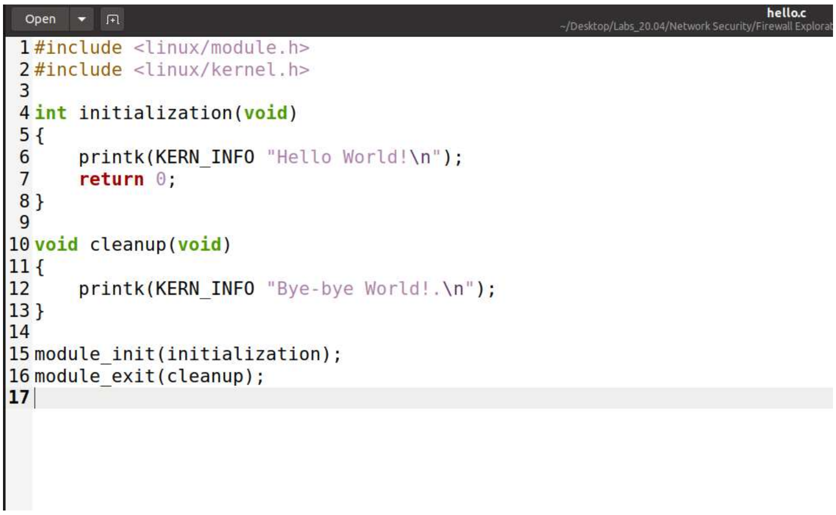
Firewall Exploration Lab

57118210 郑荔文

3 Task 1： Implementing a Simple Firewall

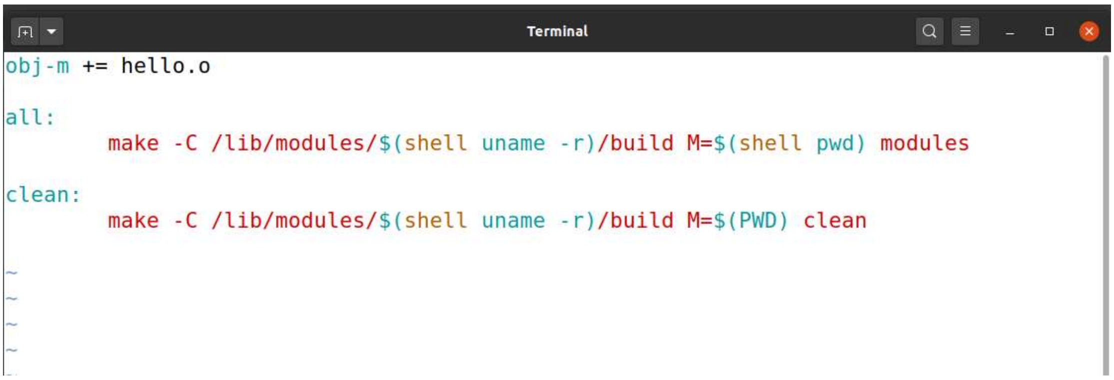
3.1 Task 1.A: Implement a Simple Kernel Module

在 file 中有如下 hello.c 文件，其功能为在模块加载时打印“Hello World！”，在模块被移除时打印“Bye-bye World！”并且这些信息在 dmesg 命令下可以查看



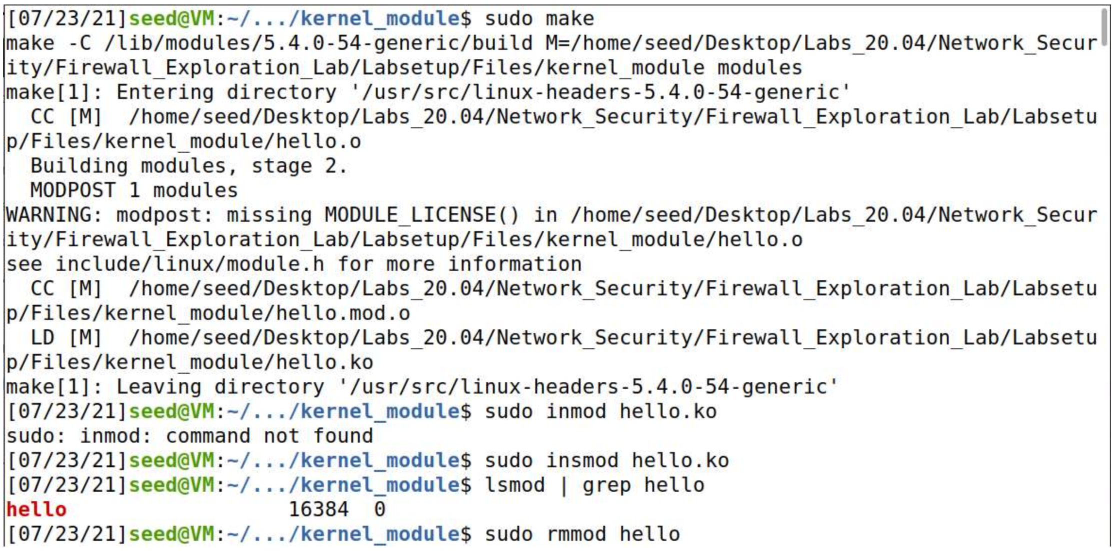
```c
#include <linux/module.h>
#include <linux/kernel.h>

int initialization(void)
{
    printk(KERN_INFO "Hello World!\n");
    return 0;
}

void cleanup(void)
{
    printk(KERN_INFO "Bye-bye World!.\n");
}

module_init(initialization);
module_exit(cleanup);
```

在同目录下还有一个 Makefile 文件，可以将上述 hello.c 文件编译成为一个可加载的内核模块

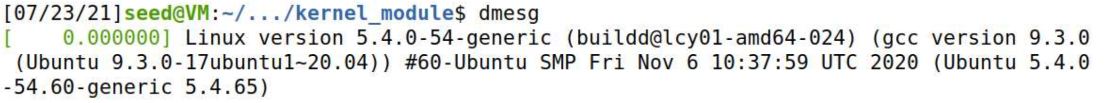


```makefile
obj-m += hello.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

~
~
~
~
```

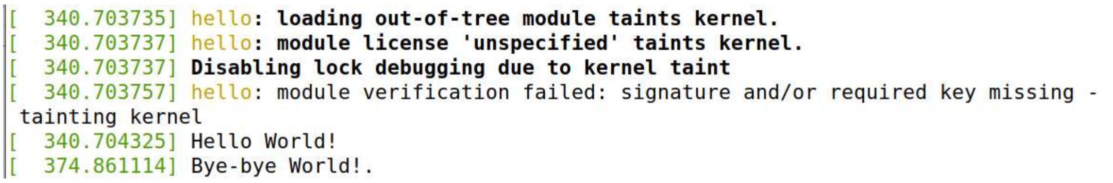
之后在 sudo make 进行编译，再采用 sudo insmod hello.ko 插入模块，采用 lsmod | grep hello 列出模块信息，sudo rmmod hello 移除该模块，采用 dmesg 查看是否出现了“Hello World！”和“Bye-bye World！”信息



```
[07/23/21]seed@VM:~/.../kernel_module$ sudo make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/Network_Secur
ity/Firewall_Exploration_Lab/Labsetup/Files/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labs_20.04/Network_Security/Firewall_Exploration_Lab/Labsetu
p/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/Desktop/Labs_20.04/Network_Secur
ity/Firewall_Exploration_Lab/Labsetup/Files/kernel_module/hello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/Desktop/Labs_20.04/Network_Security/Firewall_Exploration_Lab/Labsetu
p/Files/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/Labs_20.04/Network_Security/Firewall_Exploration_Lab/Labsetu
p/Files/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/23/21]seed@VM:~/.../kernel_module$ sudo inmod hello.ko
sudo: inmod: command not found
[07/23/21]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[07/23/21]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[07/23/21]seed@VM:~/.../kernel_module$ sudo rmmod hello
```



```
[07/23/21]seed@VM:~/.../kernel_module$ dmesg
[    0.000000] Linux version 5.4.0-54-generic (buildd@lcy01-amd64-024) (gcc version 9.3.0
 (Ubuntu 9.3.0-17ubuntu1~20.04)) #60-Ubuntu SMP Fri Nov 6 10:37:59 UTC 2020 (Ubuntu 5.4.0
-54.60-generic 5.4.65)
```



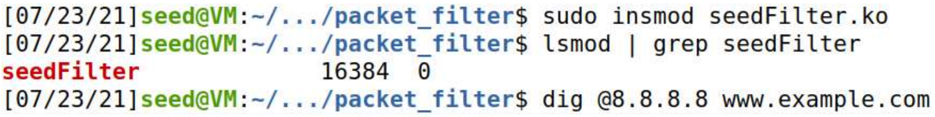
```
[  340.703735] hello: loading out-of-tree module taints kernel.
[  340.703737] hello: module license 'unspecified' taints kernel.
[  340.703737] Disabling lock debugging due to kernel taint
[  340.703757] hello: module verification failed: signature and/or required key missing -
 tainting kernel
[  340.704325] Hello World!
[  374.861114] Bye-bye World!.
```

可见再 dmesg 中出现了"Hello World！"和"Bye-bye World！"，与预期结果相符，表明该模块被插入内核模块中，之后又被移除。

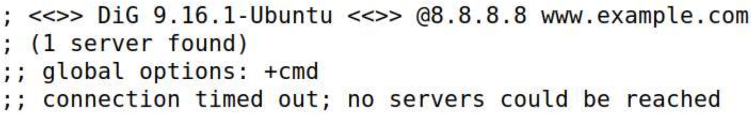
3.2 Task 1.B：Implement a Simple Firewall Using Netfilter

Tasks

1.使用 Makefile 编译 seedFilter.c,并将其加载入内核，使用 dig @8.8.8.8 www.example .com 命令产生 UDP 包，此时申请被阻塞，得不到应答



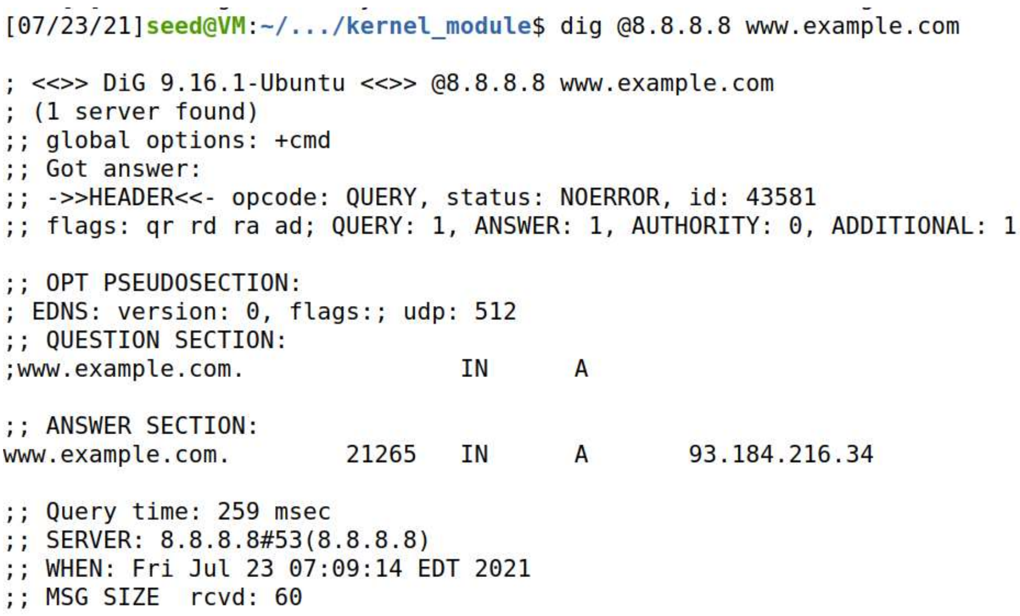
```
[07/23/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/23/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
[07/23/21]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com
```



```
; <<>> 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
```

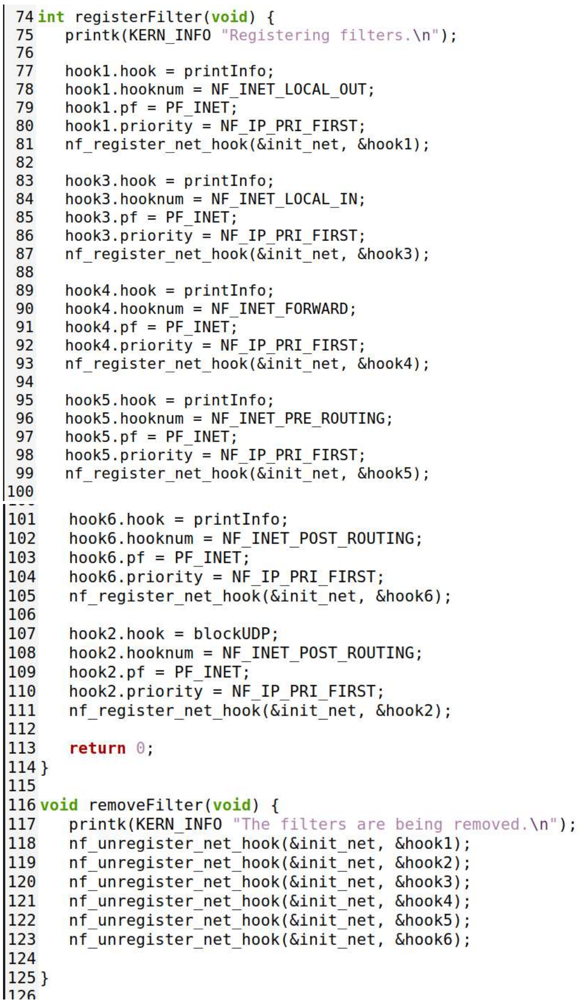
将该模块移除后，重复上述命令，则可以得到应答，说明防火墙起效

将其编译，插入内核



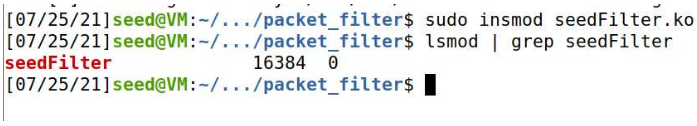
```
[07/23/21]seed@VM:~/.../kernel_module$ dig @8.8.8.8 www.example.com

; <<>> 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: 43581
;; 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.        21265   IN      A       93.184.216.34

;; Query time: 259 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Fri Jul 23 07:09:14 EDT 2021
;; MSG SIZE  rcvd: 60
```

2. 在 seedFilter.c 中修改相关代码，增加 hook



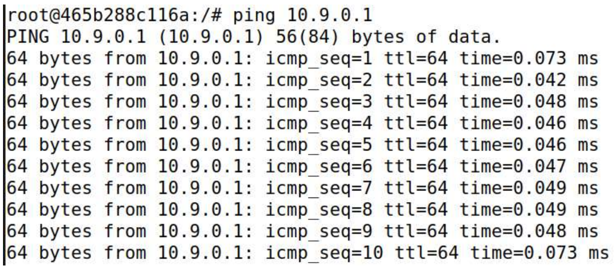
```c
int registerFilter(void) {
   printk(KERN_INFO "Registering filters.\n");

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

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

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

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

   hook6.hook = printInfo;
   hook6.hooknum = NF_INET_POST_ROUTING;
   hook6.pf = PF_INET;
   hook6.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook6);

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

   return 0;
}

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);
   nf_unregister_net_hook(&init_net, &hook6);

}
```

将其编译，插入内核



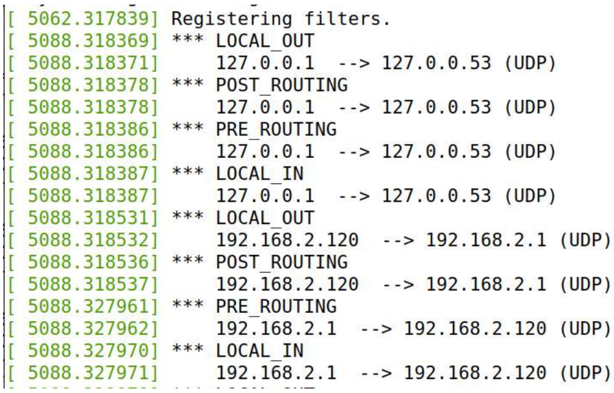
```
[07/25/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/25/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
[07/25/21]seed@VM:~/.../packet_filter$ 
```

在 10.9.0.5 上 ping 10.9.0.1



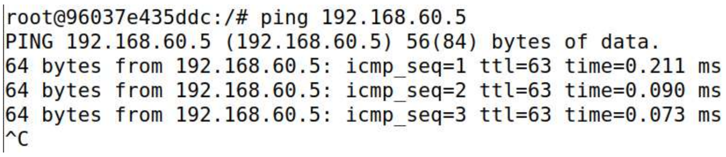
```
root@465b288c116a:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.073 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.046 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.047 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.049 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.049 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.048 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.073 ms
```

dmesg 查看信息，可以看到相关的标识



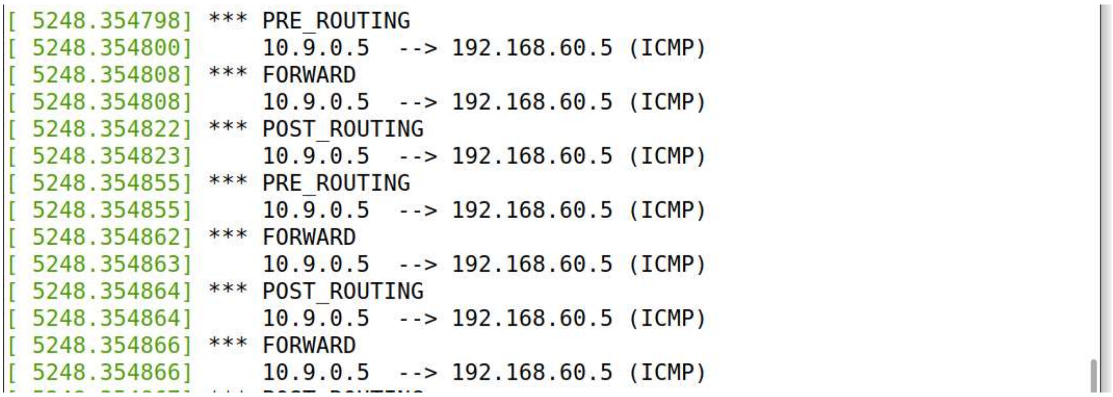
```
[ 5062.317839] Registering filters.
[ 5088.318369] *** LOCAL_OUT
[ 5088.318371]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 5088.318378] *** POST_ROUTING
[ 5088.318378]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 5088.318386] *** PRE_ROUTING
[ 5088.318386]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 5088.318387] *** LOCAL_IN
[ 5088.318387]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 5088.318531] *** LOCAL_OUT
[ 5088.318532]     192.168.2.120  --> 192.168.2.1 (UDP)
[ 5088.318536] *** POST_ROUTING
[ 5088.318537]     192.168.2.120  --> 192.168.2.1 (UDP)
[ 5088.327961] *** PRE_ROUTING
[ 5088.327962]     192.168.2.1  --> 192.168.2.120 (UDP)
[ 5088.327970] *** LOCAL_IN
[ 5088.327971]     192.168.2.1  --> 192.168.2.120 (UDP)
```

在 10.9.0.5 上 ping 192.168.60.5

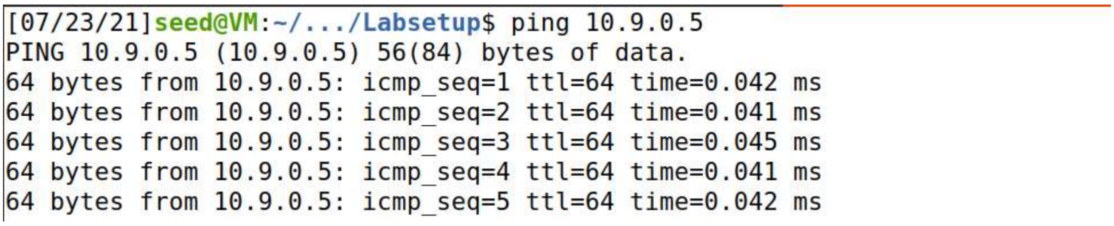


```
root@96037e435ddc:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.211 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.090 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.073 ms
^C
```

dmesg 查看信息



```
[ 5248.354798] *** PRE_ROUTING
[ 5248.354800]     10.9.0.5  --> 192.168.60.5 (ICMP)
[ 5248.354808] *** FORWARD
[ 5248.354808]     10.9.0.5  --> 192.168.60.5 (ICMP)
[ 5248.354822] *** POST_ROUTING
[ 5248.354823]     10.9.0.5  --> 192.168.60.5 (ICMP)
[ 5248.354855] *** PRE_ROUTING
[ 5248.354855]     10.9.0.5  --> 192.168.60.5 (ICMP)
[ 5248.354862] *** FORWARD
[ 5248.354863]     10.9.0.5  --> 192.168.60.5 (ICMP)
[ 5248.354864] *** POST_ROUTING
[ 5248.354864]     10.9.0.5  --> 192.168.60.5 (ICMP)
[ 5248.354866] *** FORWARD
[ 5248.354866]     10.9.0.5  --> 192.168.60.5 (ICMP)
```



```
[07/23/21]seed@VM:~/.../Labsetup$ 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=64 time=0.042 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.041 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.042 ms
```

归纳出五个 netfilter hooks 的位置：

NF_INET_PRE_ROUTING：进行完版本号，校验和等检测，进入网络层的数据包通过此点，此处可以进行目的地址转换

NF_INET_LOCAL_IN：经路由查找后，送往本机的通过此点，此处可以进行 INPUT 过滤

NF_INET_FORWARD：要转发的包通过此点，此处可以进行 FORWARD 过滤

NF_INET_LOCAL_OUT:将要通过网络设备发出的包通过此点，此处可以进行内置的源地址转换

NF_INET_POST_ROUTING：本机发出的包通过此点，此处可以进行 OUTPUT 过滤

3.需要实现以下功能：（1）阻止其他计算机 ping VM （2）阻止其他计算机 telnet VM

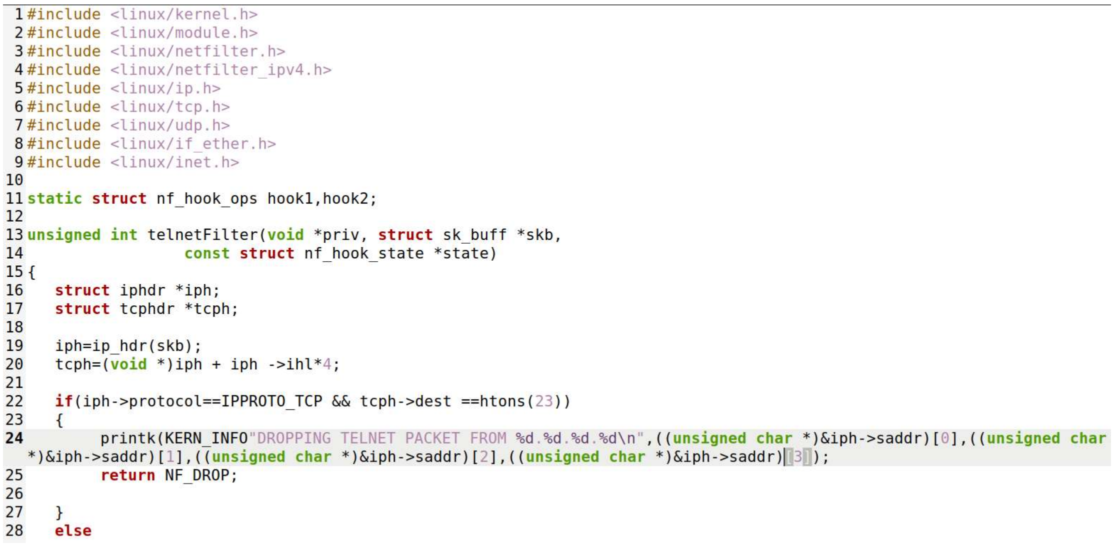
在 netfilter 中实现两个不同的 hook 函数，telnetFilter 和 ICMPFilter

在 telnetFilter 函数中提取 dst 端口为 23 且所用协议为 TCP 协议的报文，输出其 src 和 dst 的 ip 地址，并将其放入 NF_DROP 中，将其他的放入 NF_ACCEPT 中

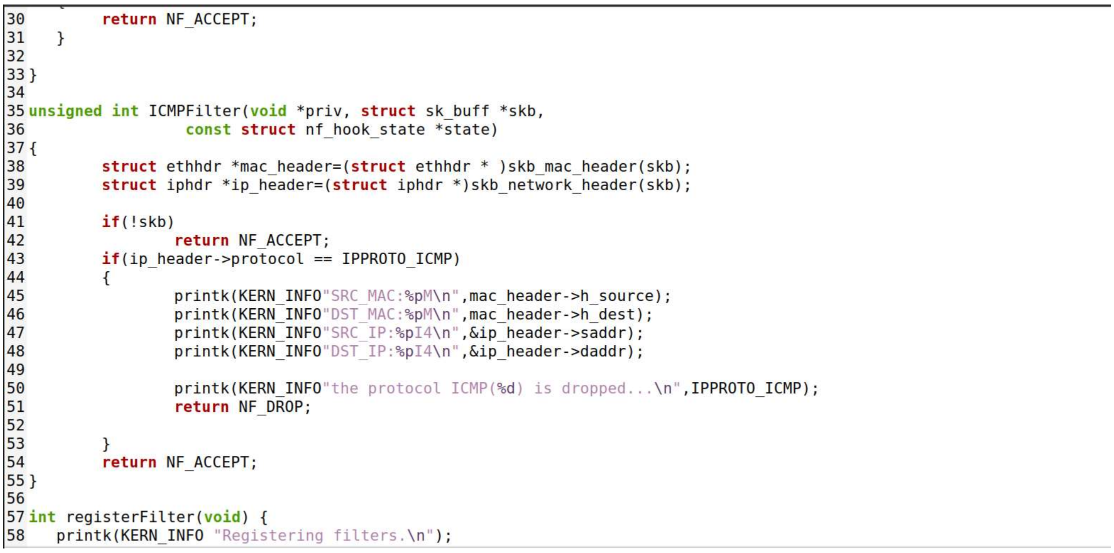
在 ICMPFilter 函数中提取所用协议为 ICMP 协议的报文，输出其 src 和 dst 的 ip 地址和 mac 地址，并将其放入 NF_DROP 中，将其他的放入 NF_ACCEPT 中

在 registerFilter 函数中，定义两个 hook，分别用 telnetFilter 和 ICMPFilter 函数注册，并在 removeFilter 中写明这两个 hook 对应的移除语句

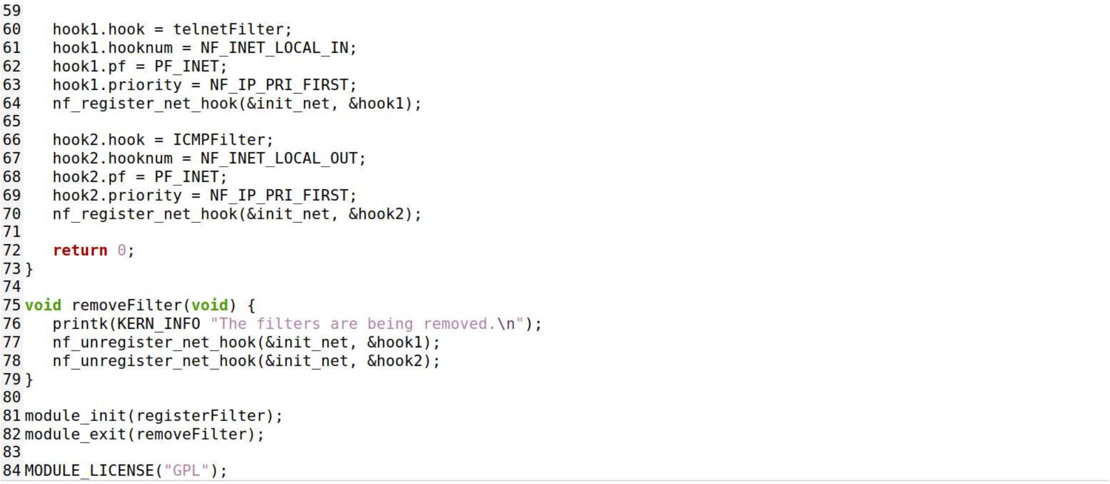
其具体代码如下：



```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>

static struct nf_hook_ops hook1,hook2;

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

   iph=ip_hdr(skb);
   tcph=(void *)iph + iph ->ihl*4;

   if(iph->protocol==IPPROTO_TCP && tcph->dest ==htons(23))
   {
        printk(KERN_INFO"DROPPING TELNET PACKET FROM %d.%d.%d.%d\n",((unsigned char *)&iph->saddr)[0],((unsigned char *)&iph->saddr)[1],((unsigned char *)&iph->saddr)[2],((unsigned char *)&iph->saddr)[3]);
        return NF_DROP;

   }
   else
```

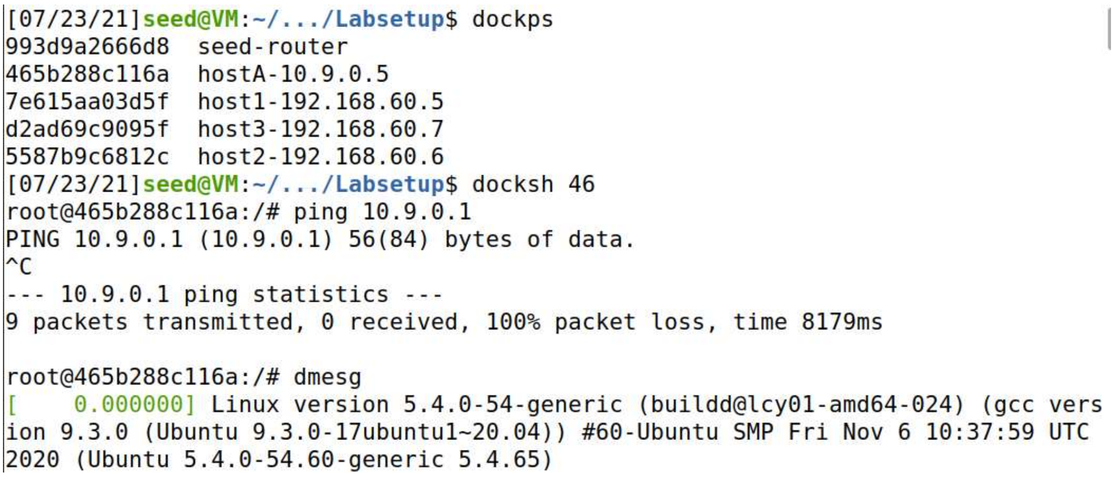


```
        return NF_ACCEPT;
   }

}

unsigned int ICMPFilter(void *priv, struct sk_buff *skb,
                 const struct nf_hook_state *state)
{
        struct ethhdr *mac_header=(struct ethhdr * )skb_mac_header(skb);
        struct iphdr *ip_header=(struct iphdr *)skb_network_header(skb);

        if(!skb)
                return NF_ACCEPT;
        if(ip_header->protocol == IPPROTO_ICMP)
        {
                printk(KERN_INFO"SRC_MAC:%pM\n",mac_header->h_source);
                printk(KERN_INFO"DST_MAC:%pM\n",mac_header->h_dest);
                printk(KERN_INFO"SRC_IP:%pI4\n",&ip_header->saddr);
                printk(KERN_INFO"DST_IP:%pI4\n",&ip_header->daddr);

                printk(KERN_INFO"the protocol ICMP(%d) is dropped...\n",IPPROTO_ICMP);
                return NF_DROP;

        }
        return NF_ACCEPT;
}

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



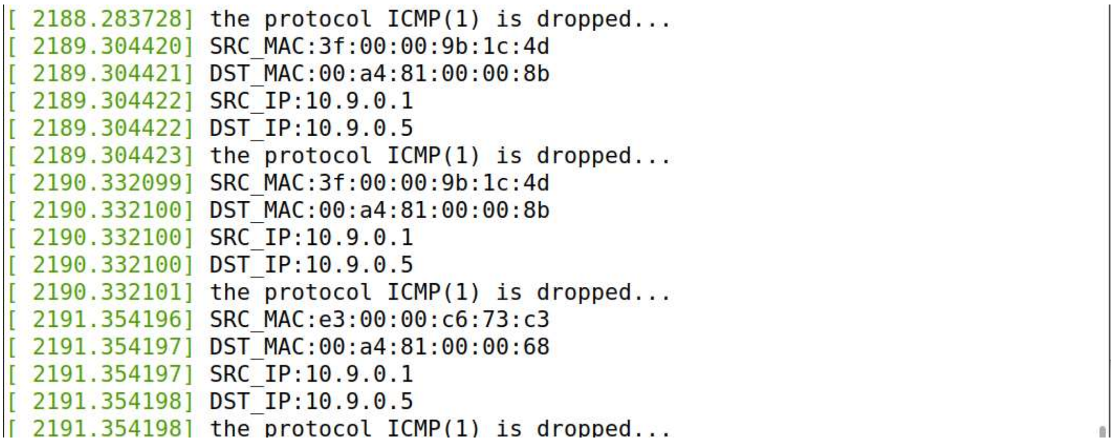
```

    hook1.hook = telnetFilter;
    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 = ICMPFilter;
    hook2.hooknum = NF_INET_LOCAL_OUT;
    hook2.pf = PF_INET;
    hook2.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook2);

    return 0;
}

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);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

在 docker 中进行测试



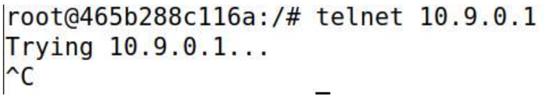
```
[07/23/21]seed@VM:~/.../Labsetup$ dockps
993d9a2666d8  seed-router
465b288c116a  hostA-10.9.0.5
7e615aa03d5f  host1-192.168.60.5
d2ad69c9095f  host3-192.168.60.7
5587b9c6812c  host2-192.168.60.6
[07/23/21]seed@VM:~/.../Labsetup$ docksh 46
root@465b288c116a:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8179ms

root@465b288c116a:/# dmesg
[    0.000000] Linux version 5.4.0-54-generic (buildd@lcy01-amd64-024) (gcc vers
ion 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)) #60-Ubuntu SMP Fri Nov 6 10:37:59 UTC
2020 (Ubuntu 5.4.0-54.60-generic 5.4.65)
```

在 10.9.0.5 的 hostA 机 ping 10.9.0.1 时，所有包都被丢弃，在 dmesg 中可以看到丢包的 ip 地址和 mac 地址信息



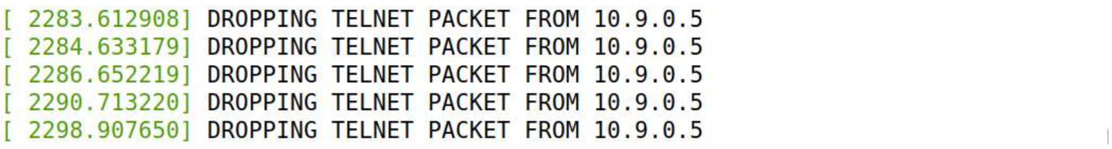
```
[ 2188.283728] the protocol ICMP(1) is dropped...
[ 2189.304420] SRC_MAC:3f:00:00:9b:1c:4d
[ 2189.304421] DST_MAC:00:a4:81:00:00:8b
[ 2189.304422] SRC_IP:10.9.0.1
[ 2189.304422] DST_IP:10.9.0.5
[ 2189.304423] the protocol ICMP(1) is dropped...
[ 2190.332099] SRC_MAC:3f:00:00:9b:1c:4d
[ 2190.332100] DST_MAC:00:a4:81:00:00:8b
[ 2190.332100] SRC_IP:10.9.0.1
[ 2190.332100] DST_IP:10.9.0.5
[ 2190.332101] the protocol ICMP(1) is dropped...
[ 2191.354196] SRC_MAC:e3:00:00:c6:73:c3
[ 2191.354197] DST_MAC:00:a4:81:00:00:68
[ 2191.354197] SRC_IP:10.9.0.1
[ 2191.354198] DST_IP:10.9.0.5
[ 2191.354198] the protocol ICMP(1) is dropped...
```

在 telnet 10.9.0.1 时，telnet 失败（一直在 trying）



```
root@465b288c116a:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
```

在 dmesg 中可以看到一直在 dropping



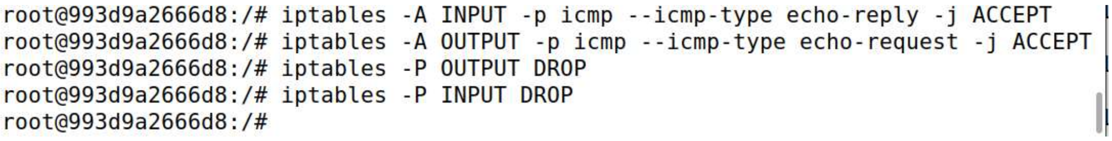
```
[ 2283.612908] DROPPING TELNET PACKET FROM 10.9.0.5
[ 2284.633179] DROPPING TELNET PACKET FROM 10.9.0.5
[ 2286.652219] DROPPING TELNET PACKET FROM 10.9.0.5
[ 2290.713220] DROPPING TELNET PACKET FROM 10.9.0.5
[ 2298.907650] DROPPING TELNET PACKET FROM 10.9.0.5
```

故完成 task 的需求，防火墙试验成功

4 Task2：Experimenting with Stateless Firewall Rules

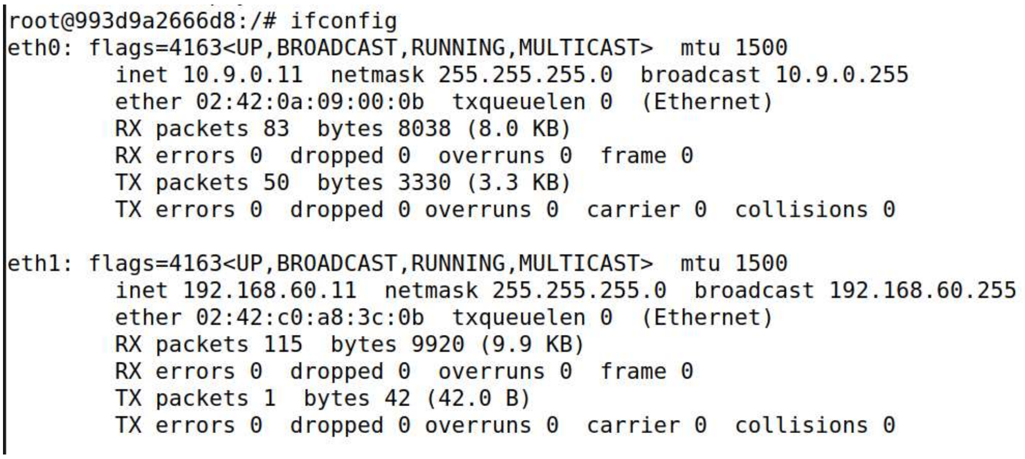
4.3 Task 2.A Protecting the Router

在路由器上输入以下命令设置 iptables,



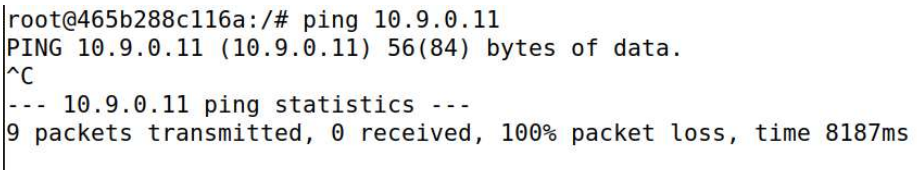
```
root@993d9a2666d8:/# iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@993d9a2666d8:/# iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
root@993d9a2666d8:/# iptables -P OUTPUT DROP
root@993d9a2666d8:/# iptables -P INPUT DROP
root@993d9a2666d8:/#
```

通过 ifconfig 命令查看路由器的 ip 地址

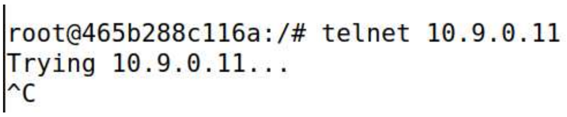


```
root@993d9a2666d8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 83  bytes 8038 (8.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 50  bytes 3330 (3.3 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 115  bytes 9920 (9.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 42 (42.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

在 10.9.0.5 的 HostA 上 ping 10.9.0.11 发现 ping 不通，通过 telnet 命令也无法远程登录



```
root@465b288c116a:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
^C
--- 10.9.0.11 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8187ms
```



```
root@465b288c116a:/# telnet 10.9.0.11
Trying 10.9.0.11...
^C
```

解释每条 rules：

（1）Iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT

-A INPUT 表示向 INPUT 链中增加一条规则，-p icmp 表示指定协议为 icmp 协议，--icmp echo-reply 表示 icmp 类型为 echo-reply，-j ACCEPT 表示封包放行，即处理完之后不再比较其他规则。

该行规则表示向 INPUT 链中增加一条对 echo-reply 类型的 ICMP 规则，在处理后不再比较其他规则封包放行。

（2）Iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT

-A OUTPUT 表示向 OUTPUT 链中增加一条规则，-p icmp 表示指定协议为 icmp 协议，--

icmp echo- request 表示 icmp 类型为 echo- request，-j ACCEPT 表示封包放行，即处理完之后不再比较其他规则。

该行规则表示向 OUTPUT 链中增加一条对 echo- request 类型的 ICMP 规则，在处理后不再比较其他规则封包放行。

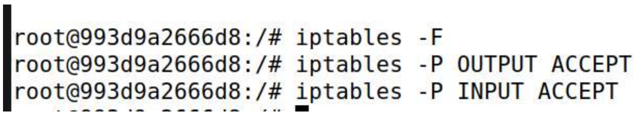
（3）iptables -P OUTPUT DROP

-P 为设置链的默认策略，该规则设置 OUTPUT 链的默认策略为丢包。

（4）iptables -P INPUT DROP

-P 为设置链的默认策略，该规则设置 INPUT 链的默认策略为丢包。

在试验之后，将 iptables 设置移除，返回其初始设置，并将 INPUT OUTPUT 默认策略改为 ACCEPT



```
root@993d9a2666d8:/# iptables -F
root@993d9a2666d8:/# iptables -P OUTPUT ACCEPT
root@993d9a2666d8:/# iptables -P INPUT ACCEPT
```

4.4 Task2.B: Protecting the Internal Network

为保护内网 192.168.60.0/24，需要使用 FORWARD 链，需要对 ICMP 流量做出如下限制：

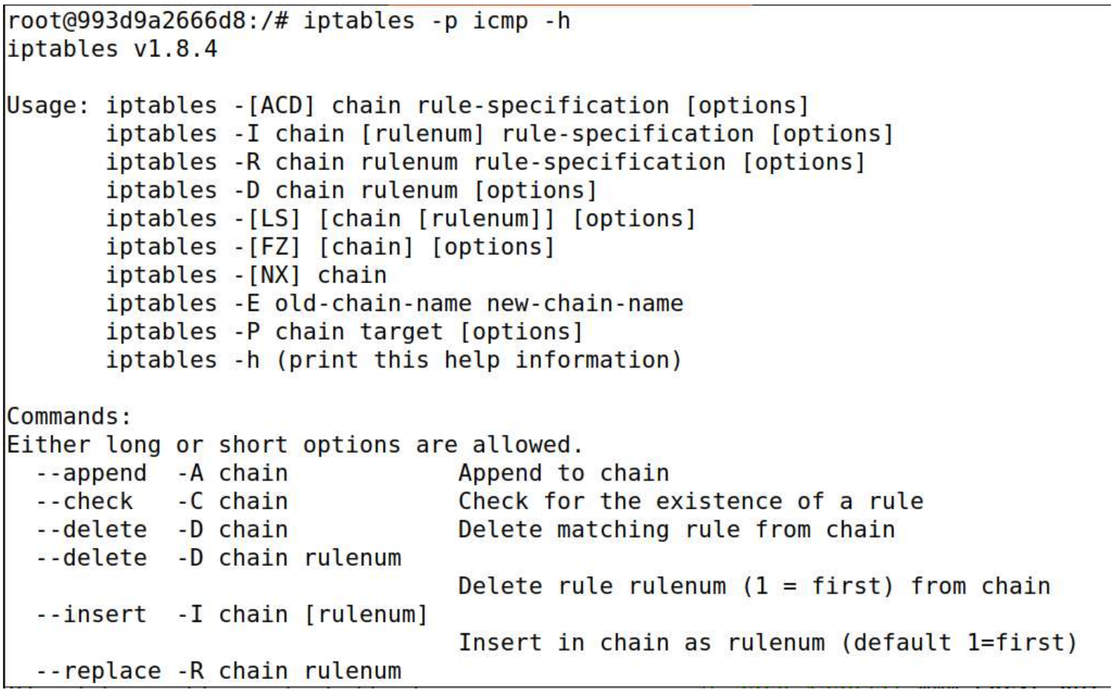
（1）外部主机不能 ping 通内部主机

（2）外部主机可以 ping 通路由器

（3）内部主机可以 ping 通外部主机

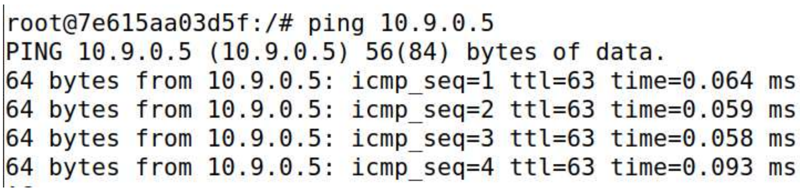
（4）其他在内外部网络之间的包应该被阻塞

首先使用 iptables -p icmp -h 查看相关信息

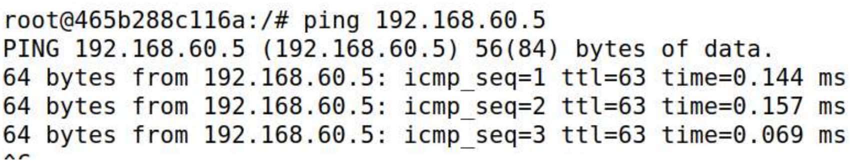


```
root@993d9a2666d8:/# iptables -p icmp -h
iptables v1.8.4

Usage: iptables -[ACD] chain rule-specification [options]
       iptables -I chain [rulenum] rule-specification [options]
       iptables -R chain rulenum rule-specification [options]
       iptables -D chain rulenum [options]
       iptables -[LS] [chain [rulenum]] [options]
       iptables -[FZ] [chain] [options]
       iptables -[NX] chain
       iptables -E old-chain-name new-chain-name
       iptables -P chain target [options]
       iptables -h (print this help information)

Commands:
Either long or short options are allowed.
  --append  -A chain            Append to chain
  --check   -C chain            Check for the existence of a rule
  --delete  -D chain            Delete matching rule from chain
  --delete  -D chain rulenum
                                Delete rule rulenum (1 = first) from chain
  --insert  -I chain [rulenum]
                                Insert in chain as rulenum (default 1=first)
  --replace -R chain rulenum
```

在不做任何保护的时候，内部外部之间可以相互 ping 通

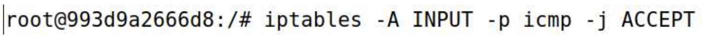


```
root@7e615aa03d5f:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.064 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.058 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.093 ms
```

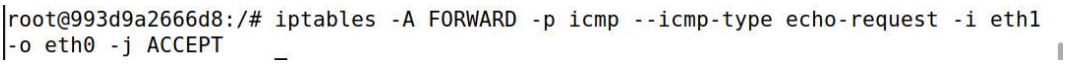


```
root@465b288c116a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.144 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.157 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.069 ms
^C
```

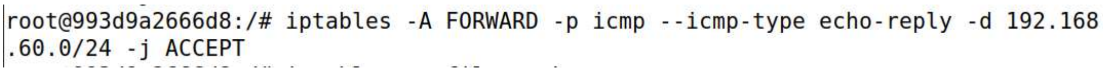
设定以下 iptables 规则，并设置 FORWARD 链的默认措施为 DROP



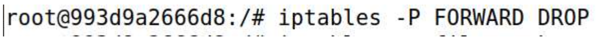
```
root@993d9a2666d8:/# iptables -A INPUT -p icmp -j ACCEPT
```



```
root@993d9a2666d8:/# iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1
-o eth0 -j ACCEPT
```

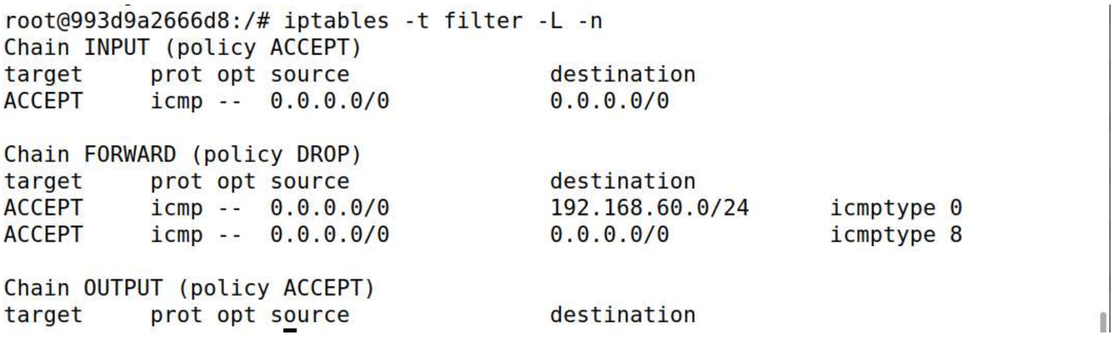


```
root@993d9a2666d8:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168
.60.0/24 -j ACCEPT
```



```
root@993d9a2666d8:/# iptables -P FORWARD DROP
```

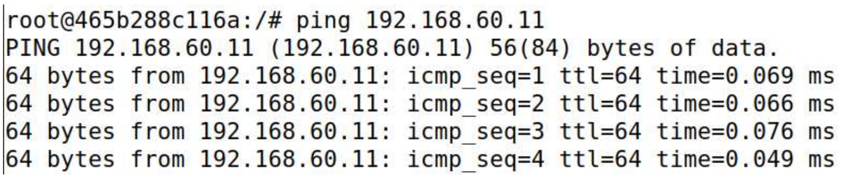
查看 filter 表的全部规则如下：



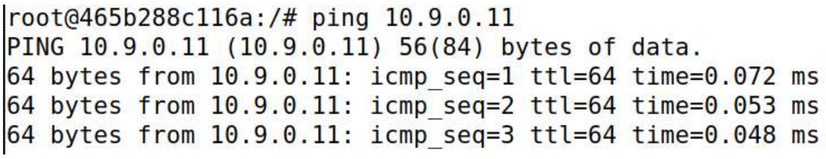
```
root@993d9a2666d8:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            192.168.60.0/24      icmptype 0
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

验证这些规则是否满足需求

1 外部主机可以 ping 通路由器（两个端口）

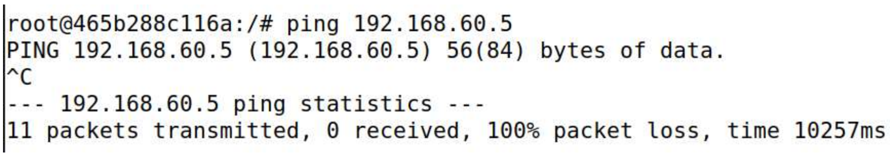


```
root@465b288c116a:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.069 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.066 ms
64 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 192.168.60.11: icmp_seq=4 ttl=64 time=0.049 ms
```



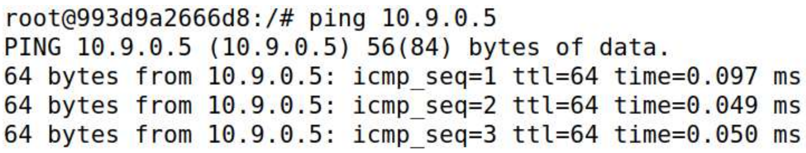
```
root@465b288c116a:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.048 ms
```

2.外部主机无法 ping 通内部



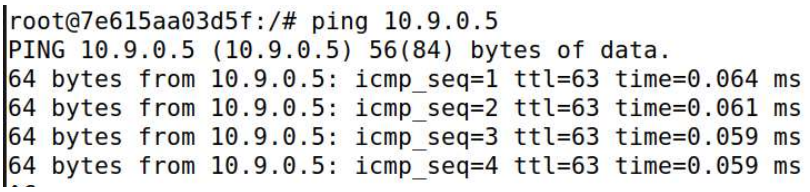
```
root@465b288c116a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
11 packets transmitted, 0 received, 100% packet loss, time 10257ms
```

3.路由器可以 ping 通外部主机



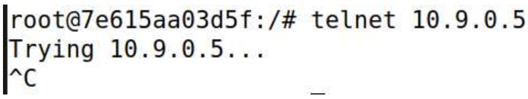
```
root@993d9a2666d8:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.097 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.049 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.050 ms
```

4.内部主机可以 ping 通外部主机



```
root@7e615aa03d5f:/# 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=0.064 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.059 ms
```

5.其他在内部主机和外部主机之间的流量被阻塞（如 telnet 不上）



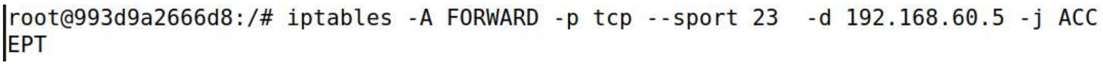
```
root@7e615aa03d5f:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
```

4.5 Task2.C：Protecting Internet Servers

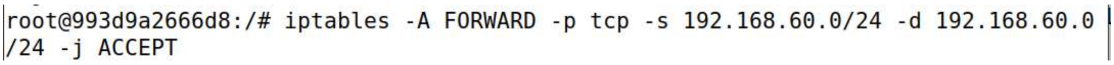
为了保护内部网的服务器（192.168.60.0/24 网段），需要满足以下需求：

（1）外部主机只能 telnet 192.168.60.5,无法 telent 其他内网服务器
（2）外部主机无法到达内部服务器
（3）内部主机可以访问内部服务器
（4）内部主机不能访问外部服务器
（5）不能使用 connection tracking 机制

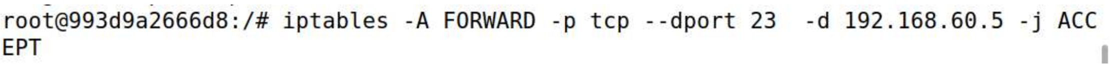
设定以下 iptables 规则，并设置 FORWARD 链的默认措施为 DROP



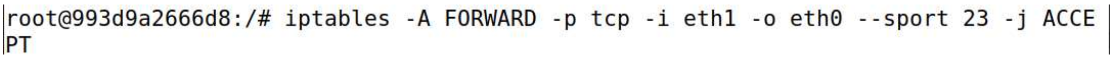
```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp --sport 23  -d 192.168.60.5 -j ACC
EPT
```



```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp -s 192.168.60.0/24 -d 192.168.60.0
/24 -j ACCEPT
```

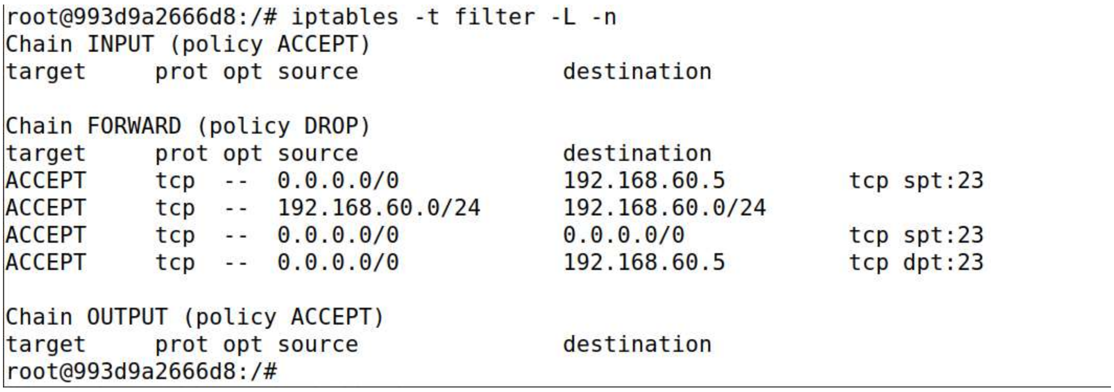


```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp --dport 23  -d 192.168.60.5 -j ACC
EPT
```



```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp -i eth1 -o eth0 --sport 23 -j ACCE
PT
```

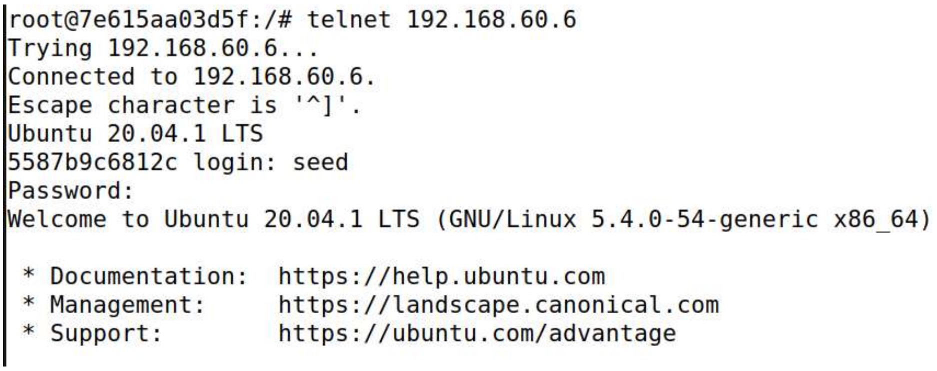
查看 filter 表的全部规则如下：



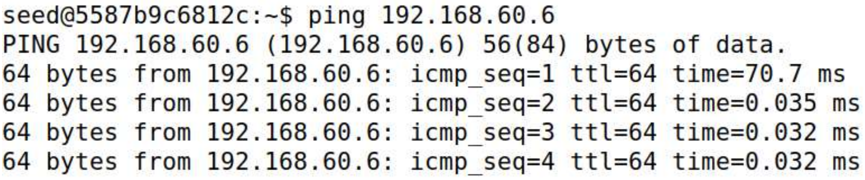
```
root@993d9a2666d8:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp spt:23
ACCEPT     tcp  --  192.168.60.0/24      192.168.60.0/24
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp spt:23
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp dpt:23

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@993d9a2666d8:/#
```

验证这些规则是否满足需求

1.内部主机可以 telnet 内部服务器，也可以 ping 通内部主机

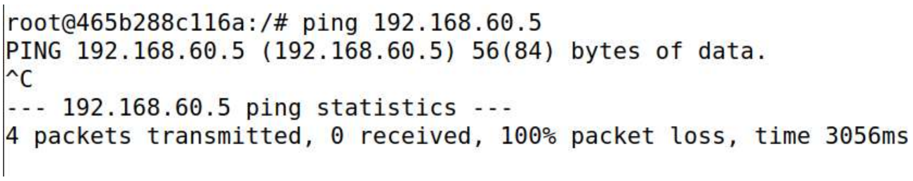


```
root@7e615aa03d5f:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5587b9c6812c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```



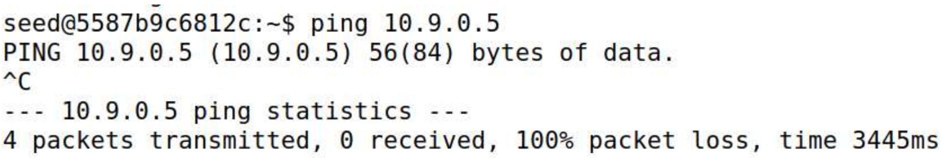
```
seed@5587b9c6812c:~$ ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
64 bytes from 192.168.60.6: icmp_seq=1 ttl=64 time=70.7 ms
64 bytes from 192.168.60.6: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 192.168.60.6: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from 192.168.60.6: icmp_seq=4 ttl=64 time=0.032 ms
```

2.外部主机无法 ping 通内部主机



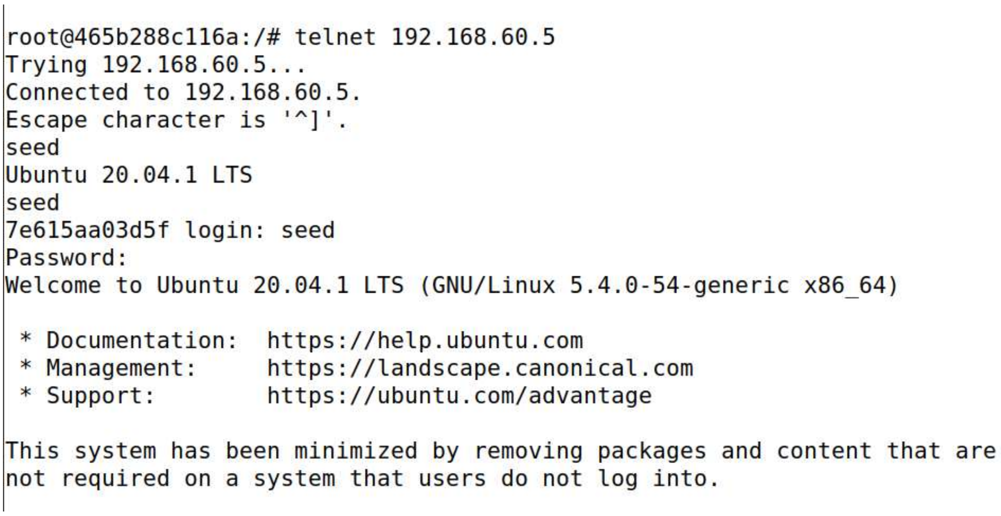
```
root@465b288c116a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3056ms
```

3.内部主机无法 ping 通外部主机



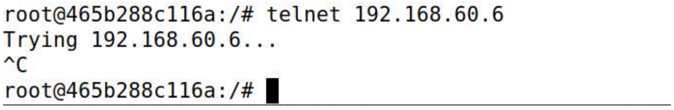
```
seed@5587b9c6812c:~$ ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3445ms
```

4.外部主机可以 telnet 到 192.168.60.5



```
root@465b288c116a:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
seed
Ubuntu 20.04.1 LTS
seed
7e615aa03d5f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.
```

5.外部主机无法 telnet 到其他内部服务器

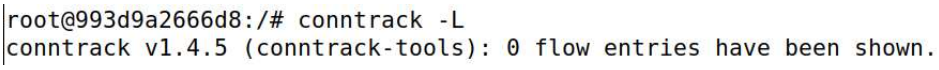


```
root@465b288c116a:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@465b288c116a:/# 
```

5 Task3：Connection Tracking and Stateful Firewall

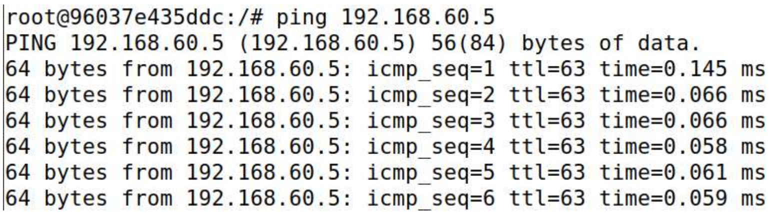
5.1 Task 3.A：Experiment with the Connection Tracking

使用 conntrack -L 命令查看连接信息



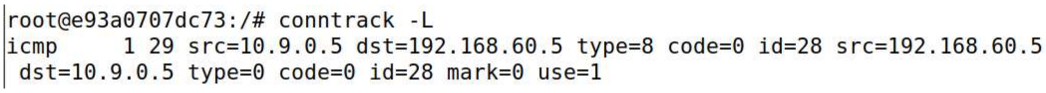
```
root@993d9a2666d8:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

ICMP 实验：在 10.9.0.5 上 ping 192.168.60.5



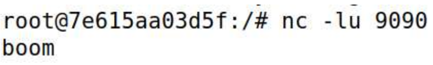
```
root@96037e435ddc:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.145 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.066 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.066 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.061 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.059 ms
```

在路由器 conntrack -L 出现如下连接追踪信息，状态维持时间为 29 秒



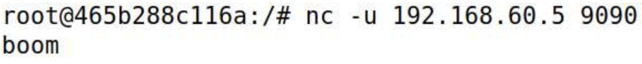
```
root@e93a0707dc73:/# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=28 src=192.168.60.5
 dst=10.9.0.5 type=0 code=0 id=28 mark=0 use=1
```

UDP 实验：在 192.168.60.5 处输入以下命令



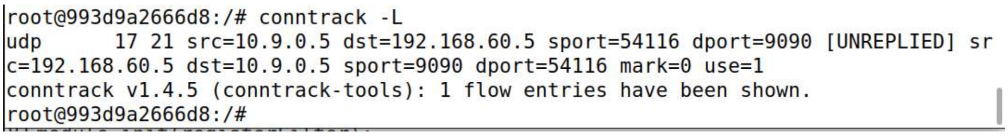
```
root@7e615aa03d5f:/# nc -lu 9090
boom
```

在 10.9.0.5 处输入以下命令



```
root@465b288c116a:/# nc -u 192.168.60.5 9090
boom
```

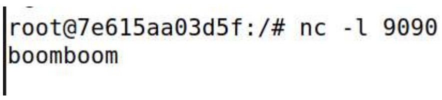
输入 boom,在回车之后，建立连接，在 192.168.60.5 出现输入信息



```
root@993d9a2666d8:/# conntrack -L
udp      17 21 src=10.9.0.5 dst=192.168.60.5 sport=54116 dport=9090 [UNREPLIED] sr
c=192.168.60.5 dst=10.9.0.5 sport=9090 dport=54116 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@993d9a2666d8:/#
```

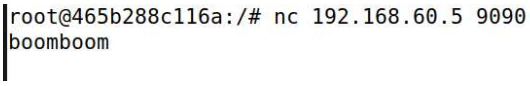
在路由器 conntrack -L 出现如下连接追踪信息，状态维持时间为 21 秒

TCP 实验：在 192.168.60.5 处输入以下命令



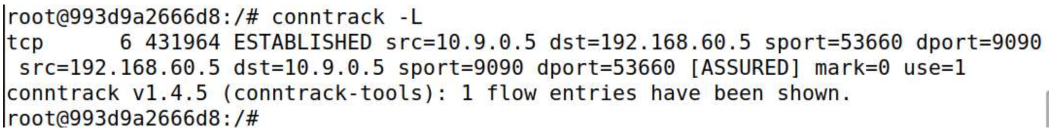
```
root@7e615aa03d5f:/# nc -l 9090
boomboom
```

在 10.9.0.5 处输入以下命令



```
root@465b288c116a:/# nc 192.168.60.5 9090
boomboom
```

输入 boomboom,在回车之后，建立连接，在 192.168.60.5 出现输入信息

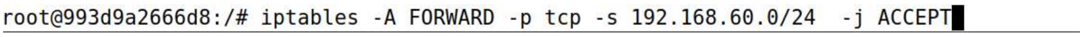


```
root@993d9a2666d8:/# conntrack -L
tcp      6 431964 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=53660 dport=9090
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=53660 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@993d9a2666d8:/#
```

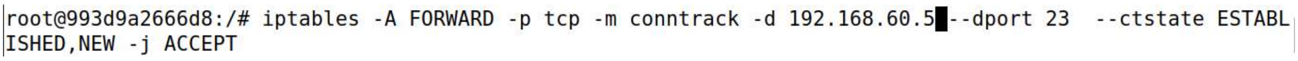
在路由器 conntrack -L 出现如下连接追踪信息，状态维持时间 431964 秒，时间明显长于 icmp 和 udp 连接

5.2 Task3.B：Setting Up a Stateful Firewall

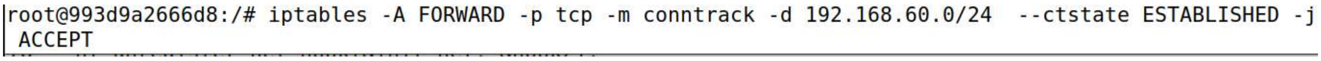
在路由器上输入以下规则



```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp -s 192.168.60.0/24  -j ACCEPT
```

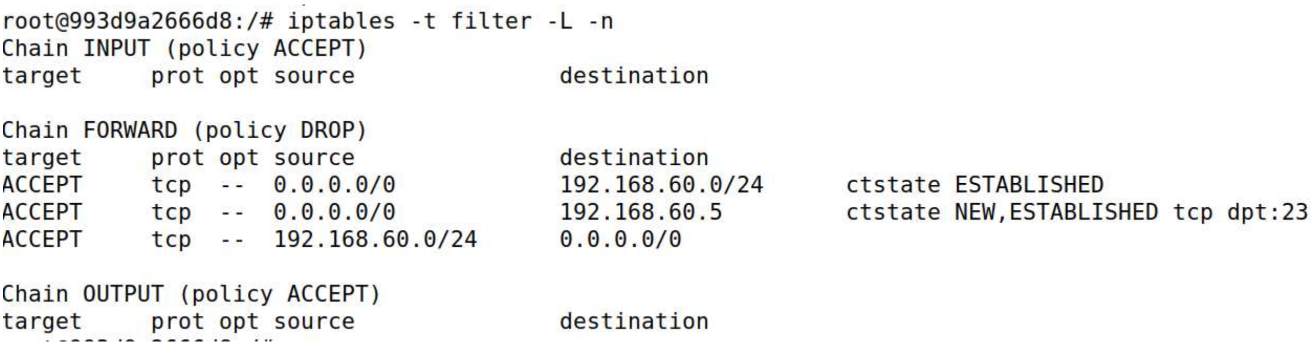


```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp -m conntrack -d 192.168.60.5 --dport 23  --ctstate ESTABL
ISHED,NEW -j ACCEPT
```



```
root@993d9a2666d8:/# iptables -A FORWARD -p tcp -m conntrack -d 192.168.60.0/24  --ctstate ESTABLISHED -j
 ACCEPT
```

查看 filter 表的全部规则如下：

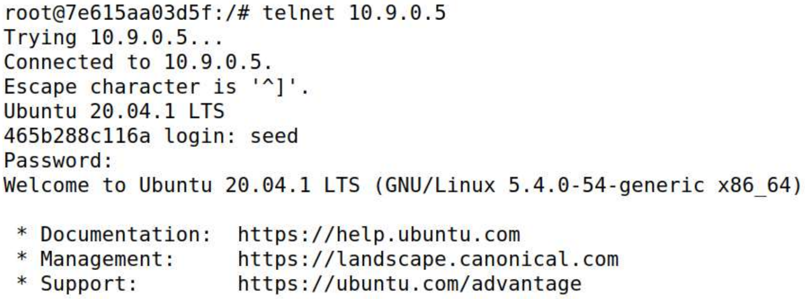


```
root@993d9a2666d8:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.0/24      ctstate ESTABLISHED
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         ctstate NEW,ESTABLISHED tcp dpt:23
ACCEPT     tcp  --  192.168.60.0/24      0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

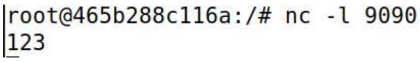
验证这些规则是否满足需求

1.内部主机可以 telnet 外部服务器，也与外部主机（10.9.0.5）建立连接

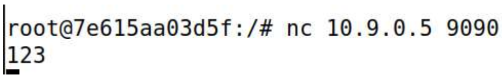
In->out



```
root@7e615aa03d5f:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
465b288c116a login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

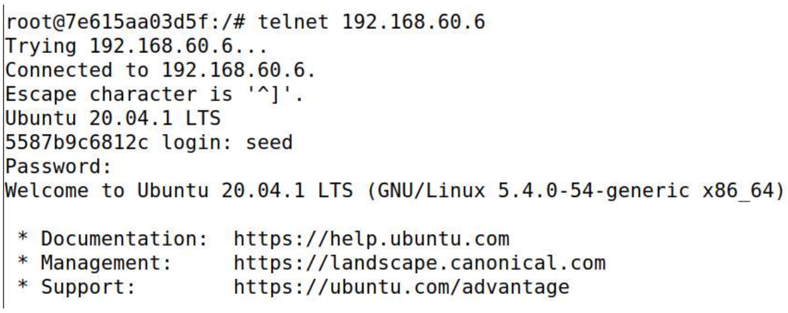


```
root@465b288c116a:/# nc -l 9090
123
```

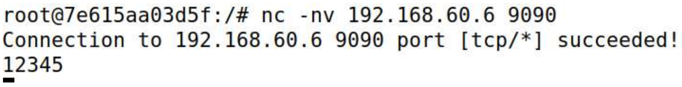


```
root@7e615aa03d5f:/# nc 10.9.0.5 9090
123
```

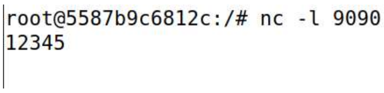
2.内部主机可以 telnet 内部主机，也可以和内部主机建立连接



```
root@7e615aa03d5f:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5587b9c6812c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

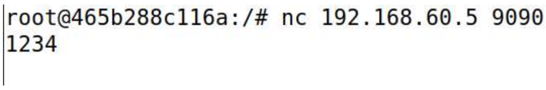


```
root@7e615aa03d5f:/# nc -nv 192.168.60.6 9090
Connection to 192.168.60.6 9090 port [tcp/*] succeeded!
12345
```

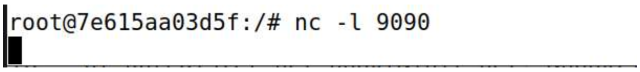


```
root@5587b9c6812c:/# nc -l 9090
12345
```

3.外部主机无法和内部主机建立连接

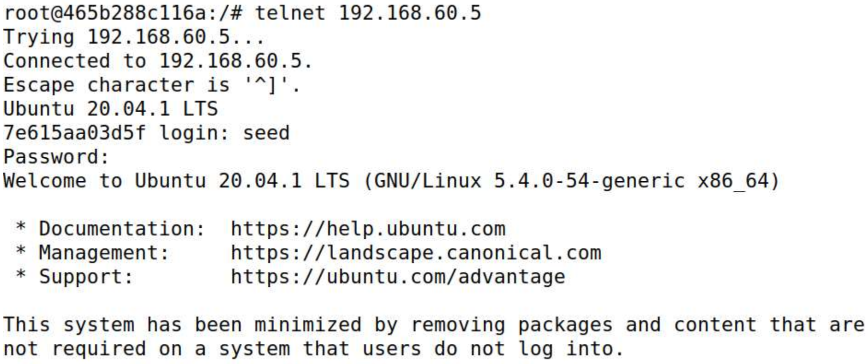


```
root@465b288c116a:/# nc 192.168.60.5 9090
1234
```

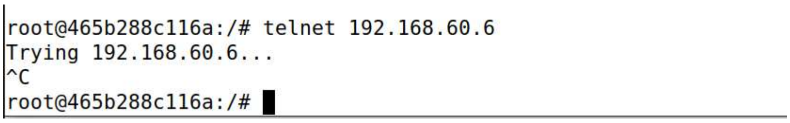


```
root@7e615aa03d5f:/# nc -l 9090
```

4.外部主机可以 telnet 192.168.60.5,且不能 telnet 其他内部主机（192.168.60.6）



```
root@465b288c116a:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7e615aa03d5f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.
```

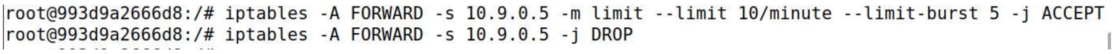


```
root@465b288c116a:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@465b288c116a:/# 
```

比较使用了连接跟踪(-m conntrack)和未使用该机制的方法，其中使用连接跟踪的方法写的规则更为简单，简化了规则设计并提高了效率，但是他需要更多的物理内存和存储空间

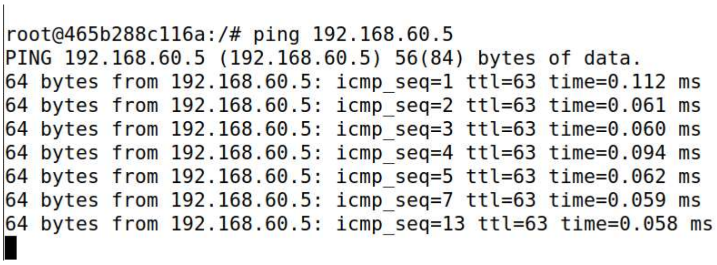
6 Task 4： Limiting Network Traffic

在路由器上运行以下命令



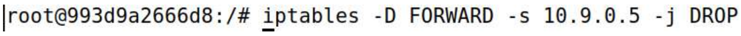
```
root@993d9a2666d8:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
root@993d9a2666d8:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
```

在 10.9.0.5 上 ping 192.168.60.5



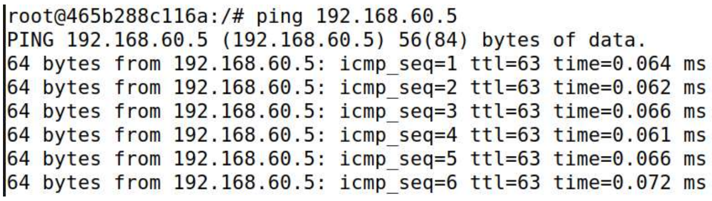
```
root@465b288c116a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.112 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.061 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.094 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.062 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.058 ms
```

可以直观的看到输出增长比较慢，即接收到的包的频率较慢，实现了对流量的限制



```
root@993d9a2666d8:/# iptables -D FORWARD -s 10.9.0.5 -j DROP
```

去掉第二行规则后重复发送



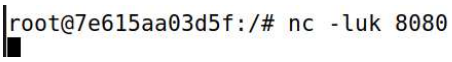
```
root@465b288c116a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.064 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.062 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.066 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.061 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.066 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.072 ms
```

直观的看到与平常 ping 的增长速度相同，即接收到的包的频率和之前相同，没有实现流量的限制

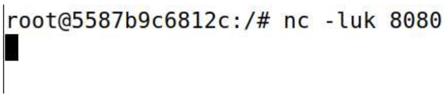
对比下表明第二行命令是需要的 ，否则起不到流量限制的效果，其原因为在默认规则为 ACCEPT 下，不满足的其他的报文也会被 accept，故所有流量都得不到限制

7 Task 5: Load Balancing

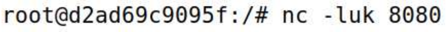
在 192.168.60.5，192.168.60.6，192.168.60.7 上分别输入以下命令，



```
root@7e615aa03d5f:/# nc -luk 8080
```



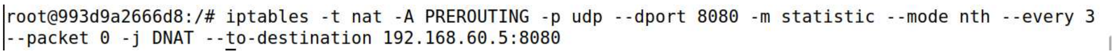
```
root@5587b9c6812c:/# nc -luk 8080
```



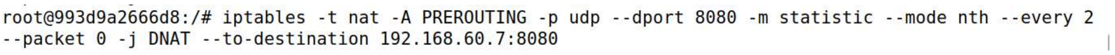
```
root@d2ad69c9095f:/# nc -luk 8080
```

1.使用 nth mode：

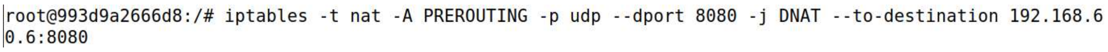
在路由器中输入以下规则，即每三个报文的第一个发送到 192.168.60.5：8080 端口，在发送后，剩下的报文中每两个报文的第一个发送到 192.168.60.7：8080 端口，剩下的所有报文发送到 192.168.60.6：8080 端口，以起到负载均衡的效果



```
root@993d9a2666d8:/# 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
```

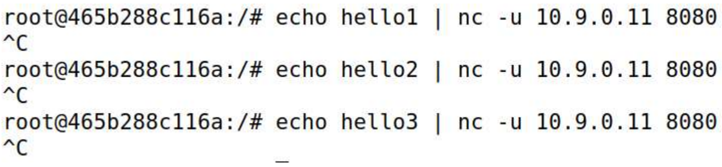


```
root@993d9a2666d8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2
--packet 0 -j DNAT --to-destination 192.168.60.7:8080
```



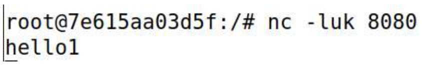
```
root@993d9a2666d8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination 192.168.6
0.6:8080
```

在 10.9.0.5 上输入以下命令



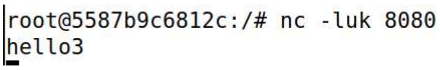
```
root@465b288c116a:/# echo hello1 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello2 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello3 | nc -u 10.9.0.11 8080
^C
```

可见 hello1 被发送到 192.168.60.5：8080 端口



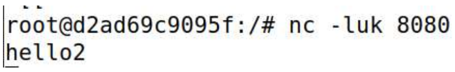
```
root@7e615aa03d5f:/# nc -luk 8080
hello1
```

hello3 被发送到 192.168.60.6：8080 端口



```
root@5587b9c6812c:/# nc -luk 8080
hello3
```

Hello2 被发送到 192.168.60.7：8080 端口

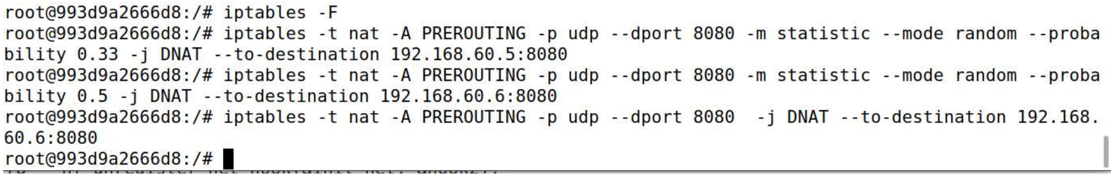


```
root@d2ad69c9095f:/# nc -luk 8080
hello2
```

实现了负载均衡

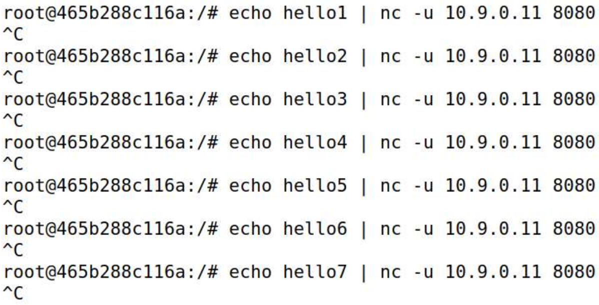
2.使用 random mode

首先清除之前的 iptables 规则，路由器中输入以下规则，即将以 0.33 的概率将报文发到 192.168.60.5：8080 端口，在发送后，剩下的报文中将报文以 0.5 的概率发送到 192.168.60.6：8080 端口，剩下的所有报文发送到 192.168.60.7：8080 端口，以起到负载均衡的效果



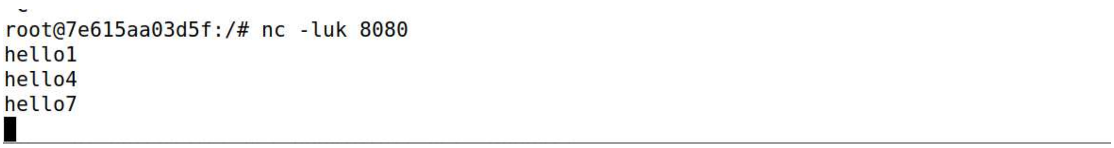
```
root@993d9a2666d8:/# iptables -F
root@993d9a2666d8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --proba
bility 0.33 -j DNAT --to-destination 192.168.60.5:8080
root@993d9a2666d8:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --proba
bility 0.5 -j DNAT --to-destination 192.168.60.6:8080
root@993d9a2666d8:/# iptables -t nat -A PREROUTING -p udp --dport 8080  -j DNAT --to-destination 192.168.
60.6:8080
root@993d9a2666d8:/# 
```

在 10.9.0.5 上输入以下命令



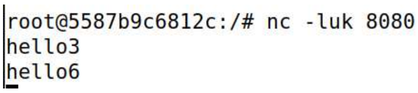
```
root@465b288c116a:/# echo hello1 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello2 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello3 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello4 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello5 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello6 | nc -u 10.9.0.11 8080
^C
root@465b288c116a:/# echo hello7 | nc -u 10.9.0.11 8080
^C
```

可见 hello1，hello4，hello7 被发送到 192.168.60.5：8080 端口



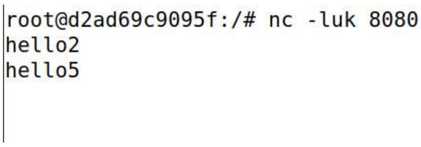
```
root@7e615aa03d5f:/# nc -luk 8080
hello1
hello4
hello7
```

Hello3，hello6 被发送到 192.168.60.6：8080 端口



```
root@5587b9c6812c:/# nc -luk 8080
hello3
hello6
```

Hello2，hello5 被发送到 192.168.60.7：8080 端口



```
root@d2ad69c9095f:/# nc -luk 8080
hello2
hello5
```

实现了负载均衡