/routertests1.srpy\ myrouter.py$，结果如下：

```
Results for test scenario ARP request: 9 passed, 0 failed, 0 pending


Passed:
1   ARP request for 192.168.1.1 should arrive on router-eth0
2   Router should send ARP response for 192.168.1.1 on router-
    eth0
3   An ICMP echo request for 10.10.12.34 should arrive on
    router-eth0, but it should be dropped (router should only
    handle ARP requests at this point)
4   ARP request for 172.16.42.1 should arrive on router-eth2
5   Router should send ARP response for 172.16.42.1 on router-
    eth2
6   ARP request for 10.10.1.2 should arrive on router-eth1, but
    the router should not respond.
7   ARP request for 10.10.1.1 should arrive on on router-eth1
8   ARP request for 10.10.0.1 should arrive on on router-eth1
9   Router should send ARP response for 10.10.0.1 on router-eth1


All tests passed!
```

测试文档说明

```
    s.add_interface('eth1', '10:00:00:00:00:00','1.0.0.0')
    s.add_interface('eth2', '20:00:00:00:00:00','2.0.0.0')
    s.add_interface('eth3', '30:00:00:00:00:00','3.0.0.0')
     #case1 ask 1.0.0.0's mac normal
    request_pkt =
create_ip_arp_request('40:00:00:00:00:00','4.0.0.0','1.0.0.0')
    s.expect(PacketInputEvent('eth1',request_pkt))
    reply_pkt = create_ip_arp_reply('10:00:00:00:00:00',
'40:00:00:00:00:00','1.0.0.0', '4.0.0.0')
    s.expect(PacketOutputEvent('eth1',reply_pkt))
    #case2 ask ip which not in router
    request_pkt =
create_ip_arp_request('40:00:00:00:00:00','4.0.0.0','5.0.0.0')
    s.expect(PacketInputEvent('eth1',request_pkt))
    s.expect(PacketInputTimeoutEvent(0.4))
    #case3 packet type not request can't process
    p = Ethernet(src="00:11:22:33:44:55", dst="66:55:44:33:22:11") +
IPv4(src="1.1.1.1", dst="2.2.2.2", protocol=IPProtocol.UDP) +
UDP(src=5555, dst=8888)
    s.expect(PacketInputEvent('eth1', p)")
    s.expect(PacketInputTimeoutEvent(0.4))
    #case4 ask 3.0.0.0's mac normal
    request_pkt =
create_ip_arp_request('60:00:00:00:00:00','6.0.0.0','3.0.0.0')
    s.expect(PacketInputEvent('eth2',request_pkt))
    reply_pkt = create_ip_arp_reply('30:00:00:00:00:00',
'60:00:00:00:00:00','3.0.0.0', '6.0.0.0')
    s.expect(PacketOutputEvent('eth2',reply_pkt))
```

1. 设置路由器的三个接口所含的ip地址和mac地址；
2. case1、4正常查询该路由接口中ip地址对应的mac地址，应从接收端口发出特定应答包；
3. case2中目的ip不在该子网内，case3数据包类型不为request，路由器均不做处理。

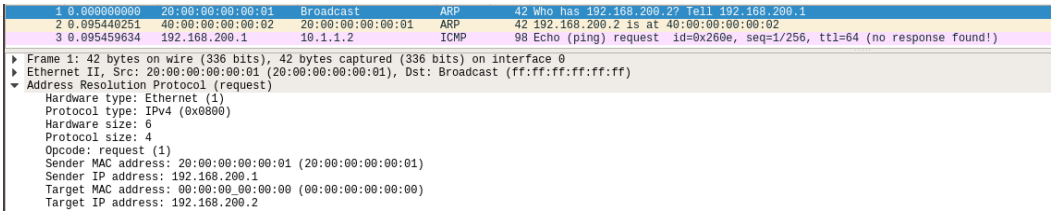
测试结果

运行$swyard\ -t\ mytests.py\ myrouter_to$，结果如下：

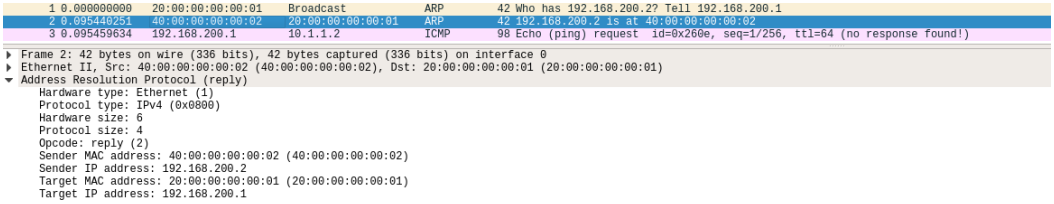
```
Results for test scenario router_arp_reply tests: 8 passed, 0 failed, 0 pending


Passed:
1   request packet Ethernet 40:00:00:00:00:00->ff:ff:ff:ff:ff:ff
    ARP | Arp 40:00:00:00:00:00:4.0.0.0
    ff:ff:ff:ff:ff:ff:1.0.0.0 arrive on eth1
2   reply packet Ethernet 10:00:00:00:00:00->40:00:00:00:00:00
    ARP | Arp 10:00:00:00:00:00:1.0.0.0
    40:00:00:00:00:00:4.0.0.0 forward to eth1
3   request packet Ethernet 40:00:00:00:00:00->ff:ff:ff:ff:ff:ff
    ARP | Arp 40:00:00:00:00:00:4.0.0.0
    ff:ff:ff:ff:ff:ff:5.0.0.0 arrive on eth1
4   the request dst ip 5.0.0.0 not in this router so nothing
    happen
5   A udp packet should arrive on eth1
6   the UDP packet current router can't process so nothing
    happen
7   request packet Ethernet 60:00:00:00:00:00->ff:ff:ff:ff:ff:ff
    ARP | Arp 60:00:00:00:00:00:6.0.0.0
    ff:ff:ff:ff:ff:ff:3.0.0.0 arrive on eth1
8   reply packet Ethernet 30:00:00:00:00:00->60:00:00:00:00:00
    ARP | Arp 30:00:00:00:00:00:3.0.0.0
    60:00:00:00:00:00:6.0.0.0 forward to eth1
```

II. 抓包测试

在mininet中运行已有topo结构，开启server2的wireshark抓包监听，执行$server2\ ping\ -c1\ 10.1.1.2$得到抓包结果如图所示：



server2发起ARP request来获取client对应的mac地址，目的mac段为全0，即进行子网段广播。



client收到ARP request包后，正确发送了ARP应答包；但是由于路由器没有实现reply和ICMP包处理机制所以无法回应。

Task3 ARP缓存表

a. 实现原理

1. 在路由表类中定义一个字典，键值对为$<ip:(mac, arp_time)>$，设置表项生存期；
2. 当ARP请求包的目的ip在该路由子网中，则将$<src_ip:(src_mac, now)>$，加入ARP缓存表；
3. 每当来一个包，便刷新当前ARP表缓存，将超时表项删除。

b. 代码编写

```
    def refresh_arp_table(self, time):#刷新ARP缓存
        for key, value in list(self.arp_table.items()):
            if time - value[1] >= self.max_arp_time:
                arp_table.pop(key)
    def forward_packet(self, port, packet):
        now_time = time.time()
        self.refresh_arp_table(now_time)#每来一个包便进行刷新
    def arp_request(self, port, packet):
        if dst_ip in self.mydic:
            self.arp_table[src_ip] = (src_mac, time.time())#加入ARP缓存
表
```

c. 实现测试

在mininet中依次执行如下命令：

$server2\ ping\ -c1\ server1$

$server1\ ping\ -c1\ server2$

$client\ ping\ -c1\ server1$

在每次更新ARP表项时，打印当前所有项，结果如下图所示：

可以看到每当来一个目的地址在路由器接口中的ARP包都会更新arp表。

同时，超时之后也会删除过期的arp表项。

4. 总结与感想

本次实验难度相对较小，只实现了ARP的应答机制，但是仍需理清MAC表、ARP缓存表、路由表的区别和机制才能正确编写。下次实现需要更加完善的路由器难度可能会上升，理解机制尤为重要。

5. 文档结构