Computer Networking-Lab-Repot

课程名称：计算机网络 任课教师：田臣/李文中

学院	Dept. of Computer Science and Technology	专业（方向）	CS
学号	181830044	姓名	董宸郅
Email	pdon#foxmail.com	开始/完成日期	4.23/5.6

实验名称：IPv4 Router: Respond to ICMP

实验目的

- 进一步完善**Router**功能
- 深入理解**Router**对于各类错误信息的处理机制
- 加强封装功能性代码的能力，降低代码耦合度
- 继续加强在实验环境下调试代码的能力

实验内容

理论知识

ICMP

定义

ICMP（Internet Control Message Protocol）Internet控制报文协议。它是TCP/IP协议簇的一个子协议，用于在IP主机、路由器之间传递控制消息。控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。

结构

ICMP报文包含在IP数据报中，属于IP的一个payload，IP头部就在ICMP报文的前面，所以一个ICMP报文包括IP头部、ICMP头部和ICMP报文，IP头部的Protocol值为1就说明这是一个ICMP报文，ICMP头部中的类型（Type）域用于说明ICMP报文的作用及格式，此外还有一个代码（Code）域用于详细说明某种ICMP报文的类型，所有数据都在ICMP头部后面。

类型

- 差错报告报文

类型值为3时：终点不可达

类型值为4时：源点抑制

类型值为5时：改变路由(Redirect）

类型值为11时：超时

类型值为12时：参数问题

- 询问报文

类型值为8或者0时：回送(Echo)请求或应答

类型值为13或14时：时间戳(Timestamp)请求或应答

实验步骤（含测试结果与关键代码）

Task 2: Responding to ICMP echo requests | Task 3: Generating ICMP error messages

`FYI`：由于此次Task 2&3的代码逻辑高度重叠，拆分开来叙述可能会牺牲整体性，所以我觉得合在一起讲更为清晰。

Coding

新增两个类方法：

`ipv4_handle(self, pkt, input_intf)`

`send_icmp_error_pkt(self, error_case, pkt, input_intf)`

- 首先介绍后者，功能是生成包含对应的ICMP错误信息的IPv4数据报并交由`ipv4_handle()`完成转发：

```
def send_icmp_error_pkt(self, error_case, pkt, input_intf):
    '''
    put together all 4 error cases to simplify code
    '''
    # drop the pkt if another 'error' occurs with the error icmp pkt
    ipv4_pkt = pkt.get_header(IPv4)
    match = False
    for entry in self.fwd_tab:
        prefixnet = IPv4Network(entry[0] + '/' + entry[1], strict=False)
        if ipv4_pkt.src in prefixnet:
            match = True
            break
    if match == False:
        return
    # create the error pkt
    tmp_icmp = ICMP()
    tmp_ip = IPv4()
    tmp_ip.protocol = IPProtocol.ICMP
    tmp_ip.ttl = 64
    tmp_ip.src = input_intf.ipaddr
    tmp_ip.dst = ipv4_pkt.src

    # ICMP error case 1: no match in forwarding table found
    if error_case == 1:
        tmp_icmp.icmptype = ICMPType.DestinationUnreachable
        tmp_icmp.icmpcode = 0
    # ICMP error case 2: TTL become zero
    elif error_case == 2:
        tmp_icmp.icmptype = ICMPType.TimeExceeded
        tmp_icmp.icmpcode = 0
    # ICMP error case 3: no ARP reply received
    elif error_case == 3:
```

```
        tmp_icmp.icmptype = ICMPType.DestinationUnreachable
        tmp_icmp.icmpcode = 1
    # ICMP error case 4: dst ip belong to the router itself while it's not an ICMP echo request
    elif error_case == 4:
        tmp_icmp.icmptype = ICMPType.DestinationUnreachable
        tmp_icmp.icmpcode = 3

    tmp_icmp.icmpdata.data = ipv4_pkt.to_bytes()[:28]
    new_pkt = Packet()
    new_pkt = Ethernet() + tmp_ip + tmp_icmp
    self.ipv4_handle(new_pkt, input_intf)
```

- 其次介绍后者，功能是处理并转发Router收到的或它自己生成的所有IPv4数据报，很多地方涉及到**4种错误**的处理：

```
def ipv4_handle(self, pkt, input_intf):
    '''
    Handle the IPv4 forwarding job of the router
    in order to simplify the code
    '''
    ipv4_pkt = pkt.get_header(IPv4)
```

  - 若目标地址属于Router自身

```
# if the dst belongs to the router itself
if ipv4_pkt.dst in self.ipaddrs:
```

    - 若为ICMP EchoRequest：

```
if ipv4_pkt.protocol == IPProtocol.ICMP and pkt.get_header(
        ICMP).icmptype == ICMPType.EchoRequest:
    # error case2
    if ipv4_pkt.ttl - 1 <= 0:
        self.send_icmp_error_pkt(2, pkt, input_intf)
        return
    # create ICMP reply
    ori_icmp = pkt.get_header(ICMP)
    tmp_icmp = ICMP()
    tmp_icmp.icmptype = ICMPType.EchoReply
    tmp_icmp.icmpdata.sequence = ori_icmp.icmpdata.sequence
    tmp_icmp.icmpdata.identifier = ori_icmp.icmpdata.identifier
    tmp_icmp.icmpdata.data = ori_icmp.icmpdata.data

    tmp_ipv4 = IPv4()
    tmp_ipv4.src = input_intf.ipaddr
    tmp_ipv4.dst = ipv4_pkt.src
    tmp_ipv4.protocol = IPProtocol.ICMP
    tmp_ipv4.ttl = 64

    new_pkt = Packet()
    new_pkt = Ethernet() + tmp_ipv4 + tmp_icmp
    self.ipv4_handle(new_pkt, input_intf)
```

    - 否则

```
# error case4
else:
    self.send_icmp_error_pkt(4, pkt, input_intf)
    return
```

- 否则

```
else:
    # denote the longest prefix match so far
    longest = 0
    # denote the updated matched entry
    matched_entry = []
    for entry in self.fwd_tab:
        prefixnet = IPv4Network(entry[0] + '/' + entry[1],
                                strict=False)
        if (ipv4_pkt.dst in prefixnet) and (longest <
                                            prefixnet.prefixlen):
            longest = prefixnet.prefixlen
            matched_entry = entry
```

  - 路由表中，最长前缀匹配失败

```
# error case1
if matched_entry == []:
    self.send_icmp_error_pkt(1, pkt, input_intf)
    return
```

  - 否则，若**TTL变为0**

```
# error case2
if ipv4_pkt.ttl - 1 <= 0:
    self.send_icmp_error_pkt(2, pkt, input_intf)
    return
```

  - 否则

```
# mainly job in lab_4
ipv4_pkt.ttl -= 1
intf = self.net.interface_by_name(matched_entry[3])
# 1. the dst is within the subnet to which the intf belong,
#    which means the next hop is the dst
if matched_entry[2] == '0.0.0.0':
    next_hop_ip = ipv4_pkt.dst
# 2. the next hop is an IP address on a router through which
the destination is reachable
else:
    next_hop_ip = IPv4Address(matched_entry[2])
# if the IP-MAC pair is already recorded in the ARP cache
table,
# then no need for an ARP request
if next_hop_ip in self.arp_tab:
    dst_macaddr = self.arp_tab[next_hop_ip]
    pkt.get_header(Ethernet).src = intf.ethaddr
    pkt.get_header(Ethernet).dst = dst_macaddr
```

```
    self.net.send_packet(matched_entry[3], pkt)
# ARP request is necessary when none recorded
else:
    # create a new ARP request using the handy API
    arp_req = create_ip_arp_request(intf.ethaddr, intf.ipaddr,
                                    next_hop_ip)
    self.net.send_packet(matched_entry[3], arp_req)
    # add this request into waiting queue
    new_entry = [
        next_hop_ip,
        time.time(), 1, matched_entry, pkt, arp_req, input_intf
    ]
    self.wait_q.append(new_entry)
```

Testing

```
Passed:
1   ICMP echo request (PING) for the router IP address
    192.168.1.1 should arrive on router-eth0.  This PING is
    directed at the router, and the router should respond with
    an ICMP echo reply.
2   Router should send an ARP request for 10.10.1.254 out
    router-eth1.
3   Router should receive ARP reply for 10.10.1.254 on router-
    eth1.
4   Router should send ICMP echo reply (PING) to 172.16.111.222
    out router-eth1 (that's right: ping reply goes out a
    different interface than the request).
5   ICMP echo request (PING) for the router IP address 10.10.0.1
    should arrive on router-eth1.
6   Router should send ICMP echo reply (PING) to 172.16.111.222
    out router-eth1.
7   ICMP echo request (PING) for 10.100.1.1 with a TTL of 1
    should arrive on router-eth1.  The router should decrement
    the TTL to 0 then see that the packet has "expired" and
    generate an ICMP time exceeded error.
8   Router should send ARP request for 10.10.123.123 out router-
    eth1.
9   Router should receive ARP reply for 10.10.123.123 on router-
    eth1.
10  Router should send ICMP time exceeded error back to
    10.10.123.123 on router-eth1.
11  A packet to be forwarded to 1.2.3.4 should arrive on router-
    eth1.  The destination address 1.2.3.4 should not match any
    entry in the forwarding table.
12  Router should send an ICMP destination network unreachable
    error back to 10.10.123.123 out router-eth1.
13  A UDP packet addressed to the router's IP address
    192.168.1.1 should arrive on router-eth1.  The router cannot
    handle this type of packet and should generate an ICMP
    destination port unreachable error.
14  The router should send an ICMP destination port unreachable
    error back to 172.16.111.222 out router-eth1.
15  An IP packet from 192.168.1.239 for 10.10.50.250 should
    arrive on router-eth0.  The host 10.10.50.250 is presumed
    not to exist, so any attempts to send ARP requests will
    eventually fail.
16  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
17  Router should try to receive a packet (ARP response), but
    then timeout.
18  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
19  Router should try to receive a packet (ARP response), but
    then timeout.
20  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
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!
```

Deploying

1. 在server1的cli中输入以下命令：

```
# from server1:
ping -c 1 192.168.100.2
```

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	0.063101208	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	0.063110642	192.168.100.1	192.168.100.2	ICMP	98	Echo (ping) request id=0x4255, seq=1/256, ttl=64 (reply in 4)
4	0.171881446	192.168.100.2	192.168.100.1	ICMP	98	Echo (ping) reply id=0x4255, seq=1/256, ttl=63 (request in 3)

router收到ICMP Echo Request之后判断正常，返回一个Echo Reply。

2. 在server1的cli中输入以下命令：

```
# from server1:
ping -c 1 -t 1 192.168.100.2
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.100.2	ICMP	98	Echo (ping) request id=0x43ae, seq=1/256, ttl=1 (no response found!)
2	0.070283870	40:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.2
3	0.070295913	Private_00:00:01	40:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
4	0.173202679	192.168.100.2	192.168.100.1	ICMP	62	Time-to-live exceeded (Time to live exceeded in transit)
5	5.168140998	Private_00:00:01	40:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
6	5.205179139	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01

router收到ICMP Request，但由于TTL减至0，返回一个**TTL exceeded**错误信息。

3. 首先，在`start_mininet.py`中，临时添加（为了使server1能够ping一个router无法转发的IP）：

```
set_route(net, 'server1', '172.16.0.0/16', '192.168.100.2')
```

然后，在server1的cli中输入以下命令：

```
# from server1:
ping -c 1 172.16.1.1
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x48cb, seq=1/256, ttl=64 (no response found!)
2	0.060067422	40:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.2
3	0.060082983	Private_00:00:01	40:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
4	0.157618350	192.168.100.2	192.168.100.1	ICMP	62	Destination unreachable (Network unreachable)
5	5.255669497	Private_00:00:01	40:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
6	5.358181303	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01

由于在router中，该目标IP地址无法成功进行最长前缀匹配，router返回一个**Destination unreachable(Network unreachable)**错误信息。

4. 在server1的cli中：

```
root@ubuntu:~/switchyard/lab_5# traceroute -N 1 -n 10.1.1.1
traceroute to 10.1.1.1 (10.1.1.1), 30 hops max, 60 byte packets
 1  * * *
 2  10.1.1.1  194.335 ms  208.033 ms  208.926 ms
```

总结与感想

由于此次任务相比之前，需要判断的条件分支更多更杂，而且在我的实现中也存在多个模块互相调用的情况，所以就更为强调对于**代码功能的封装**。前几次lab中，我几乎没有添加其余的函数，全是在router_main()中一股脑完成的；这也导致了我这次不得不对代码结构有了比较大的调整。所以之后编程时还是要从一开始就理清思路，降低代码耦合度，方便今后对某些功能的增添删改。

但总的来说，经过之前几次lab的磨砺，现在完成起来更加得心应手了，也能抽出更多时间去啃教科书。本学期不返校的通知出来之后，客观上还是让我松了一口气，毕竟之前期中期末边做作业边抽时间复习的情况这次可以不用担心了。总之，努力抓住当前机遇好好学习吧！