南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：田臣/李文中　　助教：

学院	**计算机科学与技术系**	专业（方向）	**计算机科学与技术**
学号	**191220154**	姓名	**张涵之**
Email	**1683762615@qq.com**	开始/完成日期	**2021/5/15-2021/5/16**

1. 实验名称：Lab 5: Respond to ICMP

2. 实验目的：

Respond to ICMP messages like echo requests ("pings").
Generate ICMP error messages when necessary.

3. 实验内容

a) Task 2: Responding to ICMP echo requests
b) Task 3: Generating ICMP error messages

4. 实验结果

a) Task 2 & Task 3 in test scenario:
Forwarding table is shown as below:

```
-------------------------------------------------------------------------------
network address         subnet Address          next hop address        interface
192.168.1.1             255.255.255.0           None                    router-eth0
10.10.0.1               255.255.0.0             None                    router-eth1
172.16.42.1             255.255.255.252         None                    router-eth2
172.16.0.0              255.255.0.0             192.168.1.2             router-eth0
172.16.128.0            255.255.192.0           10.10.0.254             router-eth1
172.16.64.0             255.255.192.0           10.10.1.254             router-eth1
10.100.0.0              255.255.0.0             172.16.42.2             router-eth2
-------------------------------------------------------------------------------
```

The first (set of) testcase involves replying to a ping request:
An ARP request is needed before the reply is sent.

```
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).
```

```
10:32:25 2021/05/16     INFO Receive an ICMP echo request.
10:32:25 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:25 2021/05/16     INFO Matching successful.
10:32:25 2021/05/16     INFO Sending ARP request from queue...
10:32:25 2021/05/16     INFO Sending ARP request......
10:32:25 2021/05/16     INFO Receive an ARP reply.
10:32:25 2021/05/16     INFO Deleting answered ARP entry from queue...
```

```
10:32:25 2021/05/16     INFO Fowarding packet from queue...
10:32:25 2021/05/16     INFO Deleting forwarded wait entry from queue...
```

The second testcase involves replying to a ping request directly:

```
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.
```

```
10:32:25 2021/05/16     INFO Receive an ICMP echo request.
10:32:25 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:25 2021/05/16     INFO Matching successful.
10:32:25 2021/05/16     INFO Fowarding packet from queue...
10:32:25 2021/05/16     INFO Deleting forwarded wait entry from queue...
```

The third testcase handles a TTL exceeded ping request:
An ARP request is needed before the error message is sent.

```
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.
```

```
10:32:25 2021/05/16     INFO Receive an IPV4 packet.
10:32:25 2021/05/16     INFO ICMP time exceeded.
10:32:25 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:25 2021/05/16     INFO Matching successful.
10:32:25 2021/05/16     INFO Sending ARP request from queue...
10:32:25 2021/05/16     INFO Sending ARP request......
10:32:25 2021/05/16     INFO Receive an ARP reply.
10:32:25 2021/05/16     INFO Deleting answered ARP entry from queue...
```

```
10:32:25 2021/05/16     INFO Fowarding packet from queue...
10:32:25 2021/05/16     INFO Deleting forwarded wait entry from queue...
```

The fourth testcase handles a request with unreachable destination:

```
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.
```

```
10:32:25 2021/05/16     INFO Receive an IPV4 packet.
10:32:25 2021/05/16     INFO Router doesn't know where to forward.
10:32:25 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:25 2021/05/16     INFO Matching successful.
10:32:25 2021/05/16     INFO Fowarding packet from queue...
10:32:25 2021/05/16     INFO Deleting forwarded wait entry from queue...
```

The fifth testcase fails to handle a UDP packet:

```
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.
```

```
10:32:25 2021/05/16     INFO Cannot handle UDP packet.
10:32:25 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:25 2021/05/16     INFO Matching successful.
10:32:25 2021/05/16     INFO Fowarding packet from queue...
10:32:25 2021/05/16     INFO Deleting forwarded wait entry from queue...
```

The sixth testcase deals with a destination host that does not exist:
Five ARP requests are sent but no replies are received.
After that the router sends an ICMP host unreachable error.
An ARP request is needed before the error message is sent.

```
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.
```

```
10:32:25 2021/05/16     INFO Receive an IPV4 packet.
10:32:25 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:25 2021/05/16     INFO Matching successful.
10:32:25 2021/05/16     INFO Sending ARP request from queue...
10:32:25 2021/05/16     INFO Sending ARP request......
10:32:26 2021/05/16     INFO Sending ARP request from queue...
10:32:26 2021/05/16     INFO Sending ARP request......
10:32:28 2021/05/16     INFO Sending ARP request from queue...
10:32:28 2021/05/16     INFO Sending ARP request......
10:32:29 2021/05/16     INFO Sending ARP request from queue...
10:32:29 2021/05/16     INFO Sending ARP request......
10:32:31 2021/05/16     INFO Sending ARP request from queue...
10:32:31 2021/05/16     INFO Sending ARP request......
10:32:33 2021/05/16     INFO Deleting timeout ARP entry from queue...
10:32:33 2021/05/16     INFO ARP failure.
10:32:33 2021/05/16     INFO Matching dest IP against torwaring table...
10:32:33 2021/05/16     INFO Matching successful.
10:32:33 2021/05/16     INFO Deleting timeout wait entry from queue...
10:32:33 2021/05/16     INFO Sending ARP request from queue...
10:32:33 2021/05/16     INFO Sending ARP request......
10:32:33 2021/05/16     INFO Receive an ARP reply.
10:32:33 2021/05/16     INFO Deleting answered ARP entry from queue...
```

The logic of the testcases and log info match well.

b) Task 2 & Task 3 in Mininet:

Testing TTL exceeded error (using server1):

```
root@njucs-VirtualBox:~/switchyard/lab-5-RainTreeCrow# ping -c 1 -t 1 192.168.200.1
PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data.
From 192.168.100.2 icmp_seq=1 Time to live exceeded

--- 192.168.200.1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Capturing from server1-eth0:

```
Private_00:00:01  Broadcast         ARP   42 Who has 192.168.100.2? Tell 192.168.100.1
40:00:00:00:00:01 Private_00:00:01 ARP   42 192.168.100.2 is at 40:00:00:00:00:01
192.168.100.1     192.168.200.1    ICMP  98 Echo (ping) request  id=0x1606, seq=1/256, ttl=1 (no resp
192.168.100.2     192.168.100.1    ICMP  70 Time-to-live exceeded (Time to live exceeded in transit)
```

Testing network unreachable error (using server1):

Adding an additional route in start_mininet.py imitating client:

```
set_route(net, 'server1', '192.168.200.0/24', '192.168.100.2')
set_route(net, 'server1', '172.18.0.0/16', '192.168.100.2')
set_route(net, 'server2', '10.1.0.0/16', '192.168.200.2')
```

```
root@njucs-VirtualBox:~/switchyard/lab-5-RainTreeCrow# ping -c 1 172.18.1.1
PING 172.18.1.1 (172.18.1.1) 56(84) bytes of data.
From 192.168.100.2 icmp_seq=1 Destination Net Unreachable

--- 172.18.1.1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Capturing from server1-eth0

```
Private_00:00:01   Broadcast         ARP   42 Who has 192.168.100.2? Tell 192.168.100.1
40:00:00:00:00:01  Private_00:00:01  ARP   42 192.168.100.2 is at 40:00:00:00:00:01
192.168.100.1      172.18.1.1        ICMP  98 Echo (ping) request  id=0x17d3, seq=1/256, ttl
192.168.100.2      192.168.100.1     ICMP  70 Destination unreachable (Network unreachable)
```

Testing trace route (using server1):

```
root@njucs-VirtualBox:~/switchyard/lab-5-RainTreeCrow# traceroute -N 1 -n 192.168.200.1
traceroute to 192.168.200.1 (192.168.200.1), 30 hops max, 60 byte packets
 1  192.168.100.2  17.501 ms *  44.913 ms
 2  192.168.200.1  307.293 ms  208.608 ms  215.976 ms
root@njucs-VirtualBox:~/switchyard/lab-5-RainTreeCrow# 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  192.168.100.2  93.864 ms  98.166 ms  111.247 ms
 2  10.1.1.1  309.182 ms  210.614 ms  206.034 ms
```

5. 核心代码

a) Task 2: Responding to ICMP echo requests

I 'borrowed' the functions mk_ping and mk_icmperr from the provided python test file router3_testscenario_template.py to generate packets.

#This part of code is added to the handle_packet function. First check whether the IP destination address is the same as one of the addresses of the interfaces. If the packet is also an ICMP echo request, construct an ICMP echo reply.

```
elif eth.ethertype == EtherType.IPv4:
    packet.get_header(IPv4).ttl -= 1
    ipv4 = packet.get_header(IPv4)
    icmp = packet.get_header(ICMP)
    ether = packet.get_header(Ethernet)
    log_info("Receive an IPV4 packet.")
    if ipv4.dst in [iface.ipaddr for iface in self.interfaces]:
        if ipv4.protocol == IPProtocol.UDP:
            log_info("Cannot handle UDP packet.")
            packet = mk_icmperr(hwsrc=self.net.interface_by_name(ifaceName).ethaddr,
                hwdst=ether.src,
                ipsrc=self.net.interface_by_name(ifaceName).ipaddr,
                ipdst=ipv4.src,
                xtype=ICMPType.DestinationUnreachable,
                xcode=3,
                origpkt=packet
            )
        elif icmp.icmptype == ICMPType.EchoRequest:
            log_info("Receive an ICMP echo request.")
            packet = mk_ping(hwsrc=ether.dst, hwdst=ether.src,
                ipsrc=ipv4.dst, ipdst=ipv4.src,
                reply=True, payload=icmp.icmpdata.data
            )
```

#The reply is constructed using mk_ping function. The destination address is set as the source address of the incoming echo request, and the source address is the router's interface address (which in this case is also the original request's IP destination). Other information is in payload=icmp.icmpdata.data.

#I did not see any information regarding UCP packets until I came across the log info in the provided test scenario. It says the router cannot handle this type of packet and should generate an ICMP destination port unreachable error, so I wrote according to the description. Testcase oriented programming it is.

b) Task 3: Generating ICMP error messages

For situations in which ICMP error messages are generate are listed below:

1. Error: No matching entry found when attempting to match the destination address of an IP packet with entries in the forwarding table, (i.e., the router

does not know where to forward the packet).
Solution: Send an ICMP destination network unreachable error back to the host referred to by the source address in the IP packet.

```
else:
    matched = False
    for fwEntry in self.forwardingTable:
        netAddr = IPv4Network(str(fwEntry.networkAddr) +
            "/" + str(fwEntry.subnetAddr), False
        )
        if ipv4.dst in netAddr:
            matched = True
            break
    if matched == False:
        log_info("Router doesn't know where to forward.")
        packet = mk_icmperr(hwsrc=self.net.interface_by_name(ifaceName).ethaddr,
            hwdst=ether.src,
            ipsrc=self.net.interface_by_name(ifaceName).ipaddr,
            ipdst=ipv4.src,
            xtype=ICMPType.DestinationUnreachable,
            origpkt=packet
        )
```

#The matching process is similar to Lab 4, only the packet is not directly put into wait queue if matching is successful. The part of code where the packets join the queue is moved to a separate function, send_ipv4, which will be called after every type of error is inspected and dealt with.

2. Error: The IP packet's TTL becomes zero after decrementing.
   Solution: Sending an ICMP time exceeded error message back to the host referred to by the source address in the IP packet.

```
elif ipv4.ttl == 0:
    log_info("ICMP time exceeded.")
    packet = mk_icmperr(hwsrc=self.net.interface_by_name(ifaceName).ethaddr,
        hwdst=ether.src,
        ipsrc=self.net.interface_by_name(ifaceName).ipaddr,
        ipdst=ipv4.src,
        xtype=ICMPType.TimeExceeded,
        origpkt=packet
    )
```

3. Error: ARP Failure. After 5 retransmissions of an ARP request, the router does not receive an ARP reply, which indicates there is no host that "owns" a particular IP address of the next hop or the destination host.
   Solution: Sending an ICMP destination host unreachable back to the host referred to by the source address in the IP packet.
   #This part of the logic (transmitting and retransmitting ARP requests, and dropping timeout ARP requests) used to be dealt with in class WaitQueue's member function update_queue, it is now moved to the class Router, and is modified to deal with timeout ARP requests better.

```
else:
    self.wait.arpQueue.remove(arpReq)
    if arpReq in self.wait.arpAddr:
        self.wait.arpAddr.remove(arpReq)
    log_info("Deleting timeout ARP entry from queue...")
    for entry in self.wait.waitQueue[:]:
        if entry.nextHopAddr == arpReq.nextHopAddr:
            log_info("ARP failure.")
```

```
            packet = entry.packet
            ipv4 = packet.get_header(IPv4)
            #icmp = packet.get_header(ICMP)
            ether = packet.get_header(Ethernet)
            packet = mk_icmperr(hwsrc=self.net.interface_by_name(
                entry.fromIface).ethaddr,
                hwdst=ether.src,
                ipsrc=self.net.interface_by_name(entry.fromIface).ipaddr,
                ipdst=ipv4.src,
                xtype=ICMPType.DestinationUnreachable,
                xcode=1,
                origpkt=packet
            )
            self.send_ipv4(packet, None)
            log_info("Deleting timeout wait entry from queue...")
            self.wait.waitQueue.remove(entry)
```

#The WEntry and add_entry function in Lab 4 do not remember where the packets to be forwarded come from. So, when the ARP failure error should be sent back, it does not know the source address (to be honest, I did not know the source address either, until I saw the log info in the test scenario and assume it should be the address of the interface the original input port of the packet saved to the queue, thus, a variable fromIface is added to the WEntry to solve this issue, it seems to work okay.

```
Expected event:
    Router should send an ICMP host unreachable error to
    192.168.1.239.

Failure observed:
    You called send_packet and while the output port router-eth0
    is ok, a inexact match of packet contents failed.  when
    comparing the packet you sent versus what I expected,  the
    predicate (lambda pkt:
    pkt.get_header(ICMP).icmpdata.data[:8] ==
    b'E\x00\x00\x1c\x00\x00\x00\x00' ) passed, and the predicate
    (lambda pkt: pkt.get_header(IPv4).ttl >= 8) passed. In the
    IPv4 header, src is wrong (is 10.10.50.250 but should be
    192.168.1.1).
```

4. Error: An incoming packet is destined to an IP addresses assigned to one of the router's interfaces, but the packet is not an ICMP echo request.
   Solution: Send an ICMP destination port unreachable error message back to the source address in the IP packet.

```
else:
    log_info("Not an ICMP echo request.")
    packet = mk_icmperr(hwsrc=self.net.interface_by_name(ifaceName).ethaddr,
        hwdst=ether.src,
        ipsrc=self.net.interface_by_name(ifaceName).ipaddr,
        ipdst=ipv4.src,
        xtype=ICMPType.DestinationUnreachable,
        xcode=3,
        origpkt=packet
    )
```

*One thing I didn't quite understand about the functions mk_ping is that in the manual, I am supposed to copy the echo request's sequence number, identifier and data field. At first I used the function without modifying and it passed, but it did not handle sequence and identifier, just the data field, I supposed.
Then I tried two ways to deal with the problem and chose the second one.
*If I pass in the entire icmpdata and copy it as a whole:

```
icmppkt.icmpdata = payload
reply=True, payload=icmp.icmpdata
```

I cannot pass the testcases and will fail here:

```
Expected event:
    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).

Failure observed:
    You called send_packet and while the output port router-eth1
    is ok, a inexact match of packet contents failed.  When
    comparing the packet you sent versus what I expected,  the
    predicate (lambda pkt: pkt.get_header(ICMP).icmpdata.data ==
    b'hello, world' ) passed, and the predicate (lambda pkt:
    pkt.get_header(IPv4).ttl >= 8) passed. In the ICMP header,
    icmptype is wrong (is 8 but should be 0).
```

*If I copy the three parts respectively:

```
#icmppkt.icmpdata.sequence = 42
icmppkt.icmpdata.sequence = payload.sequence
icmppkt.icmpdata.identifier = payload.identifier
#icmppkt.icmpdata.data = payload
icmppkt.icmpdata.data = payload.data
```

Then I can pass all testcases successfully. I thought the manual says icmpdata only has these three fields, but turns out to be a misunderstanding.

class switchyard.lib.packet.**ICMPEchoReply**

*It says data, identifier and sequence, but well…

#In all the cases above, the ICMP type and code are set as below:

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

An enumeration.

```
EchoReply = 0

DestinationUnreachable = 3

SourceQuench = 4

Redirect = 5

EchoRequest = 8

TimeExceeded = 11
```

Types	Codes	Description
0	0	echo reply (ping)
3	0	dest network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired

6. 总结与感想

a) I probably should not have designed so many classes in Lab 4, the entries and queues really took me a long time to modify. With so many functions calling each other, the structure became messy and confusing. I suppose the best way to solve this kind of problem is to read through Lab 3 to Lab 5 and design the structure as a whole, otherwise the classes should be designed carefully so they would easy to change. In a word, DO NOT use object-oriented programming UNLESS you have the confidence to arrange the classes neatly.

b) Reading the manual is important, DO NOT take anything for granted.