# Lab6: Firewall Exploration Lab

57118228 孙志刚

## Task 1: Implementing a Simple Firewall

### Task 1.A: Implement a Simple Kernel Module

将kernel_module这个文件夹移到一个没有空格（/home/seed/）的目录下，然后直接make编译，可以看到编译成功。

```
seed@VM: ~/kernel_module
[08/04/21]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'
```

然后下一步通过insmod hello.ko将其插入，可以这个模块已经成功进入了内核。

```
[08/04/21]seed@VM:~/kernel_module$ sudo insmod hello.ko
[08/04/21]seed@VM:~/kernel_module$ lsmod | grep hello
hello                  16384  0
[08/04/21]seed@VM:~/kernel_module$ sudo rmmod hello.ko
```

再将其移除，恢复原始环境，通过dmesg可查看日志看见其输出。

```
[08/04/21]seed@VM:~/kernel_module$ dmesg | grep  World
[28308.248936] Hello World!
[28426.484509] Bye-bye World!.
```

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

**1.使用提供的Makefile编译示例代码。将它加载到内核中，并演示防火墙按预期工作。**可以通过命令dig @8.8.8.8 [www.example.com](www.example.com)生成到谷歌的DNS服务器8.8.8.8的UDP报文。如果你的防火墙工作，你的请求将被阻止；否则，您将得到一个响应。

加载内核前，可以看到 dig @8.8.8.8 www.example 命令可以得到响应

seed@VM: ~/packet_filter

```
[08/04/21]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: 41764
;; 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.        6011    IN      A       93.184.216.34

;; Query time: 32 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Aug 04 13:39:38 EDT 2021
;; MSG SIZE  rcvd: 60
```

和前面一样，将文件拷贝到 /home/seed/ 下面进行编译。加载到内核后，可以看到防火墙生效。

seed@VM: ~/packet_filter

```
[08/04/21]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'
[08/04/21]seed@VM:~/packet_filter$ sudo insmod seedFilter.ko
[08/04/21]seed@VM:~/packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
[08/04/21]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

[08/04/21]seed@VM:~/packet_filter$
```

**2.将printInfo函数与所有的netfilter hook挂钩。使用实验结果来帮助解释每个hook函数在什么情况下会被调用。**在进行实验时，每次修改完代码，需要重新 make 编译，然后使用 sudo insmod seedFilter.ko 加载到内核，可以使用 lsmod | grep seedFilter 查看模块是否在内核，进行 dig @8.8.8.8 www.example.com 操作后，可使用 sudo dmesg -c 查看信息，每次测试后，需要运行 sudo rmmod seedFilter 从内核中移除模块。

```
NF_INET_PRE_ROUTING
NF_INET_LOCAL_IN
NF_INET_FORWARD
NF_INET_LOCAL_OUT
NF_INET_POST_ROUTING
```

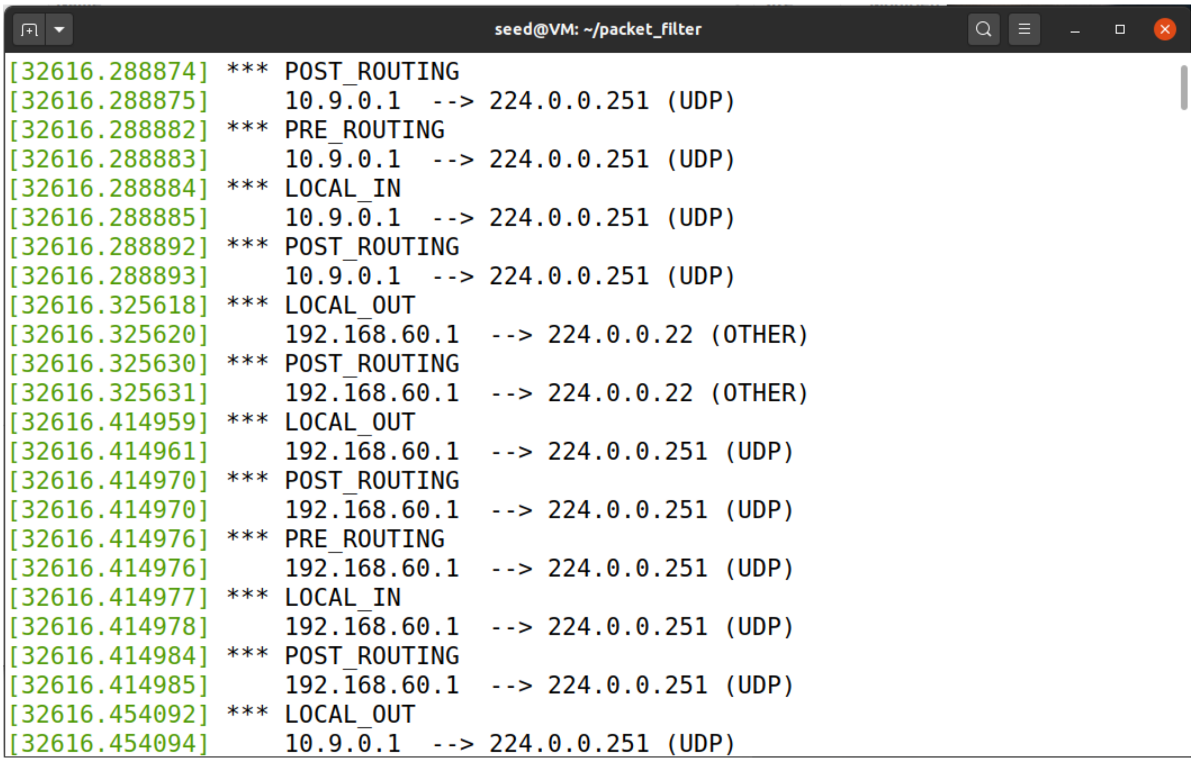
修改seedFilter.c文件

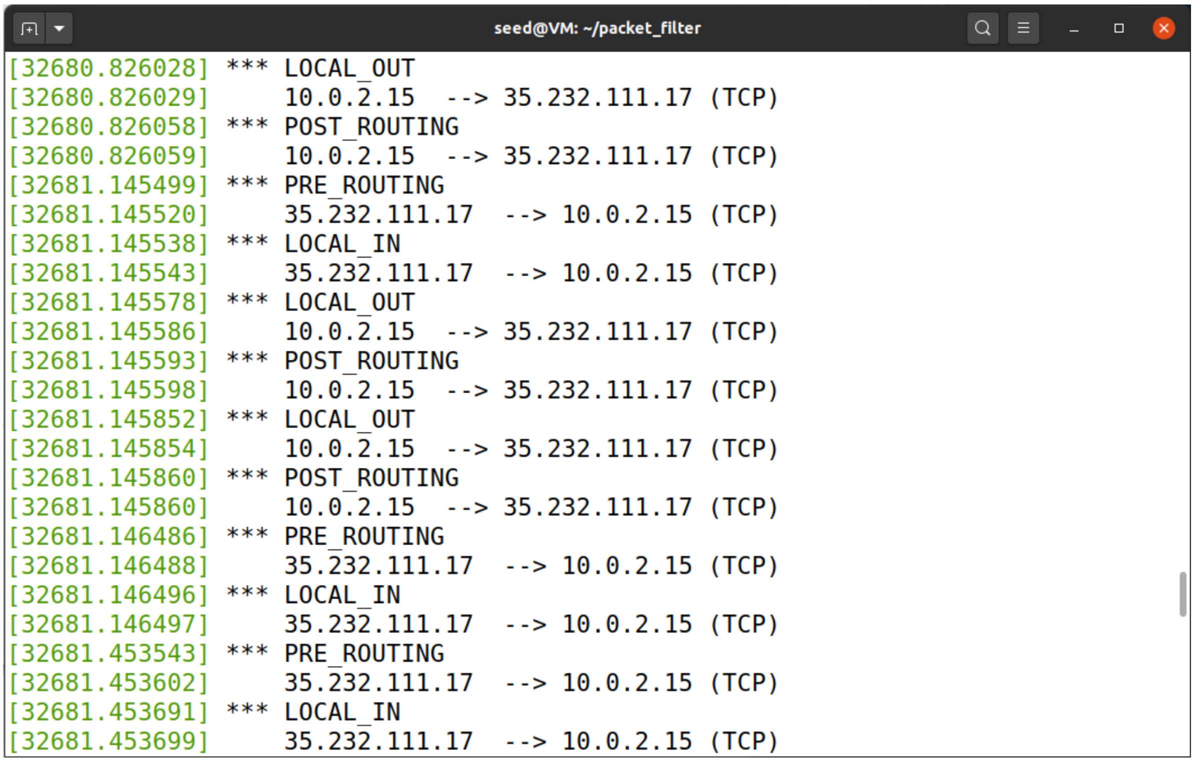
```
...
static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5;
...
int registerFilter(void) {
   printk(KERN_INFO "Registering filters.\n");

   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);


   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);
```

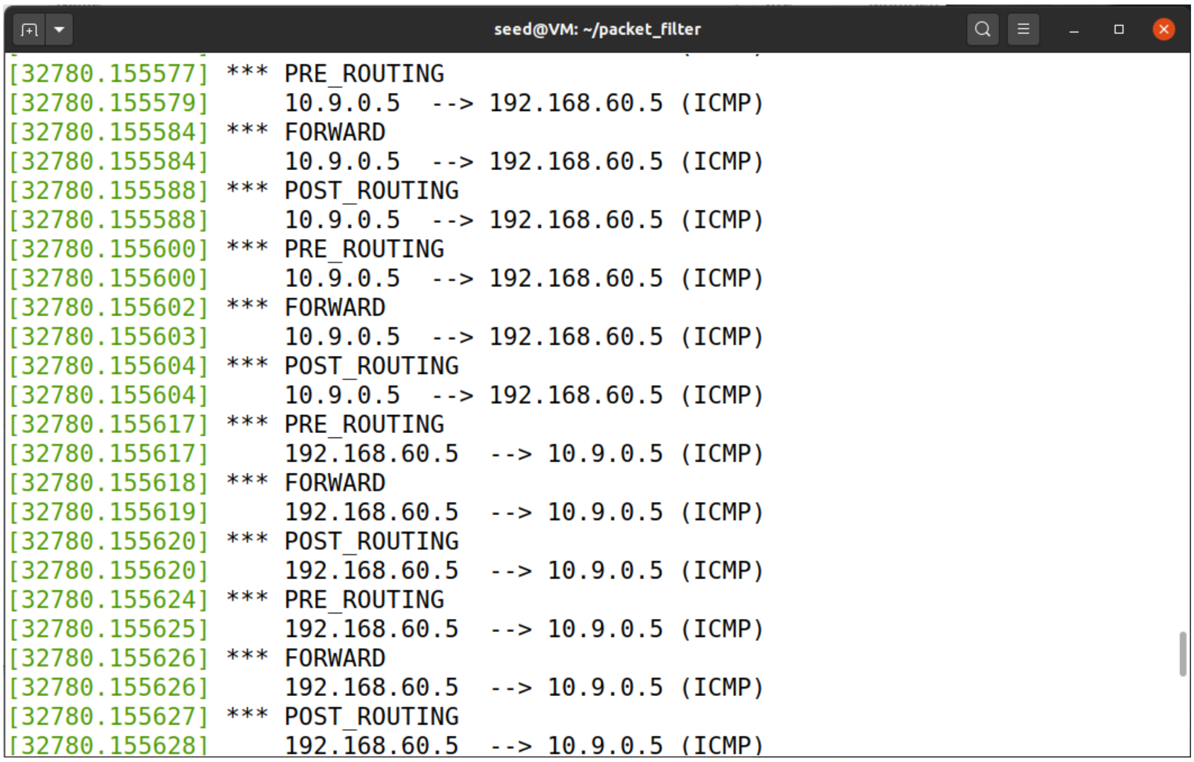
```
}
...
```

利用make命令编译可装载内核模块，并且利用insmod命令插入内核模块

在用户主机上ping内网主机192.168.60.5，得到结果如下，可知能够连接。mesg命令查看/var/log/syslog文件中的信息






结果分析：

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_INET_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_INET_LOCAL_IN** 处理以后然后传递给上层协议。

若该数据报应该被转发，则它被 **NF_INET_FORWARD** 处理挂载 **NF_INET_LOCAL_OUT** 时，本机产生的数据包将会第一个到达此HOOK ，数据经过HOOK 函数 **NF_INET_LOCAL_OUT** 处理后，进行路由选择处理，然后经过 **NF_INET_POST_ROUTING** 处理后发送出 去。

### 3.再实现两个HOOK，实现以下目的:(1)防止其他计算机ping VM，(2)防止其他计算机telnet到VM。请 实现两个不同的HOOK函数。

根据题目要求，分别设计两个钩子函数，一个为blockTELNET，判断是否为TCP协议，并且目的端口为 23，另一个为blockPING，判断是否为ICMP协议，两个函数都挂在 **NF_INET_LOCAL_IN**这个钩子下即 可。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/icmp.h>
#include <linux/udp.h>
```

```
#include <linux/if_ether.h>
#include <linux/inet.h>

static struct nf_hook_ops hook1, hook2, hook3, hook4;

unsigned int blockUDP(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct udphdr *udph;

   u16  port   = 53;
   char ip[16] = "8.8.8.8";
   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_UDP) {
       udph = udp_hdr(skb);
       if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %d\n",
&(iph->daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}

unsigned int blockTCP(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;
}

unsigned int blockICMP(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct icmphdr *icmph;

    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) {
        icmph = icmp_hdr(skb);
        if (iph->daddr == ip_addr){
             printk(KERN_WARNING "*** Dropping %pI4 (ICMP), port %d\n",
&(iph->daddr));
             return NF_DROP;
         }
    }
    return NF_ACCEPT;
}

unsigned int printInfo(void *priv, struct sk_buff *skb,
                 const struct nf_hook_state *state)
{
    struct iphdr *iph;
    char *hook;
    char *protocol;

    switch (state->hook){
      case NF_INET_LOCAL_IN:     hook = "LOCAL_IN";     break;
      case NF_INET_LOCAL_OUT:    hook = "LOCAL_OUT";    break;
      case NF_INET_PRE_ROUTING:  hook = "PRE_ROUTING";  break;
      case NF_INET_POST_ROUTING: hook = "POST_ROUTING"; break;
      case NF_INET_FORWARD:      hook = "FORWARD";      break;
```

```
    default:                   hook = "IMPOSSIBLE";   break;
  }
  printk(KERN_INFO "*** %s\n", hook); // Print out the hook info

  iph = ip_hdr(skb);
  switch (iph->protocol){
    case IPPROTO_UDP:  protocol = "UDP";   break;
    case IPPROTO_TCP:  protocol = "TCP";   break;
    case IPPROTO_ICMP: protocol = "ICMP";  break;
    default:           protocol = "OTHER"; break;

  }
  // Print out the IP addresses and protocol
  printk(KERN_INFO "    %pI4  --> %pI4 (%s)\n",
                   &(iph->saddr), &(iph->daddr), protocol);

  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 = blockICMP;
  hook3.hooknum = NF_INET_PRE_ROUTING;
  hook3.pf = PF_INET;
  hook3.priority = NF_IP_PRI_FIRST;
  nf_register_net_hook(&init_net, &hook3);

  hook4.hook = blockTCP;
  hook4.hooknum = NF_INET_PRE_ROUTING;
  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);

}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

加载内核，开启容器，在10.9.0.5容器上分别进行ping 10.9.0.1 和telnet 10.9.0.1 发现都不通

```
root@ee3a304f2907:/# 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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4102ms

root@ee3a304f2907:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@ee3a304f2907:/#
```

dmesg -c 查看：

```
[08/04/21]seed@VM:~/packet_filter$ sudo dmesg -c
[35484.237257] *** Dropping 10.9.0.1 (ICMP), port 49
[35485.268204] *** Dropping 10.9.0.1 (ICMP), port 49
[35486.290906] *** Dropping 10.9.0.1 (ICMP), port 49
[35487.314704] *** Dropping 10.9.0.1 (ICMP), port 49
[35488.338481] *** Dropping 10.9.0.1 (ICMP), port 49
[35491.409690] *** Dropping 10.9.0.1 (TCP), port 23
[35492.433663] *** Dropping 10.9.0.1 (TCP), port 23
[35494.452432] *** Dropping 10.9.0.1 (TCP), port 23
[08/04/21]seed@VM:~/packet_filter$
```

# Task 2: Experimenting with Stateless Firewall Rules

每个任务前都要清理 table 或重启路由器的 docker 。

## Task 2.A: Protecting the Router

用户主机的IP地址为10.9.0.5，路由器的IP地址为10.9.0.11，内网网段的IP地址192.168.60.0/24。

在路由器上设置以下过滤规则：（文档提供的命令是错的，如下面所示，就可以成功了。）

```
# 允许其他主机ping通防火墙
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
# 设置INPUT和OUTPUT链默认为丢包
iptables -P OUTPUT DROP
iptables -P INPUT DROP
```

结果发现，从10.9.0.5上可以ping通路由器，但无法telnet到路由器：

```
seed@VM: ~/packet_filter
[08/04/21]seed@VM:~/packet_filter$ docksh 95
root@9521273d6a2d:/# 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=0.398 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.055 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.065 ms
^C
--- 10.9.0.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4092ms
rtt min/avg/max/mdev = 0.055/0.131/0.398/0.133 ms
root@9521273d6a2d:/# telnet 10.9.0.11
Trying 10.9.0.11...
^C
root@9521273d6a2d:/#
```

将上述规则取消掉，发现可以ping和telnet

```
iptables -F
iptables -P OUTPUT ACCEPT
iptables -P INPUT ACCEPT
```

上述两条规则表示外部主机可以ping通防火墙，即其他主机可以ping通防火墙主机（即router），防火墙接收icmp的request请求报文，也可以发出icmp相应报文。

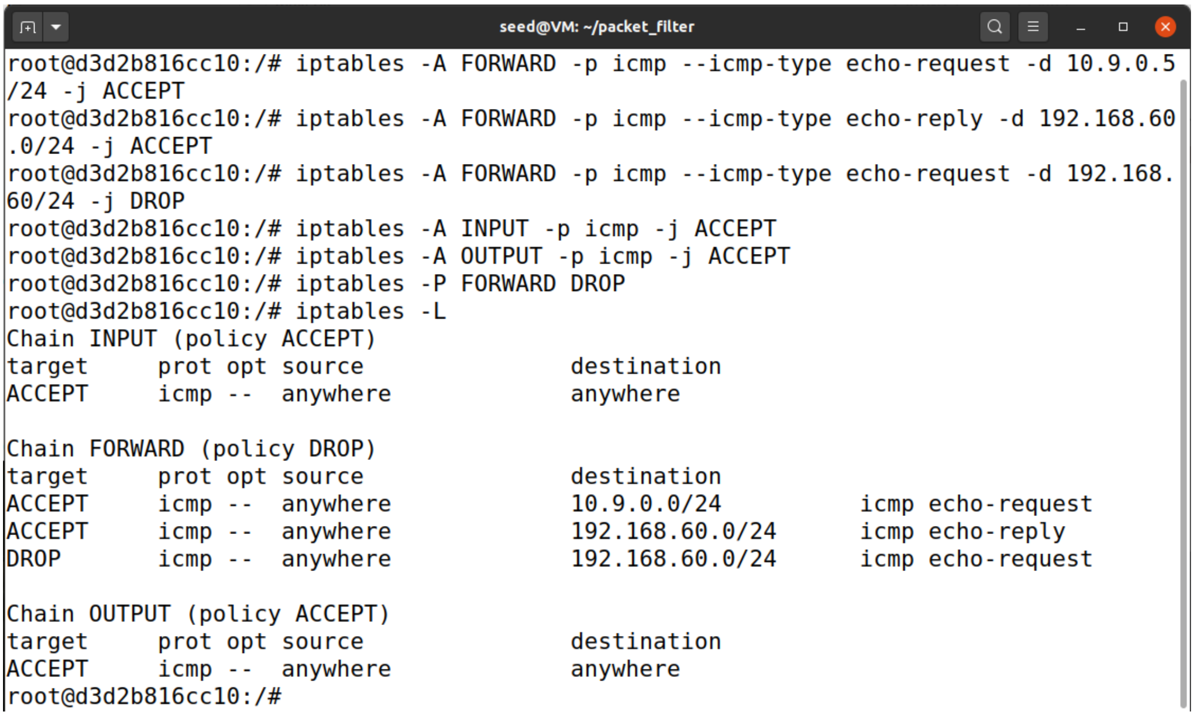
设置了**iptables -P OUTPUT DROP**后，二者无法ping通，表示丢弃所有外出的包 在单独设置了**iptables -P INPUT DROP**，可以发现，router可以ping通其他主机，但是其他主机不可以ping通router，表示所有进入的包都被丢弃了，但是外出的包不受限制。

## Task 2.B: Protecting the Internal Network

根据要求，在router上运行如下命令。

```
# 内部主机可以ping通外部主机
iptables -A FORWARD -p icmp --icmp-type echo-request -d 10.9.0.5/24 -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168.60.0/24 -j ACCEPT
# 外部主机不能ping通内部主机
iptables -A FORWARD -p icmp --icmp-type echo-request -d 192.168.60/24 -j DROP
# 路由器接受ping命令
iptables -A INPUT -p icmp -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT
# 修改策略为丢弃所有数据包
iptables -P FORWARD DROP
```

查看配置



```
seed@VM: ~/packet_filter
root@d3d2b816cc10:/# iptables -A FORWARD -p icmp --icmp-type echo-request -d 10.9.0.5/24 -j ACCEPT
root@d3d2b816cc10:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168.60.0/24 -j ACCEPT
root@d3d2b816cc10:/# iptables -A FORWARD -p icmp --icmp-type echo-request -d 192.168.60/24 -j DROP
root@d3d2b816cc10:/# iptables -A INPUT -p icmp -j ACCEPT
root@d3d2b816cc10:/# iptables -A OUTPUT -p icmp -j ACCEPT
root@d3d2b816cc10:/# iptables -P FORWARD DROP
root@d3d2b816cc10:/# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             10.9.0.0/24          icmp echo-request
ACCEPT     icmp --  anywhere             192.168.60.0/24      icmp echo-reply
DROP       icmp --  anywhere             192.168.60.0/24      icmp echo-request

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere
root@d3d2b816cc10:/#
```

从外部主机ping 路由器，可以ping 通； ping 内部主机不通； telnet 内部主机不通。

```
[08/04/21]seed@VM:~/packet_filter$ docksh 15
root@1598c767e0cf:/# 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=0.098 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.066 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.066/0.080/0.098/0.013 ms
root@1598c767e0cf:/# 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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4090ms

root@1598c767e0cf:/# telnet 192.168.60.5
Trying 192.168.60.5...
^C
root@1598c767e0cf:/#
```

内部主机ping 外部主机，可以ping 通； telnet 外部主机不通。

```
[08/04/21]seed@VM:~/packet_filter$ docksh c5
root@c5b3efcd0ec7:/# 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=0.107 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.090 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.099 ms
^C
--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2044ms
rtt min/avg/max/mdev = 0.090/0.098/0.107/0.007 ms
root@c5b3efcd0ec7:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@c5b3efcd0ec7:/#
```

## Task 2.C: Protecting Internal Servers

根据要求，在router上运行如下命令。

```
iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 -j ACCEPT
iptables -A FORWARD -p tcp --sport 23 -s 192.168.60.5 -j ACCEPT
iptables -A FORWARD -d 10.9.0.0/24 -j DROP
iptables -A FORWARD -d 192.168.60.0/24 -j DROP
```

查看配置

```
root@d3d2b816cc10:/# iptables -F
root@d3d2b816cc10:/# iptables -P OUTPUT ACCEPT
root@d3d2b816cc10:/# iptables -P INPUT ACCEPT
root@d3d2b816cc10:/# iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 -j ACCEPT
root@d3d2b816cc10:/# iptables -A FORWARD -p tcp --sport 23 -s 192.168.60.5 -j ACCEPT
root@d3d2b816cc10:/# iptables -A FORWARD -d 10.9.0.0/24 -j DROP
root@d3d2b816cc10:/# iptables -A FORWARD -d 192.168.60.0/24 -j DROP
root@d3d2b816cc10:/# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             host1-192.168.60.5.net-192.168.60.0  tcp dpt:telnet
ACCEPT     tcp  --  host1-192.168.60.5.net-192.168.60.0  anywhere            tcp spt:telnet
DROP       all  --  anywhere             10.9.0.0/24
DROP       all  --  anywhere             192.168.60.0/24

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@d3d2b816cc10:/#
```

从外部主机(10.9.0.5)telnet 192.168.60.5， 可以连接成功。

```
seed@VM: ~/packet_filter
[08/04/21]seed@VM:~/packet_filter$ docksh 15
root@1598c767e0cf:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c5b3efcd0ec7 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
```

从外部主机(10.9.0.5)telnet 192.168.60.6， 无法连接。

```
[08/04/21]seed@VM:~/packet_filter$ docksh 15
root@1598c767e0cf:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@1598c767e0cf:/#
```

从内部主机 (192.168.60.5) telnet 10.9.0.5， 无法连接，内部主机 (192.168.60.5) telnet 192.168.60.6， 连接成功。

```
seed@VM: ~/packet_filter
[08/04/21]seed@VM:~/packet_filter$ docksh c5
root@c5b3efcd0ec7:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@c5b3efcd0ec7:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c61deb4e6cd5 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

外部主机不能访问内部服务器，内部主机可以访问所有内部服务器，内部主机不可以访问外部服务器 。

所有内部主机都运行telnet服务器(侦听端口23)。外部主机只能访问192.168.60.5上的telnet服务器，不能访问其他内部主机。

# Task 3: Connection Tracking and Stateful Firewall

## Task 3.A: Experiment with the Connection Tracking

### ICMP

在用户主机上ping内网主机192.168.60.5

```
root@a6b6019f0109:/# 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=0.120 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=0.095 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.083 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.161 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.113 ms
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5148ms
rtt min/avg/max/mdev = 0.066/0.106/0.161/0.030 ms
root@a6b6019f0109:/#
```

在路由器上利用conntrack -L命令实现连接跟踪，得到结果如下

```
[08/04/21]seed@VM:~/packet_filter$ docksh 96
root@96e332e547c2:/# conntrack -L
icmp     1 26 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
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@96e332e547c2:/#
```

**ICMP** 的连接状态保持时间只有 30 秒左右。

### UDP

在用户主机上利用UDP远程连接IP地址为192.168.60.5的内网主机9090端口，并发送消息如下

```
root@a6b6019f0109:/# nc -u 192.168.60.5 9090
1234
```

```
root@6dace14d60a3:/# nc -lu 9090
1234
```

**UDP** 的连接时间也为30s.

```
root@96e332e547c2:/# conntrack -L
udp      17 23 src=10.9.0.5 dst=192.168.60.5 sport=43166 dport=9090 [UNREPLIED]
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=43166 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@96e332e547c2:/#
```

### TCP

```
root@a6b6019f0109:/# nc 192.168.60.5 9090
12345
```

```
root@6dace14d60a3:/# nc -l 9090
12345
```

```
root@96e332e547c2:/# conntrack -L
tcp      6 431988 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=56124 dport=90
90 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=56124 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@96e332e547c2:/#
```

**TCP** 的连接时间为432000s = 7200min = 120h = 5day

## Task 3.B: Setting Up a Stateful Firewall

在路由器上利用iptables命令和连接跟踪机制，创建过滤规则如下：

```
iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED
-j ACCEPT
iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 --syn -m
conntrack --ctstate NEW -j ACCEPT
iptables -A FORWARD -p tcp --dport 23 -d 10.9.0.0/24 --syn -m
conntrack --ctstate NEW -j ACCEPT
iptables -P FORWARD DROP
```

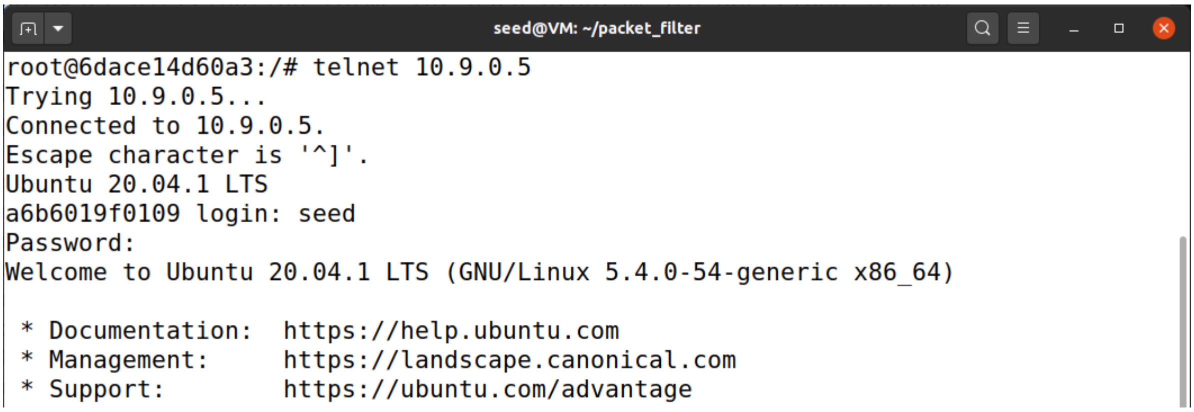
从外部主机 (10.9.0.5) telnet 192.168.60.5，连接成功。

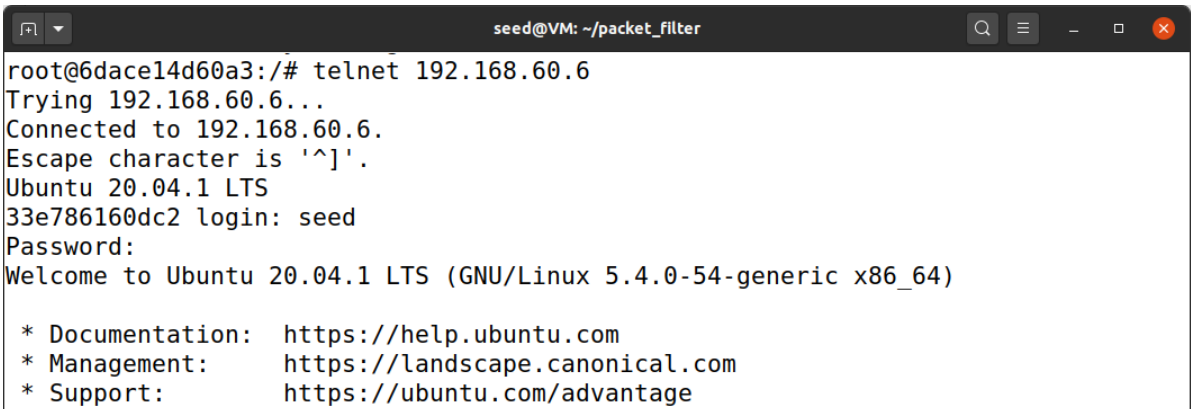
```
seed@VM: ~/packet_filter
root@a6b6019f0109:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
6dace14d60a3 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

从外部主机(10.9.0.5) telnet 192.168.0.6 不成功

```
root@a6b6019f0109:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@a6b6019f0109:/#
```

从内部主机(192.168.60.5)telnet 10.9.0.5 和192.168.60.6 ，均连接成功。




不利用连接跟踪机制的过滤规则仅对数据包的首部进行检查，其优点是处理速度快，缺点是无法定义精细的规则、不适合复杂的访问控制；而利用连接跟踪机制的过滤规则对数据包的状态也进行检查，其优点是能够定义更加严格的规则、应用范围更广、安全性更高，缺点是无法对数据包的内容进行识别。

# Task 4: Limiting Network Traffic

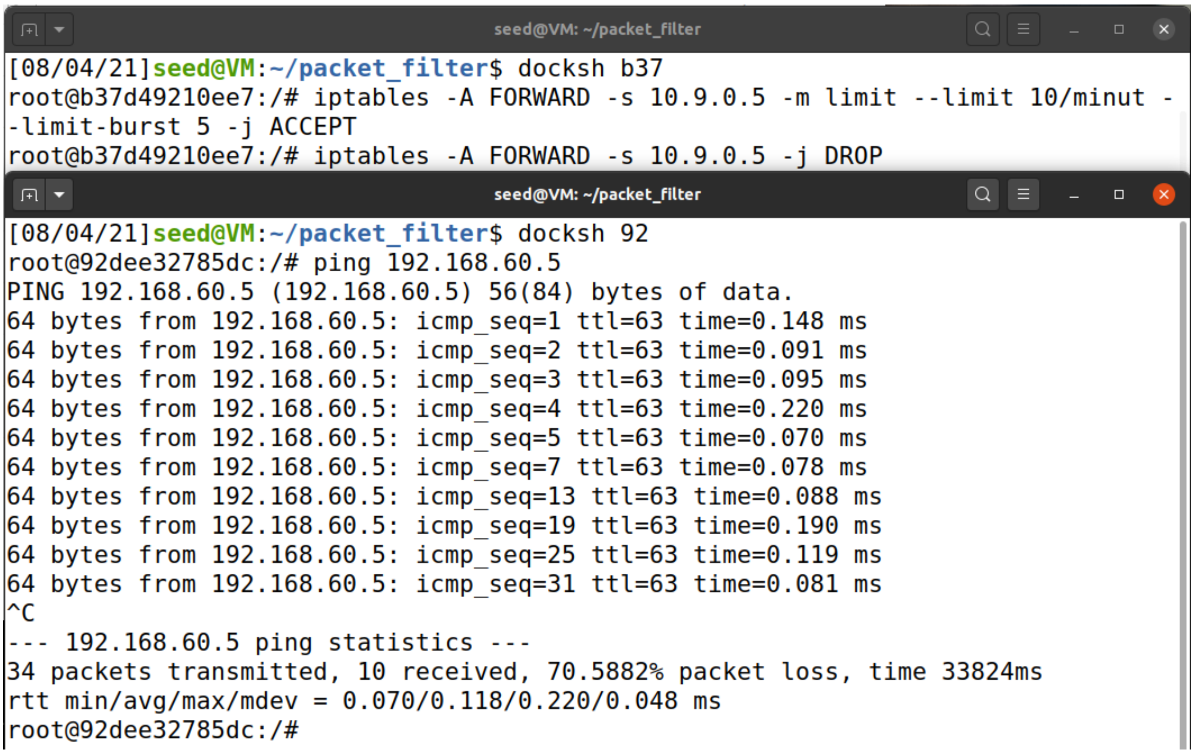
先 **iptables -F** 清空路由器配置

在路由器上利用iptables命令，创建流量限制规则如下:

```
iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minut --limit-burst 5 -j ACCEPT
iptables -A FORWARD -s 10.9.0.5 -j DROP
```

然后在10.9.0.5上ping192.168.60.5.

可以观察到前六个包的速度很快，后面每隔6秒发一个包




但部分报文因流量限制而丢失。

如果只执行第一条命令，10.9.0.5 ping 192.168.60.5 可以观察到和平时 的发包速度一样，因为 **iptables** 默认的**FORWARD** 表是接受所有包，即使超过流量限制，报文根据默认 规则也可以进行传输，可知上述第二条规则是必需的。

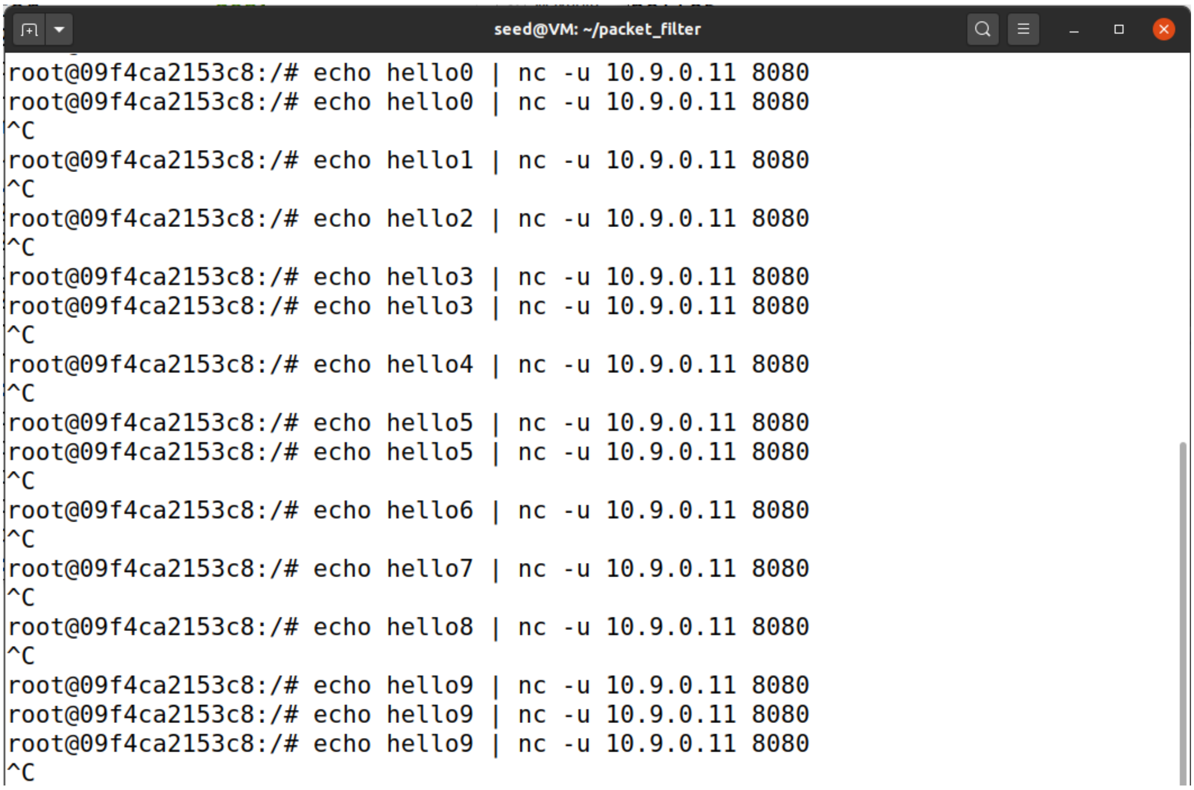
# Task 5: Load Balancing

配置如下：

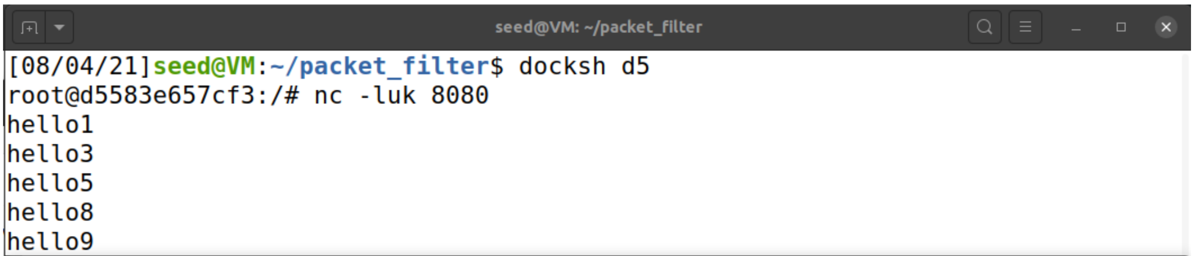
```
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode
nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode
nth --every 3 --packet 1 -j DNAT --to-destination 192.168.60.6:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode
nth --every 3 --packet 2 -j DNAT --to-destination 192.168.60.7:8080
```

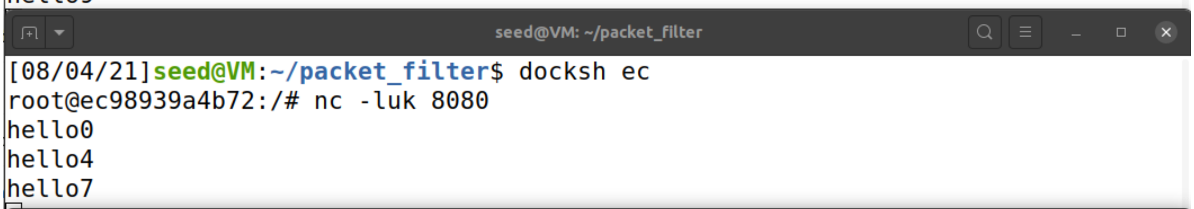
三个host上开启监听udp 8080端口：nc -luk 8080

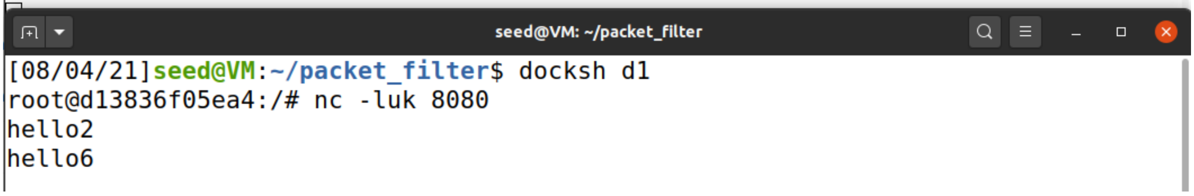
外部主机hostA 10.9.0.5 发送报文到路由器，路由器转发给三个主机：echo hello***n*** | nc -u 10.9.0.11 8080




在服务器**192.168.60.5/6/7**上监听8080端口，得到结果如下：



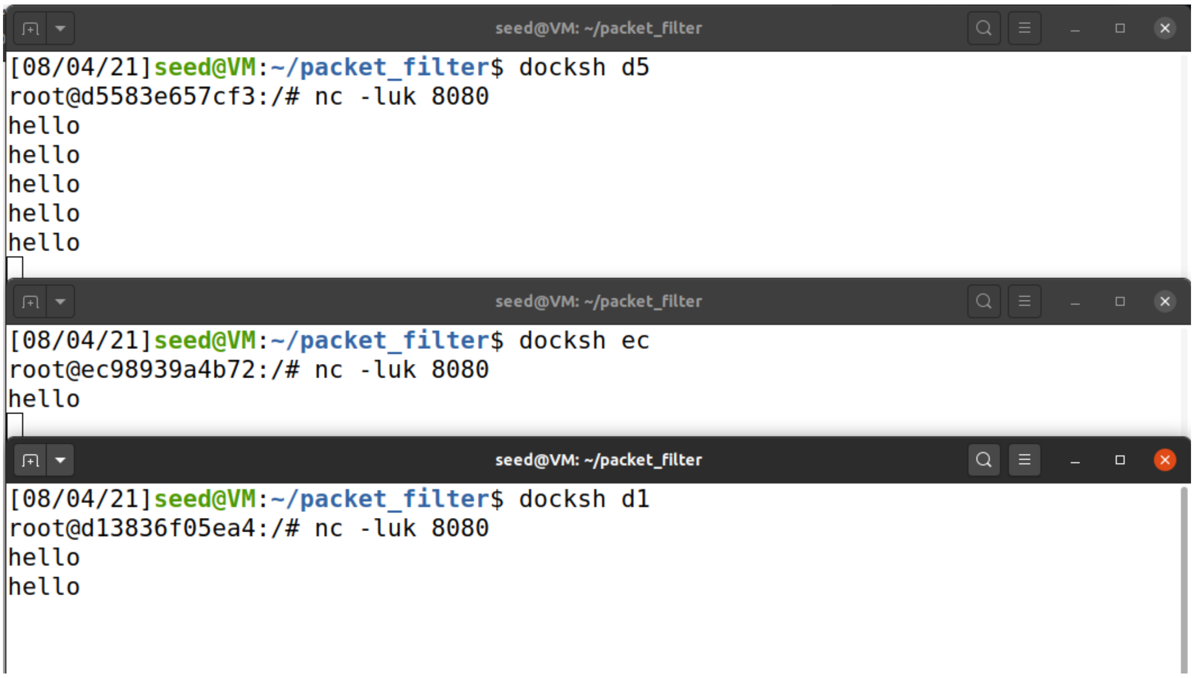




先 **iptables -F** 清空路由器配置

在路由器上利用iptables命令，采用random模式创建负载均衡规则如下

```
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.5:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.6:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.34 -j DNAT --to-destination 192.168.60.7:8080
```




虽然是等概率发送数据，但每个主机收到的数量各不相同，甚至有的差异较大，当样本数量足够多时， 应该是趋于平均的。