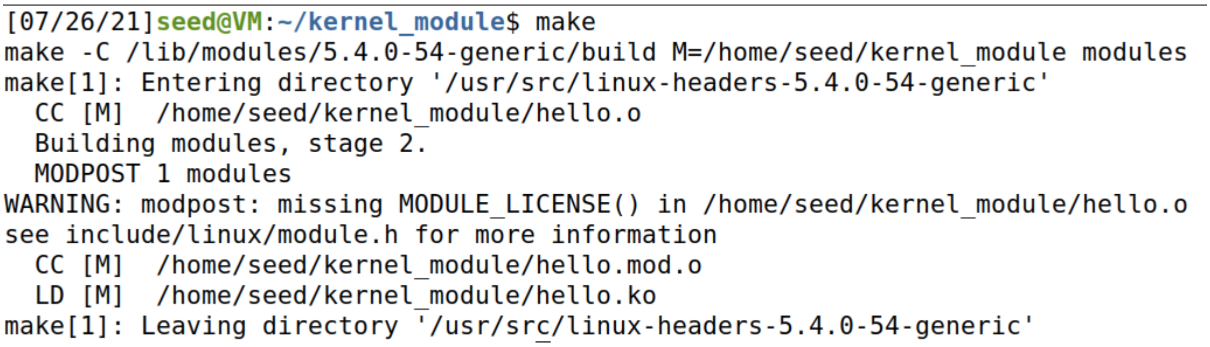
Lab6 Firewall Exploration

Task 1: Implementing a Simple Firewall

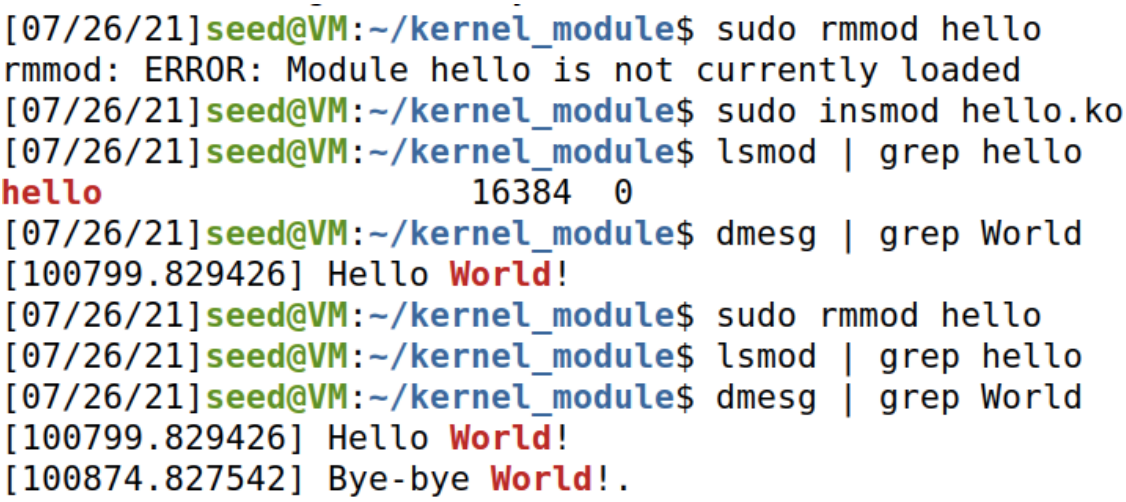
Task 1.A: Implement a Simple Kernel Module

1. 由于原始目录中有空格，故将kernel_module拷贝到/home/seed目录下再进行编译，结果如下：



```
[07/26/21]seed@VM:~/kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/kernel_module/hello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/kernel_module/hello.mod.o
  LD [M]  /home/seed/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

2. 命令测试结果如下：

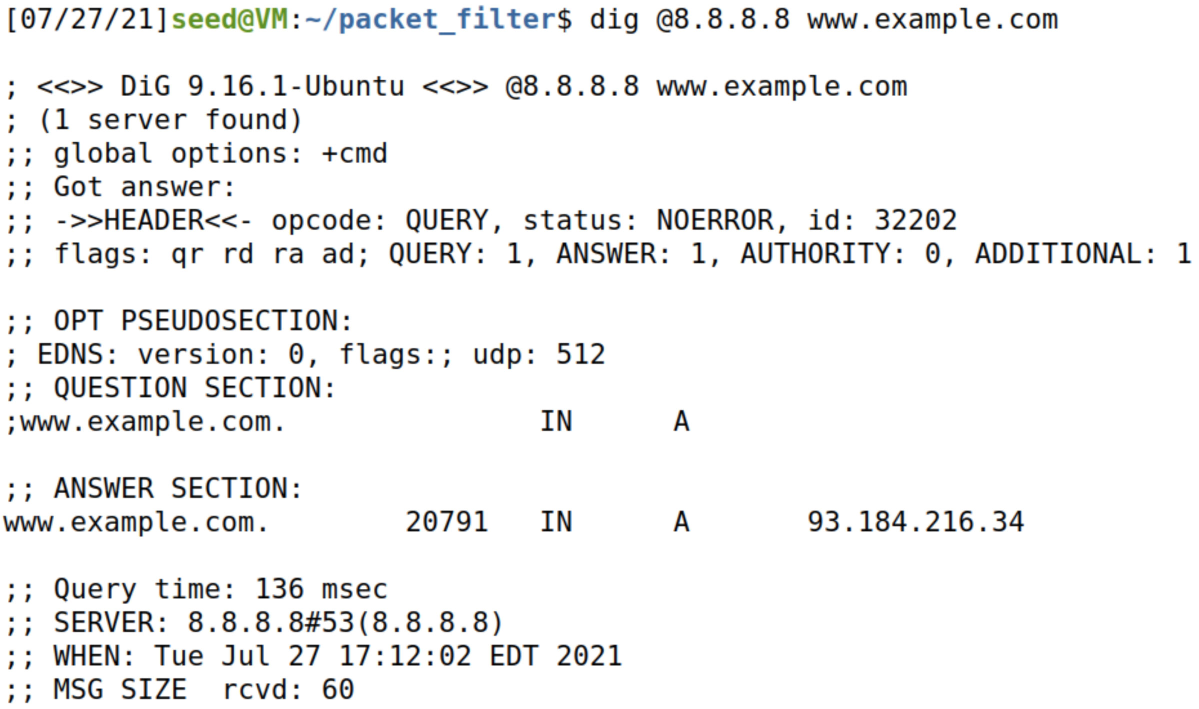


```
[07/26/21]seed@VM:~/kernel_module$ sudo rmmod hello
rmmod: ERROR: Module hello is not currently loaded
[07/26/21]seed@VM:~/kernel_module$ sudo insmod hello.ko
[07/26/21]seed@VM:~/kernel_module$ lsmod | grep hello
hello                  16384  0
[07/26/21]seed@VM:~/kernel_module$ dmesg | grep World
[100799.829426] Hello World!
[07/26/21]seed@VM:~/kernel_module$ sudo rmmod hello
[07/26/21]seed@VM:~/kernel_module$ lsmod | grep hello
[07/26/21]seed@VM:~/kernel_module$ dmesg | grep World
[100799.829426] Hello World!
[100874.827542] Bye-bye World!.
```

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

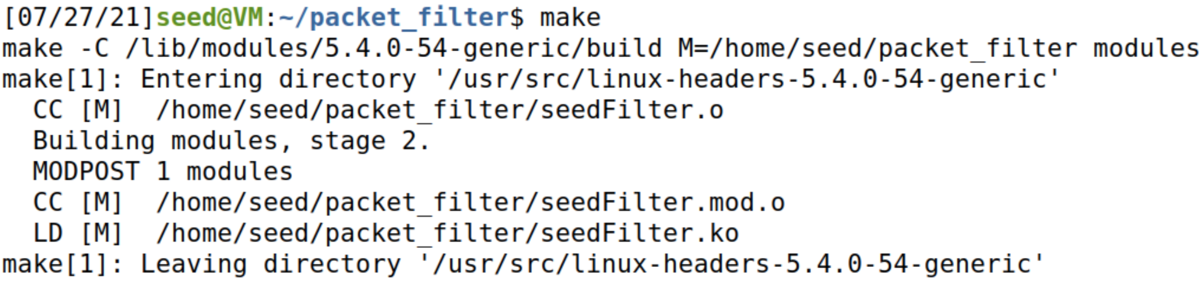
1. Compile the sample code using the provided Makefile.

1. 加载内核前，执行命令dig @8.8.8.8 www.example，结果如下：



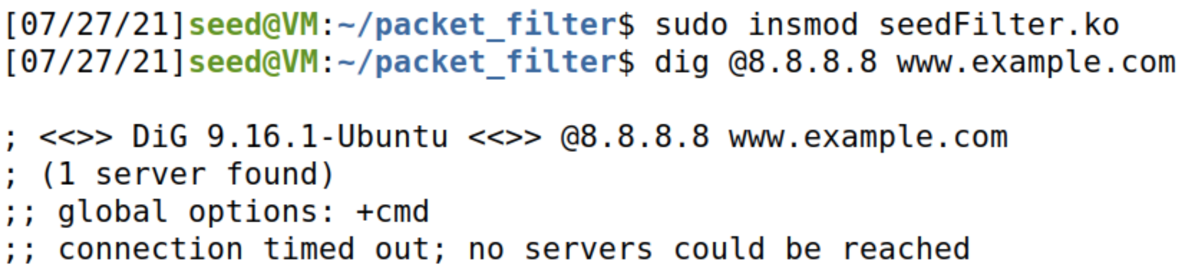
```
[07/27/21]seed@VM:~/packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32202
;; 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.        20791   IN      A       93.184.216.34

;; Query time: 136 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Jul 27 17:12:02 EDT 2021
;; MSG SIZE  rcvd: 60
```

2. 由于原始目录中有空格，故将packet_filter拷贝到/home/seed目录下再进行编译，结果如下：



```
[07/27/21]seed@VM:~/packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

3. 加载内核，再次执行命令dig @8.8.8.8 www.example，结果如下：



```
[07/27/21]seed@VM:~/packet_filter$ sudo insmod seedFilter.ko
[07/27/21]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
```

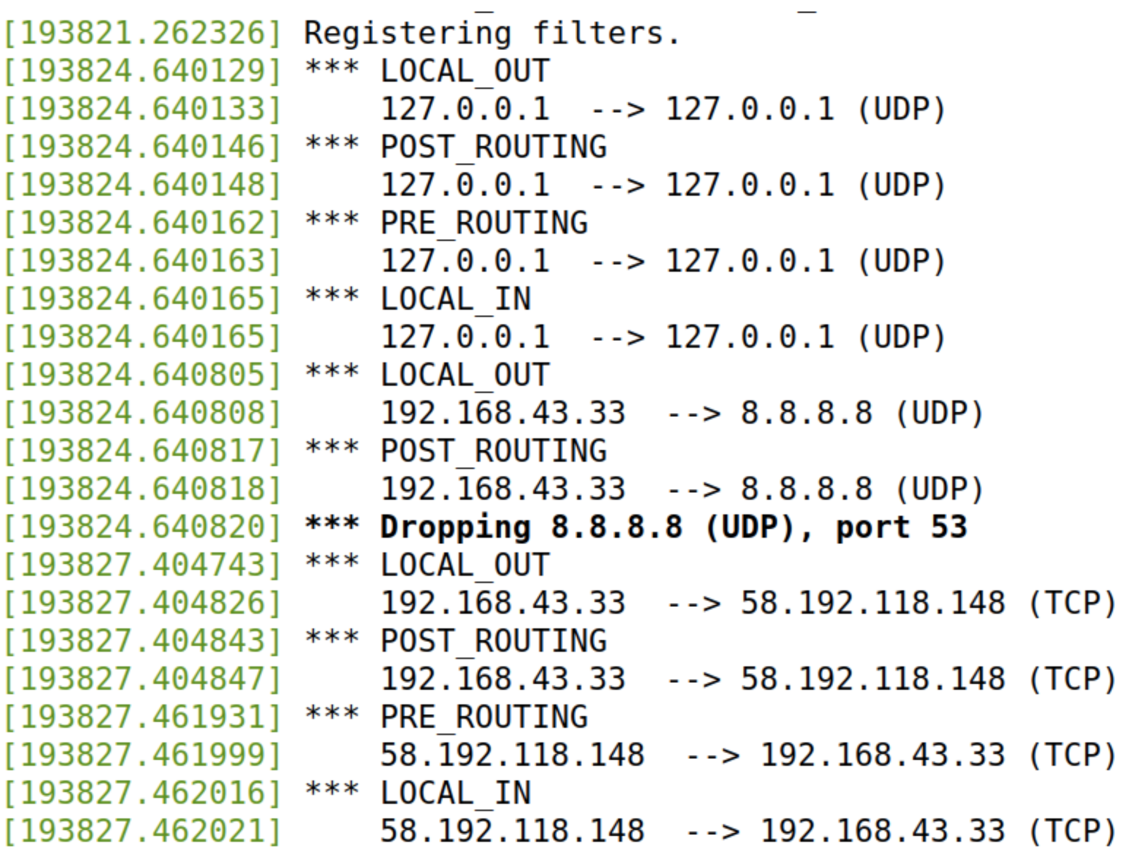
可见防火墙生效。

2. Hook the printInfo function to all of the netfilter hooks.

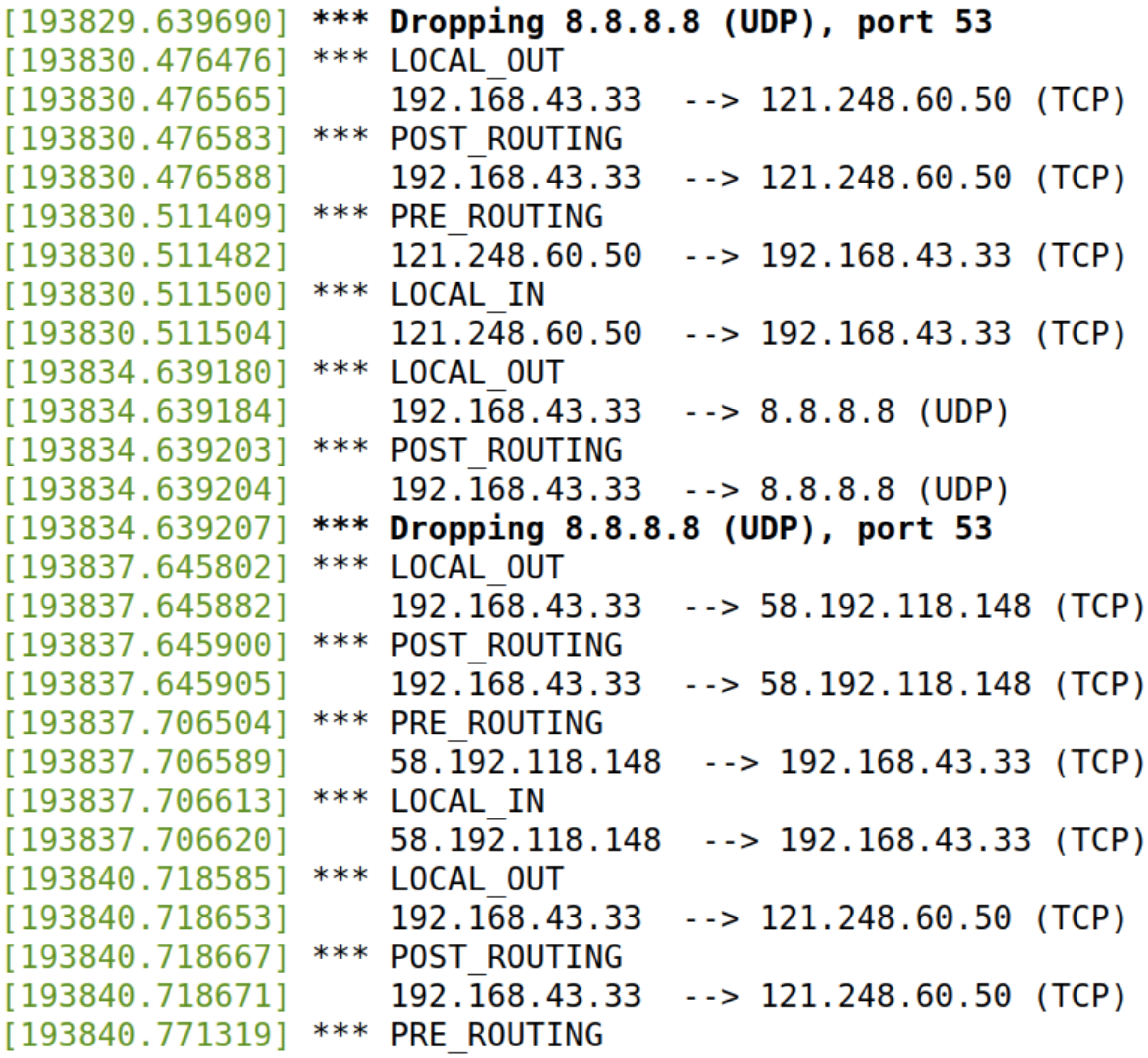
1. 内核代码修改如下：

```
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);

  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);

  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);

  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);
}
```

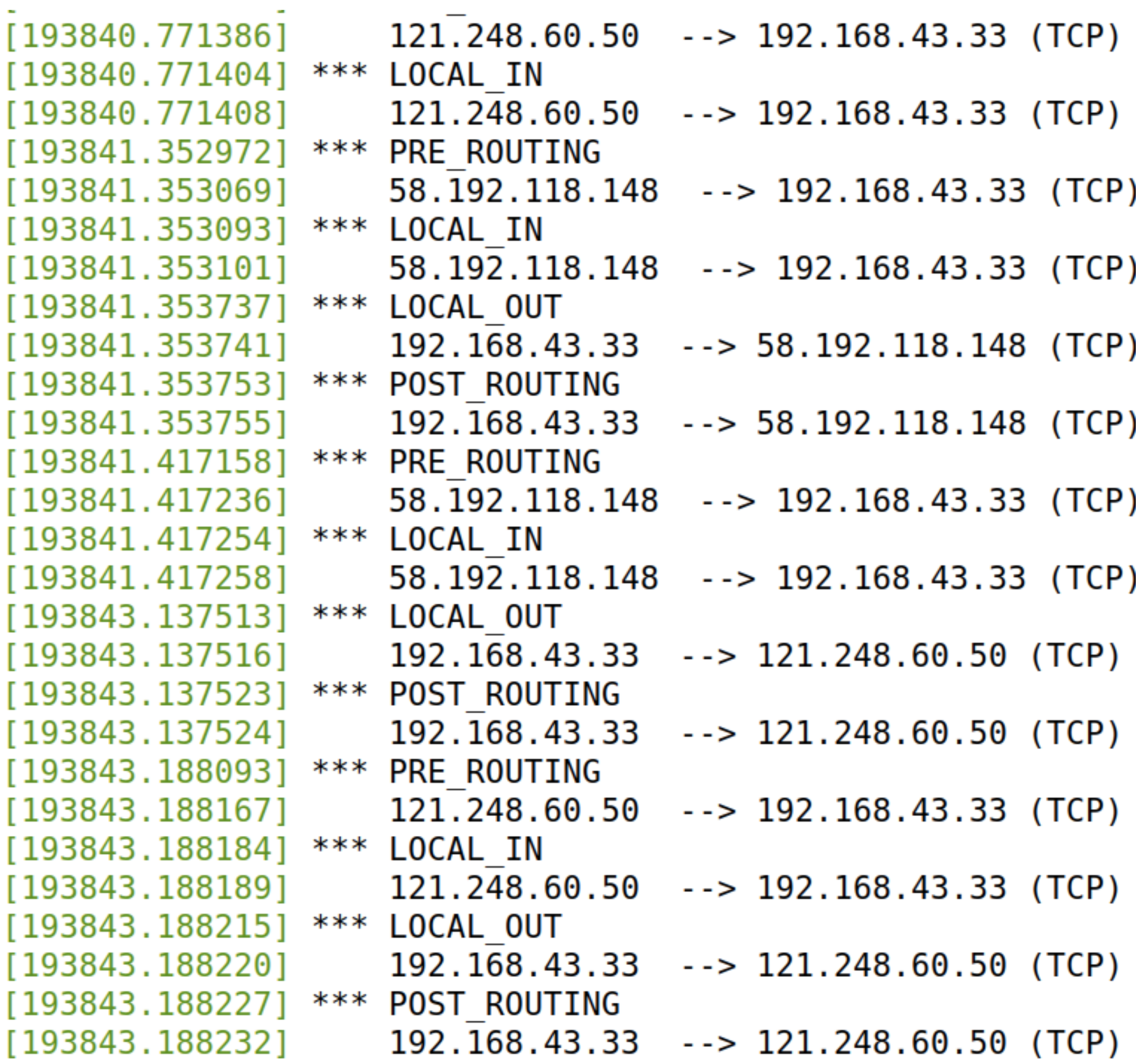
2. 编译内核，执行命令dig @8.8.8.8 www.example，sudo dmesg -c，结果如下：



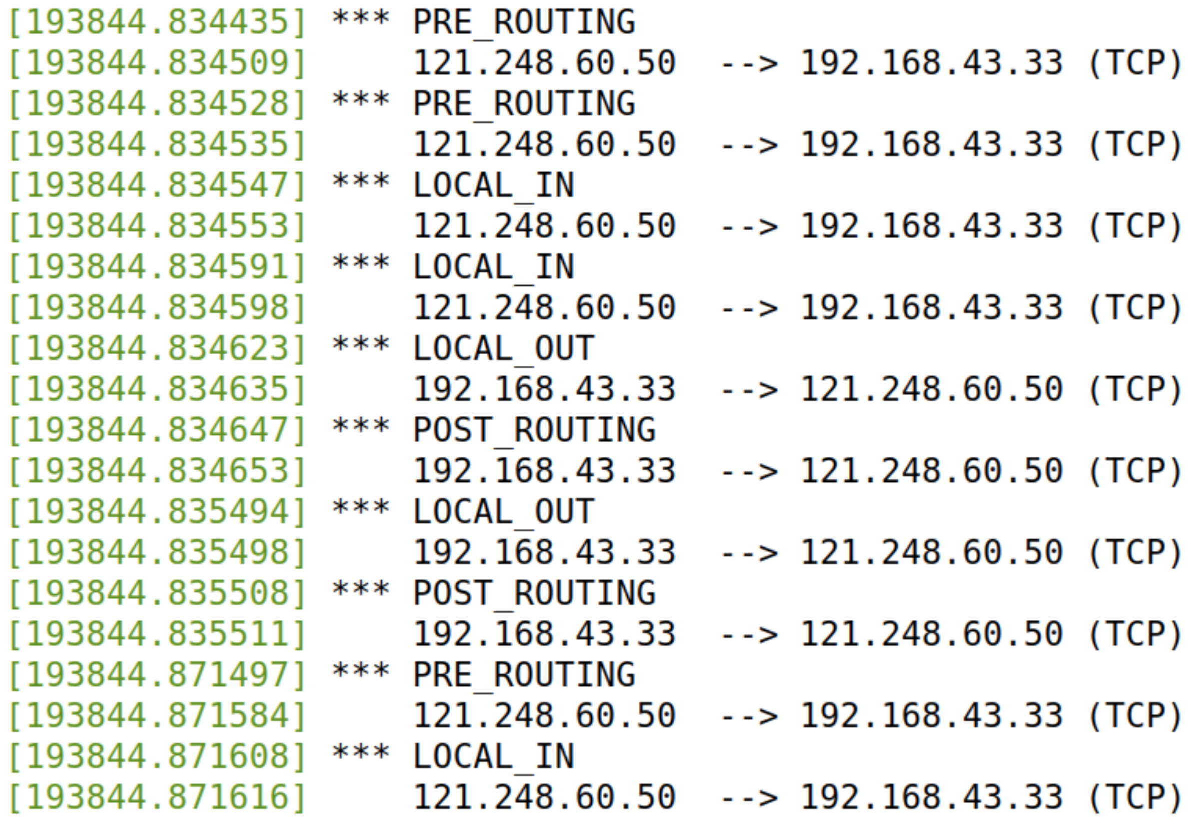
```
[193821.262326] Registering filters.
[193824.640129] *** LOCAL_OUT
[193824.640133]     127.0.0.1  --> 127.0.0.1 (UDP)
[193824.640146] *** POST_ROUTING
[193824.640148]     127.0.0.1  --> 127.0.0.1 (UDP)
[193824.640162] *** PRE_ROUTING
[193824.640163]     127.0.0.1  --> 127.0.0.1 (UDP)
[193824.640165] *** LOCAL_IN
[193824.640165]     127.0.0.1  --> 127.0.0.1 (UDP)
[193824.640805] *** LOCAL_OUT
[193824.640808]     192.168.43.33  --> 8.8.8.8 (UDP)
[193824.640817] *** POST_ROUTING
[193824.640818]     192.168.43.33  --> 8.8.8.8 (UDP)
[193824.640820] *** Dropping 8.8.8.8 (UDP), port 53
[193827.404743] *** LOCAL_OUT
[193827.404826]     192.168.43.33  --> 58.192.118.148 (TCP)
[193827.404843] *** POST_ROUTING
[193827.404847]     192.168.43.33  --> 58.192.118.148 (TCP)
[193827.461931] *** PRE_ROUTING
[193827.461999]     58.192.118.148  --> 192.168.43.33 (TCP)
[193827.462016] *** LOCAL_IN
[193827.462021]     58.192.118.148  --> 192.168.43.33 (TCP)
```



```
[193829.639690] *** Dropping 8.8.8.8 (UDP), port 53
[193830.476476] *** LOCAL_OUT
[193830.476565]     192.168.43.33  --> 121.248.60.50 (TCP)
[193830.476583] *** POST_ROUTING
[193830.476588]     192.168.43.33  --> 121.248.60.50 (TCP)
[193830.511409] *** PRE_ROUTING
[193830.511482]     121.248.60.50  --> 192.168.43.33 (TCP)
[193830.511500] *** LOCAL_IN
[193830.511504]     121.248.60.50  --> 192.168.43.33 (TCP)
[193834.639180] *** LOCAL_OUT
[193834.639184]     192.168.43.33  --> 8.8.8.8 (UDP)
[193834.639203] *** POST_ROUTING
[193834.639204]     192.168.43.33  --> 8.8.8.8 (UDP)
[193834.639207] *** Dropping 8.8.8.8 (UDP), port 53
[193837.645802] *** LOCAL_OUT
[193837.645882]     192.168.43.33  --> 58.192.118.148 (TCP)
[193837.645900] *** POST_ROUTING
[193837.645905]     192.168.43.33  --> 58.192.118.148 (TCP)
[193837.706504] *** PRE_ROUTING
[193837.706589]     58.192.118.148  --> 192.168.43.33 (TCP)
[193837.706613] *** LOCAL_IN
[193837.706620]     58.192.118.148  --> 192.168.43.33 (TCP)
[193840.718585] *** LOCAL_OUT
[193840.718653]     192.168.43.33  --> 121.248.60.50 (TCP)
[193840.718667] *** POST_ROUTING
[193840.718671]     192.168.43.33  --> 121.248.60.50 (TCP)
[193840.771319] *** PRE_ROUTING
```



```
[193840.771386]     121.248.60.50  --> 192.168.43.33 (TCP)
[193840.771404] *** LOCAL_IN
[193840.771408]     121.248.60.50  --> 192.168.43.33 (TCP)
[193841.352972] *** PRE_ROUTING
[193841.353069]     58.192.118.148  --> 192.168.43.33 (TCP)
[193841.353093] *** LOCAL_IN
[193841.353101]     58.192.118.148  --> 192.168.43.33 (TCP)
[193841.353737] *** LOCAL_OUT
[193841.353741]     192.168.43.33  --> 58.192.118.148 (TCP)
[193841.353753] *** POST_ROUTING
[193841.353755]     192.168.43.33  --> 58.192.118.148 (TCP)
[193841.417158] *** PRE_ROUTING
[193841.417236]     58.192.118.148  --> 192.168.43.33 (TCP)
[193841.417254] *** LOCAL_IN
[193841.417258]     58.192.118.148  --> 192.168.43.33 (TCP)
[193843.137513] *** LOCAL_OUT
[193843.137516]     192.168.43.33  --> 121.248.60.50 (TCP)
[193843.137523] *** POST_ROUTING
[193843.137524]     192.168.43.33  --> 121.248.60.50 (TCP)
[193843.188093] *** PRE_ROUTING
[193843.188167]     121.248.60.50  --> 192.168.43.33 (TCP)
[193843.188184] *** LOCAL_IN
[193843.188189]     121.248.60.50  --> 192.168.43.33 (TCP)
[193843.188215] *** LOCAL_OUT
[193843.188220]     192.168.43.33  --> 121.248.60.50 (TCP)
[193843.188227] *** POST_ROUTING
[193843.188232]     192.168.43.33  --> 121.248.60.50 (TCP)
```



```
[193844.834435] *** PRE_ROUTING
[193844.834509]     121.248.60.50  --> 192.168.43.33 (TCP)
[193844.834528] *** PRE_ROUTING
[193844.834535]     121.248.60.50  --> 192.168.43.33 (TCP)
[193844.834547] *** LOCAL_IN
[193844.834553]     121.248.60.50  --> 192.168.43.33 (TCP)
[193844.834591] *** LOCAL_IN
[193844.834598]     121.248.60.50  --> 192.168.43.33 (TCP)
[193844.834623] *** LOCAL_OUT
[193844.834635]     192.168.43.33  --> 121.248.60.50 (TCP)
[193844.834647] *** POST_ROUTING
[193844.834653]     192.168.43.33  --> 121.248.60.50 (TCP)
[193844.835494] *** LOCAL_OUT
[193844.835498]     192.168.43.33  --> 121.248.60.50 (TCP)
[193844.835508] *** POST_ROUTING
[193844.835511]     192.168.43.33  --> 121.248.60.50 (TCP)
[193844.871497] *** PRE_ROUTING
[193844.871584]     121.248.60.50  --> 192.168.43.33 (TCP)
[193844.871608] *** LOCAL_IN
[193844.871616]     121.248.60.50  --> 192.168.43.33 (TCP)
```

数据报从进入系统，进行IP校验以后，首先经过第一个HOOK函数NF_IP_PRE_ROUTING进行处理；

然后就进入路由代码，其决定该数据报是需要转发还是发给本机的；

若该数据报是发被本机的，则该数据经过HOOK函数NF_IP_LOCAL_IN处理以后传递给上层协议；

若该数据报应该被转发则它被NF_IP_FORWARD处理；

经过转发的数据报经过最后一个HOOK函数NF_IP_POST_ROUTING处理以后，再传输到网络上；

本地产生的数据经过HOOK函数NF_IP_LOCAL_OUT 处理后，进行路由选择处理，然后经过NF_IP_POST_ROUTING处理后发送出去。

3. Implement two more hooks to achieve the following.

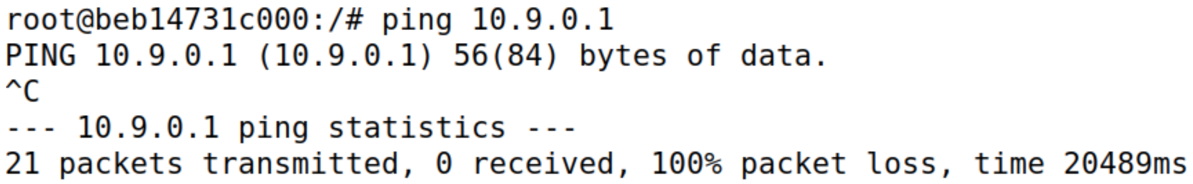
1. 内核代码修改如下：

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

  char ip[16] = "10.9.0.1";
  u32 ip_addr;

  if (!skb) return NF_ACCEPT;

  iph = ip_hdr(skb);
  // Convert the IPv4 address from dotted decimal to 32-bit binary
  in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

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

  return NF_ACCEPT;
}

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

  u16 port = 23;
  char ip[16] = "10.9.0.1";
  u32 ip_addr;

  if (!skb) return NF_ACCEPT;

  iph = ip_hdr(skb);
  // Convert the IPv4 address from dotted decimal to 32-bit binary
  in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

  if (iph->protocol == IPPROTO_TCP)
  {
    tcph = tcp_hdr(skb);
```

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

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);

    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);

    hook3.hook = block_ping;
    hook3.hooknum = NF_INET_LOCAL_IN;
    hook3.pf = PF_INET;
```

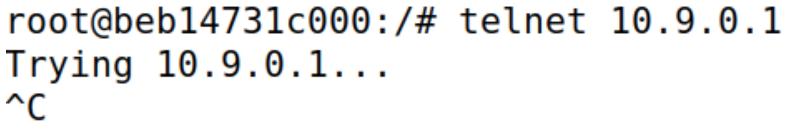
```
  hook3.priority = NF_IP_PRI_FIRST;
  nf_register_net_hook(&init_net, &hook3);

  hook4.hook = block_telnet;
  hook4.hooknum = NF_INET_LOCAL_IN;
  hook4.pf = PF_INET;
  hook4.priority = NF_IP_PRI_FIRST;
  nf_register_net_hook(&init_net, &hook4);
  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);
}
```

2. 编译内核，使用10.9.0.5 ping 10.9.0.1，结果如下：



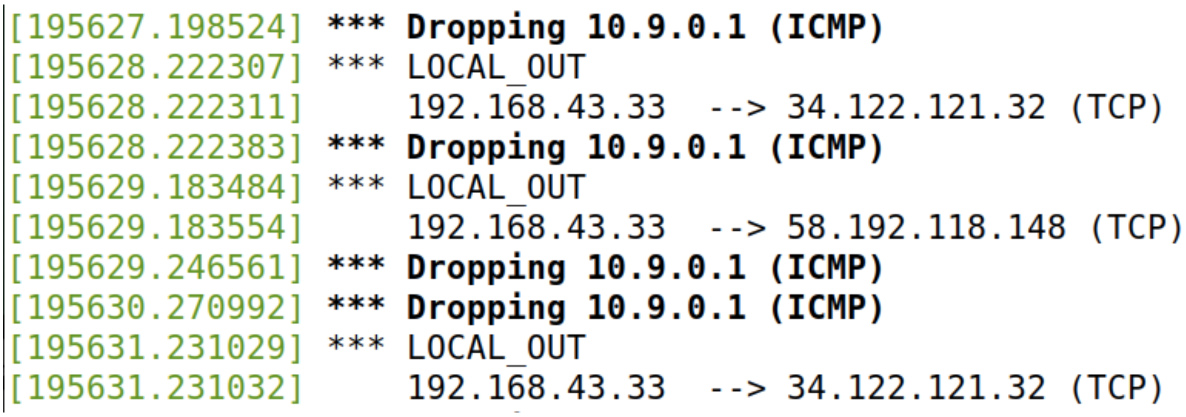
```
root@beb14731c000:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
21 packets transmitted, 0 received, 100% packet loss, time 20489ms
```

3. 使用10.9.0.5 telnet 10.9.0.1，结果如下：

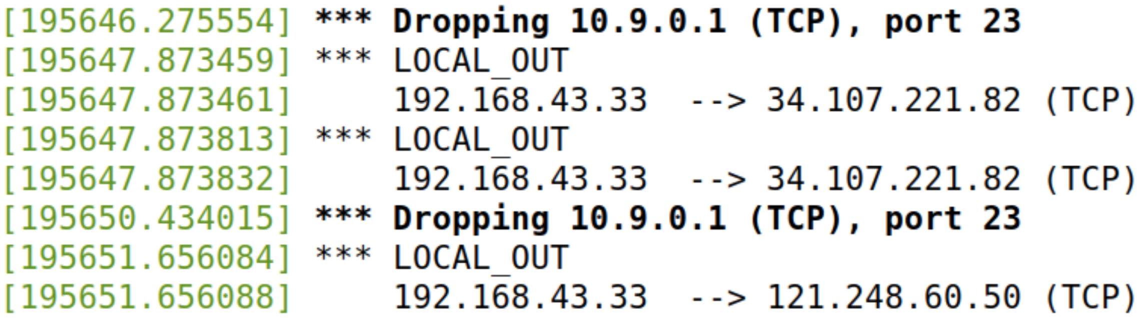


```
root@beb14731c000:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
```

4. 执行dmesg命令，结果如下：



```
[195627.198524] *** Dropping 10.9.0.1 (ICMP)
[195628.222307] *** LOCAL_OUT
[195628.222311]     192.168.43.33  --> 34.122.121.32 (TCP)
[195628.222383] *** Dropping 10.9.0.1 (ICMP)
[195629.183484] *** LOCAL_OUT
[195629.183554]     192.168.43.33  --> 58.192.118.148 (TCP)
[195629.246561] *** Dropping 10.9.0.1 (ICMP)
[195630.270992] *** Dropping 10.9.0.1 (ICMP)
[195631.231029] *** LOCAL_OUT
[195631.231032]     192.168.43.33  --> 34.122.121.32 (TCP)
```



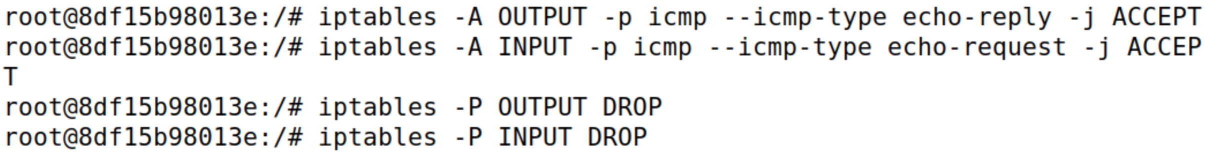
```
[195646.275554] *** Dropping 10.9.0.1 (TCP), port 23
[195647.873459] *** LOCAL_OUT
[195647.873461]     192.168.43.33  --> 34.107.221.82 (TCP)
[195647.873813] *** LOCAL_OUT
[195647.873832]     192.168.43.33  --> 34.107.221.82 (TCP)
[195650.434015] *** Dropping 10.9.0.1 (TCP), port 23
[195651.656084] *** LOCAL_OUT
[195651.656088]     192.168.43.33  --> 121.248.60.50 (TCP)
```

可见ping和telnet的相关报文都被丢弃了。

Task 2: Experimenting with Stateless Firewall Rules

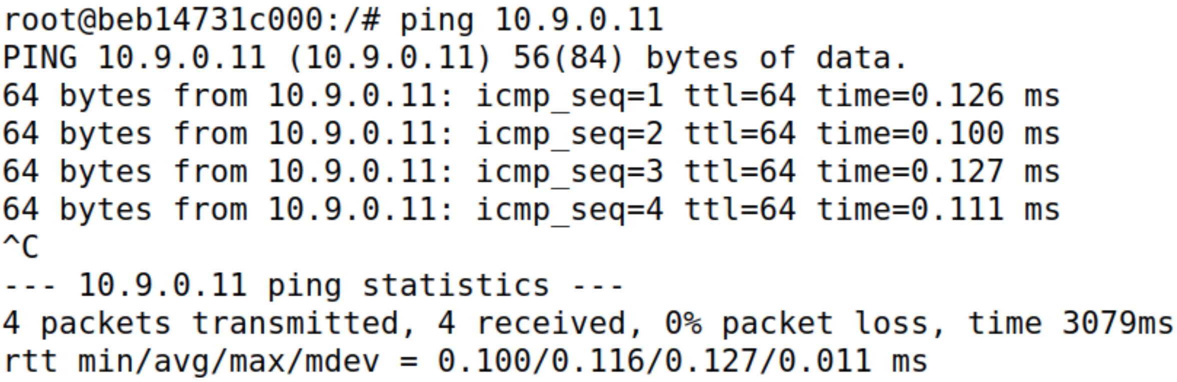
Task 2.A: Protecting the Router

1. 在路由器中执行以下命令，允许其他主机ping通防火墙，设置INPUT和OUTPUT链默认为丢包，结果如下：



```
root@8df15b98013e:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@8df15b98013e:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEP
T
root@8df15b98013e:/# iptables -P OUTPUT DROP
root@8df15b98013e:/# iptables -P INPUT DROP
```

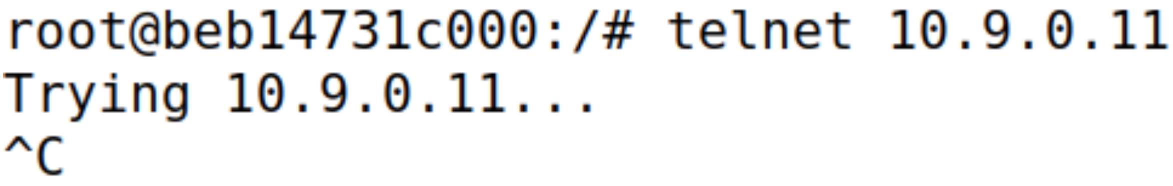
2. 执行命令ping 10.9.0.11，结果如下：



```
root@beb14731c000:/# 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=0.126 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.127 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.111 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.100/0.116/0.127/0.011 ms
```

可见路由器能够被ping通。

3. 执行命令telnet 10.9.0.11，结果如下：

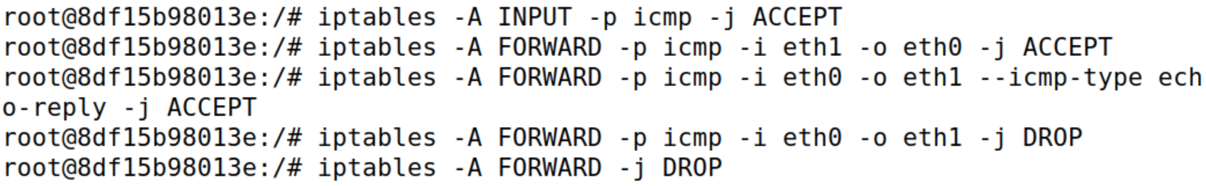


```
root@beb14731c000:/# telnet 10.9.0.11
Trying 10.9.0.11...
^C
```

可见不能够telnet到路由器。

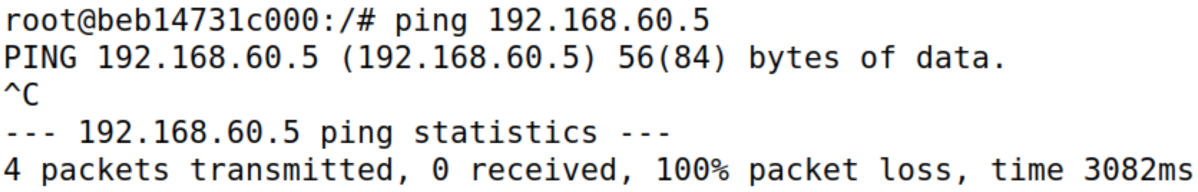
Task 2.B: Protecting the Internal Network

1. 在路由器中执行以下命令，结果如下：



```
root@8df15b98013e:/# iptables -A INPUT -p icmp -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p icmp -i eth1 -o eth0 -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p icmp -i eth0 -o eth1 --icmp-type ech
o-reply -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p icmp -i eth0 -o eth1 -j DROP
root@8df15b98013e:/# iptables -A FORWARD -j DROP
```

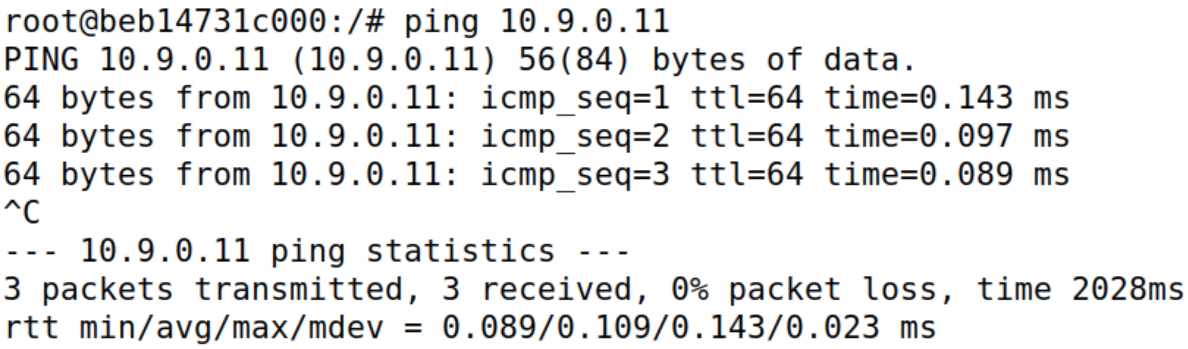
2. 使用10.9.0.5 ping 192.168.60.5，结果如下：



```
root@beb14731c000:/# 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 3082ms
```

可见外部主机无法ping通内部主机。

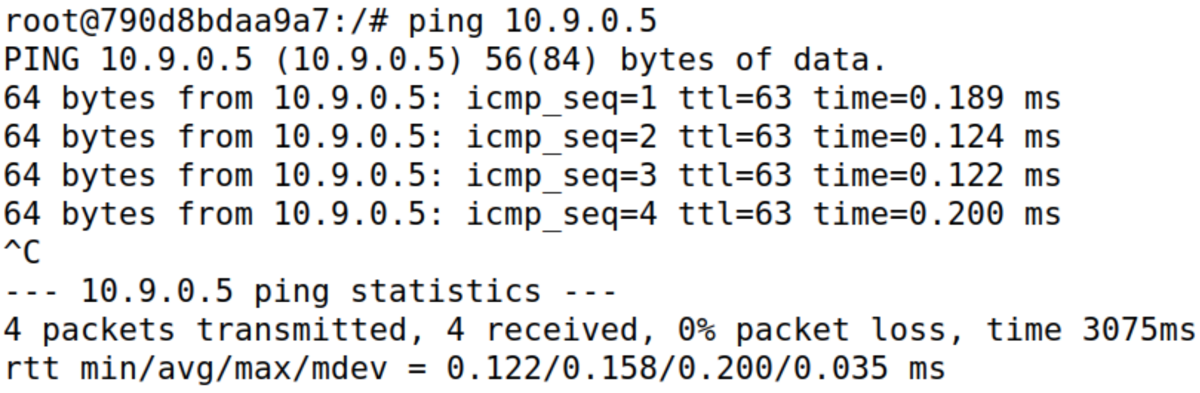
3. 使用10.9.0.5 ping 10.9.0.11，结果如下：



```
root@beb14731c000:/# 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=0.143 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.097 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.089 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2028ms
rtt min/avg/max/mdev = 0.089/0.109/0.143/0.023 ms
```

可见外部主机可以ping通路由器。

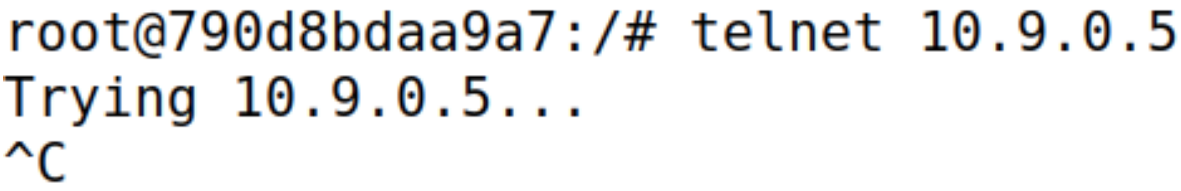
4. 使用192.168.60.5 ping 10.9.0.5，结果如下：



```
root@790d8bdaa9a7:/# 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=0.189 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.124 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.122 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.200 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3075ms
rtt min/avg/max/mdev = 0.122/0.158/0.200/0.035 ms
```

可见内部主机可以ping通外部主机。

5. 使用192.168.60.5 telnet 10.9.0.5，结果如下：

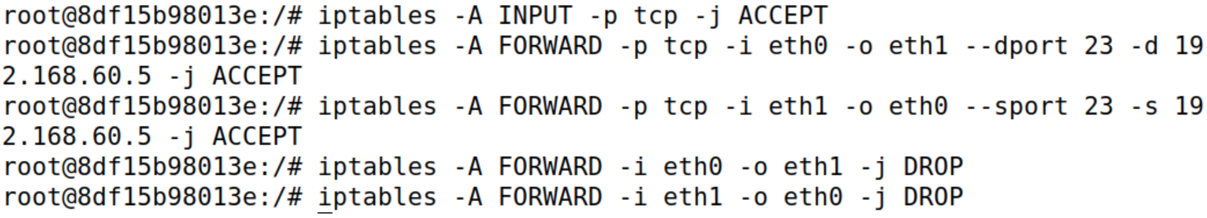


```
root@790d8bdaa9a7:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
```

可见内网和外网之间的所有其他数据包都被阻止。

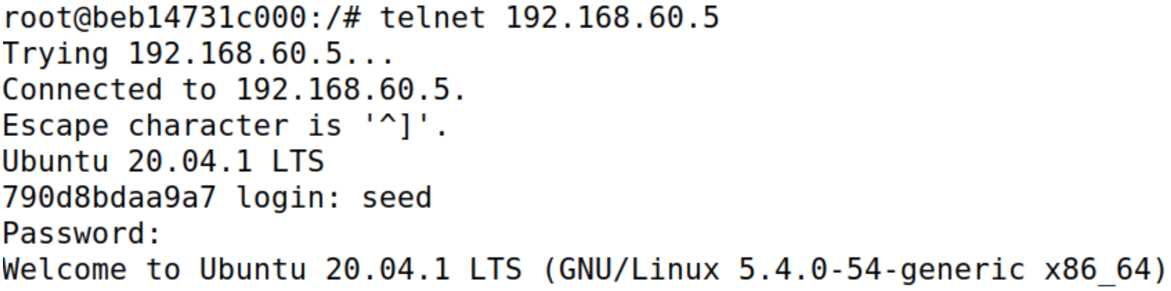
Task 2.C: Protecting Internal Servers

1. 在路由器中执行以下命令，结果如下：

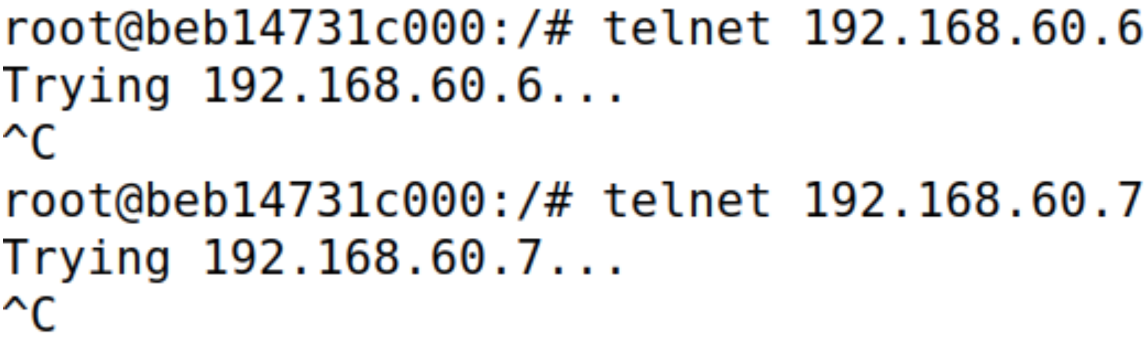


```
root@8df15b98013e:/# iptables -A INPUT -p tcp -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth0 -o eth1 --dport 23 -d 19
2.168.60.5 -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth1 -o eth0 --sport 23 -s 19
2.168.60.5 -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -i eth0 -o eth1 -j DROP
root@8df15b98013e:/# iptables -A FORWARD -i eth1 -o eth0 -j DROP
```

2. 使用10.9.0.5 telnet 192.168.60.5，telnet 192.168.60.6，telnet 192.168.60.7，结果如下：



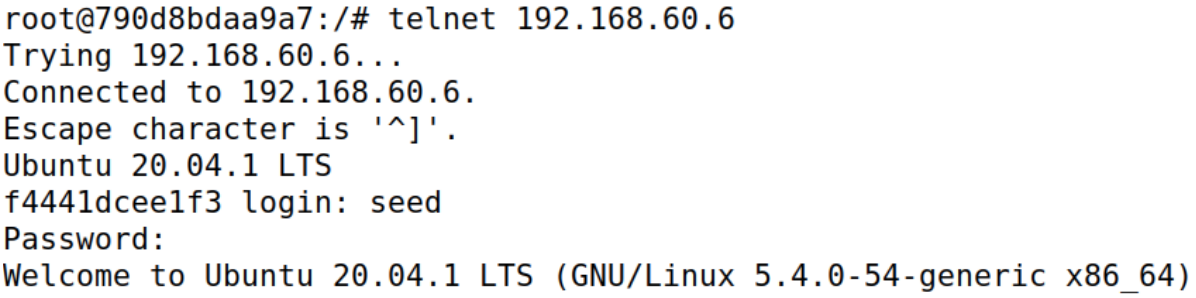
```
root@beb14731c000:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
790d8bdaa9a7 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```



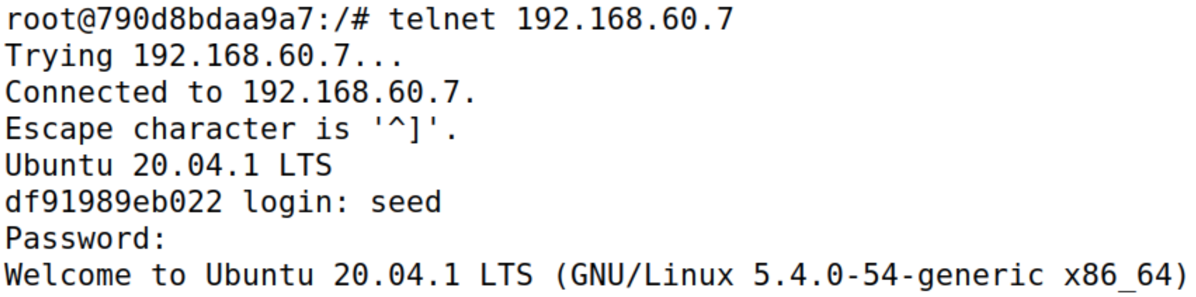
```
root@beb14731c000:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@beb14731c000:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
```

可见外部主机只能telnet192.168.60.5，而不能telnet内网中的其他主机。

3. 使用192.168.60.5 telnet 192.168.60.6，telnet 192.168.60.7，结果如下：



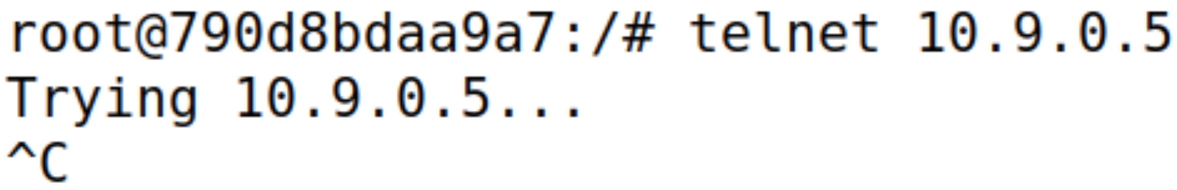
```
root@790d8bdaa9a7:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f4441dcee1f3 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```



```
root@790d8bdaa9a7:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
df91989eb022 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

可见内网主机可以访问所有内网服务器（以192.168.60.5为例）。

4. 使用192.168.60.5 telnet 10.9.0.5，结果如下：



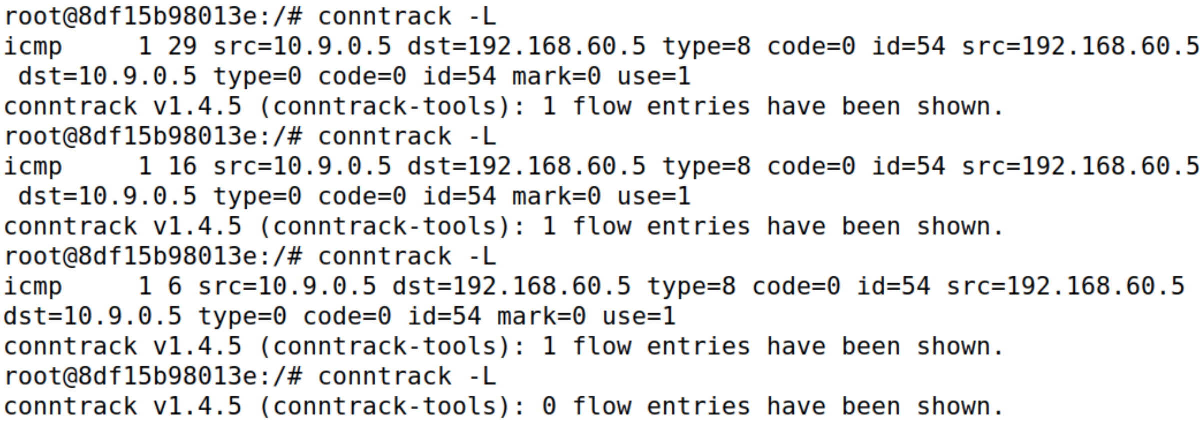
```
root@790d8bdaa9a7:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
```

可见内网主机不可以访问外网服务器（以192.168.60.5为例）。

Task 3: Connection Tracking and Stateful Firewal

Task 3.A: Experiment with the Connection Tracking

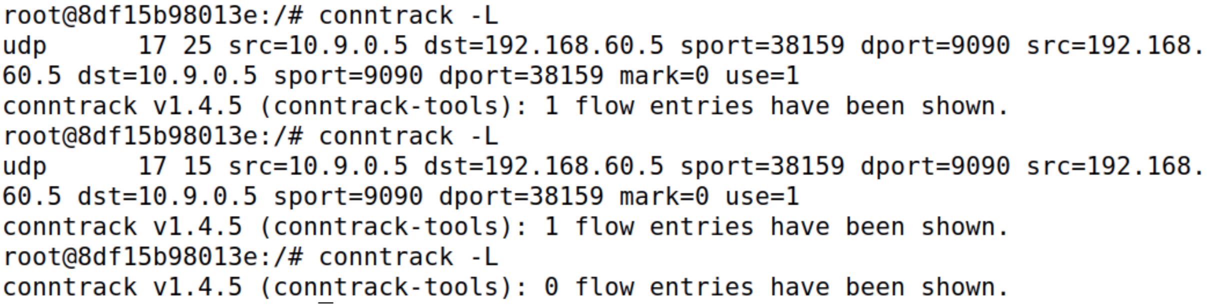
1. 使用10.9.0.5 ping 192.168.60.5，在路由器上运行命令conntrack -L，结果如下：



```
root@8df15b98013e:/# conntrack -L
icmp     1 29 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 type=0 code=0 id=54 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@8df15b98013e:/# conntrack -L
icmp     1 16 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 type=0 code=0 id=54 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@8df15b98013e:/# conntrack -L
icmp     1 6 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 type=0 code=0 id=54 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@8df15b98013e:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

可见ICMP的连接状态持续时间为30秒。

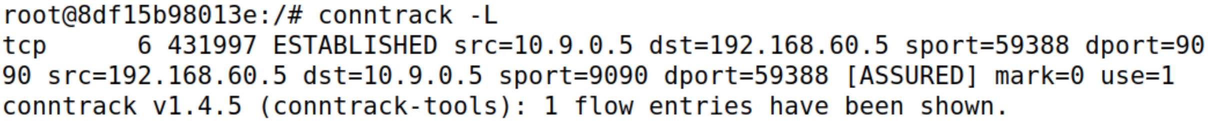
2. 在192.168.60.5上开启netcat服务nc -lu 9090， 在10.9.0.5上运行命令nc -u 192.168.60.5 9090，在路由器上运行命令conntrack -L，结果如下：



```
root@8df15b98013e:/# conntrack -L
udp      17 25 src=10.9.0.5 dst=192.168.60.5 sport=38159 dport=9090 src=192.168.
60.5 dst=10.9.0.5 sport=9090 dport=38159 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@8df15b98013e:/# conntrack -L
udp      17 15 src=10.9.0.5 dst=192.168.60.5 sport=38159 dport=9090 src=192.168.
60.5 dst=10.9.0.5 sport=9090 dport=38159 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@8df15b98013e:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

可见UDP的连接状态持续时间大概为25秒。

3. 在192.168.60.5上开启netcat服务nc -l 9090， 在10.9.0.5上运行命令nc 192.168.60.5 9090，在路由器上运行命令conntrack -L，结果如下：

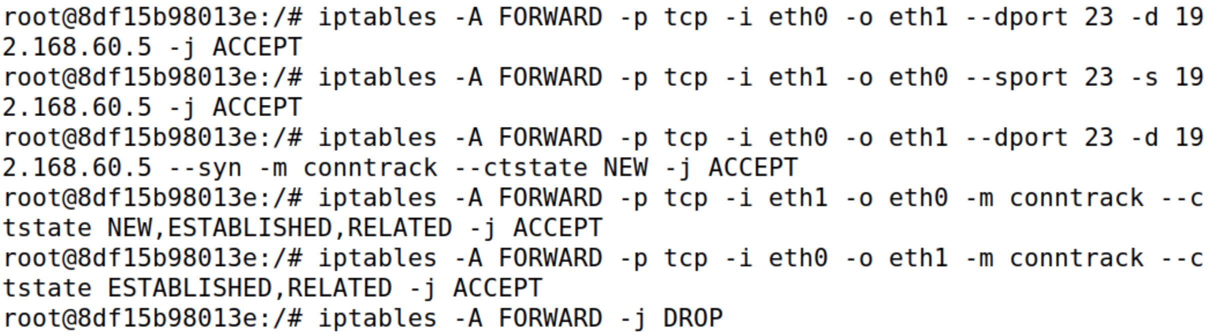


```
root@8df15b98013e:/# conntrack -L
tcp      6 431997 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=59388 dport=90
90 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=59388 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

可见TCP的连接状态持续时间为432000秒。

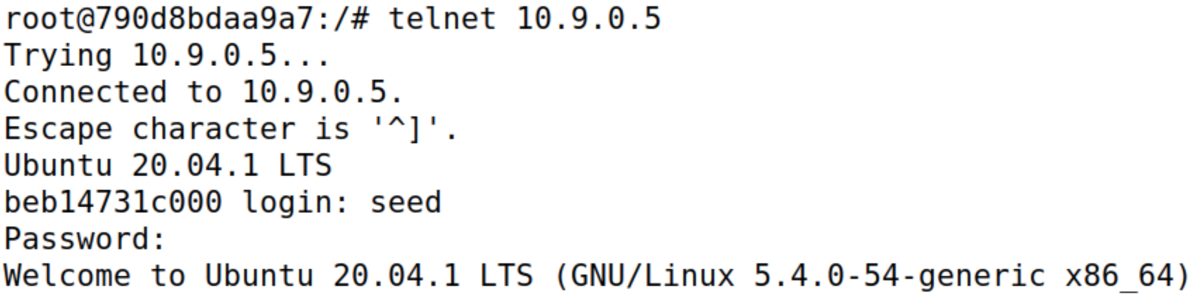
Task 3.B: Setting Up a Stateful Firewall

1. 在路由器中执行以下命令，结果如下：



```
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth0 -o eth1 --dport 23 -d 19
2.168.60.5 -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth1 -o eth0 --sport 23 -s 19
2.168.60.5 -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth0 -o eth1 --dport 23 -d 19
2.168.60.5 --syn -m conntrack --ctstate NEW -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth1 -o eth0 -m conntrack --c
tstate NEW,ESTABLISHED,RELATED -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -p tcp -i eth0 -o eth1 -m conntrack --c
tstate ESTABLISHED,RELATED -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -j DROP
```

2. 使用192.168.60.5 telnet 10.9.0.5，结果如下：

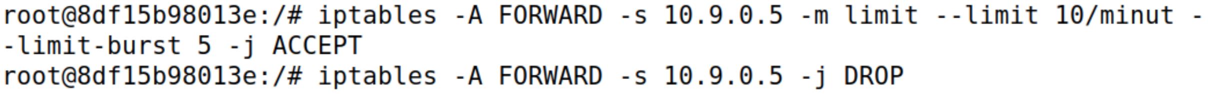


```
root@790d8bdaa9a7:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
beb14731c000 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

可见内网主机能够访问外网服务器（以192.168.60.5为例），其他访问情况与Task2.C相同。

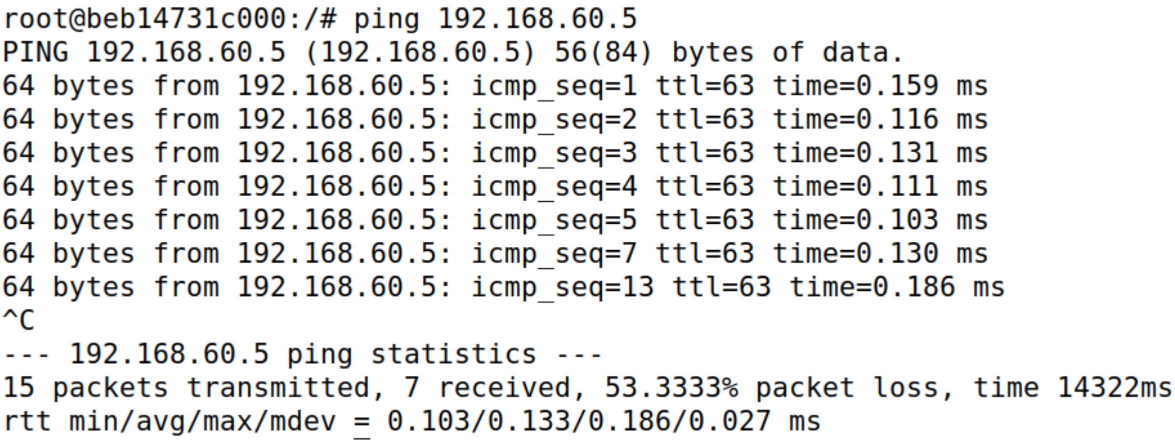
Task 4: Limiting Network Traffic

1. 在路由器中执行以下命令，结果如下：



```
root@8df15b98013e:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minut -
-limit-burst 5 -j ACCEPT
root@8df15b98013e:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
```

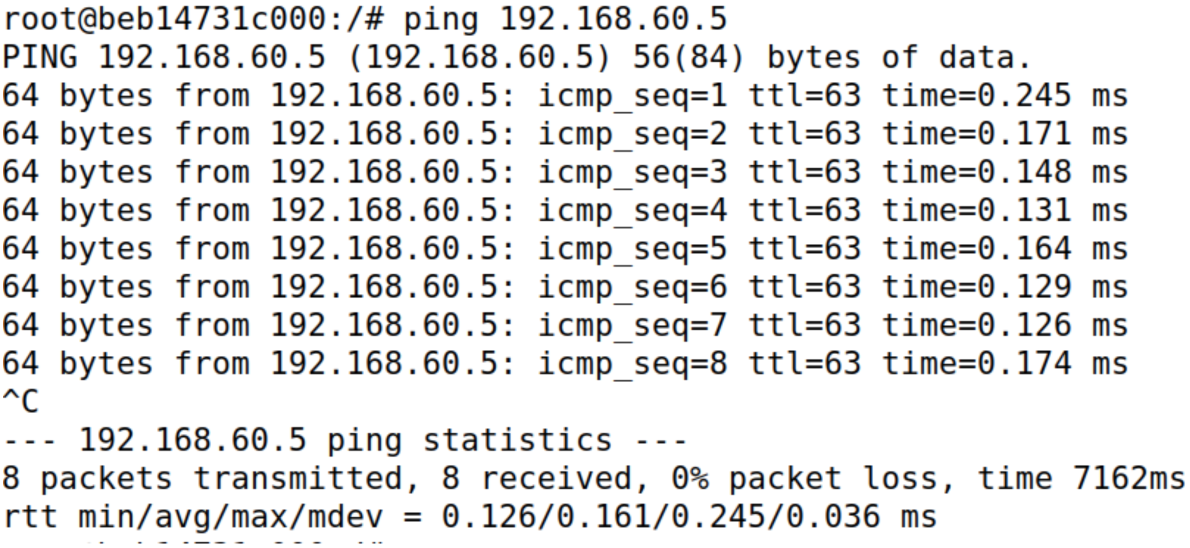
2. 使用10.9.0.5 ping 192.168.60.5，结果如下：



```
root@beb14731c000:/# 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=0.159 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.116 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.131 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.111 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.103 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.130 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.186 ms
^C
--- 192.168.60.5 ping statistics ---
15 packets transmitted, 7 received, 53.3333% packet loss, time 14322ms
rtt min/avg/max/mdev = 0.103/0.133/0.186/0.027 ms
```

可见前六个包发送很快，后面每隔六秒发送一个包。

3. 在路由器中只执行第一条命令，使用10.9.0.5 ping 192.168.60.5，结果如下：

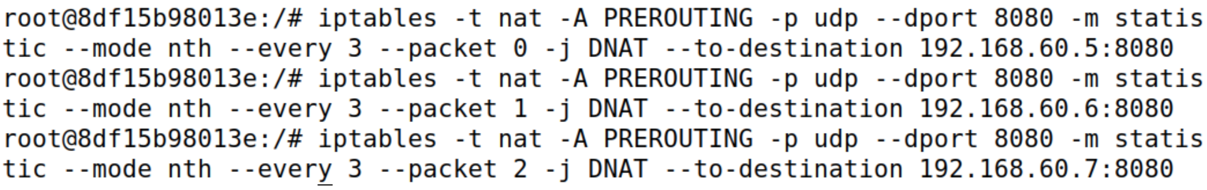


```
root@beb14731c000:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.245 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.171 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.148 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.131 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.164 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.129 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.126 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.174 ms
^C
--- 192.168.60.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7162ms
rtt min/avg/max/mdev = 0.126/0.161/0.245/0.036 ms
```

可见包发送的速度和正常一样，因为 iptables 默认的FORWARD 表是接受所有包,所以如果不写第二条命令,发包会正常进行。

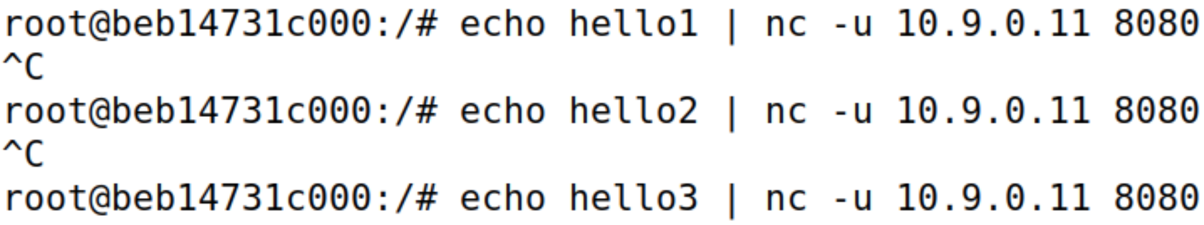
Task 5: Load Balancing

1. 在路由器中执行以下命令（**nth mode**），结果如下：

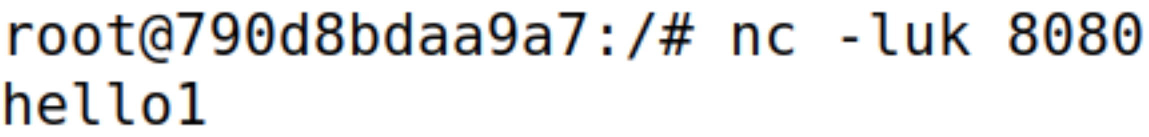


```
root@8df15b98013e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
root@8df15b98013e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 3 --packet 1 -j DNAT --to-destination 192.168.60.6:8080
root@8df15b98013e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 3 --packet 2 -j DNAT --to-destination 192.168.60.7:8080
```

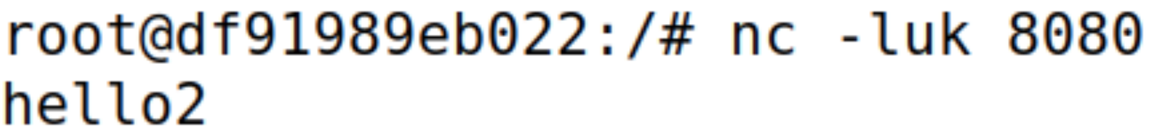
2. 在 10.9.0.5 上给路由器发报文，结果如下：



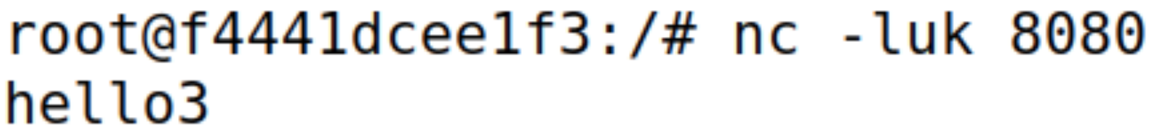
```
root@beb14731c000:/# echo hello1 | nc -u 10.9.0.11 8080
^C
root@beb14731c000:/# echo hello2 | nc -u 10.9.0.11 8080
^C
root@beb14731c000:/# echo hello3 | nc -u 10.9.0.11 8080
```



```
root@790d8bdaa9a7:/# nc -luk 8080
hello1
```



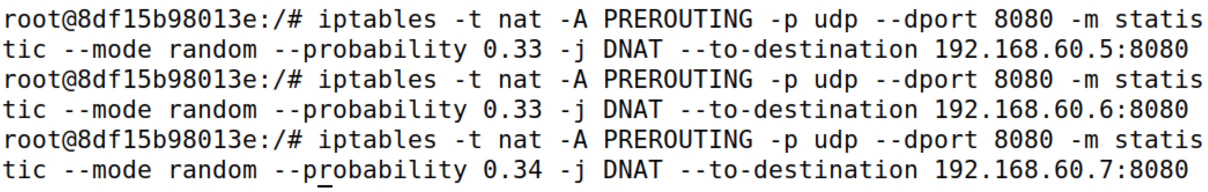
```
root@df91989eb022:/# nc -luk 8080
hello2
```



```
root@f4441dcee1f3:/# nc -luk 8080
hello3
```

可见由于负载均衡，各个主机监听到的报文数量平均。

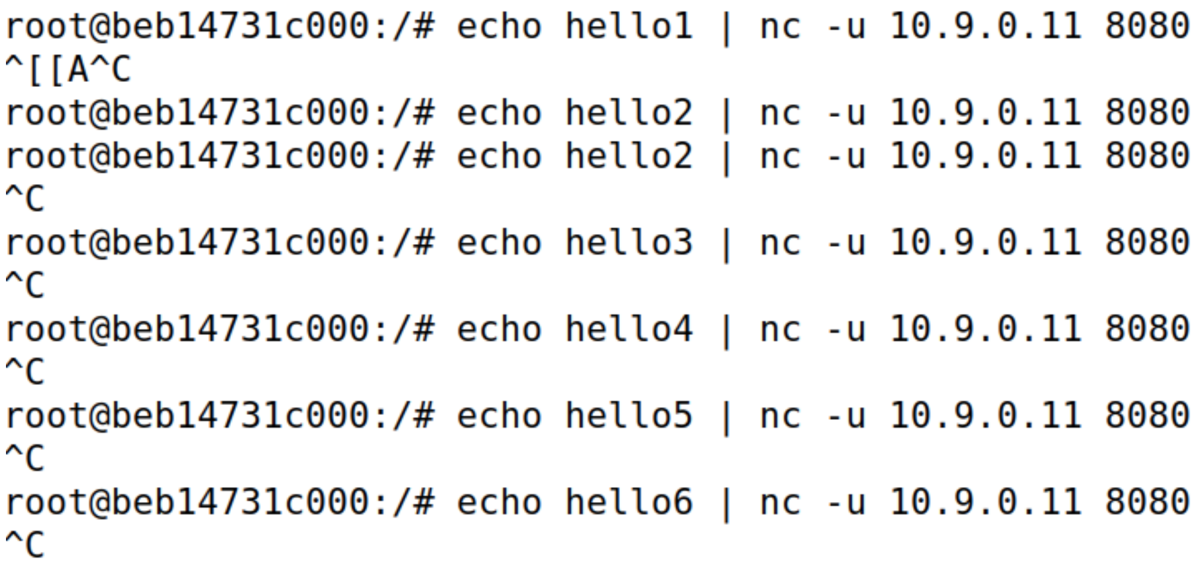
3. 在路由器中执行以下命令（**random mode**），结果如下：



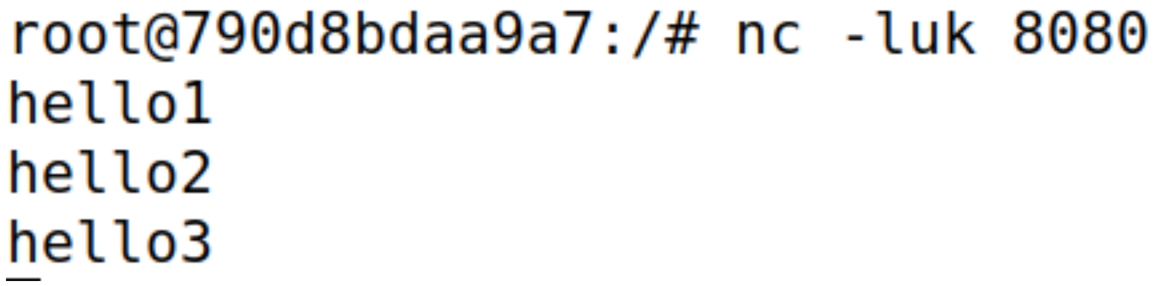
```
root@8df15b98013e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.5:8080
root@8df15b98013e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.6:8080
root@8df15b98013e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.34 -j DNAT --to-destination 192.168.60.7:8080
```

可见在样本数量小的时候，主机收到的报文数量并不平均，但当样本数量足够大时，由于负载均衡，各个主机监听到的报文数量平均。

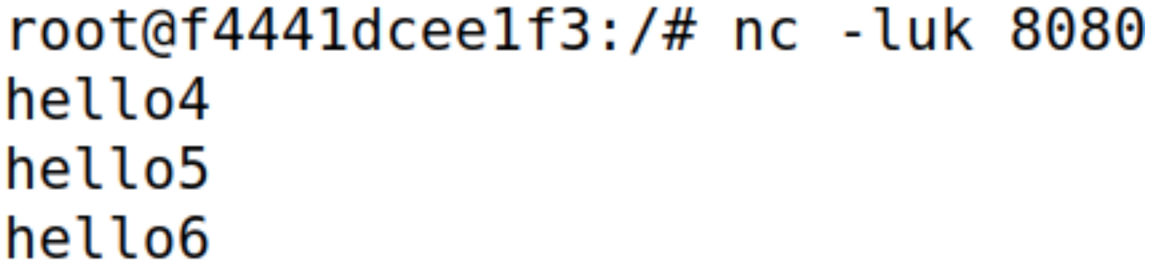
4. 在 10.9.0.5 上给路由器发报文，结果如下：



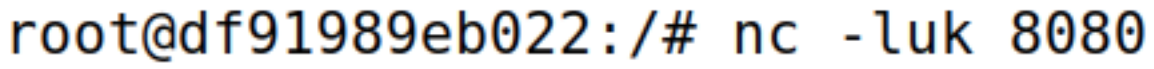
```
root@beb14731c000:/# echo hello1 | nc -u 10.9.0.11 8080
^[[A^C
root@beb14731c000:/# echo hello2 | nc -u 10.9.0.11 8080
root@beb14731c000:/# echo hello2 | nc -u 10.9.0.11 8080
^C
root@beb14731c000:/# echo hello3 | nc -u 10.9.0.11 8080
^C
root@beb14731c000:/# echo hello4 | nc -u 10.9.0.11 8080
^C
root@beb14731c000:/# echo hello5 | nc -u 10.9.0.11 8080
^C
root@beb14731c000:/# echo hello6 | nc -u 10.9.0.11 8080
^C
```



```
root@790d8bdaa9a7:/# nc -luk 8080
hello1
hello2
hello3
```



```
root@f4441dcee1f3:/# nc -luk 8080
hello4
hello5
hello6
```



```
root@df91989eb022:/# nc -luk 8080
```

可见在样本数量小的时候，主机收到的报文数量并不平均，但当样本数量足够大时，由于负载均衡，各个主机监听到的报文数量平均。