Lab 4: Forwarding Packets

191220138 杨飞洋

实验目的

学习路由器的工作原理，并通过代码实现其中的一部分。

具体功能为：接收并转发那些会被发送给其他主机的数据包，对没有已知mac地址的ip地址发出ARP请求。

背景知识

interface接口

class switchyard.lib.interface.**Interface**(*name*, *ethaddr*, *ipaddr=None*, *netmask=None*, *ifnum=None*, *iftype=<InterfaceType.Unknown: 1>*) ¶

ethaddr
Get the Ethernet address associated with the interface

ifnum
Get the interface number (integer) associated with the interface

iftype
Get the type of the interface as a value from the InterfaceType enumeration.

ipaddr
Get the IPv4 address associated with the interface

ipinterface
Returns the address assigned to this interface as an IPInterface object. (see documentation for the built-in ipaddress module).

name
Get the name of the interface

netmask
Get the IPv4 subnet mask associated with the interface

ARP接口

ARP (address resolution protocol) header

class switchyard.lib.packet.**Arp**(***kwargs*)

hardwaretype

operation

protocoltype

senderhwaddr

senderprotoaddr

targethwaddr

targetprotoaddr

IPv4接口

IP version 4 header

class switchyard.lib.packet.**IPv4**(***kwargs*) [source]

Represents an IP version 4 packet header. All properties relate to specific fields in the header and can be inspected and/or modified.

Note that the field named "hl" ("h-ell") stands for "header length". It is the size of the header in 4-octet quantities. It is a read-only property (cannot be set).

Note also that some IPv4 header option classes are available in Switchyard, but are currently undocumented.

dscp

dst

ecn

flags

fragment_offset

hl

ipid

options

protocol

src

tos

total_length

ttl

address接口

```
from switchyard.lib.address import *
netaddr = IPv4Network('172.16.0.0/255.255.255.0')
netaddr.prefixlen # -> 24
```

Ethernet接口

```
ether = Ethernet()
ether.src = srchw
ether.dst = 'ff:ff:ff:ff:ff:ff'
ether.ethertype = EtherType.ARP
arp = Arp(operation=ArpOperation.Request,
          senderhwaddr=srchw,
          senderprotoaddr=srcip,
          targethwaddr='ff:ff:ff:ff:ff:ff',
          targetprotoaddr=targetip)
arppacket = ether + arp
```

packet接口

get_header(*hdrclass*, *returnval=None*)

Return the first header object that is of class hdrclass, or None if the header class isn't found.

get_header_by_name(*hdrname*)

Return the header object that has the given (string) header class name. Returns None if no such header exists.

get_header_index(*hdrclass*, *startidx=0*)

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.

has_header(*hdrclass*)

Return True if the packet has a header of the given hdrclass, False otherwise.

实现逻辑

IP Forwarding Table Lookup

构建forwarding table

首先需要创建`forwarding table`，我采用的是`list`结构，在`router`类中定义为`self.ftable`，`list`中的每个元素为`[IPv4Network,IPv4Address:next_ip,interface.name]`，其创建过程定义在`router`类中的`build()`函数中。

信息来源有2个：

1.net.interface()中自带的信息

可以用`interface`接口中的`ipaddr`和`netmask`获取接口的ip和子网掩码，再用`IPv4Network`的构造函数就可以生成一个`IPv4Network`的对象。由于不知道`next_ip`，就设置为`0.0.0.0`，interface.name天然就有。

代码如下：

```
for intf in self.interfaces:
    p = []
    p.append(IPv4Network(str(intf.ipaddr) + '/' + str(intf.netmask),False))
    p.append(IPv4Address("0.0.0.0"))
    p.append(intf.name)
    self.ftable.append(p)
```

2.forwarding_table.txt

txt文件的每一行记录了ip，子网掩码，next_ip，interface.name

可用python中的`open()`方法读取文件，再用`split()`方法分割字符串，还要处理一下行末的回车，可以用字符串切片解决。

代码如下：

```
with open("forwarding_table.txt","r") as f:
    for item in f:
        p = []
        l = len(item) - 1
        if(item[l] == '\n'):
            item = item[:l]
        data = item.split(" ",4)
        p.append(IPv4Network(data[0] + "/" + data[1]))
        p.append(IPv4Address(data[2]))
        p.append(data[3])
        self.ftable.append(p)
```

匹配目的地 IP 地址与转发表

路由器接收的 IP 数据包中的目的地地址应与转发表匹配，如果表中的两个项目匹配，应使用最长的前缀匹配。

如果表中没有匹配，只需丢弃数据包。如果数据包是路由器本身（即，目的地地是路由器的接口之一的地址），也删除/忽略包。

最长前缀匹配，因为使用了`IPv4Network`类，可以用`in`和`prefixlen`来进行最长前缀匹配，定义在`router`类中的`match()`函数中。

该函数返回匹配结果在`forwarding_table`中的索引，如果没有匹配则返回-1，代码如下：

```
def match(self,ipaddr):
    index = -1
    length = 0
    ipaddrs = [intf.ipaddr for intf in self.interfaces]
    if ipaddr not in ipaddrs:
        i = 0
        while i < len(self.ftable):
            if ipaddr in self.ftable[i][0] and self.ftable[i][0].prefixlen >
length:
                index = i
                length = self.ftable[i][0].prefixlen
            i = i + 1
    return index
```

Forwarding the Packet and ARP

发送ARP请求和转发数据包

要转发数据包，需要配置`packet`中的`ethernet`属性，主要是要找到目的mac地址

首先在查找转发表之前将IP包的ttl减1，如果没有查找结果，直接pass。

查找到结果之后，如果`next_ip = 0.0.0.0`，则将该`packet`中的`ipv4.dst`赋值给`next_ip`，如果`next_ip`在`arp_table`中，则可以直接找到相对应的mac地址，利用`send_packet()`函数将其发出去。如果不在，则将其放进等待队列中。

这里说明一下等待队列，我没有新建其他的对象，就还是以`list`形式存在`router`类中，命名为`queue`，其元素形式为`[packet,interface,next_ip,time,num]`，其中`num`记录发了多少次请求，`time`记录时间。

上述逻辑记录在`handle_packet()`函数中，部分代码为：

```
packet[ipdx].ttl -= 1
ipv4 = packet[ipdx]
index = self.match(ipv4.dst)
if index == -1:
    pass
else:
    next_ip = self.ftable[index][1]
    intfname = self.ftable[index][2]
    intf = self.net.interface_by_name(intfname)
    if next_ip == IPv4Address('0.0.0.0'):
        next_ip = ipv4.dst
    if next_ip == intf.ipaddr:
        pass
    else:
        if next_ip in self.arptable.keys():
            mac = self.arptable[next_ip]
            eth_index = packet.get_header_index(Ethernet)
            packet[eth_index].src = intf.ethaddr
```

```
            packet[eth_index].dst = mac
            self.net.send_packet(intfname, packet)
        else:
            self.insert(packet,intf,next_ip,time.time())
```

那该如何找到目的mac地址呢？需要向外发送ARP请求，定义在`forward()`函数中。

对于队列中的元素：

1.如果`arp_table`有所更新，能够找到对应的mac地址，则将数据包发送出去，并且从等待队列中移除。

2.如果发送的请求次数 >= 5次，直接丢弃。

3.否则如果距离上一次发送时间超过1s，向外发送一个arp请求，关于构造arp请求的方式，在背景知识：Ethernet接口中，有相关的示例代码，可以沿用，注意`targethwaddr`为`'ff:ff:ff:ff:ff:ff'`，因为是向外广播。最后还是用`send_packet()`方法发出，代码如下：

```
def forward(self):
        for item in self.queue:
            pkt = item[0]
            intf = item[1]
            next_ip = item[2]
            t = item[3]
            num = item[4]
            if next_ip in self.arptable.keys():
                eth_index = pkt.get_header_index(Ethernet)
                pkt[eth_index].dst = self.arptable[next_ip]
                self.net.send_packet(intf.name,pkt)
                self.queue.remove(item)
            elif time.time() - t > 1.0:
                if num >= 5:
                    self.queue.remove(item)
                else:
                    ether = Ethernet()
                    ether.src = intf.ethaddr
                    ether.dst = "ff:ff:ff:ff:ff:ff"
                    ether.ethertype = EtherType.ARP
                    arp = Arp(operation=ArpOperation.Request,
                            senderhwaddr=intf.ethaddr,
                            senderprotoaddr=intf.ipaddr,
                            targethwaddr='ff:ff:ff:ff:ff:ff',
                            targetprotoaddr=next_ip)
                    packet = ether+arp
                    item[3] = time.time()
                    item[4] += 1
                    self.net.send_packet(intf.name,packet)
```

很明显这个过程需要持续地进行，我没有新开一个线程来做，而是直接将其放在`start()`函数地`while`循环中了

```
while True:
    self.forward()
```

```
    try:
        ...
```

发出去了arp请求，当然会收到reply，此时需要接收这个reply包，并且根据它的源mac地址来设置目的mac地址。

首先在`handle_packet()`函数中判断是否是reply包，如果是，将这个包作为参数传递给`send()`函数

```python
elif arp.operation == ArpOperation.Reply:
    self.send(arp)
```

`send()` 函数的功能就是在等待队列 `queue` 中找到对应的数据包，并且将其发出去，最后在队列中删去。

```python
def send(self,arp):
    ip = arp.senderprotoaddr
    mac = arp.senderhwaddr
    for item in self.queue:
        if ip == item[2]:
            packet = item[0]
            intf = item[1]
            eth_index = packet.get_header_index(Ethernet)
            packet[eth_index].src = intf.ethaddr
            packet[eth_index].dst = mac
            self.net.send_packet(item[1].name,packet)
            self.queue.remove(item)
```

到这里，代码逻辑就实现完成了。

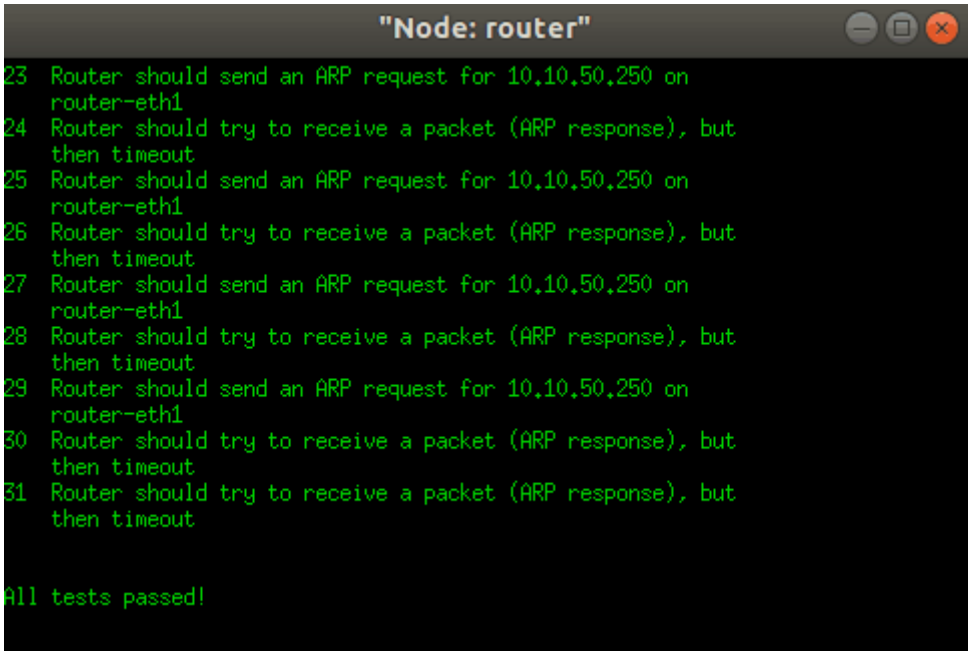
测试结果

testscenario

在router中用下述语句测试：

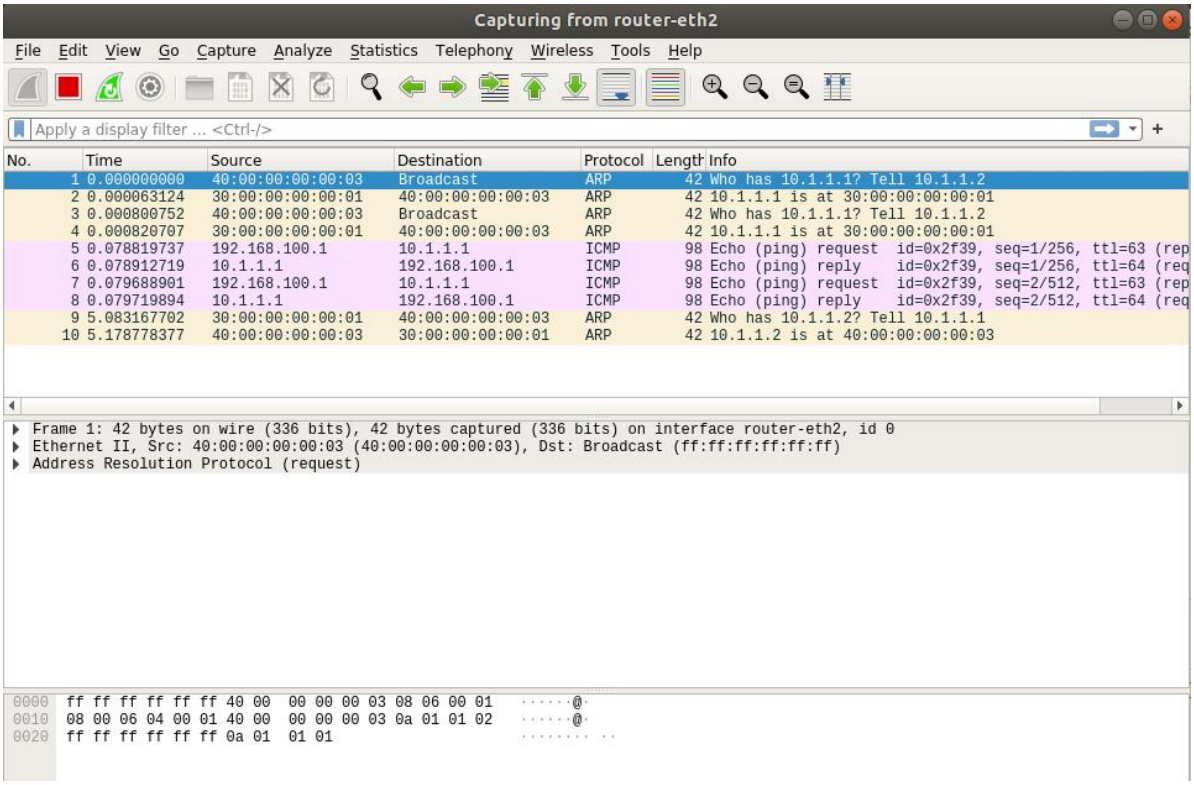
`swyard -t testcases/myrouter2_testscenario.srpy myrouter.py`

结果通过。

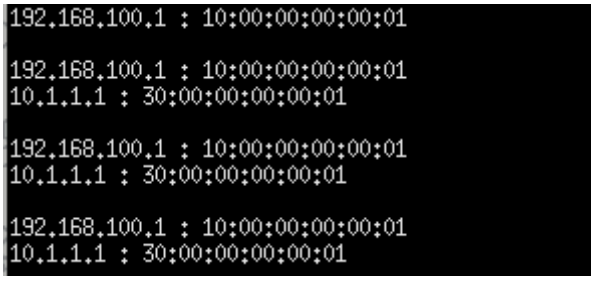


Mininet

从server1中输入：`ping -c2 10.1.1.1`，给client发送两个回声请求，监测router-eth2结果如下：



`arp_table`打印结果如下：



分析结果可知，首先server1广播寻找`10.1.1.1`的位置，此时`client`回应，同时路由器的`arp_table`更新`server1`的信息。

回应后`server1`接收到`reply`，更新`arp_table`并且发包，接着`client`回发，这样两遍，就得到了wireshark的结果。

实验心得

python虽然代码很简洁，并且有很多现成的东西可以用，但是对函数参数，函数的返回值类型不明确，这会使得，在新接触一个模块时，对其不了解会造成很大的困难。

对switchyard的API Reference更加熟悉，但是还有很多东西需要了解。