

# Computer Networks\*

## Lab Report III

LaTeX

\* Teacher: Chen Tian, Wenzhong Li. TAs:

助教-陈衍庆

助教-吴昌容

助教-黄晓洁

助教-段建辉

助教-刘柯鑫

助教-方毓楚

助教-李想

1<sup>st</sup> 张逸凯 171840708 (转专业到计科补课, 非重修)

Department of Computer Science and Technology

Nanjing University

zykhelloha@gmail.com

171840708 张逸凯 Lab 3

### Task 1 Preparation

### Task 2: Handle ARP Requests

Respond to the ARP request 的 Logic

test 的相关情况

自己写test case (附加)

ping Router from another Server1

wireshark 视角

### Task 3: Cached ARP Table

如何construct ARP table

打印ARP table并分析变化

分析ARP表中的表项变化

从my test case角度:

从ping, wireshark角度

Implementation Details (附加)

### Appendix

Key points in Chinese

Lab 3, 4, 5 overview

Details

Notes

ARP Review:

ICMP Review:

Testing Your Code

Your Task

Task 1: Preparation

Task 2: Handle ARP Requests

Procedure

Received ARP Requests

Received Other Packets

Coding

Task 3: Cached ARP Table

Construct cached ARP Table

FAQ

### Summarize the problems:

interface\_by\_ipaddr 方法不友好, 如果找不到则抛出异常

自己构造测试用例时:

在s.expect 中回复的ARP包需要一致

real mode 下报错:

Main experimental steps

real mode

## Task 1 Preparation

```
myrouter.py store_mininet.py
kaik@kai-virtual-machine:~/switchyard/lab_3$ ls
myrouter.py  routertests1full.srpy  routertests1.srpy  start_mininet.py
kaik@kai-virtual-machine:~/switchyard/lab_3$
```

## Task 2: Handle ARP Requests

### Respond to the ARP request 的逻辑

✓ Show how you implement the logic of responding to the ARP request.

```
1  # 以上有外面的while 及其他处理 具体见完整代码...
2  arp = pkt.get_header(Arp)
3
4  # 找目的地IP 对应的 路由器interface:
5  targetIntf = None
6  for i in self.net.interfaces():
7      if arp.targetprotoaddr == i.ipaddr:
8          targetIntf = i
9  if targetIntf != None
10     # APR 请求.
11     if arp.operation == ArpOperation.Request:
12         arpReply = create_ip_arp_reply(targetIntf.ethaddr, arp.senderhwaddr,
13                                         targetIntf.ipaddr, arp.senderprotoaddr)
14         self.net.send_packet(dev, arpReply)
15     elif arp.operation == ArpOperation.Reply:
16         pass
```

如上代码分析：

1. 拿到ARP包header后，搜索路由器的interface有没有 和 目的地的IP地址匹配。
2. 如果这是一个ARP请求，则通过 `create_ip_arp_reply` 构造一个包
3. 从哪里来的，发回哪里去。

总结为：严格按照手册上的要求，在找到对应IP的端口后，实现 `senderhwaddr`，`targethwaddr`，`senderprotoaddr`，`targetprotoaddr` 的顺序构建ARP回复包，对于其他情况的包不做处理(比如找不到对应的相同IP的interface或者非ARP请求包)。

## test 的相关情况

✓ In the report, show the test result of your router.

(Optional) If you have written the test files yourself, show how you test your ARP responding logic.

```
kai@kai-virtual-machine: ~/switchyard/lab_3
File Edit View Search Terminal Help

kai@kai-virtual-machine:~/switchyard/lab_3$ swyard -t routertests1.srpy myrouter.py
11:42:57 2020/03/28      INFO Starting test scenario routertests1.srpy

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!
```

```
kai@kai-virtual-machine: ~/switchyard/lab_3
File Edit View Search Terminal Help

kai@kai-virtual-machine:~/switchyard/lab_3$
kai@kai-virtual-machine:~/switchyard/lab_3$ swyard -t routertests1full.srpy myrouter.py
16:23:49 2020/04/04      INFO Starting test scenario routertests1full.srpy

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!
```

## 自己写test case (附加)

首先我根据助教哥test case的输出还原了所有测试用例：

```
kai@kai-virtual-machine: ~/switchyard/lab_3
File Edit View Search Terminal Help
All tests passed!

kai@kai-virtual-machine:~/switchyard/lab_3$ swyard -t myroutertest.py myrouter.
py
15:38:14 2020/03/28      INFO Starting test scenario myroutertest.py

Results for test scenario switch tests: 6 passed, 0 failed, 0 pending

Passed:
1  (ARP Request) DST IP: 192.168.1.1, arrive on router-eth0
2  (ARP Request) DST IP: 192.168.1.1, response from router-eth0
3  (ICMP NOT Respond) DST IP: 10.10.12.34, arrive on router-
  eth1
4  (ARP NOT Respond) DST IP: 10.10.0.2, arrive on router-eth1
5  (ARP Request) DST IP: 10.10.0.1, arrive on router-eth1
6  (ARP Request) DST IP: 10.10.0.1, response from router-eth1

All tests passed!

kai@kai-virtual-machine:~/switchyard/lab_3$
```

代码以及详解如下：

```
1  # --- lab 3 ---
2  def arpRequestLab3TestCase(s, srcMac, dstMac, srcIp, dstIp, srcInterface):
3      # create_ip_arp_request(senderhwaddr, senderprotoaddr, targetprotoaddr)
4      testpkt = create_ip_arp_request(srcMac, srcIp, dstIp)
5      s.expect(PacketInputEvent(srcInterface, testpkt, display=Ethernet), "
  (ARP Request) DST IP: {}, arrive on {}".format(dstIp, srcInterface))
6
7      # create_ip_arp_reply(senderhwaddr, targethwaddr, senderprotoaddr,
  targetprotoaddr)
8      replypkt = create_ip_arp_reply(dstMac, srcMac, dstIp, srcIp)
9      s.expect(PacketOutputEvent(srcInterface, replypkt, display=Ethernet), "
  (ARP Request) DST IP: {}, response from {}".format(dstIp, srcInterface))
10
11
12  def notRespondLab3TestCase(s, srcMac, dstMac, srcIp, dstIp, srcInterface,
  isArpRqst=False):
13      if isArpRqst == False:
14          testpkt = mk_pkt(srcMac, dstMac, srcIp, dstIp)
15          s.expect(PacketInputEvent(srcInterface, testpkt, display=Ethernet),
  "(ICMP NOT Respond) DST IP: {}, arrive on {}".format(dstIp, srcInterface))
16
17      else:
18          testpkt = create_ip_arp_request(srcMac, srcIp, dstIp)
19          s.expect(PacketInputEvent(srcInterface, testpkt, display=Ethernet),
  "(ARP NOT Respond) DST IP: {}, arrive on {}".format(dstIp, srcInterface))
```

上面是封装了的测试函数，分别实现了

- 判断ARP请求是否正常进入并回复
- 判断无关的ICMP包是否被drop
- 判断找不到的ARP请求是否被drop（在目前还不需要处理这个ARP包）

调用方法如下：

结合上面的测试输出图，我们可以证明这是正确的。

```
1  def router_tests():
```

```

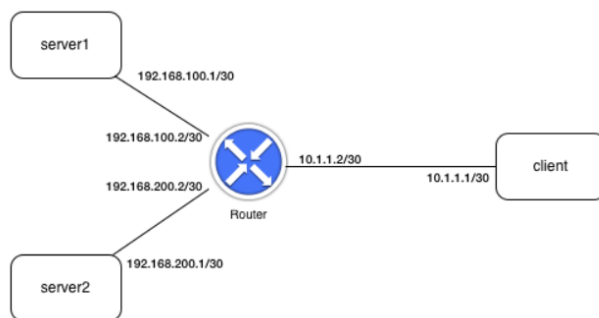
2     s = TestScenario("switch tests")
3     s.add_interface('router-eth0', '10:00:00:00:00:01',
ipaddr='192.168.1.1')
4     s.add_interface('router-eth1', '10:00:00:00:00:02', ipaddr='10.10.0.1')
5
6     host = [
7         ["20:00:00:00:00:01", "192.168.1.205", "192.168.1.1"],
8         ["20:00:00:00:00:02", "192.168.1.206", "10.10.0.2"],
9         ["30:00:00:00:00:01", "192.168.1.207", "10.10.0.1"]
10    ]
11
12    arpRequestLab3TestCase(s, host[0][0], '10:00:00:00:00:01', host[0][1],
host[0][2], 'router-eth0')
13    notRespondLab3TestCase(s, host[1][0], '10:00:00:00:00:02', host[1][1],
'10.10.12.34', 'router-eth1')
14    notRespondLab3TestCase(s, host[1][0], '10:00:00:00:00:02', host[1][1],
host[1][2], 'router-eth1', isArpRqst=True)
15    arpRequestLab3TestCase(s, host[2][0], '10:00:00:00:00:02', host[2][1],
host[2][2], 'router-eth1')
16
17    return s

```

## ping Router from another Server

So here is our example. Your task is: ping the router from another host (server1 or server2). Using Wireshark to prove that you have handled ARP requests well. ☒ Write the procedure and analysis in your report with screenshots.

这里我用server来ping router:



由上图可知我们需要在server1视角:

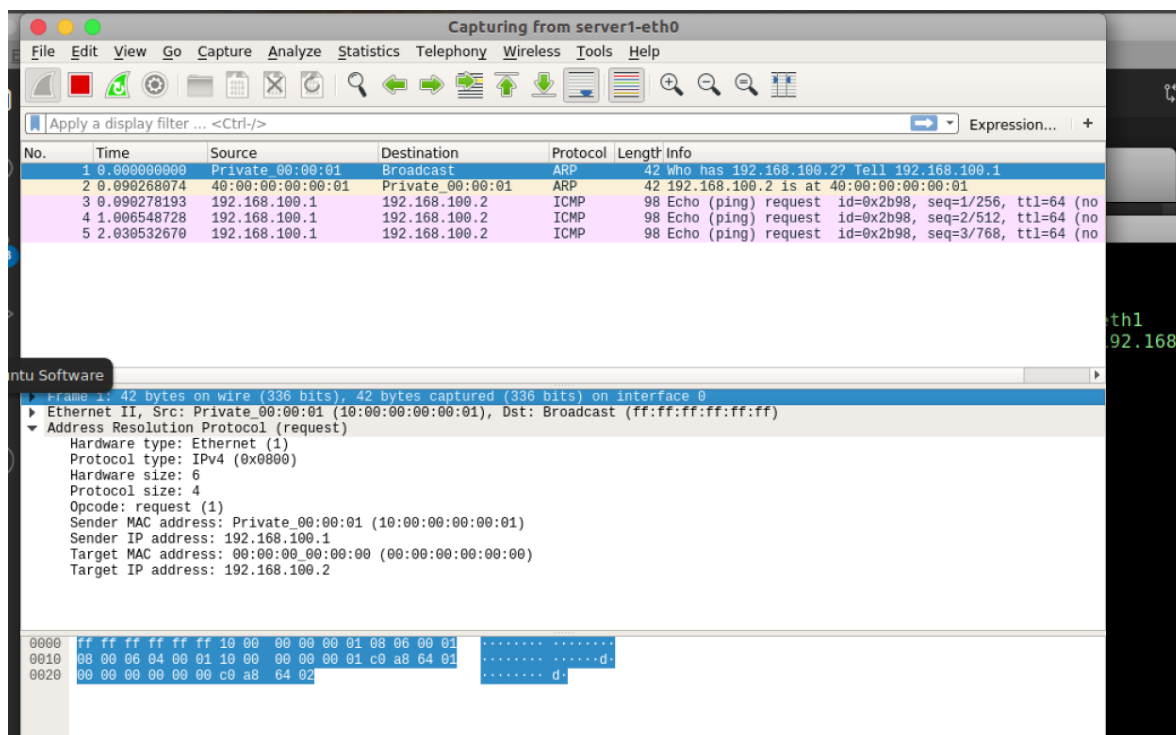
```
1 | ping -c3 192.168.100.2
```

回复的ARP包应为:

- source IP, MAC: 路由器192.168.100.2 及此端口的MAC.
- destination IP, MAC: server1中192.168.100.1 及其MAC.

.

wireshark 视角



## Task 3: Cached ARP Table

### 如何construct ARP table

✅ In your report, show how you construct the ARP table.

建立ARP table可以根据手册的要求，使用dict数据结构，代码以及分析如下：

```

1  # 在Router类中定义 arpTable 字典结构：
2  def __init__(self, net):
3      self.net = net
4      # other initialization stuff here
5      self.arpTable = {}
6
7  # 在路由器核心结构中 记录并更新ARP表。
8  # 如果之前已经存在了这个表项，就进行更新。
9  if arp.operation == ArpOperation.Request:
10     # 发包相关 ...
11     arpReply = create_ip_arp_reply(targetIntf.ethaddr, arp.senderhwaddr,
12     targetIntf.ipaddr, arp.senderprotoaddr)
13     self.net.send_packet(dev, arpReply)
14     # 更新表项：
15     self.arpTable[targetIntf.ipaddr] = targetIntf.ethaddr
16 elif arp.operation == ArpOperation.Reply:
17     self.arpTable[arp.senderprotoaddr] = arp.senderhwaddr

```

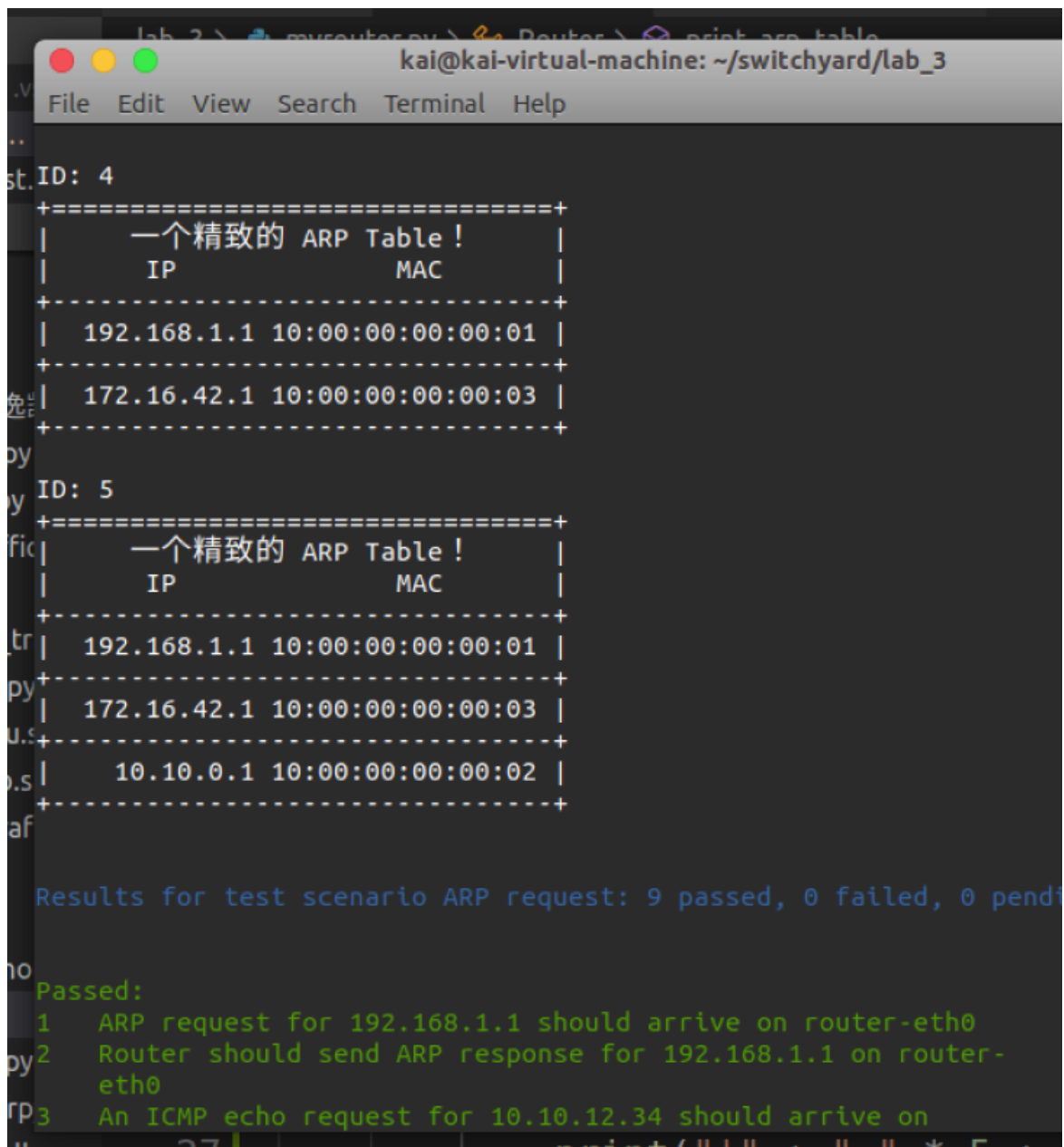
### 打印ARP table并分析变化

✅ In your report, show the cached ARP table with screenshots. Explain how entries have changed.

使用助教哥推荐的 面向对象 方法创建类成员函数，封装打印ARP表的过程：

```
1 def print_arp_table(self):
2     if self.arpTable:
3         print("=" * 32 + "=")
4         print("|" + " " * 5 + "一个精致的 ARP Table!" + " " * 5 + "|")
5         print("|          IP          MAC          |")
6         print("=" * 32 + "=")
7
8         for ip, mac in self.arpTable.items():
9             print("|", str(ip).rjust(12, ' '), str(mac), "|")
10            print("=" * 32 + "=")
11        print("")
```

运行结果：



```
lab 3 \ myrouter.py \ Router \ print_arp_table
kai@kai-virtual-machine: ~/switchyard/lab_3
File Edit View Search Terminal Help
st.ID: 4
=====+
|          一个精致的 ARP Table!          |
|          IP          MAC          |
+-----+
| 192.168.1.1 10:00:00:00:00:01 |
+-----+
| 172.16.42.1 10:00:00:00:00:03 |
+-----+
免
py
y ID: 5
=====+
|          一个精致的 ARP Table!          |
|          IP          MAC          |
+-----+
| 192.168.1.1 10:00:00:00:00:01 |
+-----+
| 172.16.42.1 10:00:00:00:00:03 |
+-----+
U.s.+
o.s| 10.10.0.1 10:00:00:00:00:02 |
+-----+
af

Results for test scenario ARP request: 9 passed, 0 failed, 0 pending
no
Passed:
1 ARP request for 192.168.1.1 should arrive on router-eth0
py2 Router should send ARP response for 192.168.1.1 on router-eth0
rp3 An ICMP echo request for 10.10.12.34 should arrive on
ll raw 27 | print("|" + " " * 5 + "|")
```

## 分析ARP表中的表项变化



## 从my test case角度:

像上一节一样编写自己的测试用例:

```
You, 7 days ago • Update at 2020-3-28 19:13:57 finished task 1
arpRequestLab3TestCase(s, host[0][0], '10:00:00:00:00:01', host
[0][1], host[0][2], 'router-eth0')

notRespondLab3TestCase(s, host[1][0], '10:00:00:00:00:02', host
[1][1], '10.10.12.34', 'router-eth1')

notRespondLab3TestCase(s, host[1][0], '10:00:00:00:00:02', host
[1][1], host[1][2], 'router-eth1', isArpRqst=True)

arpRequestLab3TestCase(s, host[2][0], '10:00:00:00:00:02', host
[2][1], host[2][2], 'router-eth1')

return s
```

```
kai@kai-virtual-machine: ~/switchyard/lab_3
File Edit View Search Terminal Help

kai@kai-virtual-machine:~/switchyard/lab_3$ swyard -t myroutertest.py myrouter.p
y
19:27:31 2020/04/04      INFO Starting test scenario myroutertest.py
ID: 0
=====+
|      一个精致的 ARP Table !      |
|      IP              MAC          |
|-----+-----+
|      192.168.1.1  10:00:00:00:00:01 |
|-----+-----+

ID: 1
=====+
|      一个精致的 ARP Table !      |
|      IP              MAC          |
|-----+-----+
|      192.168.1.1  10:00:00:00:00:01 |
|-----+-----+

ID: 2
=====+
|      一个精致的 ARP Table !      |
|      IP              MAC          |
|-----+-----+
|      192.168.1.1  10:00:00:00:00:01 |
|-----+-----+
|      10.10.0.1   10:00:00:00:00:02 |
|-----+-----+
```



```
Results for test scenario switch tests: 6 passed, 0 failed, 0 pending

Passed:
1 (ARP Request) DST IP: 192.168.1.1, arrive on router-eth0
2 (ARP Request) DST IP: 192.168.1.1, response from router-eth0
3 (ICMP NOT Respond) DST IP: 10.10.12.34, arrive on router-eth1
4 (ARP NOT Respond) DST IP: 10.10.0.2, arrive on router-eth1
5 (ARP Request) DST IP: 10.10.0.1, arrive on router-eth1
6 (ARP Request) DST IP: 10.10.0.1, response from router-eth1

All tests passed!

kai@kai-virtual-machine:~/switchyard/lab_3$
```

这里使用的正是上一节我 仿制助教的测试文件写的 自己的测试用例；测试过程是基本一致的。

我们可以发现：

- 第一个需要回复的ARP包经过之后(ID: 0)，在ARP表中建立了一个表项。
- 然后来了一个不需要回复的ICMP包，此时因为不是ARP包，不输出。
- 然后来了一个不需要回复的ARP包，此时当前表输出(ID: 1)。
- 然后来了一个需要回复的ARP包，并且与之前不同(不是更新，而是添加)，所以输出了添加新表项的表(ID: 2)。

## 从ping, wireshark角度

同手册中的例子，使用命令：

```
1 | client ping -c3 10.1.1.2
```

首先我们来看real mode中站在router视角的表项输出：

```
root@kai-virtual-machine:~/switchyard/lab_3# source ../syenv/bin/activate
(syenv) root@kai-virtual-machine:~/switchyard/lab_3# swyard myrouter.py
17:58:04 2020/04/04 INFO Saving iptables state and installing switchyard rules
17:58:04 2020/04/04 INFO Using network devices: router-eth2 router-eth1 router-eth0
ID: 0
+=====+
| 一个精致的 ARP Table! |
| IP MAC |
+-----+
| 10.1.1.2 40:00:00:00:00:03 |
+-----+
```

这是正确的，因为在请求ARP时直接发给了路由，路由就记录下这个表项。

接下来我们键入：

```
1 | server1 ping -c3 client
```

```
ID: 25
+=====+
| 一个精致的 ARP Table! |
| IP MAC |
+-----+
| 10.1.1.2 40:00:00:00:00:03 |
+-----+
| 192.168.100.2 40:00:00:00:00:01 |
+-----+
```

因为此时server1对于目的地的MAC仍然是不知道的，属于第一次发包，所以ARP表会被加入一个新表项。

当我再次键入：

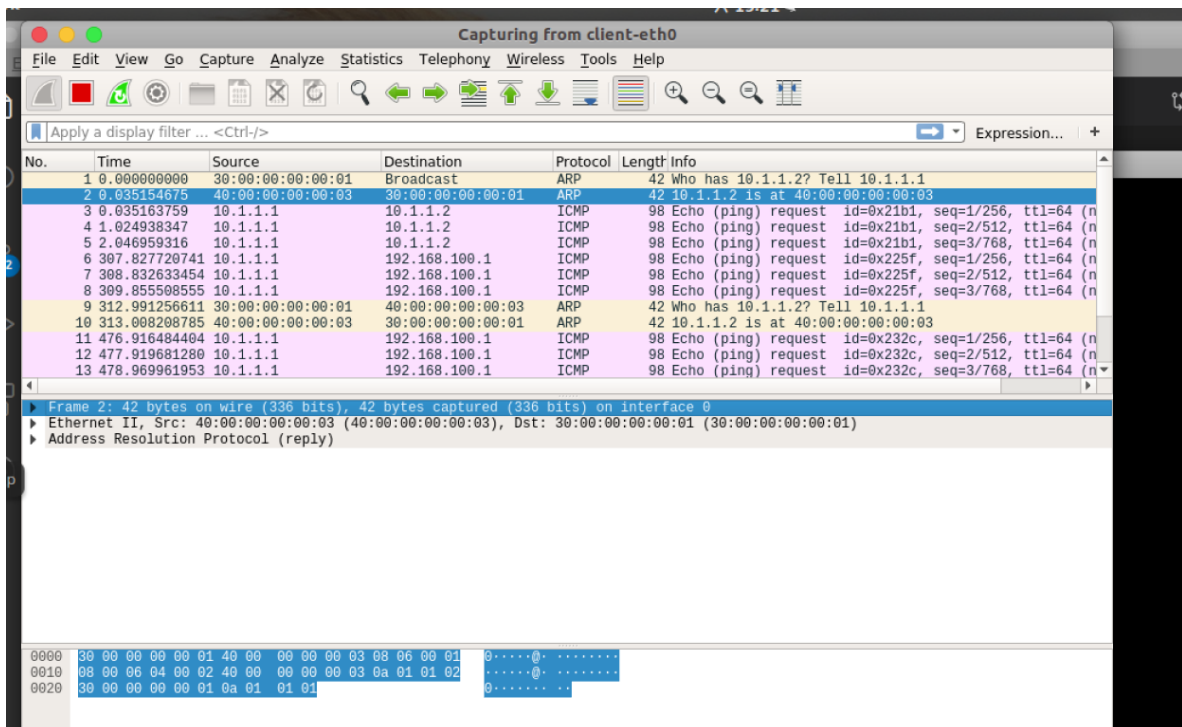
```
1 | server1 ping -c3 client
```

我们发现ARP包不再更新，再没有任何输出，这说明了它已经收到了ARP回复包，知道MAC了。

```
| 192.168.100.2 40:00:00:00:00:01 |
+-----+
^C19:12:05 2020/04/04      INFO Restoring saved iptables state

(syenv) root@kai-virtual-machine:~/switchyard/lab_3# swyard myrouter.py
19:15:44 2020/04/04      INFO Saving iptables state and installing switchyard rules
19:15:44 2020/04/04      INFO Using network devices: router-eth1 router-eth2 router-eth0
```

wireshark中的结果与上一节是一致的：



## Implementation Details (附加)

查文档看函数，查文档非常关键。

<code>add_header(ph)</code>	<a href="#">[source]</a>
Add a PacketHeaderBase derived class object, or a raw bytes object as the next "header" item in this packet. Note that 'header' may be a slight misnomer since the last portion of a packet is considered application payload and not a header per se.	
<code>add_payload(ph)</code>	<a href="#">[source]</a>
Alias for add_header	
<code>static from_bytes(raw, first_header)</code>	<a href="#">[source]</a>
Create a new packet by parsing the contents of a bytestring	
<code>get_header(hdrclass, returnval=None)</code>	<a href="#">[source]</a>
Return the first header object that is of class hdrclass, or None if the header class isn't found.	
<code>get_header_by_name(hdrname)</code>	<a href="#">[source]</a>
Return the header object that has the given (string) header class name. Returns None if no such header exists.	
<code>get_header_index(hdrclass, startidx=0)</code>	<a href="#">[source]</a>
Return the first index of the header class hdrclass starting at startidx (default=0), or -1 if the header class isn't found in the list of headers.	
<code>has_header(hdrclass)</code>	<a href="#">[source]</a>
Return True if the packet has a header of the given hdrclass, False otherwise.	
<code>headers()</code>	<a href="#">[source]</a>
Return a list of packet header names in this packet.	
<code>insert_header(idx, ph)</code>	<a href="#">[source]</a>
Insert a PacketHeaderBase-derived object at index idx the list of headers. Any headers previously in the Packet from index idx:len(ph) are shifted to make room for the new packet.	
<code>num_headers()</code>	<a href="#">[source]</a>
Return the number of headers in the packet.	
<code>prepend_header(ph)</code>	<a href="#">[source]</a>
Insert a PacketHeader object at the beginning of this packet (i.e., as the first header of the packet).	

## Appendix

以下是本次实验过程中不需要展示在实验报告中，但对我本次实验至关重要，方便以后复习的内容。

### Key points in Chinese

#### Lab 3, 4, 5 overview

既然您已经构建了一个简单的学习型以太网交换机，并且对Switchyard框架更加满意，那么您将可以使用它来做更多更酷的事情。在从实验3到实验5的分配中，您将完成一系列任务，最终创建一个功能齐全的IPv4路由器。从总体上讲，您的路由器将具有以下功能：

- 响应/发出**ARP**请求
- 接收数据包，并使用查找表将其转发到目的地
- 响应/生成**ICMP**消息

#### Details

为了创建具有上述功能的酷路由器，您将实现5个主要功能：

1. 响应对分配给路由器接口的地址的**ARP**（地址解析协议）请求。
2. 对没有已知以太网**MAC**地址的**IP**地址发出**ARP**请求。路由器通常将必须向其他主机发送数据包，并且需要以太网**MAC**地址来完成。
3. 接收和转发到达链接并发往其他主机的数据包。转发过程的一部分是在转发信息库中执行地址查找（"最长前缀匹配"查找）。您最终将只在路由器中使用"静态"路由，而不是实现诸如**RIP**或**OSPF**之类的动态路由协议。
4. 响应**Internet**控制消息协议（**ICMP**）消息，例如回声请求（"**ping**"）。
5. 必要时（例如，当**IP**数据包的**TTL**（生存时间）值减小到零时）生成**ICMP**错误消息。

您可以在以下实验任务中找到有关这些功能的更多详细信息：

实验3：完成项目1。

实验4：完成项目2和项目3。

实验5：完成项目4和项目5。

## Notes

### ARP Review:

ARP是用于将IP地址解析为MAC地址的协议。主要问题是，尽管IP地址用于跨网络转发IP数据包，但在特定的物理网络中仍需要将数据包发送到主机或路由器的链接级地址。因此，网络中的主机需要保持IP与链路层地址之间的映射。主机可以使用ARP在其物理网络中广播针对特定IP地址的查询消息，以便适当的主机可以使用其链路层地址来答复该查询。

### ICMP Review:

ICMP是允许路由器密切监视Internet运行的主要协议之一。网络设备（例如路由器）使用ICMP消息发送错误消息以指示各种问题，例如无法访问的目标主机/网络或数据包的TTL过期。ping是一种非常常用的网络管理实用程序，它使用ICMP Echo请求/应答数据包来验证主机的可达性，并收集有关网络状态的信息（例如，平均RTT，数据包丢失百分比等）。

## Testing Your Code

就像之前的任务一样，您应该通过编写自己的测试用例来测试实现的正确性。作为您的友好TA，我应该警告您，与上一个任务相比，在此任务中您将实现更多功能，并且路由器将同时处理更多事件。因此，您应该考虑测试实施正确性的合理方法。您可以做的一件事是创建仅测试某些功能的测试分开的测试用例。创建仅测试某些功能的测试分开的测试用例。创建仅测试某些功能的测试分开的测试用例。这将使您查看路由器的各个部分是否正常工作。然后，您可以生成更大的测试用例，以确保单独的模块可以正常工作而不会相互破坏。与往常一样，不要忘记考虑极端情况。

## Your Task

在此练习的源目录中，有一个Python文件可用作入门模板：`myrouter.py`。该文件包含Router类的概述，并且当前包含构造函数（`__init__`）和`router_main`方法。这只是一个入门模板：您可以按照自己喜欢的任何方式重构和重新设计代码。

本练习的主要任务是修改Router类，完成任务。

## Task 1: Preparation

本节不难看懂。

注意：所有修改都应该在lab\_3目录下的文件上进行。我们会检查和比较git的承诺，以判断你的作品的原创性。所以记住，每次完成一项小任务时，都要全力以赴。

## Task 2: Handle ARP Requests

### Procedure

当收到一个数据包时，判断它是否是一个ARP请求。

### Received ARP Requests

对于ARP数据包标头，有两种地址类型，以及每种地址类型的源和目的地，总共有四个地址。对于一个ARP请求，需要填写源以太网IP地址以及目标IP地址。请注意，源IP地址和目标IP地址在ARP头中分别称为`senderprotoaddr`和`targetprotoaddr`。源和目标以太网地址分别称为`senderhwaddr`和`targethwaddr`。目标以太网地址未填写：因为这是所请求的地址。

请注意，ARP的数据包头类名为`Arp`，因此要从传入数据包（如果存在）中获取ARP头，您可以执行以下操作：

```
1 | arp = packet.get_header(Arp)
```

对于每个ARP请求，您应该确定ARP头中的 `targetprotoaddr` 字段（目的地IP地址）是否是分配给路由器上接口之一的IP地址。

请记住，可以通过在Router类的 `self.net` 属性中存储的网络对象上调用 `interfaces` 方法（或等效地调用 `ports` 方法）来获取为路由器配置的接口的列表。还要注意，您可以按以太网地址，名称或IP地址 查找端口。请参阅文档以了解适当的方法。

如果目的地IP地址是分配给路由器接口的IP地址，则应创建并发送适当的ARP答复。（如果未将目标IP地址分配给路由器的一个接口，则即使您具有足够的信息，也不应使用ARP答复进行响应。）ARP应答发送的接口应该与ARP请求到达的接口相同。

Switchyard文档详细说明了接口方法返回的内容。您可能希望在Router类的构造函数中调用这个方法，并为Router创建一些内部数据结构，以便它能够跟踪自己的接口。

## Received Other Packets

如果您在路由器中收到的数据包不是ARP请求，则暂时应将其忽略（丢弃）。在以后的练习中，您将在路由器中处理更多的传入数据包类型。

## Coding

Switchyard文档中的一节介绍了数据包解析和构造的工作方式。强烈建议您阅读背景知识。您可能还会发现该API参考对数据包解析很有帮助。您还可以使用两个帮助器功能（在 `switchyard.lib.packet` 中定义，该功能已在模板文件中导入）。

- `create_ip_arp_reply(senderhwaddr, targethwaddr, senderprotoaddr, targetprotoaddr)`
- `create_ip_arp_request(senderhwaddr, senderprotoaddr, targetprotoaddr)`

注意，上面的这两个函数返回一个完整的包对象，包括以太网和Arp报头，都是填好的。

## Task 3: Cached ARP Table

### Construct cached ARP Table

您最终需要在路由器中存储目标IP地址和以太网MAC地址之间的映射(您可以假设存在一对一的映射)。原因很简单：当您将一个IP包发送到另一个主机时，您还需要与目标IP地址相关联的以太网地址。如果您"记住"在路由器上接收到的来自ARP请求的任何源IP/以太网MAC对，它可能会帮助您避免构造和发送一个ARP请求来获取相同的信息。这种功能可能有助于在将来构建路由器的阶段中使用。缓存表类似于以太网学习交换机中使用的表。对于每个条目，都有两个或多个字段需要IP和MAC地址。缓存的ARP表可能看起来像：

IP	MAC Address
10.1.2.3	01:02:03:04:05:06
...	...

当路由器收到带有ARP标头的数据包时，请添加或更新缓存的ARP表的条目。例如，如果存在一个以以太网源地址为 `01:02:03:04:05:06`，且IP源地址为 `10.1.2.3` 的ARP请求，则路由器的ARP表可以建立 (IP, MAC) 的条目，您还可以在此表中看到IP地址是唯一的。

通常，ARP表具有超时机制。它指示ARP缓存中的MAC地址可以驻留的时间。在此任务中，实现此机制是可选的。

## FAQ



问：路由器在以下情况下应该做什么：某个IP地址的数据包到达路由器，并发送ARP请求以查找MAC地址。在收到ARP答复之前，路由器会收到另一个具有相同IP地址的数据包（非ARP），是否再次发送ARP请求？就是一个ARP请求还在路上，另一个同样IP地址的包来了。

答：否，在这种情况下，您不会重新传输第二个数据包的ARP请求。更一般而言，当某个IP地址有未解决的ARP请求时，您的路由器可能会收到许多有关该IP地址的数据包。在这种情况下，您的路由器不应发送任何新的ARP请求或更新初始ARP请求的时间戳。但是，您的路由器应该缓冲新的数据包，以便一旦收到ARP答复就可以将它们传输到目标主机。重要的是：如果您的路由器为具有未解决ARP请求的目标主机缓冲了多个数据包，则在收到相应的ARP答复后，这些数据包必须按照它们到达路由器的顺序转发到目标主机。

问：当ARP请求到达路由器时，该IP地址不对应路由器的任何一个接口，则路由器是否需要泛洪ARP请求？还是丢弃它？

答：在这种情况下，您的路由器应丢弃数据包。注意：在第1阶段，您应该检查目标IP地址是否是分配给路由器接口之一的IP地址。但是，对于第二阶段和第三阶段，请检查目标IP地址是否为ARP表中的IP地址。

问：（不是在本次lab中涉及）...

问：部分指令说明该数据包是否适合我们（即，目标 `ipaddr` 位于 `net.interfaces` 中），然后将其丢弃。但是后来，指令说如果该数据包是给我们直接连接的邻居之一的，那么我们将其转发。

`net.interfaces()` 与直接连接的网络有何不同？

答：`net.interfaces()` 是路由器本身的接口信息。例如，`eth1`的 `ipaddr` 可能为 `192.168.1.1`。但是，您可以从这些接口使用 `ipaddr` 和 `netmask` 获得接口直接连接到的子网。因此，如果相同的 `eth1` 的网络掩码为 `255.255.255.0`，则子网将是应用于 `ipaddr` 的掩码，即 `192.168.1.0`。因此，如果到达路由器的数据包以 `192.168.1.100` 之类的IP地址为目的地址，则您将在转发表中找到匹配项，最终将数据包转发出 `eth1`。但是，如果到达路由器的数据包发往 `192.168.1.1`，则您将其丢弃，因为这是接口的确切IP地址。

## Summarize the problems:

---

`interface_by_ipaddr` 方法不友好, 如果找不到则抛出异常

```
interface_by_ipaddr(ipaddr)
```

Given an IP address, return the interface that 'owns' this address

我们当然可以用 `catch` 来捕获这个异常，但是由OS的知识知道这样会降低速度。

解决方法就是不用它，使用 `for-loop` 遍历所有 `interface`，判断是不是和目的地的IP相同。

### 自己构造测试用例时:

一定要封装，可以节省很多代码量。

在 `s.expect` 中回复的ARP包需要一致

```
(dstIp, srcInterface))
48 kai@kai-virtual-machine: ~/switchyard/lab_3
49 File Edit View Search Terminal Help
This is the Switchyard equivalent of the blue screen of death.
As far as I can tell, here's what happened:

Expected event:
  (ARP Request)IP: 192.168.1.1, response from router-eth0

Failure observed:
  You called send_packet and while the output port router-eth0
  is ok, an exact match of packet contents failed. In the
  Ethernet header, dst is wrong (is 20:00:00:00:00:01 but
  should be ff:ff:ff:ff:ff:ff); src is wrong (is
  10:00:00:00:00:01 but should be 20:00:00:00:00:01), In the
  Arp header, operation is wrong (is 2 but should be 1);
  senderhwaddr is wrong (is 10:00:00:00:00:01 but should be
  20:00:00:00:00:01); senderprotoaddr is wrong (is 192.168.1.1
  but should be 192.168.1.205); targetethwaddr is wrong (is
  20:00:00:00:00:01 but should be ff:ff:ff:ff:ff:ff);
  targetprotoaddr is wrong (is 192.168.1.205 but should be
  192.168.1.1).
```

real mode 下报错:

```
17081986 30:00:00:00:00:01
root@kai-virtual-machine:~/switchyard/lab_3# source ../syenv/bin/activate
(syenv) root@kai-virtual-machine:~/switchyard/lab_3# swyard myrouter.py
15:50:41 2020/03/28 INFO Saving iptables state and installing switchyard rules
15:50:41 2020/03/28 INFO Using network devices: router-eth2 router-eth0 router-eth1
^C15:50:57 2020/03/28 INFO Restoring saved iptables state
(syenv) root@kai-virtual-machine:~/switchyard/lab_3# swyard myrouter.py
15:54:42 2020/03/28 INFO Saving iptables state and installing switchyard rules
15:54:43 2020/03/28 INFO Using network devices: router-eth1 router-eth2 router-eth0
15:56:44 2020/03/28 CRITICAL Exception while running your code: 'str' object has no attribute 'format'

*****
This is the Switchyard equivalent of the blue screen of death.
Here (repeating what's above) is the failure that occurred:

Traceback (most recent call last):
  File "/home/kai/switchyard/syenv/lib/python3.6/site-packages/switchyard/llnetreal.py", line 272, in main_real
    start_usercode(usercode_entry_point, netobj, options.codearg)
AttributeError: 'str' object has no attribute 'format'
*****

I'm throwing you into the Python debugger (pdb) at the point of failure.
If you don't want pdb, use the --nopdb flag to avoid this fate.

> /home/kai/switchyard/lab_3/myrouter.py(51)router_main()
-> print("Request create_ip_arp_reply: ({}, {}, {}, {})".format(targetIntf.ethwaddr, arp.senderhwaddr, targetIntf.ipaddr,
  arp.senderprotoaddr))
```

是 str 的 format 的错误。

## Main experimental steps

### real mode

开启mininet:

```
1 | sudo python start_mininet.py
```

```
1 | mininet> xterm switch
2 |
3 | 在 server1 结点上开启wireshark(蹲着端口)
4 | mininet> server1 wireshark &
```

mininet经过switch xterm中开启的py文件, 例如:



```
1 | xterm中:
2 | 激活环境
3 | swyard myswitch_to.py
```

之后在 mininet CLI> ping 等操作即经过myswitch\_to.

## *test mode*

```
1 | 终端:
2 | swyard -t switchtests_to.srpy myswitch_to.py
```

## Summary

---

还是有一些难点的，做出来就很开心，计网实验，有趣!!!

哈哈这次实验似乎比较简单，我做得比较好的应该是封装了自己测试用例函数(测试过程分析完全)，用了面向对象思想，输出精致的**ARP**表。

谢谢助教哥的批改！辛苦了！我会继续努力的！😊