# 实验报告

计算机网络-Lab5

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

# 目录

# 一、 实验目的

在 lab4 的基础上，使我们的路由器能够回应 ICMP 报文；同时，它自己也会组装一个 ICMP 报错信息（例如，发现一个包 TTL 到期了）。

# 二、 实验结果

- 路由器可以正确回应 ICMP echo request；
- 路由器可以生成合适的 ICMP 错误，并发往正确的目的地；
- 实验代码在单线程模式下通过了官方测试集；
- 自己设计了测试集，代码可以在单线程模式下通过；
- 在 Mininet 简单仿真测试环境下，单模式、多线程可以正常工作（利用 Wireshark 抓包验证）；
- 发现了一个非常诡异的 BUG（见实验内容）。

# 三、 实验内容

## 1. Preparation

只要我们完成了 Lab4，那么这一阶段的事情只剩下阅读手册和 STFW 了。

## 2. Responding to ICMP echo requests

当路由器收到一个包，该包的目的地是路由器的某个端口，并且包的类型为 ICMP request，那我们就得 reply。生成 reply 非常简单，我们只需对换 IP 首部、以太网首部的源和目的地址，然后拷贝 request 中的 `sequence number`、identifier 和 `data` 即

可（该三者都是在 icmpdata 成员中）。

```
# just swap the src and dst mac and ip
ipv4_header.src = ori_packet.get_header(IPv4).dst
ipv4_header.dst = ori_packet.get_header(IPv4).src
request_icmp_header = ori_packet.get_header(ICMP)
icmp_header.icmpdata = ICMPEchoReply()
icmp_header.icmpdata.data = request_icmp_header.icmpdata.data
icmp_header.icmpdata.identifier = request_icmp_header.icmpdata.identifie
icmp_header.icmpdata.sequence = request_icmp_header.icmpdata.sequence
ethernet_header.src = ori_packet.get_header(Ethernet).dst
ethernet_header.dst = ori_packet.get_header(Ethernet).src
```

IPv4 首部的 protocol 要设为 ICMP，ttl 也要设为一个非 0 值。

```
ipv4_header = IPv4()
ipv4_header.protocol = IPProtocol.ICMP
ipv4_header.ttl = 64 + 1 # ttl will decrease by 1 in ha
```

在收到 ping 的时候，我们生成这个包。为了提高代码复用率（其实是想偷个懒），我们递归地调用转发包的函数(这步骚操作,我自己都佩服我自己),即把刚生成的 reply 作为一个普通的转发的包来对待。这样，就可以处理查表、ARP 请求等一系列可能会遇到的问题。

```
# Check if the packet is destined for this router, drop it
if ip_header.dst in [iface.ipaddr for iface in self.net.interfaces()]:
    # ICMP request
    if ip_header.protocol is IPProtocol.ICMP and packet.get_header(ICMP).icmptype
        icmp_reply = self.make_icmp_packet(recv, icmp_type=ICMPType.EchoReply)
        self.handle_packet((timestamp, ifaceName, icmp_reply))
```

这样递归调用，代价也是有的。首先，转发时，TTL 字段减 1，所以正如上图，我们设置 TTL 时必须显式地加 1；其次，这会导致不必要的性能损耗，如重复检查 IP 首部、检查目的地是否合法等等。

## 3. Generating ICMP error messages

思路和上一部分几乎一样：在恰当的地方生成 ICMP 包，然后递归地调用转发包的函数。以下几个细节需要注意。

对于 ARP request 超过 5 次而没有回应，需要对等待该回应的每一个包的源发出 ICMP 错误。由于我们递归调用，转发包必须留到检查等待队列完毕一次性全部发送，

否则等待队列会在遍历时新加入包（那些不知道目的地 MAC 的 ICMP 错误报文）而报错。

```
if times >= 5:
    # del self.waiting_packets[next_hop_ip]
    time_out_record.append(next_hop_ip)
    # send icmp error
    for timestamp, ifaceName, packet in recv_packets
        icmp_error = self.make_icmp_packet(
            (timestamp, ifaceName, packet),
            icmp_type=ICMPType.DestinationUnreachabl
            icmp_code=ICMPTypeCodeMap[ICMPType.Desti
        )
        icmp_errors.append(icmp_error)
```

ICMP 源地址，不是像 ping 那样取自 request 的目的地，而是要定为路由器发出该包的接口。此外，`icmptype` 须先于 `icmpcode` 赋值，否则也会报错（这检查也忒严格了）。

```
# need to set src according to the interface of router
ipv4_header.src = self.net.interface_by_name(ifaceName).ipaddr
ipv4_header.dst = ori_packet.get_header(IPv4).src
icmp_header.icmptype = icmp_type
icmp_header.icmpcode = icmp_code
```

对于 `icmpdata` 字段，我们需要将其存储原数据包从 IPv4 首部开始的 28 字节。所以，我们应当先“剥下”以太网首部（保险起见，我后面又给贴回去了）。

```
# we need first 28 bytes from ipv4 header, so ethernet_header has to be deleted
del ori_packet[Ethernet]
icmp_header.icmpdata.data = ori_packet.to_bytes()[:28]
# restore ethernet header of original packet
ori_packet.add_header(ethernet_header)
```

这样，连带调用的地方，我们成功地把本实验的代码量硬生生压缩到了不到 50 行！

```
220
221 >     def make_icmp_packet(self,
254
```

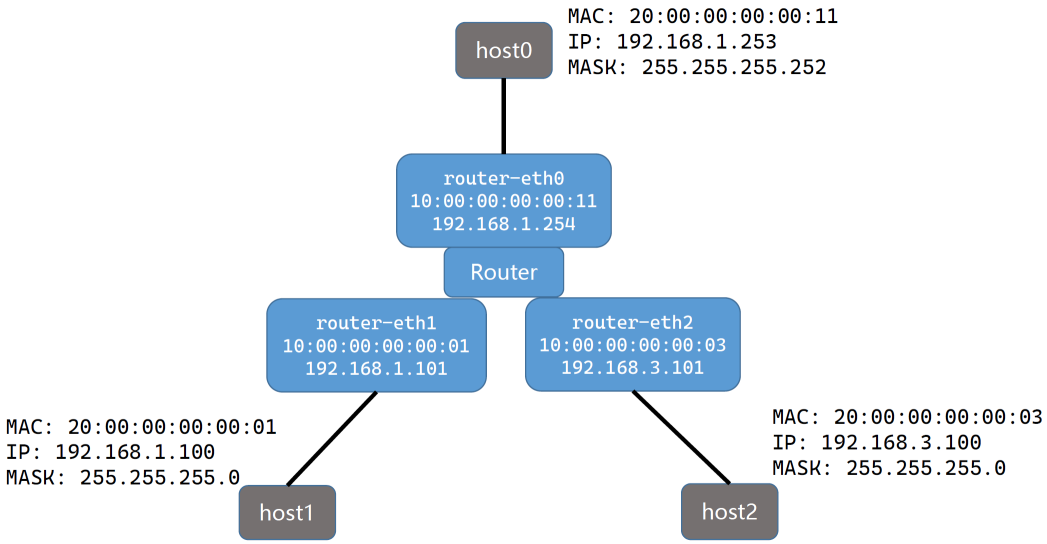
# 4. Testing

首先，贴下绿油油的官方测试集！（单线程和多线程均通过测试）

```
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!
```

在我们自己的测试集中，直接沿用上个实验设计的拓扑。这里再贴一下网络图和初始的路由器转发表。




| IP | Mask | Next Hop | Interface |
|---|---|---|---|
| 192.168.1.0 | 255.255.255.0 | 192.168.1.100 | router-eth1 |
| 192.168.1.252 | 255.255.255.252 | 192.168.1.253 | router-eth0 |
| 192.168.3.0 | 255.255.255.0 | 192.168.3.100 | router-eth2 |

我们的几个测试用例简介如下：

- host0 向路由器发送 ARP 请求，以获得 eth0 接口的 MAC；同时，路由器应记住 host0 的 MAC；
- host0 向路由器发出 ping，路由器应直接 ping 回去；
- host1 向路由器发出 ping，路由器先通过 ARP 询问 host1 的 MAC，得到回复后在 ping 回去；
- host1 向路由器端口发出一个普通的 IPv4 包，路由器应向 host1 发送一个 ICMP 错误；
- host1 向 host2 发出一个 TTL 为 1 的 IPv4 包，路由器应丢弃该包，并向 host1 发送一个 ICMP 错误；
- host1 向某个不在路由器表中的 IP 发送一个普通的 IPv4 包，路由器应丢弃该包，并向 host1 发送一个 ICMP 错误；
- host0、host1 先后向 host2 发送一个普通 IPv4 包，路由器将向 host2 发出 ARP 询问。出于某种原因，host2 没有回复。在连续询问 5 次后，路由器将先后向 host0、host1 发送一个 ICMP 错误。

详细的测试用例的源代码可以看 Github 上的提交文件。这里仅展示一下最终成果。

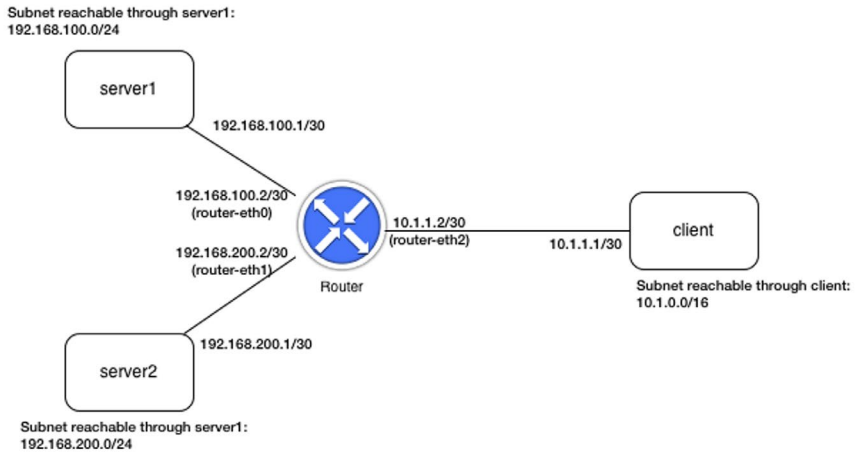
```
18  An IPv4 packet should arrive on router-eth1
19  An ICMP error should arrive on router-eth1
20  Other hosts should not receive any packet
21  An IPv4 packet should arrive on router-eth0
22  An ARP request should arrive on router-eth2
23  An IPv4 packet should arrive on router-eth1
24  router should resend ARP request
25  router should resend ARP request
26  router should resend ARP request
27  router should resend ARP request
28  An ICMP error should arrive on router-eth0
29  An ICMP error should arrive on router-eth1
30  Other hosts should not receive any packet


All tests passed!
```

吐槽一下，测试用例的代码量和出的 BUG，比写路由器本身还多……

## 5. Deploying

在仿真测试环境下，我们的配置和转发表如下图：




| IP | Mask | Next Hop | Interface |
|---|---|---|---|
| 192.168.100.0 | 255.255.255.0 | 192.168.100.1 | router-eth0 |
| 192.168.200.0 | 255.255.255.0 | 192.168.200.1 | router-eth1 |
| 10.1.0.0 | 255.255.0.0 | 10.1.1.1 | router-eth2 |

在这里，我们把注意力聚焦在 server1 上。

首先，`server1 ping -c1 192.168.100.2`(router-eth0)，此时路由器端口将给予一个 reply。

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | Private_00:00:01 | Broadcast | ARP | 42 | Who has 192.168.100. |
| 2 | 0.070090975 | 40:00:00:00:00:01 | Private_00:00:01 | ARP | 42 | 192.168.100.2 is at |
| 3 | 0.070107447 | 192.168.100.1 | 192.168.100.2 | ICMP | 98 | Echo (ping) request |
| 4 | 0.172703614 | 192.168.100.2 | 192.168.100.1 | ICMP | 98 | Echo (ping) reply |

接着，`server1 ping -c1 -t1 10.1.1.1`(client)，由于在路由器内 TTL 将减为1，故 server1 将收到一个 ICMP error。

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 19.085348849 | 192.168.100.1 | 10.1.1.1 | ICMP | 98 | Echo (ping) request |
| 6 | 19.109753262 | 192.168.100.2 | 192.168.100.1 | ICMP | 70 | Time-to-live exceeded |

然后，给 server1 配置，告诉他 123.123.123.0/24 可到达（修改 start_mininet.py），

在 ping 这个地址。路由器查表没找到，将返回一个 ICMP error。

```
set_route(net, 'server1', '123.123.123.0/24', '192.168.100.2')
set_route(net, 'server2', '10.1.0.0/16', '192.168.200.2')
```

```
7 33.992142754  192.168.100.1   123.123.123.1   ICMP   98 Echo (ping) request  id=
8 34.094345819  192.168.100.2   192.168.100.1   ICMP   70 Destination unreachable
```

最后，`server1 traceroute 192.168.200.1`。server1 成功找出了到 server2 的路径!

```
mininet> server1 traceroute 192.168.200.1
traceroute to 192.168.200.1 (192.168.200.1), 64 hops max
  1   192.168.100.2  56.005ms  101.860ms  105.454ms
  2   192.168.200.1  304.389ms  101.187ms  105.331ms
```

上述过程使用多线程版完成。

## 6. A mysterious BUG

实测，无论是单线程还是多线程，在第一次 ping 时有概率（重点）出现无限递归调用的问题；然而，代码设计在理论上不可能出现这样的问题。

```
**********************************************************************

This is the Switchyard equivalent of the blue screen of death.
Here (repeating what's above) is the failure that occurred:

Traceback (most recent call last):
  File "/home/njucs/NetLabWork/lab-05-SkyerWalkery/syenv/lib/python3.6/site-pack
ages/switchyard/llnetreal.py", line 272, in main_real
    _start_usercode(usercode_entry_point, netobj, options.codearg)
RecursionError: maximum recursion depth exceeded while calling a Python object
**********************************************************************


I'm throwing you into the Python debugger (pdb) at the point of failure.
If you don't want pdb, use the --nopdb flag to avoid this fate.

> /usr/lib/python3.6/copy.py(146)deepcopy()
```

为了找出原因，我做了如下尝试。

完全仿照产生报错的发包，使用 Switchyard 写了 Test scenario，进行压力测试 `watch -n 10 swyard -t testcases/myrouter3_testscenario.srpy myrouter.py`。然

而，经过漫长的等待，一次错误都没有出现过。

接着，我在 Mininet 环境下，使用插桩（即到处插 print，输出相关信息，例如包）的方法检查。然而，经过无数次尝试，仍然一次错误都没有出现……

最后，删除插桩，重装 python 虚拟环境，再次测试，BUG 又回来了（而且还是概率性触发）。

该 BUG 已经超出我目前的理解范围，故保留原代码，也希望助教可以在闲暇之余指点迷津。

- ***后续更新***

在启动 mininet 后，怀疑仿真环境下，链路中的包可能在关掉 xterm 后仍然存在。这意味着，同一次 start mininet 测试下，先前的测试可能会对后续测试产生影响。

# 四、 核心代码

要不您往上边翻一翻，都有截图；或者，上 Github 看看代码？

# 五、 总结与感想

- 相比起上个实验，似乎简单了点？
- 当要实现的功能较多时，设计代码结构、提高代码复用率大有讲究。本实验在 lab4 的基础上对不少地方的结构设计做了调整；
- 协议之间紧密关联，设计确实精妙；
- 看到自己的路由器正常工作时，确实蛮有成就感的。