

Lab 3: Respond to ARP

191220138 杨飞洋

实验目的

学习路由器的工作原理，完成对ARP的响应，并且初步构建一个forwarding table。

背景知识

interface 接口

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

get_header()函数

```
def get_header(self, hdrclass, returnval=None):
    """
    Return the first header object that is of
    class hdrclass, or None if the header class isn't
    found.
    """
    if isinstance(hdrclass, str):
        return self.get_header_by_name(hdrclass)

    for hdr in self._headers:
        if isinstance(hdr, hdrclass):
            return hdr
    return returnval
```

ARP接口

ARP (address resolution protocol) header

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

hardwaretype

operation

protocoltype

senderhwaddr

senderprotoaddr

targethwaddr

targetprotoaddr

实现逻辑

首先在构造函数中初始化自己的forwarding table，即 `table`，用 `dict` 这个数据结构，`key = ipaddr, value = macaddr`

在利用 `net` 类的 `interface()` 接口来初始化自己的接口 `interfaces`，这样就不用每再调用了。

构造函数如下：

```
def __init__(self, net: switchyard.llnetbase.LLNetBase):
    self.net = net
    self.table = {}
    self.interfaces = self.net.interfaces()
    # other initialization stuff here
```

接下里就是编写 `handle_packet()` 函数。

这个函数是在 `start()` 函数中被循环调用的：

```
while True:
    try:
        recv = self.net.recv_packet(timeout=1.0)
    except NoPackets:
        continue
    except Shutdown:
        break
    self.handle_packet(recv)
```

考察 `recv_packet()` 函数的性质，可知 `handle_packet()` 函数的参数是一个 `tuple`，这也是 `handle_packet()` 函数第一行 `timestamp, ifaceName, packet = recv` 的意义。

然后用 `get_header()` 方法来判断接收到的是不是一个ARP包。

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

如果不是ARP类型的，这个函数直接返回None，handle中止。

如果是ARP类型的，则继续。首先根据Arp包的源IP地址和源MAC地址可以更新forwarding table，更新完了之后，为了测试将table打印出来。

遍历 interfaces，如果Arp包的目的IP地址即 targetprotoaddr 和接口的IP地址即 ipaddr 一样，则用 create_ip_arp_reply(senderhwaddr, targethwaddr, senderprotoaddr, targetprotoaddr) 构建一个数据包

其中

```
senderhwaddr = intf.ethaddr
targethwaddr = arp.senderhwaddr
senderprotoaddr = intf.ipaddr
targetprotoaddr = arp.senderprotoaddr
```

再用 net 类的 send_packet() 函数将这个数据包发送出去，注意出去的接口和进来的接口是一样的，都是 ifaceName

具体实现代码如下：

```
arp = packet.get_header(Arp)
if(arp):
    self.table[arp.senderprotoaddr] = arp.senderhwaddr
    for key,value in self.table.items():
        print(key,":",value)
        print(" ")
    for intf in self.interfaces:
        if(arp.targetprotoaddr == intf.ipaddr):
            Packet = create_ip_arp_reply(intf.ethaddr, arp.senderhwaddr,
            intf.ipaddr,
            arp.senderprotoaddr)
            self.net.send_packet(ifaceName,Packet)
```

测试结果

测试代码

在router中用下述语句测试：

```
swyard -t testcases/myrouter1_testscenario.srpy myrouter.py
```

结果通过。

```

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!

```

Mininet

ping the router from server1

实验结果:

The image shows a Wireshark packet capture window titled "Capturing from server1-eth0". The packet list shows five packets:

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-------------------|------------------|----------|--------|--|
| 1 | 0.000000000 | Private_00:00:01 | Broadcast | ARP | 42 | Who has 192.168.100.2? Tell 192.168.100.1 |
| 2 | 0.062386005 | 40:00:00:00:00:01 | Private_00:00:01 | ARP | 42 | 192.168.100.2 is at 40:00:00:00:00:01 |
| 3 | 0.062395555 | 192.168.100.1 | 192.168.100.2 | ICMP | 98 | Echo (ping) request id=0x3bf8, seq=1/256, ttl=64 (no |
| 4 | 1.026843377 | 192.168.100.1 | 192.168.100.2 | ICMP | 98 | Echo (ping) request id=0x3bf8, seq=2/512, ttl=64 (no |
| 5 | 2.050921981 | 192.168.100.1 | 192.168.100.2 | ICMP | 98 | Echo (ping) request id=0x3bf8, seq=3/768, ttl=64 (no |

The packet details pane shows the first packet (Frame 1) as an Ethernet II frame with source Private_00:00:01 and destination Broadcast (ff:ff:ff:ff:ff:ff). The protocol is Address Resolution Protocol (request).

The packet bytes pane shows the raw data of the first packet:

```

0000  ff ff ff ff ff 10 00 00 00 01 08 06 00 01  .....
0010  08 00 06 04 00 01 10 00 00 00 01 c0 a8 64 01  .....
0020  00 00 00 00 00 00 c0 a8 64 02  .....

```

路由器首先会收到一个ARP请求到它自己的IP地址，需要对这个请求做出响应。这个过程可以对应前两行。

然后router会收到来自server1的一个ICMP回送请求，会无事发生，这个过程可以对应后三行。

forwarding table

打印结果:

```
192.168.1.100 : 30:00:00:00:00:01
192.168.1.100 : 30:00:00:00:00:01
10.10.1.1 : 60:00:de:ad:be:ef
192.168.1.100 : 30:00:00:00:00:01
10.10.1.1 : 60:00:de:ad:be:ef
10.10.5.5 : 70:00:ca:fe:c0:de
```

forwarding table 只有在收到ARP包时会更新数据。

根据testcase的说明，整个过程一共发送了三次ARP请求，则每次都更新一下，进行打印就是上面的结果。

至于ip地址和mac地址的数据，因为这是把源的地址更新进去的，在testcase的说明中没有显示的说明。

实验心得

通过编写路由器对ARP的响应，更加了解了路由器的工作原理。

对switchyard的API Reference更加熟悉。

python语言十分简练好用。