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	计算机科学与技术系	专业（方向）	计算机科学与技术
学号	201220102	姓名	武雅琛
Email	201220102@smail.nju.edu.cn	开始/完成日期	2022.3.21

1. 实验名称

Respong to ARP

2. 实验目的

- **进一步熟悉Switchyard框架。**

  通过自行学习Switchyard相关**API的实现和接口**进一步熟悉lab提供的Switchyard框架。

- **了解IPv4网络ARP包头的结构和功能，构造和解析**

  根据实验手册了解ARP分组包头的具体组成结构，并且可以通过实验解析ARP包头的内容以及在此基础上构造出新的ARP分组。

- **学习和巩固三层IPv4路由器设备的基本功能及其工作细节**

  根据实验手册学习IPv4路由器的基本功能：接受ARP包，并且对符合要求的ARP包构造ARP包作出回复。

3. 实验内容

task 1:Preparation

task 2:Handle ARP request

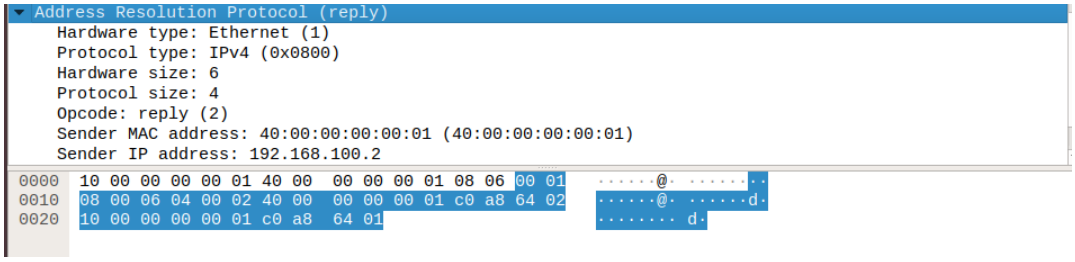
- 这一步要求路由器在接受分组后有能力**判断分组是否是ARP分组**，并且能够在此基础上通过**解析ARP请求分组的内容**确定是否需要**构造出对应的ARP回复分组**回复请求。
- 任务主要有以下几个部分：
  - **判断ARP分组**：可以利用Switchyard提供的API得到分组的ARP包头部分，如果不为空，则为一个ARP分组。

    在实验中分组没有携带数据，大致结构如下：

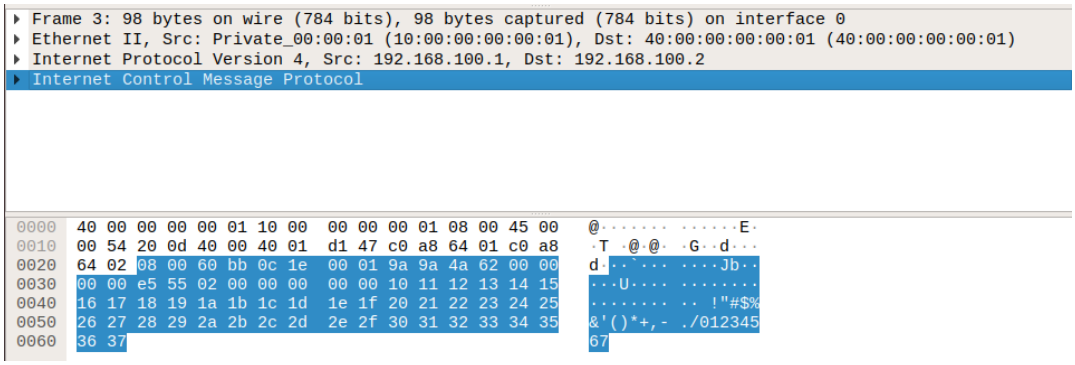
以太网包头：type:Arp	ARP包头

    并且在本实验ARP包头大小为**28bytes**，具体结构如下：

可以看到ARP包头携带了**源端口和目的端口的mac地址和IP地址。**



此处又用一个不是ARP分组的包头内容举例：



这启示我可以通过API得到分组的**ARP包头的内容，如果为空**，即不存在这一组成部分就可以判断ARP分组了。

- **判断是否为ARP请求分组**：得到ARP分组的包头部分后，首先判断是request和reply，我们无需处理reply分组。
- **判断目的IP地址**：得到ARP分组的ARP包头部分之后，同时发现包头携带了`target IP address`，只需判断该地址是否在路由器端口IP的列表中即可。
- **构造ARP回复**：此处目的端口的mac地址和IP地址都可以直接从接受到的ARP包的源地址中取得即可，**特别注意的是源端口的地址要通过IP地址寻到端口来取得IP地址。**

  ***ps:*这是因为*ARP*请求的目的端口*mac*地址为全*0*，需要接收请求的设备查询*mac*地址后填写返回正确的*mac*地址。**
- 具体实现
  - 利用Switchyard提供的API即可填写正确的参数即可。

    主要使用到的API如下：

```
arp = packet.get_header(Arp)
create_ip_arp_reply(intf.ethaddr, arp.senderhwaddr, intf.ipaddr,
arp.senderprotoaddr)
```

task 3:Cached ARP request

- 这一步要求路由器承载了上一个task的任务的基础上具有缓存ARP表的能力，为接下来的实验做准备。
- 首先要考虑更新ARP表的时机：接收到ARP请求，这包括了两种情景，**子网内和子网外**的ARP请求分组，对于**子网外的且目的端口不在路由器上的ARP请求分组**，虽然不由路由器回复，但是**也要缓存表项**方便传送回复分组。
- ARP表要求可以存储**IP地址和mac地址的一一映射表**，并且拥有在**接收一个新的ARP分组或者超时时**更新表项的能力。
  - 在逻辑上ARP高速缓存表如下：

IP_address	MAC_address	timestamp
111.111.111.111	aa:bb:cc:dd:ee:ff	12345.12345
...	...	...

- 首先我选用了`dict`**存储表格**，以达到**一一映射的**目的，在实现上我选择用`list`**取代了**`dict`**中**`value`**的内容**来满足我可以在字典中存储更多信息的需求。
- 其次为了在程序设计时对数据和操作做更好的抽象和封装，我选择创建了`Arptable`**类**来封装了相关的高速缓存表的数据和操作。

```
#myrouter.py
class Arptable(object):
    def __init__(self, table):
        ...
    def timeout(self):
        ...

    def update_Arptable(self, ipaddr, macaddr):
        ...
```

- 此外还要考虑如何检测表项超时：
  - 比较容易想到的是如果可以有**并发**，可以管理另一个进程在进行超时监督，一旦超时移出转发表。这种方法作为想法容易想到，也最接近超时机制的本意。但**实现起来却比较困难。**
  - 所以，我实现了一种**伪更新**的效果：只有在更新或查询ARP表的时候才检验是否超时。如果发生超时，删除对应的表项。

    （这样意味着有些表项只要*ARP*表没有发生访问操作，即使超时了也不会立即被移除，但是在上层看到的效果是和逻辑上的超时机制相同的）

4. 实验结果

task 2:Handle ARP request

Testing

- **测试结果如图所示：**
  - 测试1、2确定路由器有能力给出**正确的回复ARP分组。**
  - 测试3确定路由器具有判断分组**是否为ARP请求分组**有选择的作出应答。
  - 测试4、5确定路由器在识别ARP请求分组的基础上可以判断是否为**发送到路由器某一个端口的ARP分组**有选择的作出应答。
  - 同时，总共接收了三次ARP请求分组，发生了三次表项更新，其中一次的IP地址**不对应路由器端口的地址**，所以**只需要更新ARP高速缓存表。**

```
(syenv) njucs@njucs-VirtualBox:~/lab-03-Wumaomaomao$ swyard -t testcases/myrouter1_testscenario.srpy myrouter.py
12:04:11 2022/04/07     INFO Starting test scenario testcases/myrouter1_testscenario.srpy
12:04:11 2022/04/07     INFO ipaddr:192.168.1.100 macaddr:30:00:00:00:00:01 timestamp:1649304251.1558313
12:04:11 2022/04/07     INFO In ARP request port router-eth0 receive a arp packet from 192.168.1.100 to 192.168.1.1
12:04:11 2022/04/07     INFO arp reply:Ethernet 10:00:00:00:00:01->30:00:00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.1.1 3
0:00:00:00:00:01:192.168.1.100
12:04:11 2022/04/07     INFO ipaddr:192.168.1.100 macaddr:30:00:00:00:00:01 timestamp:1649304251.1558313
12:04:11 2022/04/07     INFO ipaddr:10.10.1.1 macaddr:60:00:de:ad:be:ef timestamp:1649304251.1566784
12:04:11 2022/04/07     INFO ipaddr:192.168.1.100 macaddr:30:00:00:00:00:01 timestamp:1649304251.1558313
12:04:11 2022/04/07     INFO ipaddr:10.10.1.1 macaddr:60:00:de:ad:be:ef timestamp:1649304251.1566784
12:04:11 2022/04/07     INFO ipaddr:10.10.5.5 macaddr:70:00:ca:fe:c0:de timestamp:1649304251.15703
12:04:11 2022/04/07     INFO In ARP request port router-eth1 receive a arp packet from 10.10.5.5 to 10.10.0.1
12:04:11 2022/04/07     INFO arp reply:Ethernet 10:00:00:00:00:02->70:00:ca:fe:c0:de ARP | Arp 10:00:00:00:00:02:10.10.0.1 70:
00:ca:fe:c0:de:10.10.5.5

Results for test scenario ARP request: 6 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 10.10.1.2 should arrive on router-eth1, but
    the router should not respond.
5   ARP request for 10.10.0.1 should arrive on on router-eth1
6   Router should send ARP response for 10.10.0.1 on router-eth1


All tests passed!
```

Deploying

打开wireshark，启动路由器代码。在Mininet中输入以下指令

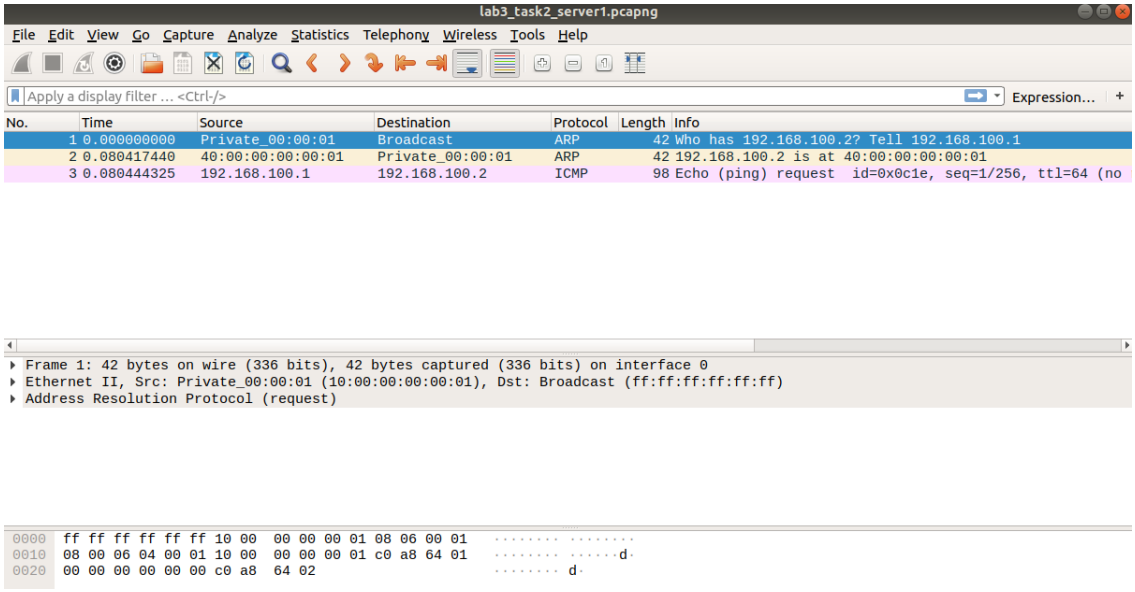
```
Mininet> server1 ping -c 1 192.168.100.2
```

在router的`xterm`看到运行日志：说明路由器**收到了server1的ARP请求分组并且作出回复。**

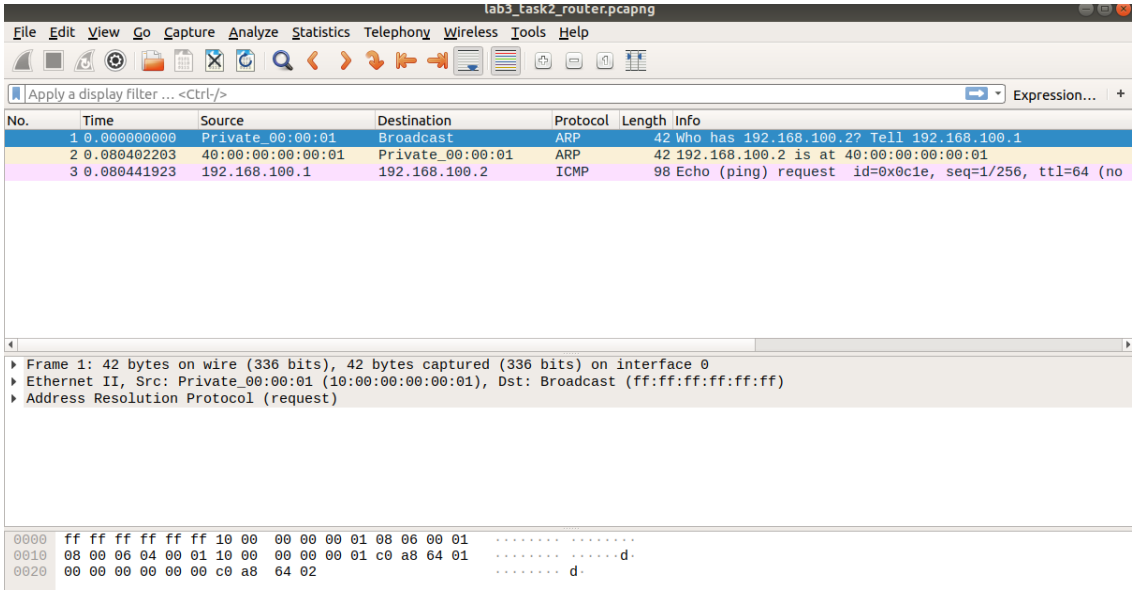
```
root@njucs-VirtualBox:~/lab-03-Wumaomaomao# source ../switchyard/syenv/bin/acti
vate
(syenv) root@njucs-VirtualBox:~/lab-03-Wumaomaomao# swyard myrouter.py
21:37:28 2022/04/06     INFO Saving iptables state and installing switchyard rul
es
21:37:29 2022/04/06     INFO Using network devices: router-eth1 router-eth2 rout
er-eth0
21:37:48 2022/04/06     INFO ipaddr:192.168.100.1 macaddr:10:00:00:00:00:01 time
stamp:1649252268.3138337
21:37:48 2022/04/06     INFO In njucs-VirtualBox port router-eth0 receive a arp
packet from 192.168.100.1 to 192.168.100.2
21:37:48 2022/04/06     INFO arp reply:Ethernet 40:00:00:00:00:01->10:00:00:00:0
0:01 ARP | Arp 40:00:00:00:00:01:192.168.100.2 10:00:00:00:00:01:192.168.100.1
```

在server1和router的wireshark上可以看到捕获结果：

- **server1**:server1发送ARP请求分组（可以看到在以太网包头中mac地址以广播形式给出，在ARP包头中为0），**询问192.168.100.2的mac地址**，并且**收到**了同一端口**返回的ARP分组**。发出了一个ICMP分组，但是由于**router没有实现有关的回复逻辑**，所以石沉大海了。



- **router**：可以看到从router这里也看到了对应的报文，说明二者之间完成了符合要求双向通信。



task 3:Cached ARP request

Deploying

- 创建默认的Mininet拓扑网络。
- 首先我连续在Mininet中输入两条命令：

```
Mininet> client ping -c 1 10.1.1.2
Mininet> server1 ping -c 1 192.168.100.2
```

- 在router的xterm终端中查看日志：

  可以看到通过运行日志打印出的ARP高速缓存表，共有两个表项，对应了**client和server1发送分组的两个端口的ip地址和mac地址。**

```
(syenv) root@njucs-VirtualBox:~/lab-03-Wumaomaomao# swyard myrouter.py
21:55:28 2022/04/06     INFO Saving iptables state and installing switchyard rul
es
21:55:29 2022/04/06     INFO Using network devices: router-eth2 router-eth1 rout
er-eth0
21:55:38 2022/04/06     INFO ipaddr:10.1.1.1 macaddr:30:00:00:00:00:01 timestamp
:1649253338.2728364
21:55:38 2022/04/06     INFO In njucs-VirtualBox port router-eth2 receive a arp
packet from 10.1.1.1 to 10.1.1.2
21:55:38 2022/04/06     INFO arp reply:Ethernet 40:00:00:00:00:03->30:00:00:00:0
0:01 ARP | Arp 40:00:00:00:00:03:10.1.1.2 30:00:00:00:00:01:10.1.1.1
21:55:47 2022/04/06     INFO ipaddr:10.1.1.1 macaddr:30:00:00:00:00:01 timestamp
:1649253338.2728364
21:55:47 2022/04/06     INFO ipaddr:192.168.100.1 macaddr:10:00:00:00:00:01 time
stamp:1649253347.5515056
21:55:47 2022/04/06     INFO In njucs-VirtualBox port router-eth0 receive a arp
packet from 192.168.100.1 to 192.168.100.2
21:55:47 2022/04/06     INFO arp reply:Ethernet 40:00:00:00:00:01->10:00:00:00:0
0:01 ARP | Arp 40:00:00:00:00:01:192.168.100.2 10:00:00:00:00:01:192.168.100.1
```

- 等待三分钟后，逻辑上之前插入的表项应当已经发生**超时被删除**，所以我输入：

```
Mininet> server2 ping -c 1 192.168.200.2
```

- 这一次日志更新，可以发现只**剩余一个表项**了，也就是刚刚ping的server2的表项，符合预期。

```
21:58:04 2022/04/06     INFO ipaddr:192.168.200.1 macaddr:20:00:00:00:00:01 time
stamp:1649253484.3125992
21:58:04 2022/04/06     INFO In njucs-VirtualBox port router-eth1 receive a arp
packet from 192.168.200.1 to 192.168.200.2
21:58:04 2022/04/06     INFO arp reply:Ethernet 40:00:00:00:00:02->20:00:00:00:0
0:01 ARP | Arp 40:00:00:00:00:02:192.168.200.2 20:00:00:00:00:01:192.168.200.1
^C22:00:39 2022/04/06     INFO Restoring saved iptables state
```

- 另外还有一种情况，ARP发送到子网外，也会更新高速缓存表，这个功能在task2:Testing中有所涉略。

5. 核心代码

task 1:Preperation

task 2:Handle ARP request

```
# myrouter.py
...
# 首先得到路由器的端口IP表以供查询接收的ARP分组判断是否发送到该设备
my_interface = self.net.interfaces()
my_ip = [intf.ipaddr for intf in my_interface]
...
# 从接收的分组中分离出ARP包头
arp = packet.get_header(Arp)
if arp: #首先判断是否为ARP分组，否则不处理
    if arp.operation == 1:#其次判断类型，只接收请求分组
        self.arptable.update_Arptable(arp.senderprotoaddr, arp.senderhwaddr)#对ARP请求的源端口缓存
        if arp.targetprotoaddr in my_ip:#如果ARP分组的目的端口在路由器上部署
            log_info(f"In {self.net.name} port {ifaceName} receive a arp packet from{arp.senderprotoaddr} to {arp.targetprotoaddr}")
            intf = self.net.interface_by_ipaddr(arp.targetprotoaddr)#通过IP地址得到端口对象
            Arp_reply = create_ip_arp_reply(intf.ethaddr, arp.senderhwaddr, intf.ipaddr, arp.senderprotoaddr)#构造ARP回复分组
            log_info(f"arp reply:{Arp_reply}")
            self.net.send_packet(ifaceName, Arp_reply)
```

task 3:Cached ARP request

```
# myrouter.py
#这里主要涉及的是ARP缓存表类的实现
class Arptable(object):
    def __init__(self, table):
        #Arptable类只包括一个固定成员：一个用于存储表格的字典
        self.table = table
    def timeout(self):#超时机制的实现
        delete_key =[];
        timestamp = time.time()
        #遍历字典：找到超时的表项，用list存储key值（这是因为在python里同时遍历dict和删除表项是非法操作）
        for key in self.table:
            if (self.table[key][1] + 10 <= timestamp):
                delete_key.append(key);
         #依次删除
        for key in delete_key:
            self.table.pop(key)

    def update_Arptable(self, ipaddr, macaddr):#更新表项：也是主要被调用的接口
        self.timeout()#首先用超时机制淘汰超时的表项
        timestamp = time.time()
        self.table[ipaddr] = [macaddr, timestamp]#创建新表项，注意加入时间戳
        for key in self.table:#每次更新后利用日志输出现在的表项内容
            log_info(f"ipaddr:{key} macaddr:{self.table[key][0]} timestamp:{self.table[key][1]}")

```

6.总结与感想

- 在本实验中路由器被赋予了**“认识”ARP请求分组并且回复ARP分组**的能力。可以看出ARP协议是三层IP网络重要的辅助协议，实现了三层地址到二层地址的转换。