Lab 5: Respond to ICMP

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：接收并回复ICMP包，并对一些异常情况做出处理。

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

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.

icmp接口

ICMP (Internet control message protocol) header (v4)

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

A mother class for all ICMP message types. It holds a reference to another object that contai type. Just setting the icmptype causes the data object to change (the change happens autom change, but it only changes to some valid code given the new icmptype.

Represents an ICMP packet header for IPv4.

icmpcode

icmpdata

icmptype

class switchyard.lib.packet.common.**ICMPType**

An enumeration.

EchoReply = 0

DestinationUnreachable = 3

SourceQuench = 4

Redirect = 5

EchoRequest = 8

TimeExceeded = 11

icmpdata

```
>>> list(ICMPTypeCodeMap[ICMPType.DestinationUnreachable])
[ <DestinationUnreachable.ProtocolUnreachable: 2>,
  <DestinationUnreachable.SourceHostIsolated: 8>,
  <DestinationUnreachable.FragmentationRequiredDFSet: 4>,
  <DestinationUnreachable.HostUnreachable: 1>,
  <DestinationUnreachable.DestinationNetworkUnknown: 6>,
  <DestinationUnreachable.NetworkUnreachableForTOS: 11>,
  <DestinationUnreachable.HostAdministrativelyProhibited: 10>,
  <DestinationUnreachable.DestinationHostUnknown: 7>,
  <DestinationUnreachable.HostPrecedenceViolation: 14>,
  <DestinationUnreachable.PrecedenceCutoffInEffect: 15>,
  <DestinationUnreachable.NetworkAdministrativelyProhibited: 9>,
  <DestinationUnreachable.NetworkUnreachable: 0>,
  <DestinationUnreachable.SourceRouteFailed: 5>,
  <DestinationUnreachable.PortUnreachable: 3>,
  <DestinationUnreachable.CommunicationAdministrativelyProhibited: 13>,
  <DestinationUnreachable.HostUnreachableForTOS: 12> ]
```

实现逻辑

Responding to ICMP echo requests

首先处理正常的情况，即发来的包是ICMP request并且接收方是路由器本身，路由器向源IP发送一个ICMP reply。

回发的reply包中的`sequence,identifier,data`都沿用收到的包中的数据，只需要将目的ip和源ip交换一下即可。

再将这个包交给`process_pkt`函数。

`match`函数就是在查找forwarding table，返回数组下标，查找失败返回-1，这里的`src_index = self.match(ipc4.src)`，即为packet的源IP的查找结果。

```
                if icmp_idx != -1 and packet[icmp_idx].icmptype ==
ICMPType.EchoRequest:
                    #dst is this router
                    i = ICMP()
                    i.icmptype = ICMPType.EchoReply
                    i.icmpdata.sequence = packet[icmp_idx].icmpdata.sequence
                    i.icmpdata.identifier = packet[icmp_idx].icmpdata.identifier
                    i.icmpdata.data = packet[icmp_idx].icmpdata.data
                    ipv4.dst,ipv4.src = ipv4.src,ipv4.dst
                    packet[icmp_idx] = i
                    self.process_pkt(src_index,packet,ipv4.dst)
                    return
```

这里谈一下`process_pkt`函数，显然这是一个处理packet的函数，有三个参数，其中`index`是由在`forwarding table`中最长前缀匹配`packet`的目的IP得到的数组下标，如果`index == -1`，说明在`forwarding table`中没有匹配,直接`return`。`packet`就是要发出去的包，`dstip`是目的ip，接着查找forwarding table进行转发和arp request就行了。

说明一下`global from_intf`，这个参数是`hanble_packet()`函数中的`ifromFace`，用来记录这个包是从哪个接口进入router的，方便回发。

这里说明一个特殊情况，就是为什么当`index == -1`时直接返回呢。此时相当于`packet`这个包想从`ifromFace`发回一个icmp reply或者是其他形式的包，但是目的IP在forwarding table中找不到，即这个包本来的源IP在forwarding table中找不到，则此时应该发给路由器本身一个`network unreachable`，不过自己给自己发包就没什么意义，我在这里就直接return了。

这个函数在后续也会持续使用，这里先解释一下。

下面是`process_pkt()`函数的实现。

```
def process_pkt(self,index,packet,dstip):
        global from_intf
        if index == -1:
            return
        next_ip = self.ftable[index][1]
        if next_ip == IPv4Address('0.0.0.0'):
            next_ip = dstip
        print(next_ip)
        print(self.ftable[index][2])
        if next_ip in self.arptable.keys():
            intfname = self.ftable[index][2]
            intf = self.net.interface_by_name(intfname)
            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:
            print(next_ip)
            self.insert(packet,self.net.interface_by_name(self.ftable[index]
[2]),next_ip,time.time())
```

ICMP destination port unreachable

不过不可能所有的包都是规范可靠的，总有很多异常情况，我们讨论四种，第一种是 icmp destination port unreachable。

这意味着，接收到的包的目的IP就是router的某个interface， 不过这个包不是icmp echo request，可能是UDP之类的。

这时就要给源IP发送一个icmp destination port unreachable error。

我是将收到的packet改装一下，就当成回发的error包了，用`build_icmp()`函数实现。

需要修改`icmptype、icmpcode、origdgramlen`，再配置一下IP就行了。

其中`icmptype、icmpcode`都能从手册中找到，`origdgramlen`就是packet原来的长度，可以用`len(packet)`得到。

`<DestinationUnreachable.PortUnreachable: 3>`

需要配置一下ip的protocol为`IPProtocol.ICMP`，目的IP和源IP和原来的相反。

这里的`Src = self.net.interface_by_name(ifaceName).ipaddr , Dst = ipv4.src`

ttl设为10

这个`build_icmp()`函数是生成异常包的一个函数，在后续都会用，这里就先解释了一下。

实现代码如下：

```
def build_icmp(self,Type,Code,pkt,Src,Dst,length):
        print("nonetype1")
        i = pkt.get_header_index(Ethernet)
        del pkt[i]
        icmp = ICMP()
        icmp.icmptype = Type
        icmp.icmpcode = Code
        icmp.icmpdata.data = pkt.to_bytes()[:28]
        icmp.icmpdata.origdgramlen = length
        ip = IPv4()
        ip.protocol = IPProtocol.ICMP
        ip.ttl = 10
        ip.src = Src
        ip.dst = Dst
        #return Ethernet() + ip + icmp
        return Ethernet() + ip + icmp
```

那么在`handle_packet()`函数中只需要两个函数就能实现对这个情况的处理了，就是上面所说的`build_icmp()`和`process_pkt()`函数。

```
packet =
self.build_icmp(ICMPType.DestinationUnreachable,3,packet,Src,Dst,length)
self.process_pkt(src_index,packet,Dst)
```

ICMP time exceeded

现在讨论第二种异常，如果收到的包的目的IP不是路由器本身的话，就先检查一下ttl，ttl自减之后如果`ttl == 0`，则这个包的寿命就结束了，就给源IP发一个超时异常。

在这里先解释一下为什么不先检查ttl呢，因为如果此时ttl减到0了，同时目的IP是路由器本身，则相当于这个包正好在最后一跳到达了目的地，那正好回发reply就行了，回发超时异常是不合理的。

将`icmpcode = ICMPType.TimeExceeded`，`icmpcode = ICMPCodeTimeExceeded.TTLExpired: 0`传入`build_icmp()`函数

得到新的packet包后，在交给`process_pkt`函数处理就行了。

代码如下：

```
elif packet[ipdx].ttl == 0:
        packet = self.build_icmp(ICMPType.TimeExceeded,0,packet,Src,Dst,length)
        self.process_pkt(src_index,packet,Dst)
```

ICMP destination network unreachable

讨论第三种异常，顾名思义，目的IP找不到，这是在转发表中找不到目的IP的情况。

首先用match函数进行查找目的IP

`dst_index = self.match(ipv4.dst)`

如果查找不到，将`icmpcode = ICMPType.DestinationUnreachable`，`icmpcode = <DestinationUnreachable.NetworkUnreachable: 0>`传入`build_icmp()`函数，直接用`insert()`函数放进等待队列就行了，`next_ip`设为`Dst`，即packet的源IP

代码如下：

```
elif dst_index == -1:
    packet =
self.build_icmp(ICMPType.DestinationUnreachable,0,packet,Src,Dst,length)
    self.insert(packet,self.net.interface_by_name(ifaceName),Dst,time.time())
```

ICMP destination host unreachable

讨论最后一种情况，找不到host，即arp request得不到回复，超过五次后就触发这个错误。

在遍历等待队列queue时，发现 num >= 5即已经发过5次arp request的包，如果他是一个icmp的包，则生成一个异常包。

将`icmpcode = ICMPType.DestinationUnreachable`，`icmpcode = <DestinationUnreachable.HostUnreachable: 1>`传入`build_icmp()`函数，并且用全局变量`from_intf`可以得到这个异常包的源IP，异常包的目的IP就是packet的源IP

将这个包交给`process_pkt()`函数处理就好了。

代码如下：

```
                if num >= 5:
                    if pkt.get_header_index(ICMP)!= -1:
                        global from_intf
                        print(from_intf)
                        ip = self.net.interface_by_name(from_intf).ipaddr
                        ipv4 = pkt.get_header(IPv4)
                        pkt =
self.build_icmp(ICMPType.DestinationUnreachable,1,pkt,ip,ipv4.src,len(pkt))
                        src_index = self.match(ipv4.src)
                        self.process_pkt(src_index,pkt,ipv4.src)
                    self.queue.remove(item)
```

这就讨论完所有的异常情况了，在配合lab4 和 lab3实现的功能，就初步完成了路由器的逻辑了。

测试结果

testscenario

用下述语句测试：

`swyard -t testcases/router3_testscenario.srpy myrouter.py`

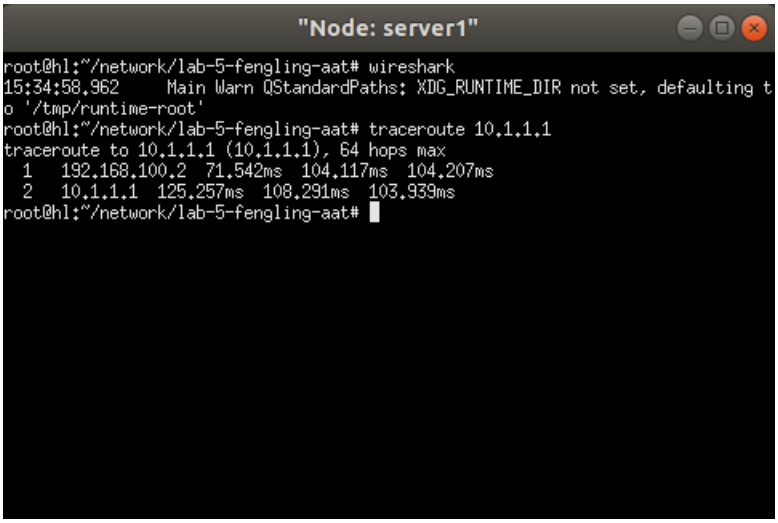
结果通过。

```
21  Router should try to receive a packet (ARP response), but
    then timeout.
22  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
23  Router should try to receive a packet (ARP response), but
    then timeout.
24  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
25  Router should try to receive a packet (ARP response), but
    then timeout.  At this point, the router should give up and
    generate an ICMP host unreachable error.
26  Router should send an ARP request for 192.168.1.239.
27  Router should receive ARP reply for 192.168.1.239.
28  Router should send an ICMP host unreachable error to
    192.168.1.239.


All tests passed!
```

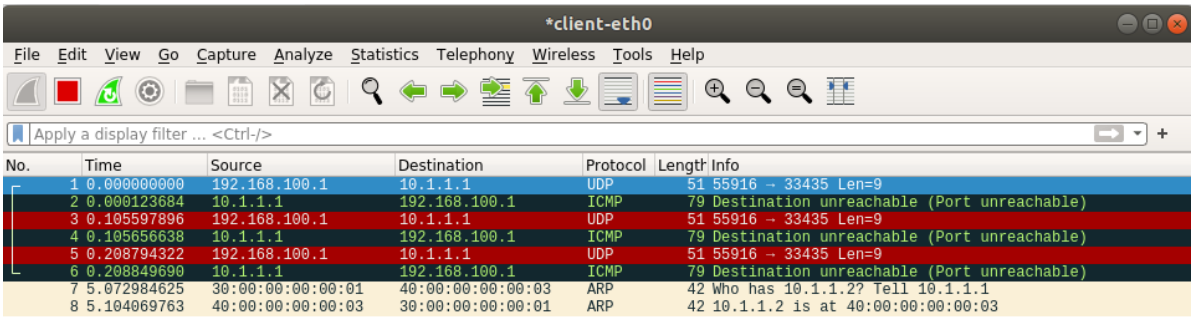
Mininet

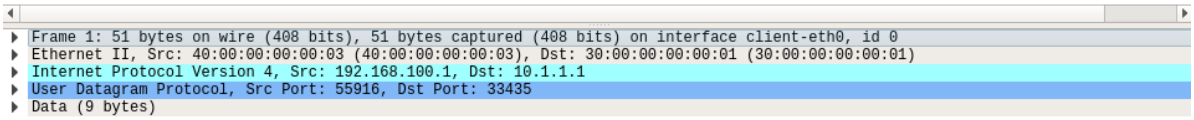
从server1中输入：`traceroute 10.1.1.1`，相当于server1给client发送一个UDP包，结果如下：



单看上面的结果不明显，需要在wireshark中进行监测。

监测client结果如下：





可以看到触发了**ICMP destination port unreachable**异常，至于为什么有3次，应该是traceroute的频率太快了导致的。

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。

大概完成了一个路由器的逻辑还是很有成就感的，经过了多次代码重构之后，结构自我感觉还是比较良好且稳定的。