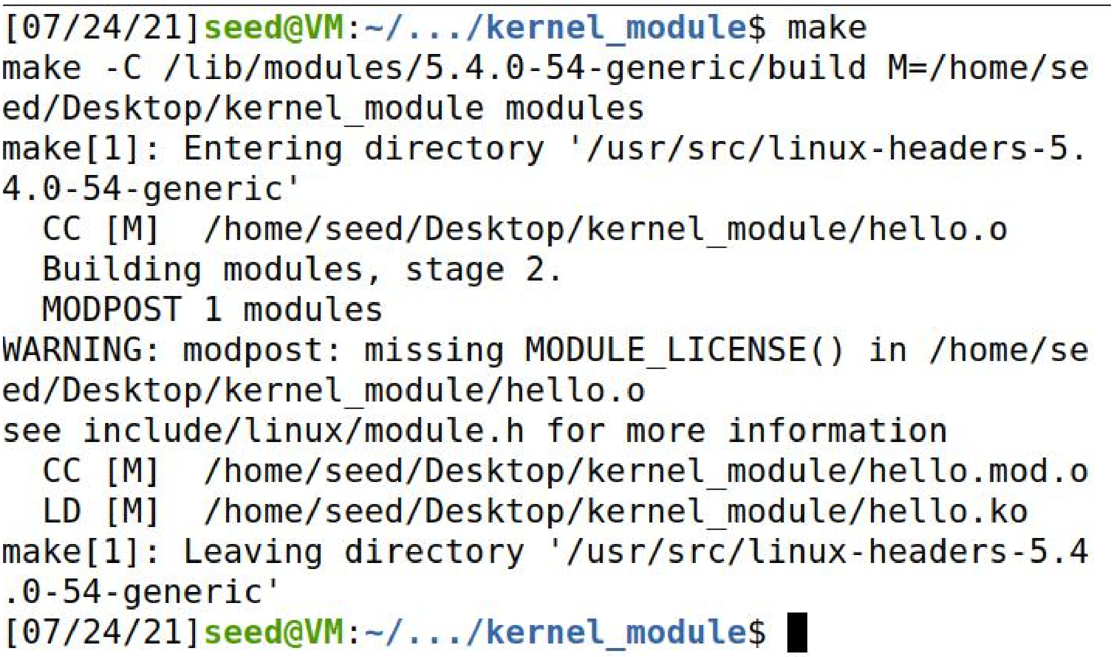
Firewall Exploration Lab

学号：57118225 姓名：宋雨帆

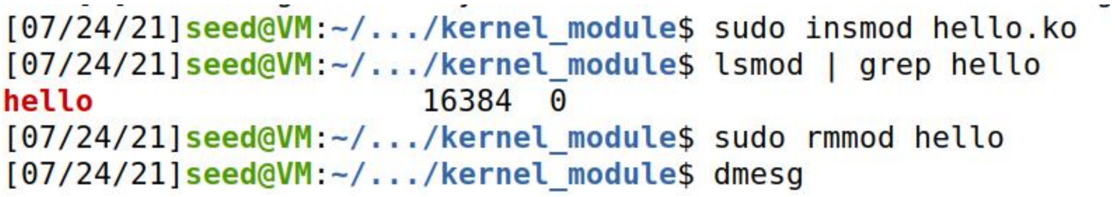
Task1.A

（1）将 kernel_module 移动到桌面，并编译



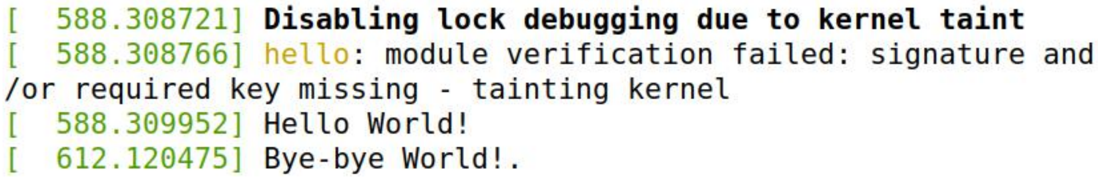
```
[07/24/21]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/Desktop/kernel_module/hello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/Desktop/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/24/21]seed@VM:~/.../kernel_module$ 
```

（2）输入指令添加模块



```
[07/24/21]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[07/24/21]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[07/24/21]seed@VM:~/.../kernel_module$ sudo rmmod hello
[07/24/21]seed@VM:~/.../kernel_module$ dmesg
```

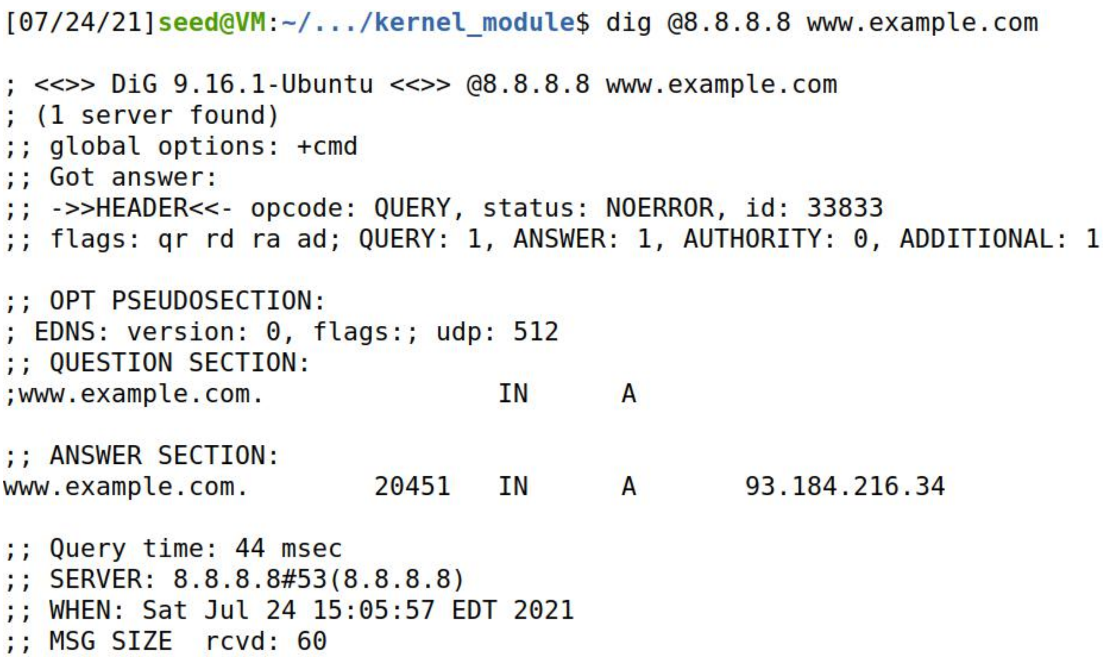
（3）安装成功



```
[  588.308721] Disabling lock debugging due to kernel taint
[  588.308766] hello: module verification failed: signature and/or required key missing - tainting kernel
[  588.309952] Hello World!
[  612.120475] Bye-bye World!.
```

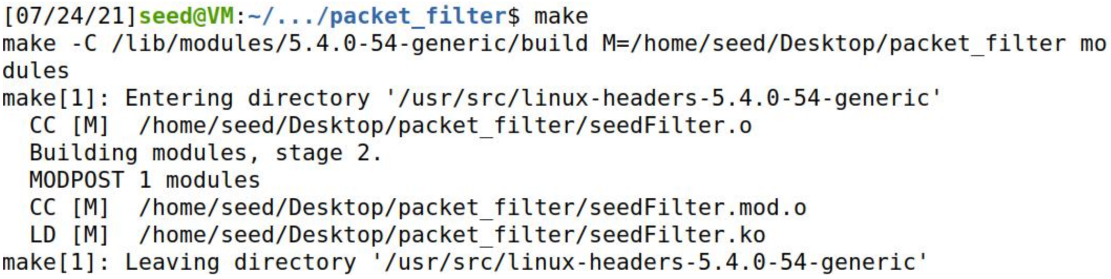
Task1.B.1

（1）攻击前使用 dig 指令进行查看



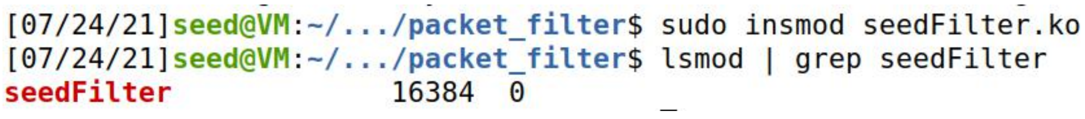
```
[07/24/21]seed@VM:~/.../kernel_module$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33833
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        20451   IN      A       93.184.216.34

;; Query time: 44 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sat Jul 24 15:05:57 EDT 2021
;; MSG SIZE  rcvd: 60
```

（2）编译 packet_filter 内核



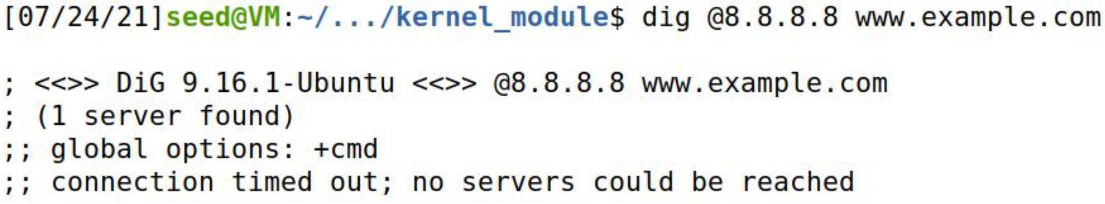
```
[07/24/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter mo
dules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

（3）输入指令加载模块



```
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

（4）重新输入 dig 指令

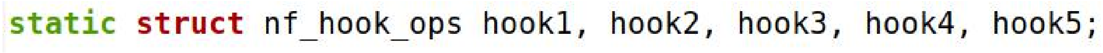


```
[07/24/21]seed@VM:~/.../kernel_module$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

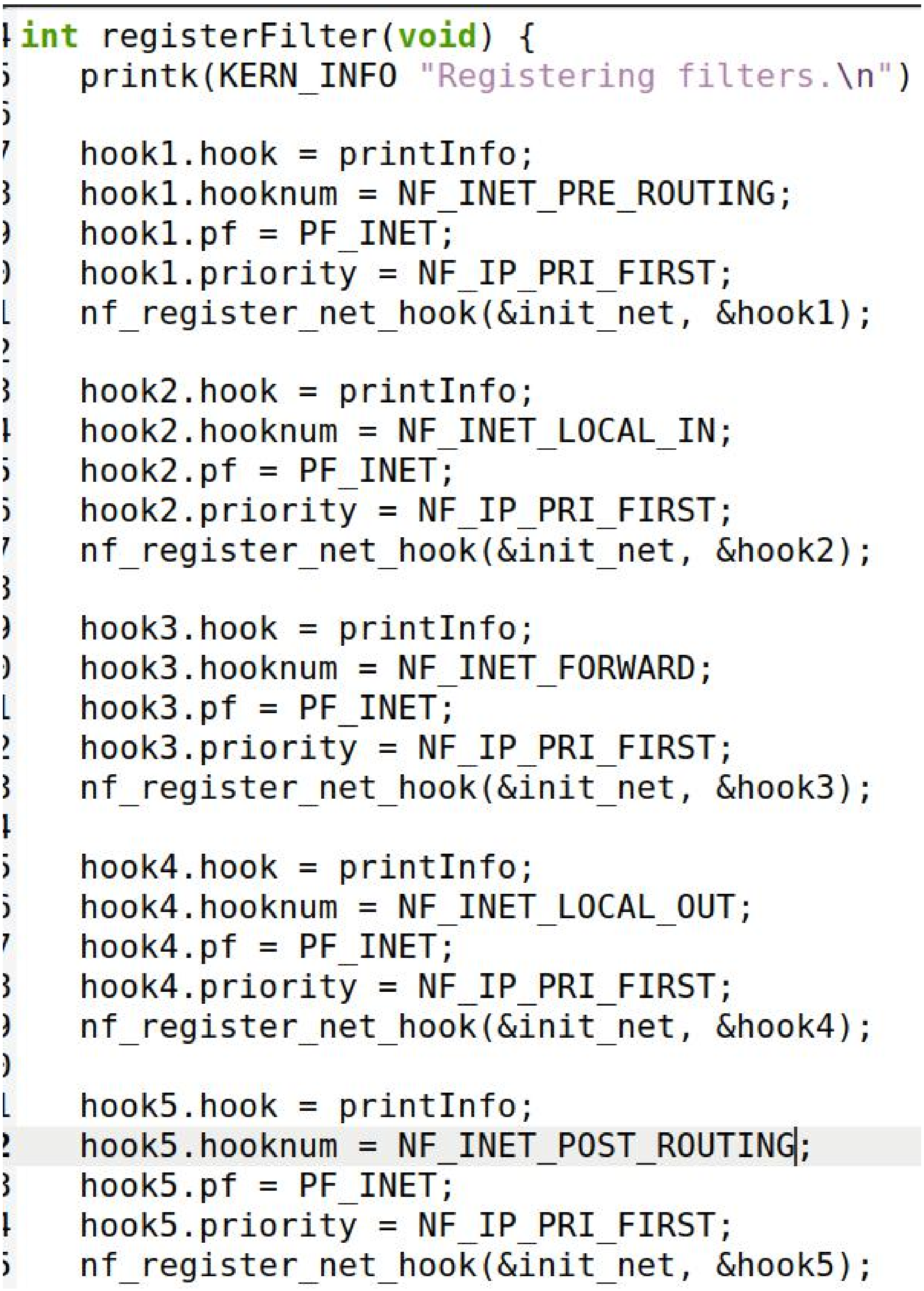
可以看到无法连接

Task1.B.2

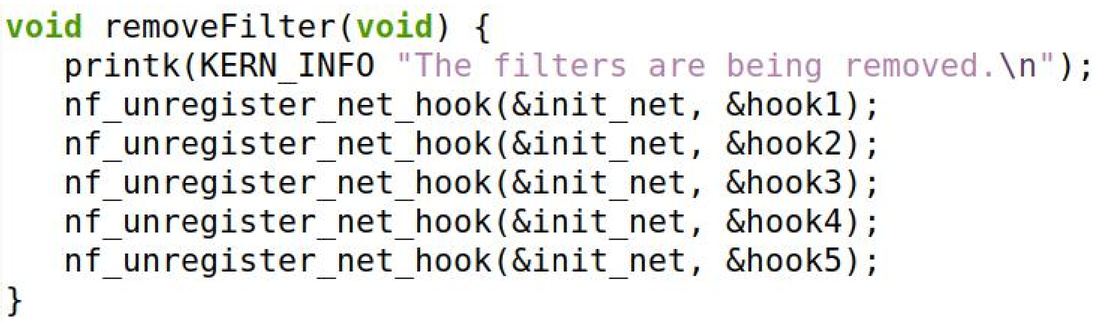
（1）修改程序，新增 hook 并将其与 printInfo 关联



```
static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5;
```

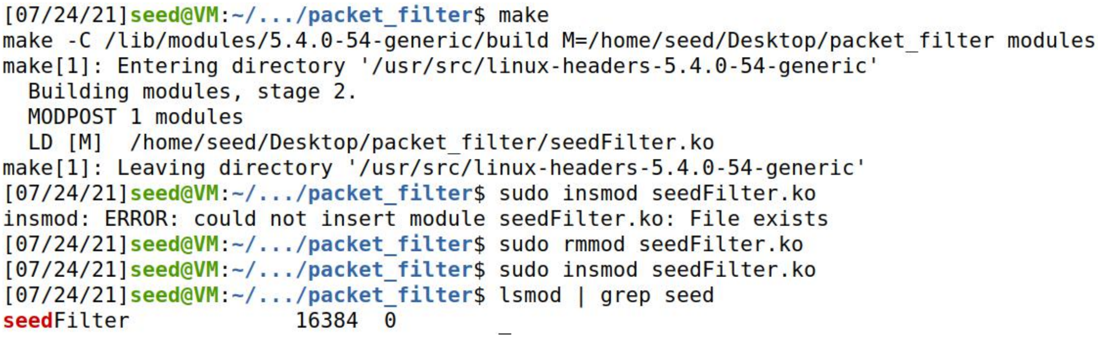


```c
int registerFilter(void) {
    printk(KERN_INFO "Registering filters.\n")

    hook1.hook = printInfo;
    hook1.hooknum = NF_INET_PRE_ROUTING;
    hook1.pf = PF_INET;
    hook1.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook1);

    hook2.hook = printInfo;
    hook2.hooknum = NF_INET_LOCAL_IN;
    hook2.pf = PF_INET;
    hook2.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook2);

    hook3.hook = printInfo;
    hook3.hooknum = NF_INET_FORWARD;
    hook3.pf = PF_INET;
    hook3.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook3);

    hook4.hook = printInfo;
    hook4.hooknum = NF_INET_LOCAL_OUT;
    hook4.pf = PF_INET;
    hook4.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook4);

    hook5.hook = printInfo;
    hook5.hooknum = NF_INET_POST_ROUTING;
    hook5.pf = PF_INET;
    hook5.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook5);
```



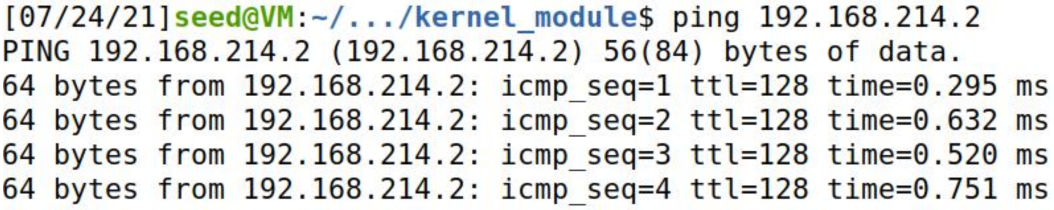
```c
void removeFilter(void) {
   printk(KERN_INFO "The filters are being removed.\n");
   nf_unregister_net_hook(&init_net, &hook1);
   nf_unregister_net_hook(&init_net, &hook2);
   nf_unregister_net_hook(&init_net, &hook3);
   nf_unregister_net_hook(&init_net, &hook4);
   nf_unregister_net_hook(&init_net, &hook5);
}
```

（2）重新编译和加载



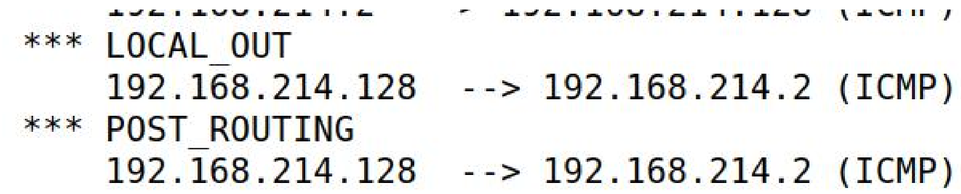
```
[07/24/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/seed/Desktop/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
insmod: ERROR: could not insert module seedFilter.ko: File exists
[07/24/21]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
```

（3）ping 内网，输入 dmesg 查看日志



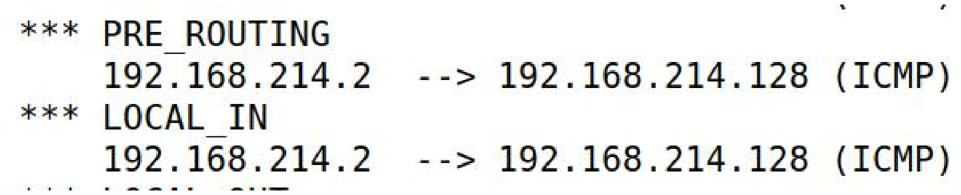
```
[07/24/21]seed@VM:~/.../kernel_module$ ping 192.168.214.2
PING 192.168.214.2 (192.168.214.2) 56(84) bytes of data.
64 bytes from 192.168.214.2: icmp_seq=1 ttl=128 time=0.295 ms
64 bytes from 192.168.214.2: icmp_seq=2 ttl=128 time=0.632 ms
64 bytes from 192.168.214.2: icmp_seq=3 ttl=128 time=0.520 ms
64 bytes from 192.168.214.2: icmp_seq=4 ttl=128 time=0.751 ms
```

可以看到，对于本地产生向外部网络发出的数据包，会依次经过 NF_INET_LOCAL_OUT 和 NF_INET_POST_ROUTING 两个 hook 点的处理。推测前者是本地产生包时触发，后者是向外发送时触发。



```
 *** LOCAL_OUT
     192.168.214.128  --> 192.168.214.2 (ICMP)
 *** POST_ROUTING
     192.168.214.128  --> 192.168.214.2 (ICMP)
```

发往本地的数据包，会依次经过 NF_INET_PRE_ROUTING 和 NF_INET_LOCAL_IN 两个 hook 点的处理。推测前者是判断是发往外部还是本地接收时触发，后者是发往本地时触发。



```
 *** PRE_ROUTING
     192.168.214.2  --> 192.168.214.128 (ICMP)
 *** LOCAL_IN
     192.168.214.2  --> 192.168.214.128 (ICMP)
```

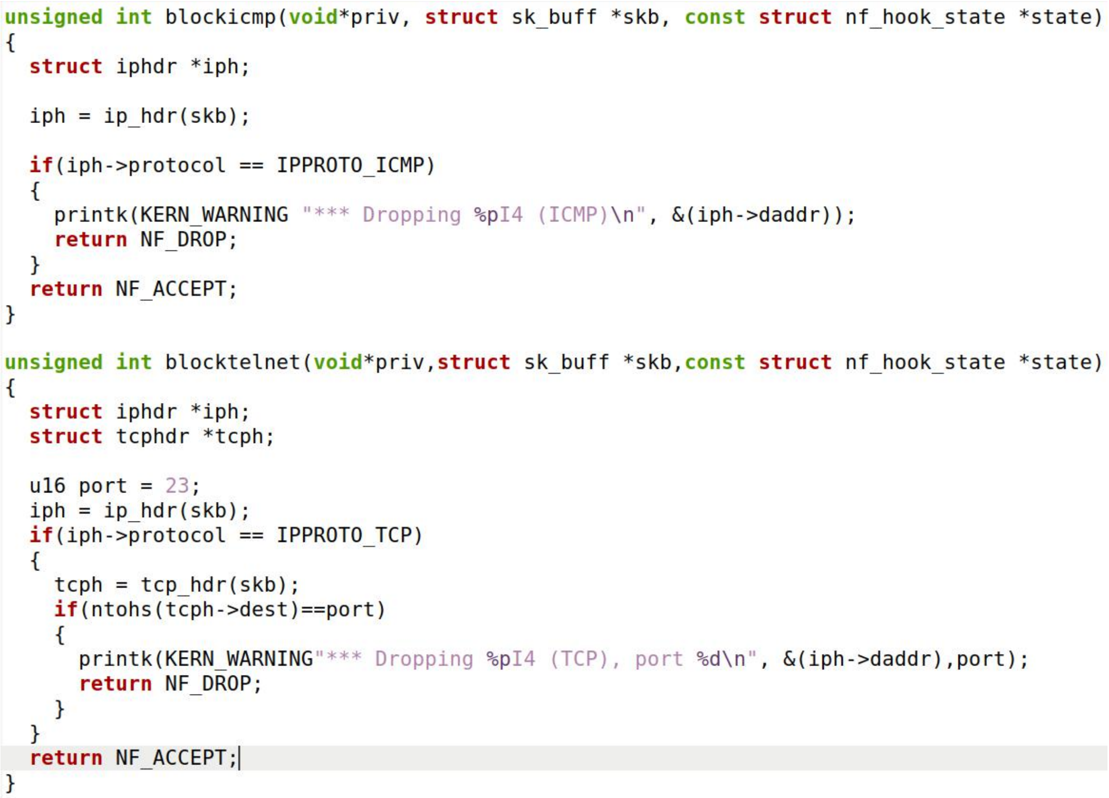
在日志中没有出现 NF_INET_FORWARD 点的活动

因此得到结论：

NF_INET_LOCAL_OUT	触发条件：由本地产生的包
NF_INET_POST_ROUTING	触发条件：向外部网络发出的包
NF_INET_PRE_ROUTING	触发条件：本地接收到的包（不包括本地产生的），用于判断是否向外转发
NF_INET_LOCAL_IN	触发条件：发往本地而不需要转发的包
NF_INET_FORWARD	触发条件：需要向外转发的数据包

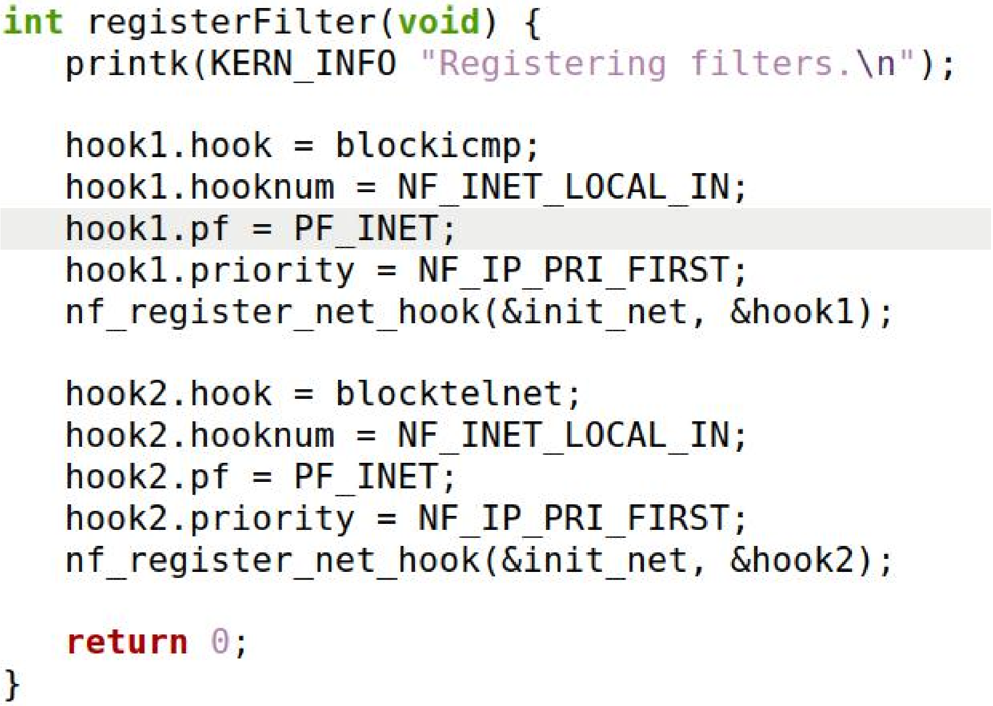
Task1.B.3

（1）增加两个 bock 函数，分别用于拦截 icmp 报文和 telnet 报文



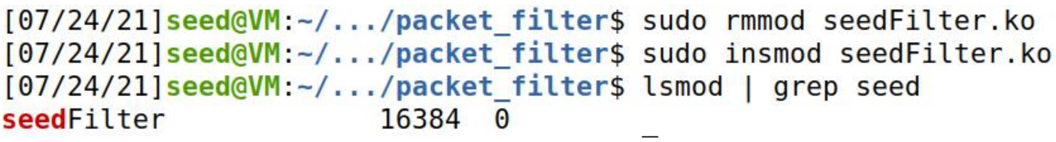
```
unsigned int blockicmp(void*priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
  struct iphdr *iph;

  iph = ip_hdr(skb);

  if(iph->protocol == IPPROTO_ICMP)
  {
    printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n", &(iph->daddr));
    return NF_DROP;
  }
  return NF_ACCEPT;
}

unsigned int blocktelnet(void*priv,struct sk_buff *skb,const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  u16 port = 23;
  iph = ip_hdr(skb);
  if(iph->protocol == IPPROTO_TCP)
  {
    tcph = tcp_hdr(skb);
    if(ntohs(tcph->dest)==port)
    {
      printk(KERN_WARNING"*** Dropping %pI4 (TCP), port %d\n", &(iph->daddr),port);
      return NF_DROP;
    }
  }
  return NF_ACCEPT;
}
```

其中可以看到，这里没有判断目标 ip 是否为主机，这是因为利用 task1.B.2 中的知识，所有触发 NF_INET_LOCAL_IN 的都是发往本地的包，因此无需再比较 ip，只需要将函数注册为 NF_INET_LOCAL_IN 点即可。



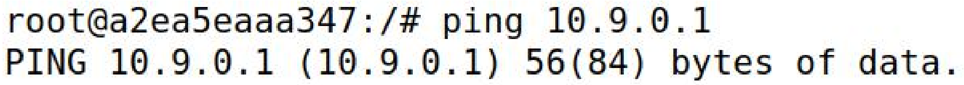
```
int registerFilter(void) {
   printk(KERN_INFO "Registering filters.\n");

   hook1.hook = blockicmp;
   hook1.hooknum = NF_INET_LOCAL_IN;
   hook1.pf = PF_INET;
   hook1.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook1);

   hook2.hook = blocktelnet;
   hook2.hooknum = NF_INET_LOCAL_IN;
   hook2.pf = PF_INET;
   hook2.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook2);

   return 0;
}
```

（2）重新编译并加载模块



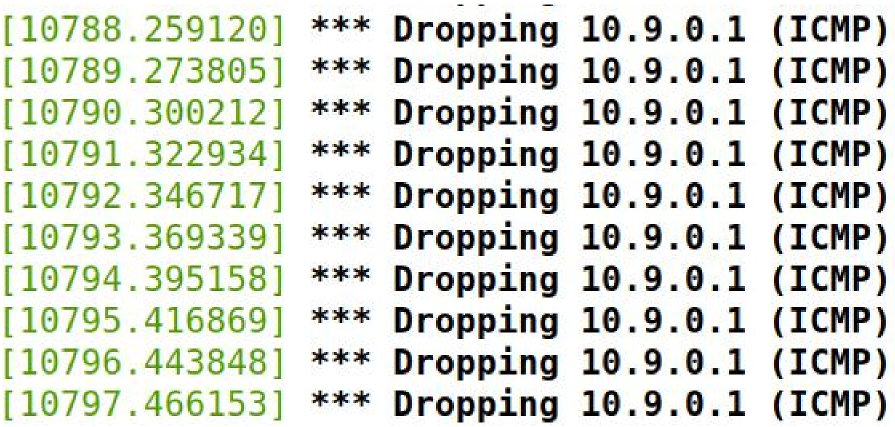
```
[07/24/21]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
```

（3）登录 hostA 主机，并 ping 10.9.0.1（虚拟机），可以看到没有 ping 通



```
root@a2ea5eaaa347:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
```

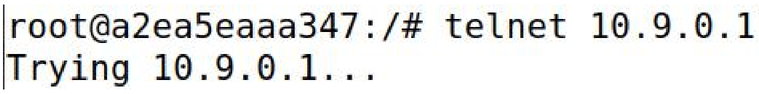
输入 dmesg 查看



```
[10788.259120] *** Dropping 10.9.0.1 (ICMP)
[10789.273805] *** Dropping 10.9.0.1 (ICMP)
[10790.300212] *** Dropping 10.9.0.1 (ICMP)
[10791.322934] *** Dropping 10.9.0.1 (ICMP)
[10792.346717] *** Dropping 10.9.0.1 (ICMP)
[10793.369339] *** Dropping 10.9.0.1 (ICMP)
[10794.395158] *** Dropping 10.9.0.1 (ICMP)
[10795.416869] *** Dropping 10.9.0.1 (ICMP)
[10796.443848] *** Dropping 10.9.0.1 (ICMP)
[10797.466153] *** Dropping 10.9.0.1 (ICMP)
```

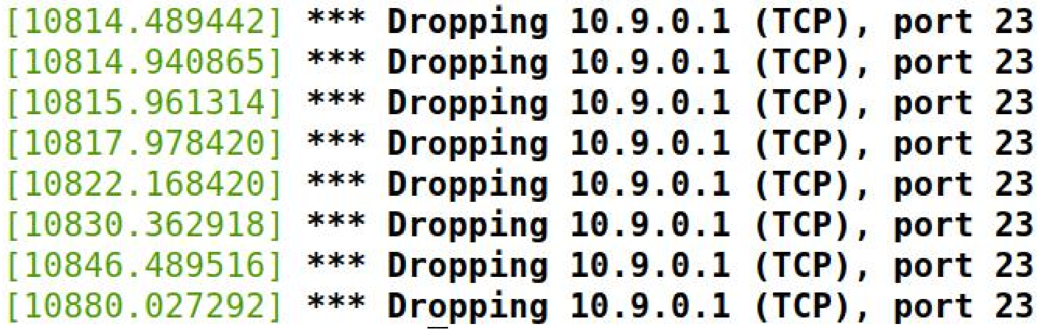
可以看到，icmp 报文 都被丢弃了，说明拦截成功

（4）使用 telnet 10.9.0.1 尝试登录，可以看到登录失败



```
root@a2ea5eaaa347:/# telnet 10.9.0.1
Trying 10.9.0.1...
```

输入 dmesg 命令查看

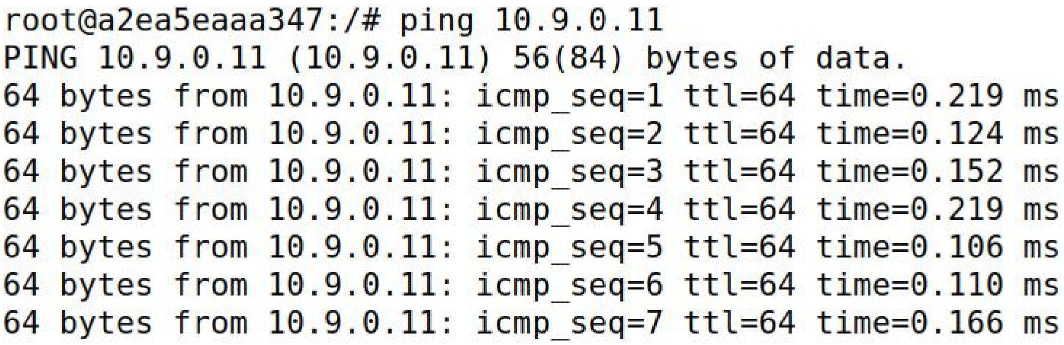


```
[10814.489442] *** Dropping 10.9.0.1 (TCP), port 23
[10814.940865] *** Dropping 10.9.0.1 (TCP), port 23
[10815.961314] *** Dropping 10.9.0.1 (TCP), port 23
[10817.978420] *** Dropping 10.9.0.1 (TCP), port 23
[10822.168420] *** Dropping 10.9.0.1 (TCP), port 23
[10830.362918] *** Dropping 10.9.0.1 (TCP), port 23
[10846.489516] *** Dropping 10.9.0.1 (TCP), port 23
[10880.027292] *** Dropping 10.9.0.1 (TCP), port 23
```

可以看到，丢弃了很多 tcp 报文，从而阻止了登录，说明拦截成功。

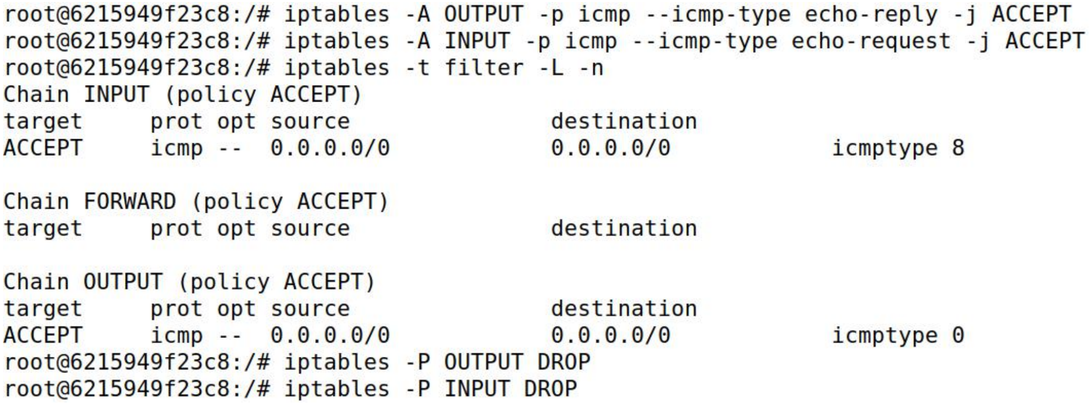
Task2.A

（1）未进行操作前，在主机上 ping 路由器，成功



```
root@a2ea5eaaa347:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.219 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.124 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.152 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.219 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.106 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.110 ms
64 bytes from 10.9.0.11: icmp_seq=7 ttl=64 time=0.166 ms
```

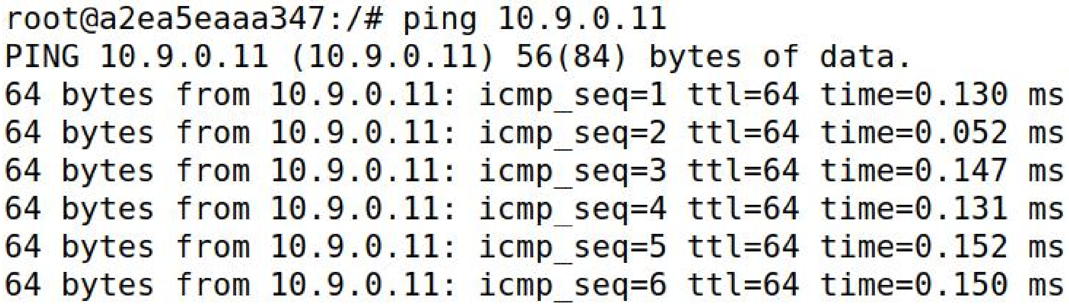
（2）输入规则



```
root@6215949f23c8:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@6215949f23c8:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@6215949f23c8:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
root@6215949f23c8:/# iptables -P OUTPUT DROP
root@6215949f23c8:/# iptables -P INPUT DROP
```

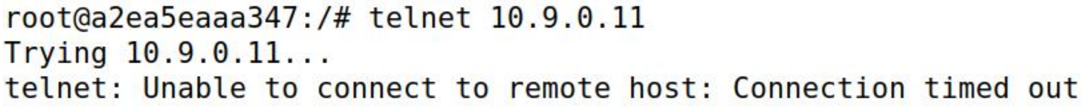
（3）测试

使用 ping 指令，成功



```
root@a2ea5eaaa347:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.130 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.147 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.131 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.152 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.150 ms
```

使用 telnet 指令，失败

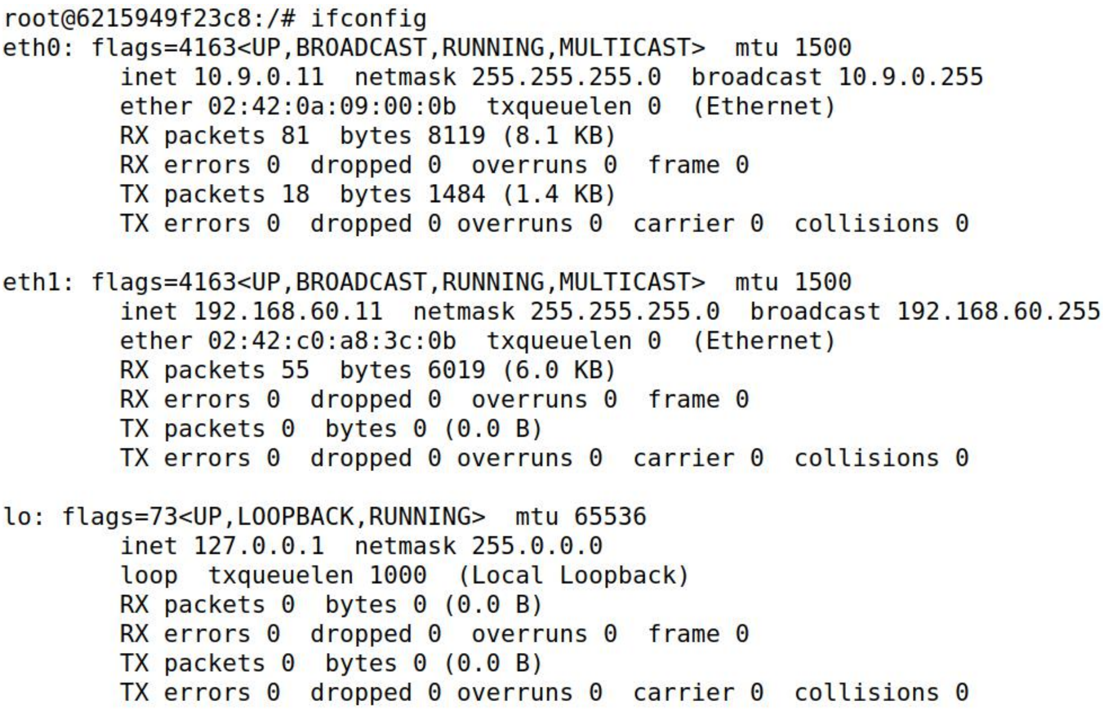


```
root@a2ea5eaaa347:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

说明规则编写正确。

Task2.B

（1）在路由器上输入 ifconfig 查看端口对应 ip



```
root@6215949f23c8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 81  bytes 8119 (8.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1484 (1.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.11  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:0b  txqueuelen 0  (Ethernet)
        RX packets 55  bytes 6019 (6.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

可以看到10.9.0.11对应的端口(eth0)是面向外网的，192.168.60.11(eth1)对应的端口是面向内网的。

（2）根据要求指定规则如下：

①OUTPUT INPUT FORWARD丢弃

```
iptables -P OUTPUT DROP
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

（不允许内外流量交互）

②对于FORWARD 只有icmp请求报文由内部端口（eth1)进入，外部端口（eth0)，才接收：

```
iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-request -o eth0 -j ACCEPT
```

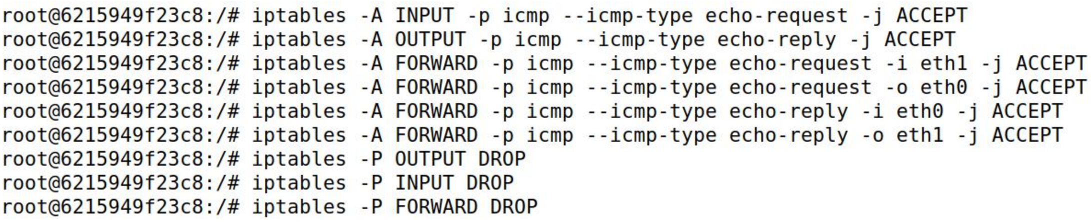
对于FORWARD 只有icmp应答报文由外部端口（eth0）进入，外部端口（eth1)，才接收：

```
iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-reply -o eth1 -j ACCEPT
```

（从而保证外部不能ping内部，内部能ping外部）

③对于input和output ，允许输入的icmp请求和应答报文，保证外部主机能够ping路由器

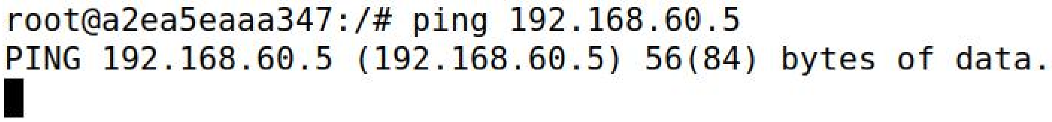
```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
```



```
root@6215949f23c8:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@6215949f23c8:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -p icmp --icmp-type echo-request -o eth0 -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -o eth1 -j ACCEPT
root@6215949f23c8:/# iptables -P OUTPUT DROP
root@6215949f23c8:/# iptables -P INPUT DROP
root@6215949f23c8:/# iptables -P FORWARD DROP
```

（3）测试

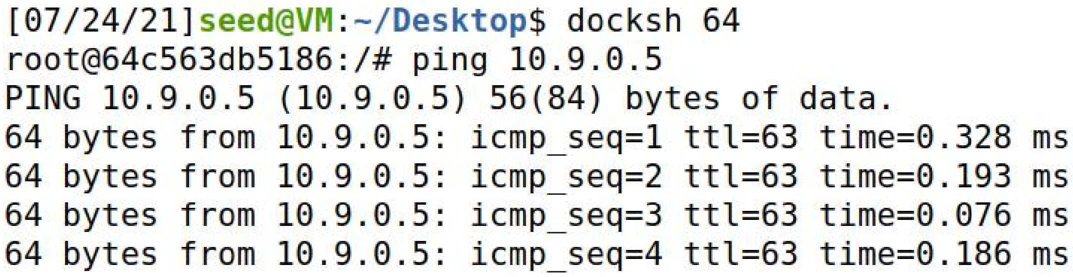
①外网 ping 内网



```
root@a2ea5eaaa347:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

不能实现 ping 操作，符合要求。

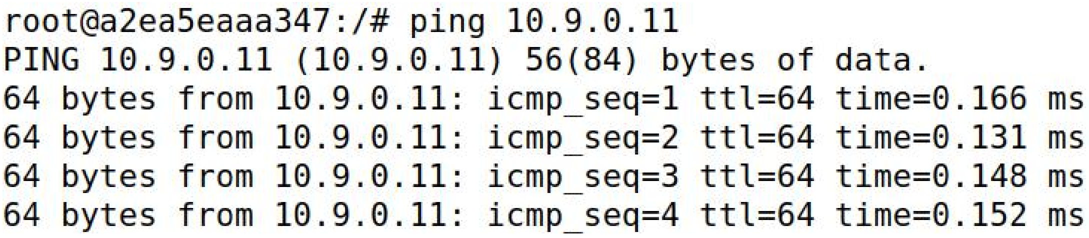
②内网 ping 外网



```
[07/24/21]seed@VM:~/Desktop$ docksh 64
root@64c563db5186:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.328 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.193 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.076 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.186 ms
```

能够实现 ping 操作，符合要求。

③外网 ping 路由

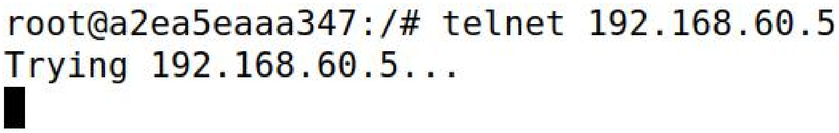


```
root@a2ea5eaaa347:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.166 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.148 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.152 ms
```

能够实现 ping 操作，符合要求。

④内外其它流量的交互

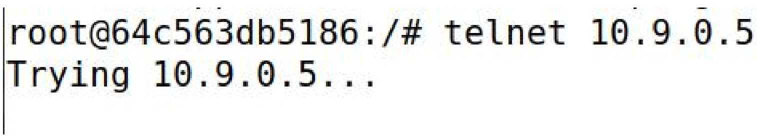
外网 telnet 内网：



```
root@a2ea5eaaa347:/# telnet 192.168.60.5
Trying 192.168.60.5...
```

无法登录，符合要求。

内网 telnet 外网：

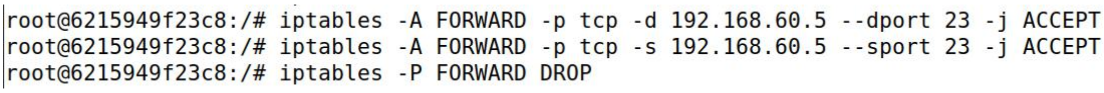


```
root@64c563db5186:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

无法登录，符合要求。

Task2.C

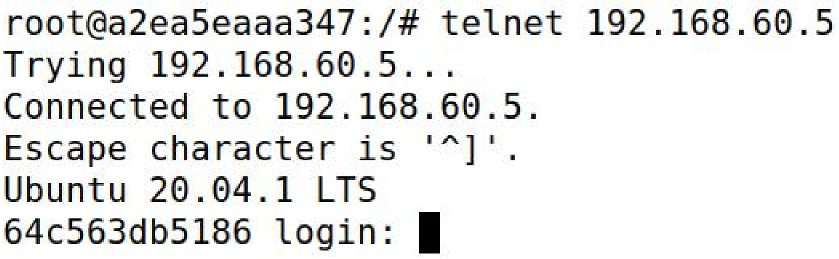
（1）设计策略如下



```
root@6215949f23c8:/# iptables -A FORWARD -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -p tcp -s 192.168.60.5 --sport 23 -j ACCEPT
root@6215949f23c8:/# iptables -P FORWARD DROP
```

只允许 192.168.60.5 主机的 23 端口的流量进行转发，保证其能被外部和内部主机登录，而外部主机到其它内部主机则不行；本地主机的相互访问不需要转发，因此 FORWARD 对其它报文的丢弃不会影响内部主机间的登录。

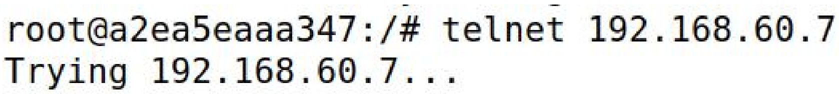
①外部 telnet192.168.60.5



```
root@a2ea5eaaa347:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
64c563db5186 login: 
```

成功。

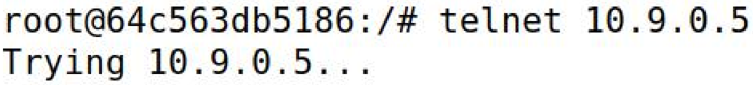
②外部 telnet 其它主机



```
root@a2ea5eaaa347:/# telnet 192.168.60.7
Trying 192.168.60.7...
```

无法登录，符合要求。

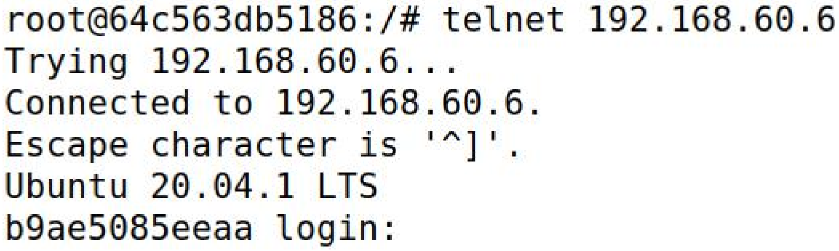
③内部 telnet 外部



```
root@64c563db5186:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

无法登录，符合要求。

④内部 telnet 内部



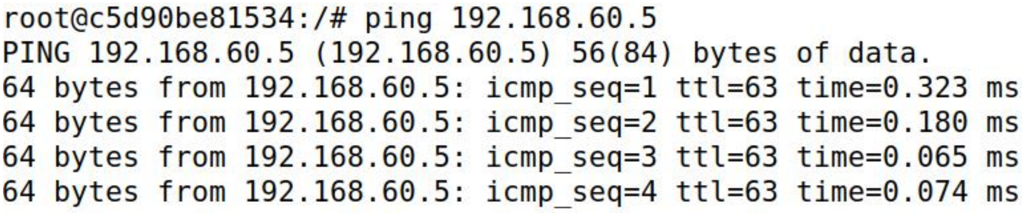
```
root@64c563db5186:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b9ae5085eeaa login:
```

成功。

Task3.A

ICMP

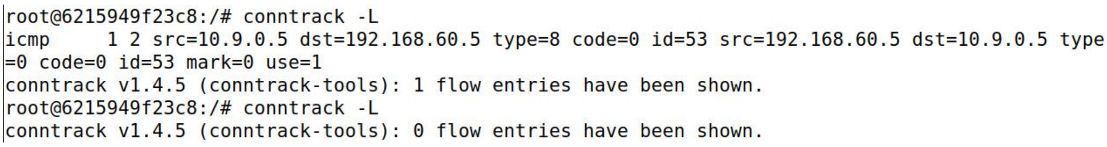
（1）在 10.9.0.5 上 ping 192.168.60.5 主机



```
root@c5d90be81534:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.323 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.180 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.074 ms
```

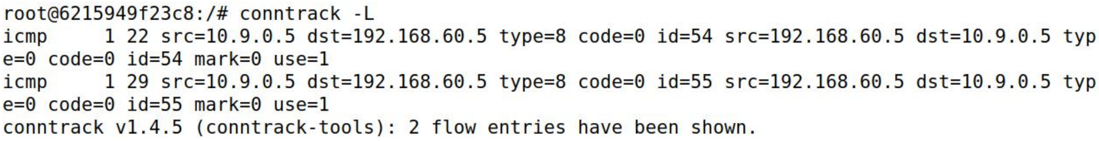
（2）查看追踪信息

可以看到一个 icmp。



```
root@6215949f23c8:/# conntrack -L
icmp     1 2 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=53 src=192.168.60.5 dst=10.9.0.5 type
=0 code=0 id=53 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@6215949f23c8:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

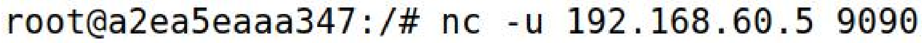
如果 30s 内再次 ping，则会显示两个 flow。



```
root@6215949f23c8:/# conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=54 src=192.168.60.5 dst=10.9.0.5 typ
e=0 code=0 id=54 mark=0 use=1
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=55 src=192.168.60.5 dst=10.9.0.5 typ
e=0 code=0 id=55 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
```

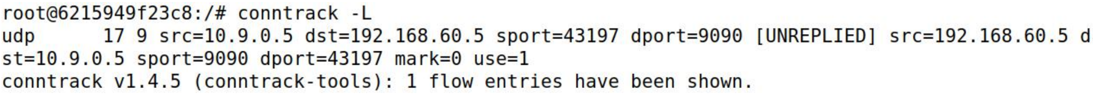
UDP

（1）在主机 10.9.0.5 上输入指令



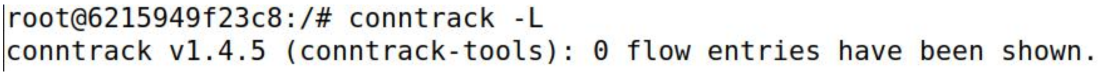
```
root@a2ea5eaaa347:/# nc -u 192.168.60.5 9090
```

（2）查看追踪信息



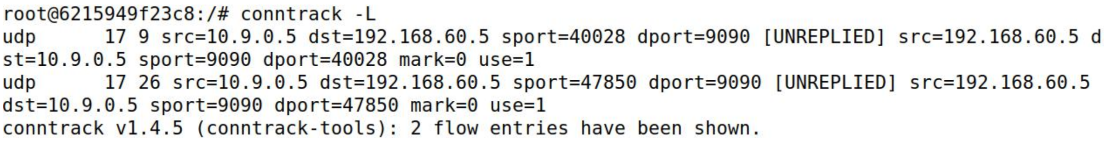
```
root@6215949f23c8:/# conntrack -L
udp      17 9 src=10.9.0.5 dst=192.168.60.5 sport=43197 dport=9090 [UNREPLIED] src=192.168.60.5 d
st=10.9.0.5 sport=9090 dport=43197 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

时间同样为 30s，且停止登录后会倒计时到 0。



```
root@6215949f23c8:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

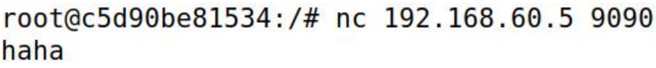
同样，30s 内再次连接会出现两条记录。



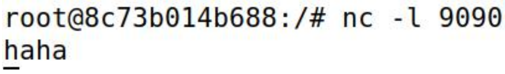
```
root@6215949f23c8:/# conntrack -L
udp      17 9 src=10.9.0.5 dst=192.168.60.5 sport=40028 dport=9090 [UNREPLIED] src=192.168.60.5 d
st=10.9.0.5 sport=9090 dport=40028 mark=0 use=1
udp      17 26 src=10.9.0.5 dst=192.168.60.5 sport=47850 dport=9090 [UNREPLIED] src=192.168.60.5
dst=10.9.0.5 sport=9090 dport=47850 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
```

TCP

（1）在当前机和目标主机上输入指令，建立连接

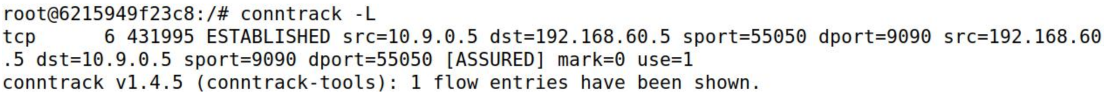


```
root@c5d90be81534:/# nc 192.168.60.5 9090
haha
```



```
root@8c73b014b688:/# nc -l 9090
haha
```

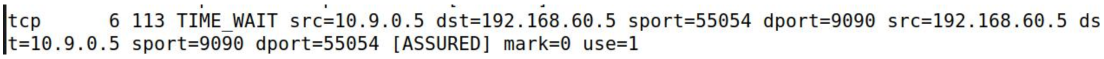
（2）查看追踪



```
root@6215949f23c8:/# conntrack -L
tcp      6 431995 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=55050 dport=9090 src=192.168.60
.5 dst=10.9.0.5 sport=9090 dport=55050 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

可以看到，存活时间在 430000s 以上

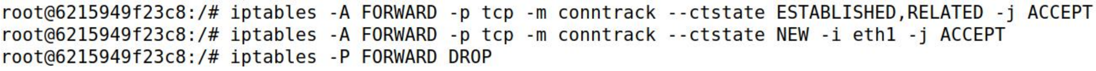
（3）结束连接，可以看到记录仍然存在，但只存在 120s，120s 倒计时结束后将消失。



```
tcp      6 113 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=55054 dport=9090 src=192.168.60.5 ds
t=10.9.0.5 sport=9090 dport=55054 [ASSURED] mark=0 use=1
```

Task3.B

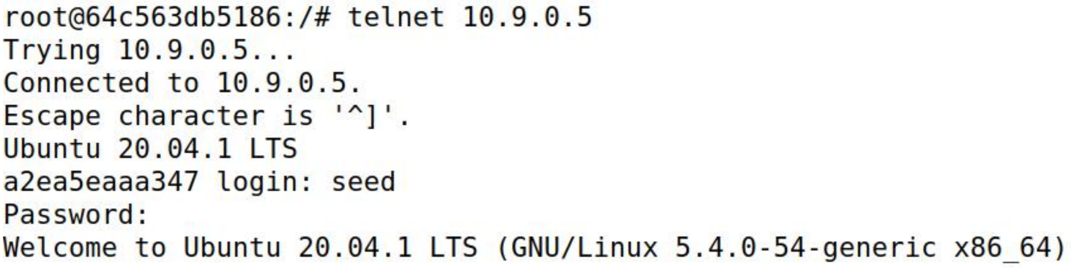
（1）编写规则如下



```
root@6215949f23c8:/# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -p tcp -m conntrack --ctstate NEW -i eth1 -j ACCEPT
root@6215949f23c8:/# iptables -P FORWARD DROP
```

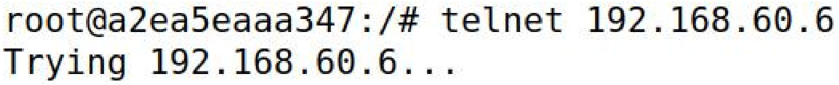
（2）测试

内网 telnet 外网，成功



```
root@64c563db5186:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a2ea5eaaa347 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

外网 telnet 内网，无法连接

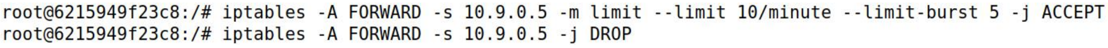


```
root@a2ea5eaaa347:/# telnet 192.168.60.6
Trying 192.168.60.6...
```

因此规则正确 ，符合实验要求。

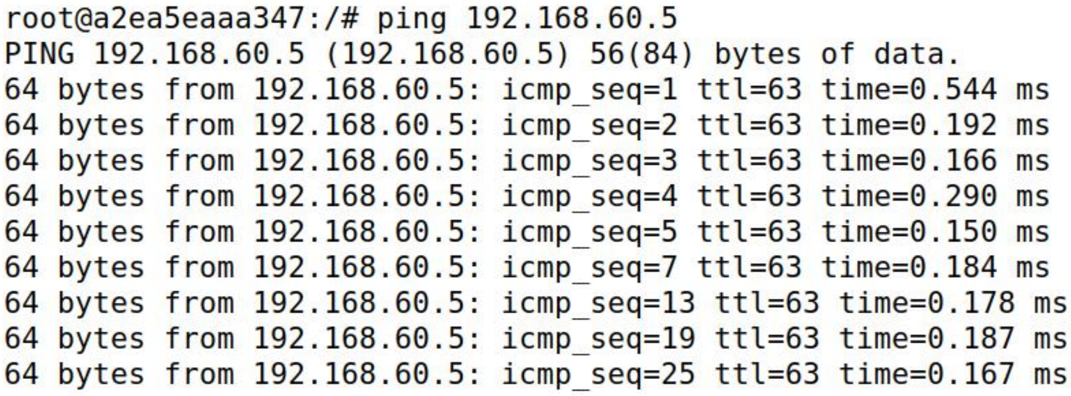
Task4

（1）在路由器输入规则



```
root@6215949f23c8:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
root@6215949f23c8:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
```

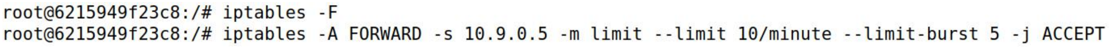
（2）在 10.9.0.5 主机尝试 ping



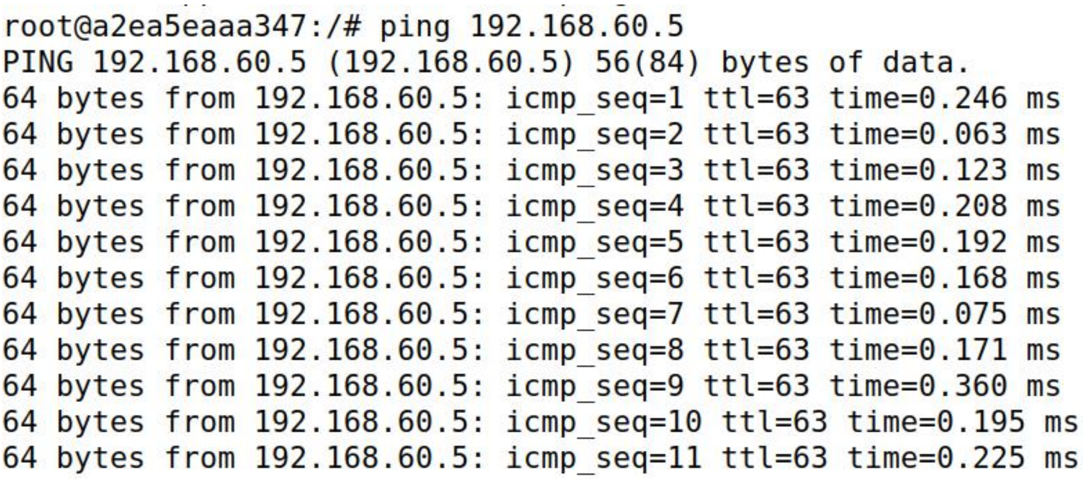
```
root@a2ea5eaaa347:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.544 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.192 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.166 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.290 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.150 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.184 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.178 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.187 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.167 ms
```

可以看到，前几个报文速度非常快，后面速度较慢，平均 6s 一个，符合要求，说明规则发挥了作用

（3）去掉第二条规则，再次尝试 ping



```
root@6215949f23c8:/# iptables -F
root@6215949f23c8:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
```



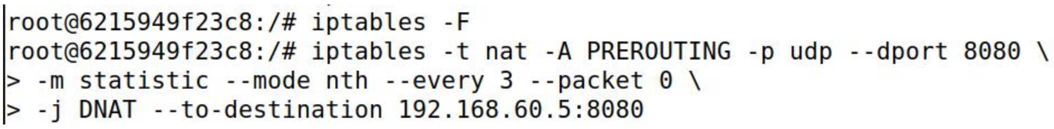
```
root@a2ea5eaaa347:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.246 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.063 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.123 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.208 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.192 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.168 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.075 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.171 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.360 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.195 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.225 ms
```

可以看到，报文发送速度很快，并没有减慢到每分钟 10 个，说明规则失效了。
这是因为没有第二条规则将报文设置为默认 DROP，所有报文（无论是否满足第一个规则）都会从默认的 ACCEPT 规则通过，从而导致第一条规则也失效。

Task5

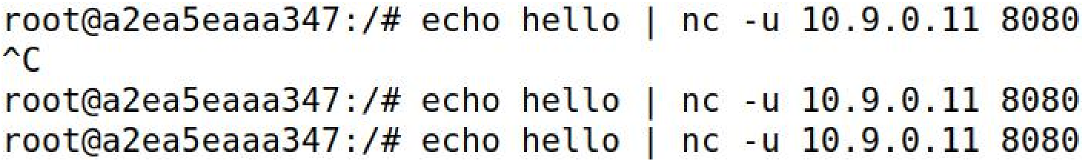
轮询：

（1）在路由器上输入规则



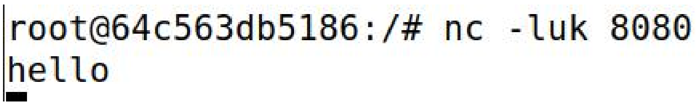
```
root@6215949f23c8:/# iptables -F
root@6215949f23c8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 \
> -m statistic --mode nth --every 3 --packet 0 \
> -j DNAT --to-destination 192.168.60.5:8080
```

（2）在 192.168.60.5 上开启 nc -luk 8080 监听，在 10.9.0.5 处连接并输入 hello



```
root@a2ea5eaaa347:/# echo hello | nc -u 10.9.0.11 8080
^C
root@a2ea5eaaa347:/# echo hello | nc -u 10.9.0.11 8080
root@a2ea5eaaa347:/# echo hello | nc -u 10.9.0.11 8080
```

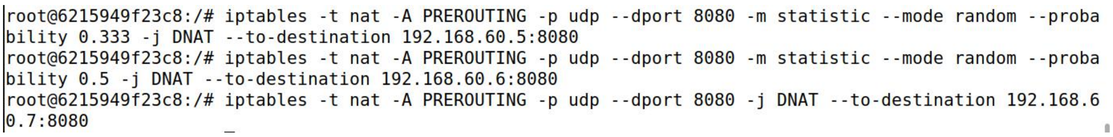
（3）在 192.168.60.5 上查看，可以看到发送者没发送 3 次，接收者再会收到 1 次路由器转发的报文，说明路由规则正确。



```
root@64c563db5186:/# nc -luk 8080
hello
```

随机：

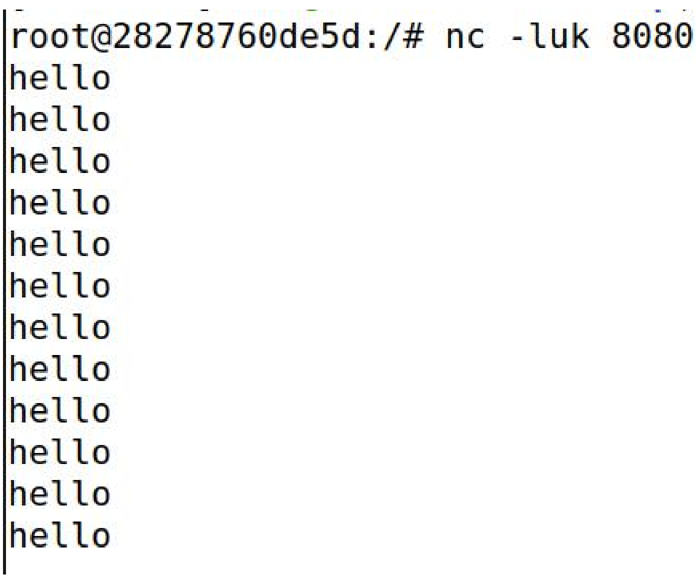
（1）指定规则如下



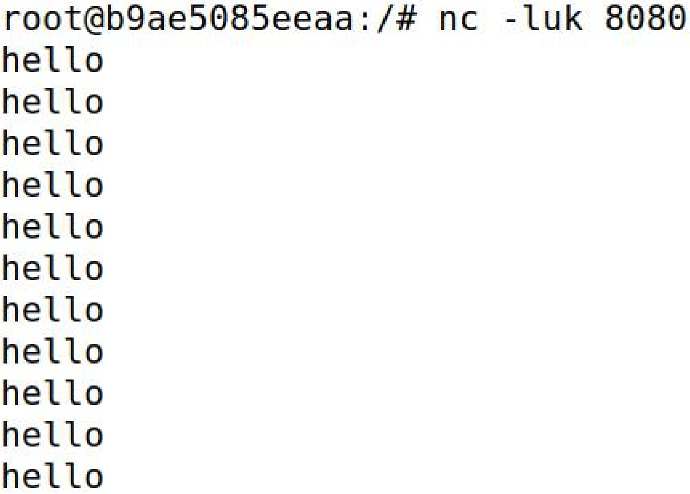
```
root@6215949f23c8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --proba
bility 0.333 -j DNAT --to-destination 192.168.60.5:8080
root@6215949f23c8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --proba
bility 0.5 -j DNAT --to-destination 192.168.60.6:8080
root@6215949f23c8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination 192.168.6
0.7:8080
```

其中第一个的概率为 0.333，第二个概率为 0.5，当前两个没有命中，则默认命中第三条

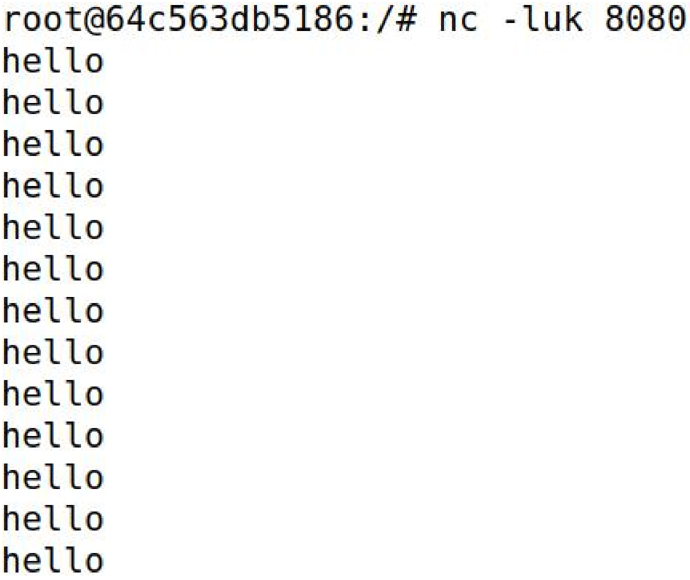
（2）在三台主机上都输入 nc -luk 8080 打开监听，在 10.9.0.5 主机上建立连接并不断输入 hello，得到三台主机上的输出结果如下：



```
root@28278760de5d:/# nc -luk 8080
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```



```
root@b9ae5085eeaa:/# nc -luk 8080
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```



```
root@64c563db5186:/# nc -luk 8080
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

可以看到路由确实实现了分发，由于样本数量太少，不能做到每个主机上收到的报文各 1/3，但总体上符合负载均衡的要求。