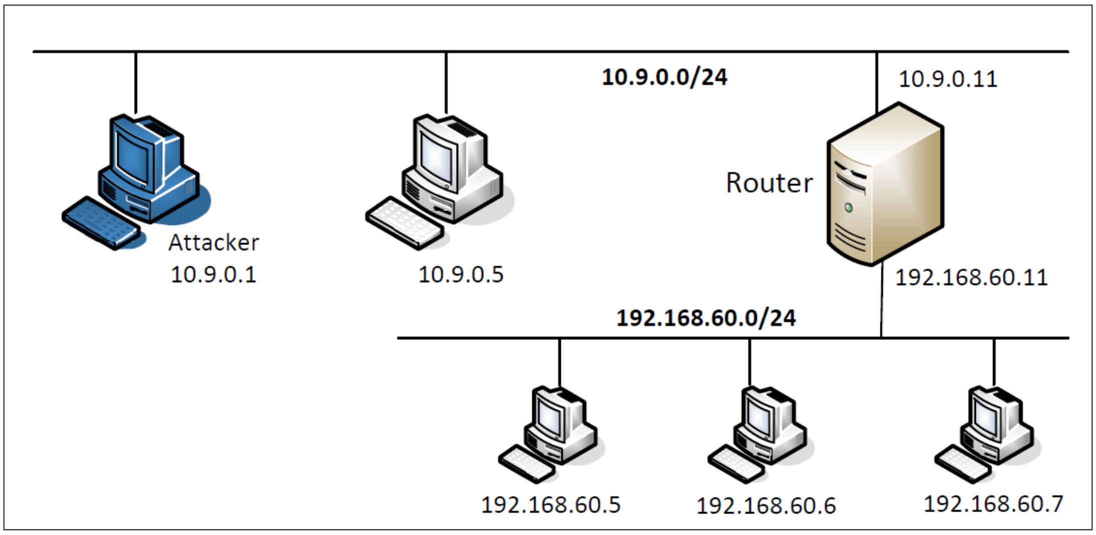
lab6-report

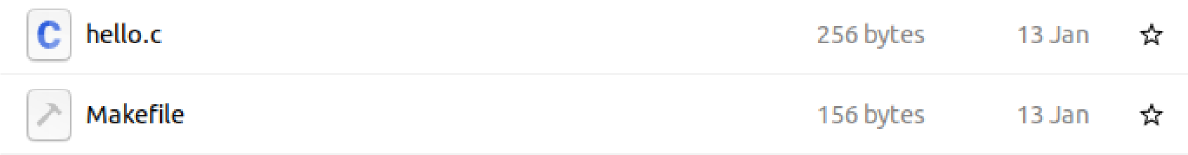
57118115陈烨

>

网络拓朴



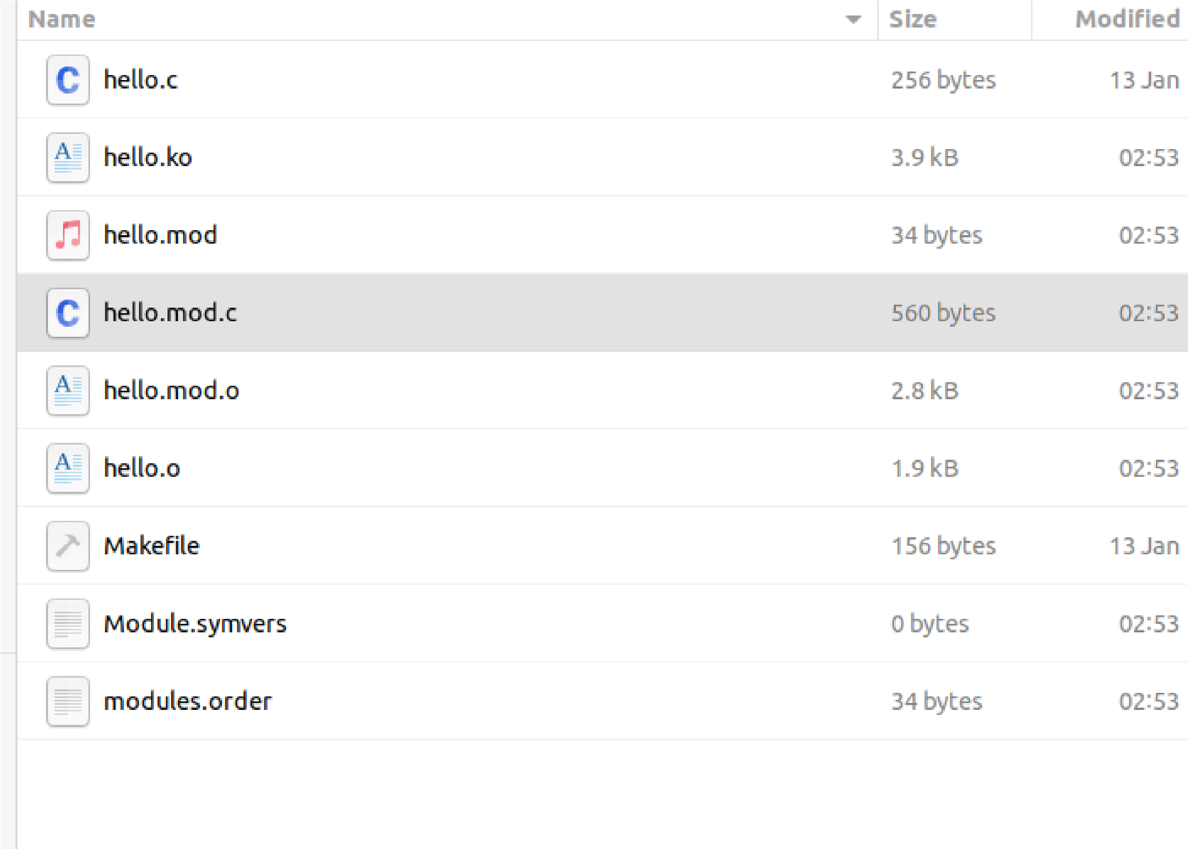
Task1A



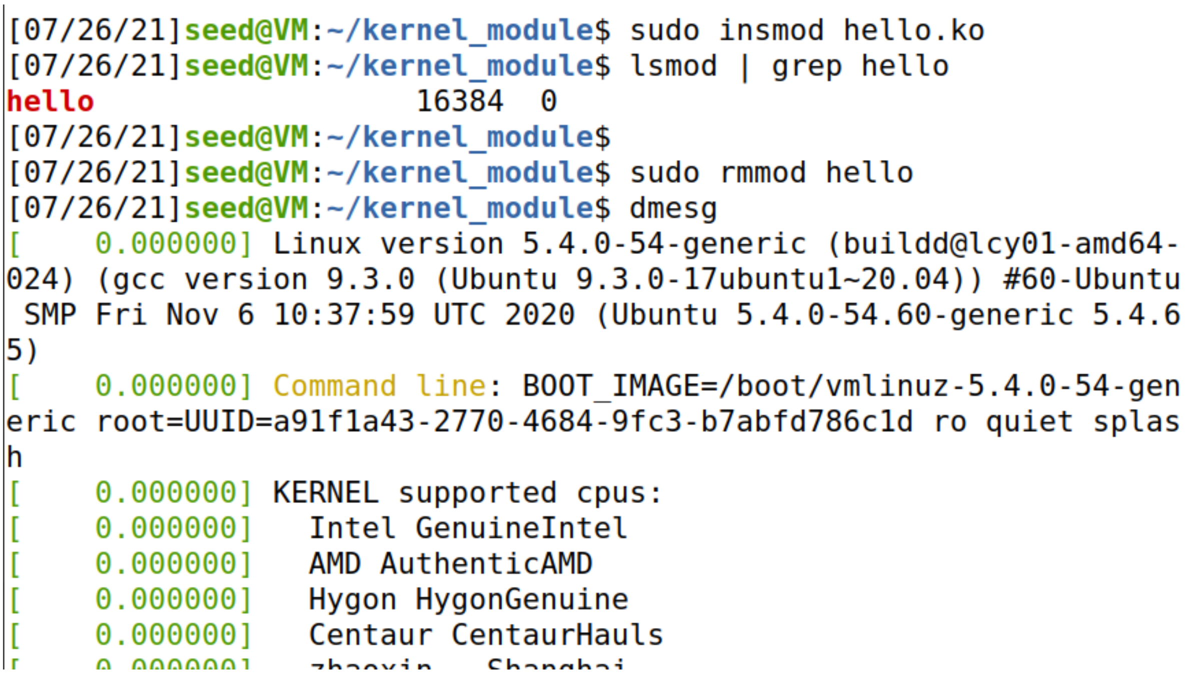
hello.c	256 bytes	13 Jan	☆
Makefile	156 bytes	13 Jan	☆

文件夹copy到/home/seed make后

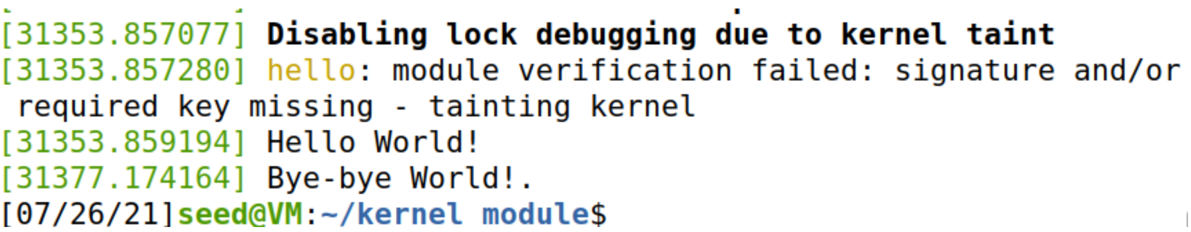
```
[07/26/21]seed@VM:~/kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/kernel_mo
dule modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-gener
ic'
  CC [M]  /home/seed/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/kernel_mo
dule/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-generi
c'
```



Name	Size	Modified
hello.c	256 bytes	13 Jan
hello.ko	3.9 kB	02:53
hello.mod	34 bytes	02:53
hello.mod.c	560 bytes	02:53
hello.mod.o	2.8 kB	02:53
hello.o	1.9 kB	02:53
Makefile	156 bytes	13 Jan
Module.symvers	0 bytes	02:53
modules.order	34 bytes	02:53



```
[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$
[07/26/21]seed@VM:~/kernel_module$ sudo rmmod hello
[07/26/21]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 5.4.6
5)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.4.0-54-gen
eric root=UUID=a91f1a43-2770-4684-9fc3-b7abfd786c1d ro quiet splas
h
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
```



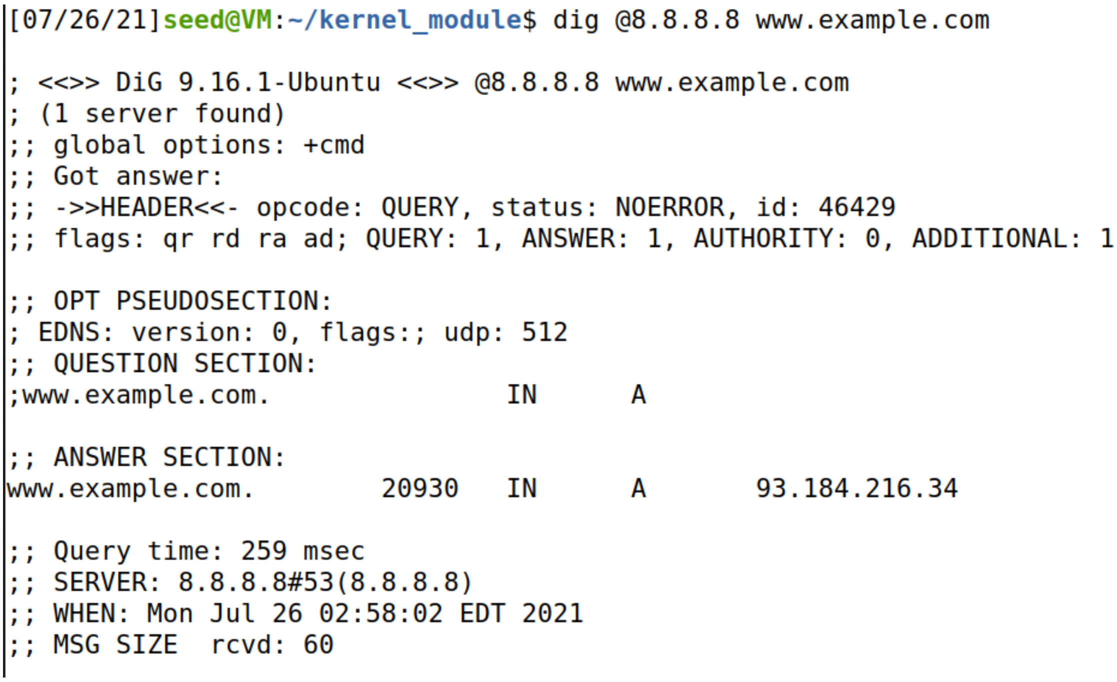
```
[31353.857077] Disabling lock debugging due to kernel taint
[31353.857280] hello: module verification failed: signature and/or
 required key missing - tainting kernel
[31353.859194] Hello World!
[31377.174164] Bye-bye World!.
[07/26/21]seed@VM:~/kernel_module$
```

Task1B

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。可以通过以下命令生成到谷歌的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。

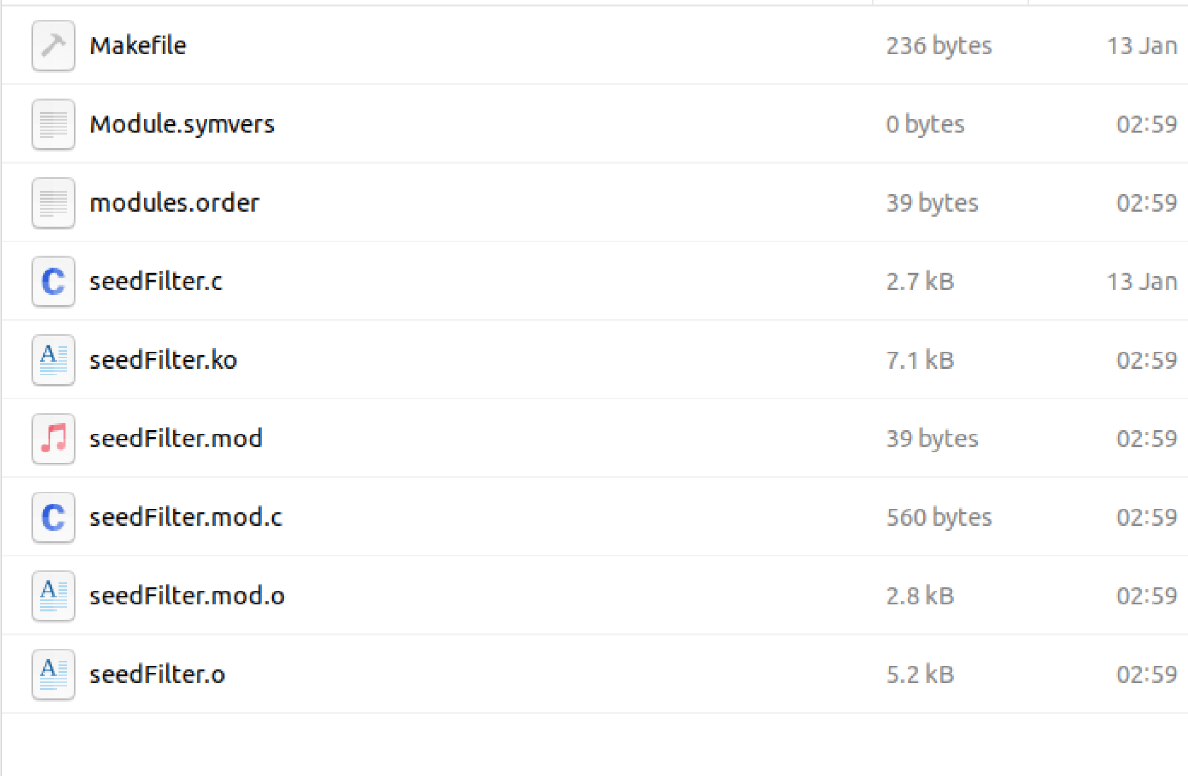
以LKM的形式编写包过滤程序，然后将其插入到内核中的包处理路径

在主机上利用dig查询www.example.com的DNS如下，可知能够获得相关信息。



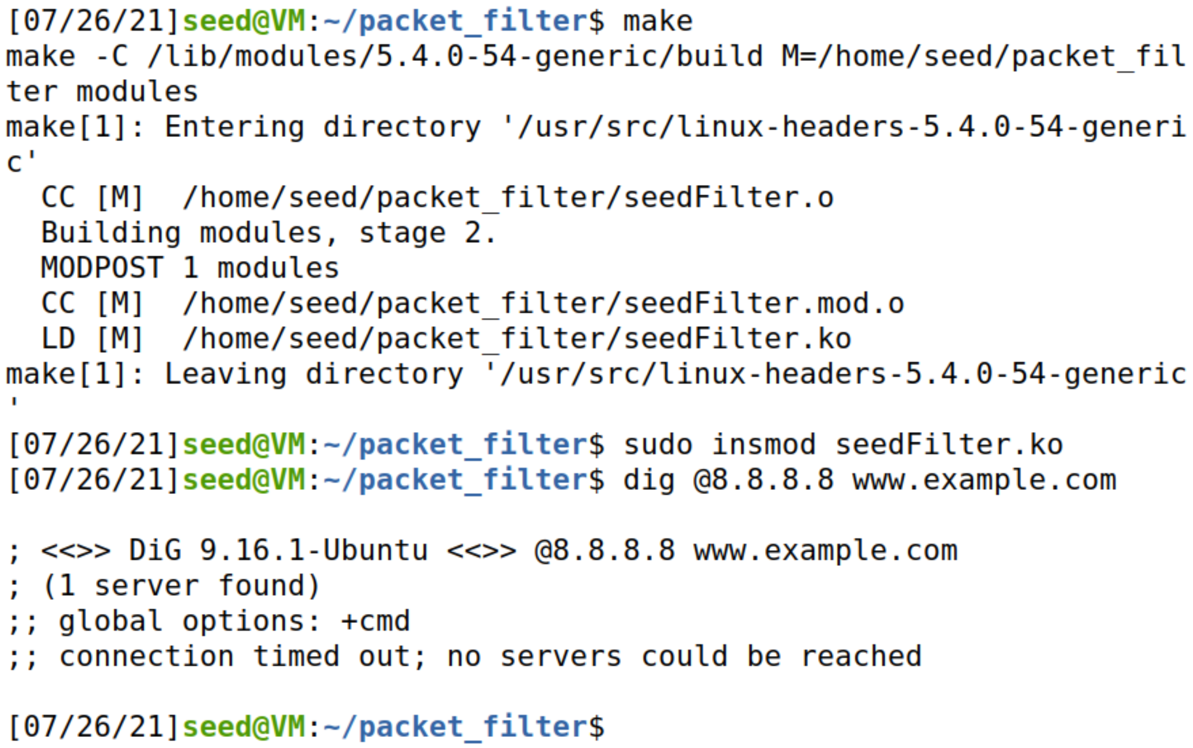
```
[07/26/21]seed@VM:~/kernel_module$ dig @8.8.8.8 www.example.com

; <<>> 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: 46429
;; 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.        20930   IN      A       93.184.216.34

;; Query time: 259 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Jul 26 02:58:02 EDT 2021
;; MSG SIZE  rcvd: 60
```

编译加载seedFilter LKM



Makefile	236 bytes	13 Jan
Module.symvers	0 bytes	02:59
modules.order	39 bytes	02:59
seedFilter.c	2.7 kB	13 Jan
seedFilter.ko	7.1 kB	02:59
seedFilter.mod	39 bytes	02:59
seedFilter.mod.c	560 bytes	02:59
seedFilter.mod.o	2.8 kB	02:59
seedFilter.o	5.2 kB	02:59

再次请求：



```
[07/26/21]seed@VM:~/packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/packet_fil
ter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generi
c'
  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
'
[07/26/21]seed@VM:~/packet_filter$ sudo insmod seedFilter.ko
[07/26/21]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

[07/26/21]seed@VM:~/packet_filter$
```

可知无法获得相关信息。

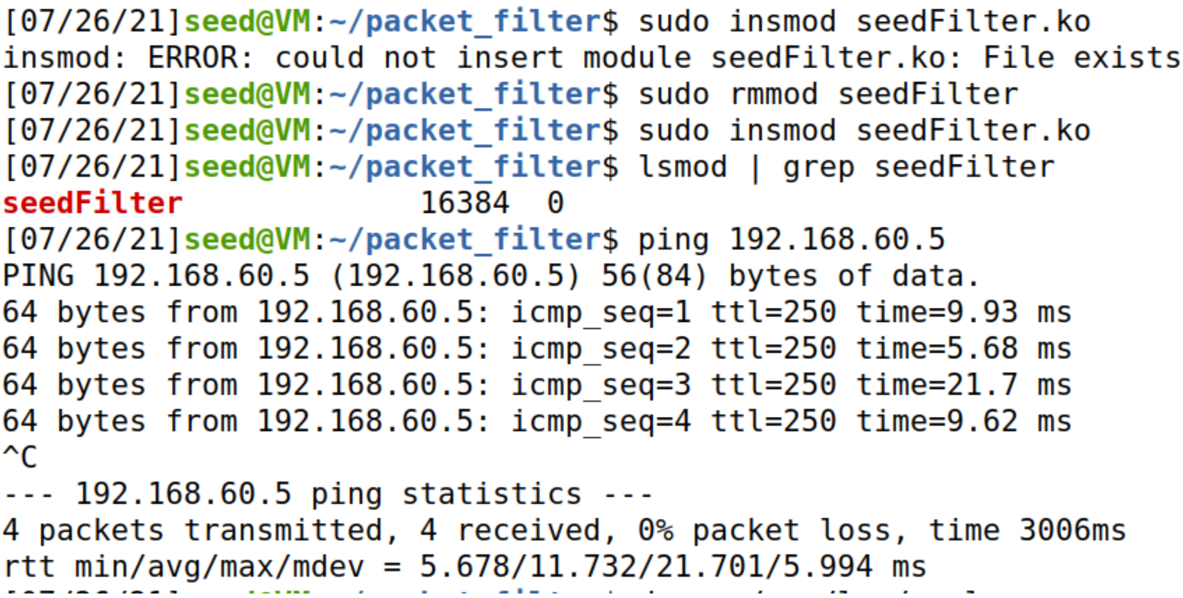
2.将printInfo函数与所有的netfilter hook挂钩。使用实验结果来帮助解释每个hook函数在什么情况下会被调用。

修改seedFilter.c文件

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


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



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_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");
```

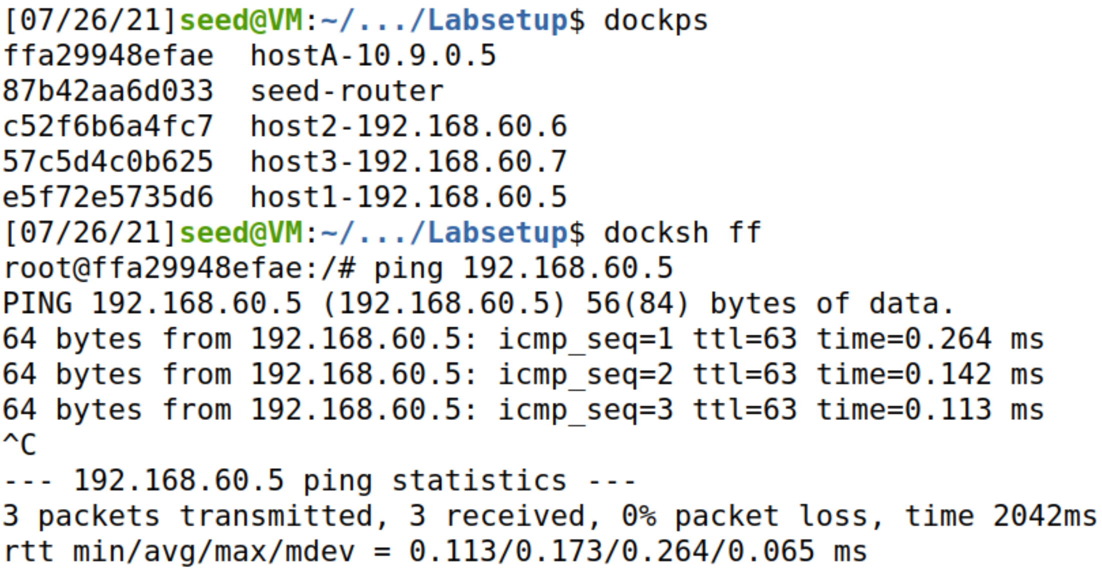
```
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
    nf_unregister_net_hook(&init_net, &hook5);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");

```

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

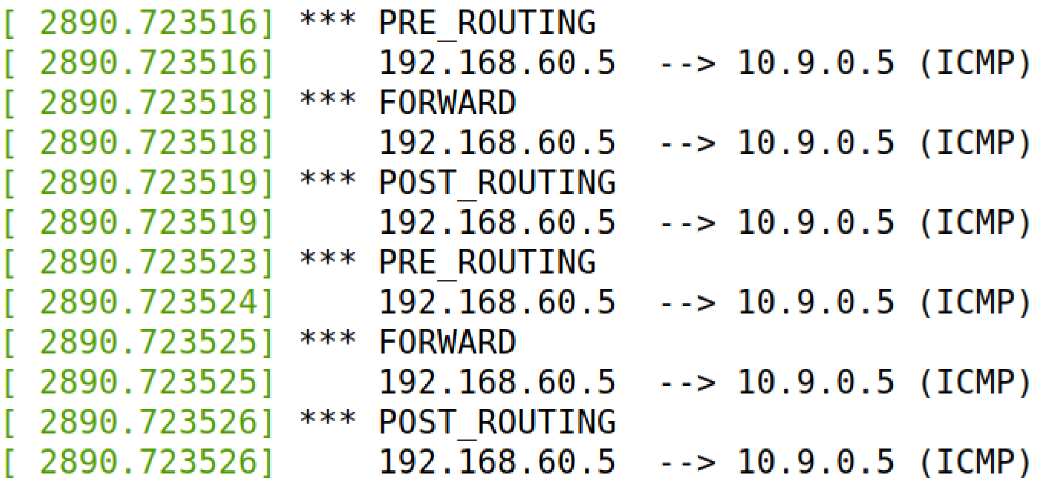


```
[07/26/21]seed@VM:~/packet_filter$ sudo insmod seedFilter.ko
insmod: ERROR: could not insert module seedFilter.ko: File exists
[07/26/21]seed@VM:~/packet_filter$ sudo rmmod seedFilter
[07/26/21]seed@VM:~/packet_filter$ sudo insmod seedFilter.ko
[07/26/21]seed@VM:~/packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
[07/26/21]seed@VM:~/packet_filter$ 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=250 time=9.93 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=250 time=5.68 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=250 time=21.7 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=250 time=9.62 ms
^C
--- 192.168.60.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 5.678/11.732/21.701/5.994 ms
```

在用户主机上ping内网主机，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

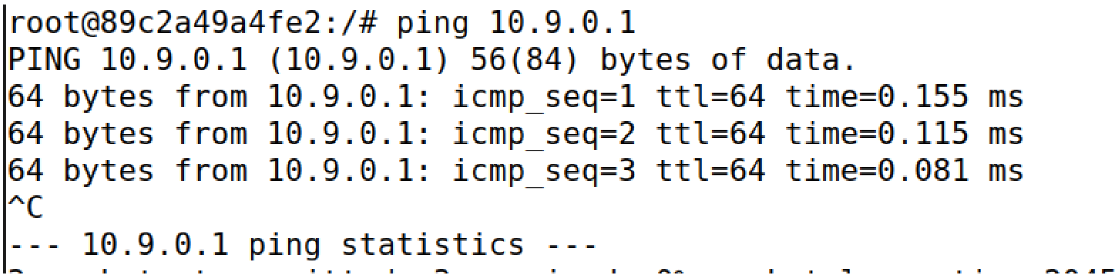


```
[07/26/21]seed@VM:~/.../Labsetup$ dockps
ffa29948efae  hostA-10.9.0.5
87b42aa6d033  seed-router
c52f6b6a4fc7  host2-192.168.60.6
57c5d4c0b625  host3-192.168.60.7
e5f72e5735d6  host1-192.168.60.5
[07/26/21]seed@VM:~/.../Labsetup$ docksh ff
root@ffa29948efae:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.264 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.142 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.113 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.113/0.173/0.264/0.065 ms
```



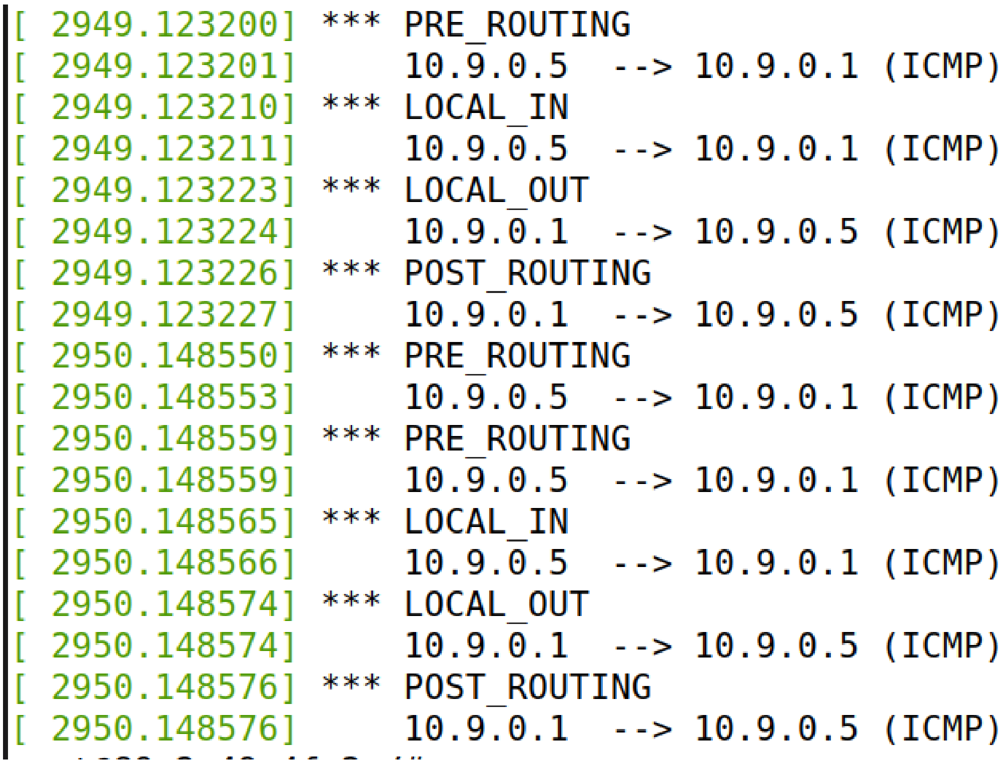
```
[ 2890.723516] *** PRE_ROUTING
[ 2890.723516]     192.168.60.5  --> 10.9.0.5 (ICMP)
[ 2890.723518] *** FORWARD
[ 2890.723518]     192.168.60.5  --> 10.9.0.5 (ICMP)
[ 2890.723519] *** POST_ROUTING
[ 2890.723519]     192.168.60.5  --> 10.9.0.5 (ICMP)
[ 2890.723523] *** PRE_ROUTING
[ 2890.723524]     192.168.60.5  --> 10.9.0.5 (ICMP)
[ 2890.723525] *** FORWARD
[ 2890.723525]     192.168.60.5  --> 10.9.0.5 (ICMP)
[ 2890.723526] *** POST_ROUTING
[ 2890.723526]     192.168.60.5  --> 10.9.0.5 (ICMP)
```

在用户主机上ping攻击者主机，得到结果如下，可知能够连接



```
root@89c2a49a4fe2:/# 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=0.155 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.115 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.081 ms
^C
--- 10.9.0.1 ping statistics ---
```

查看dmesg



```
[ 2949.123200] *** PRE_ROUTING
[ 2949.123201]     10.9.0.5  --> 10.9.0.1 (ICMP)
[ 2949.123210] *** LOCAL_IN
[ 2949.123211]     10.9.0.5  --> 10.9.0.1 (ICMP)
[ 2949.123223] *** LOCAL_OUT
[ 2949.123224]     10.9.0.1  --> 10.9.0.5 (ICMP)
[ 2949.123226] *** POST_ROUTING
[ 2949.123227]     10.9.0.1  --> 10.9.0.5 (ICMP)
[ 2950.148550] *** PRE_ROUTING
[ 2950.148553]     10.9.0.5  --> 10.9.0.1 (ICMP)
[ 2950.148559] *** PRE_ROUTING
[ 2950.148559]     10.9.0.5  --> 10.9.0.1 (ICMP)
[ 2950.148565] *** LOCAL_IN
[ 2950.148566]     10.9.0.5  --> 10.9.0.1 (ICMP)
[ 2950.148574] *** LOCAL_OUT
[ 2950.148574]     10.9.0.1  --> 10.9.0.5 (ICMP)
[ 2950.148576] *** POST_ROUTING
[ 2950.148576]     10.9.0.1  --> 10.9.0.5 (ICMP)
```

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。

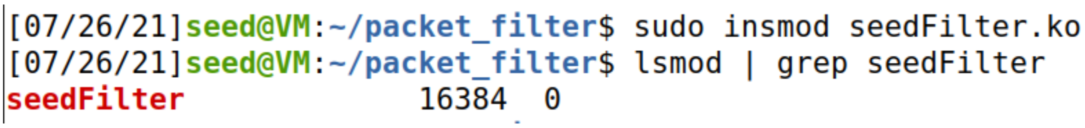
挂载NF_INET_LOCAL_OUT 时，本机产生的数据包将会第一个到达此HOOK， 数据经过HOOK 函数NF_INET_LOCAL_OUT 处理后，进行路由选择处理，然后经过NF_INET_POST_ROUTING 处理后发送出去。

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，

```
#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/icmp.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 \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");
```

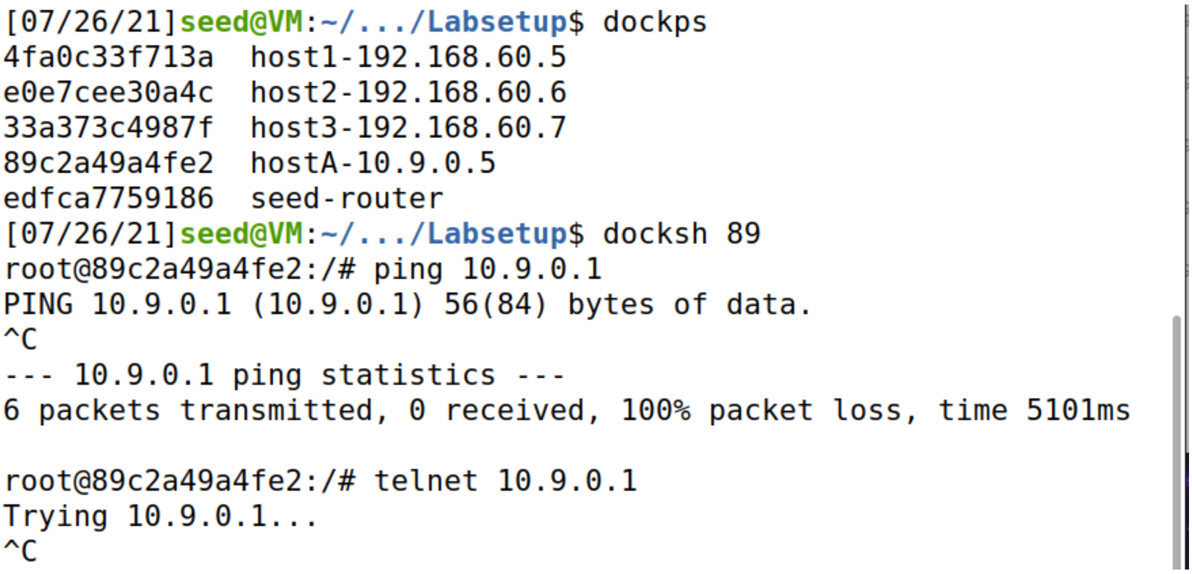
加载内核



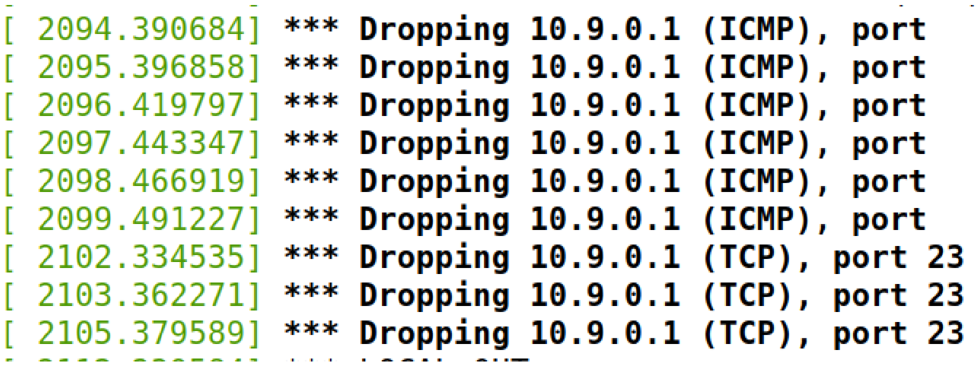
```
[07/26/21]seed@VM:~/packet_filter$ sudo insmod seedFilter.ko
[07/26/21]seed@VM:~/packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

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

发现都不通过，dmesg查看：



```
[07/26/21]seed@VM:~/.../Labsetup$ dockps
4fa0c33f713a  host1-192.168.60.5
e0e7cee30a4c  host2-192.168.60.6
33a373c4987f  host3-192.168.60.7
89c2a49a4fe2  hostA-10.9.0.5
edfca7759186  seed-router
[07/26/21]seed@VM:~/.../Labsetup$ docksh 89
root@89c2a49a4fe2:/# 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 ---
6 packets transmitted, 0 received, 100% packet loss, time 5101ms

root@89c2a49a4fe2:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
```



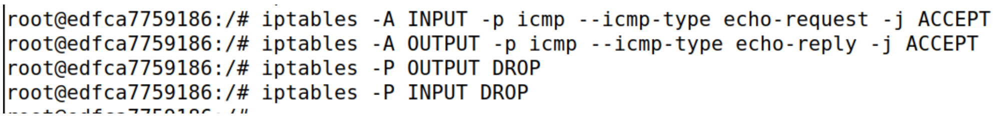
```
[ 2094.390684] *** Dropping 10.9.0.1 (ICMP), port
[ 2095.396858] *** Dropping 10.9.0.1 (ICMP), port
[ 2096.419797] *** Dropping 10.9.0.1 (ICMP), port
[ 2097.443347] *** Dropping 10.9.0.1 (ICMP), port
[ 2098.466919] *** Dropping 10.9.0.1 (ICMP), port
[ 2099.491227] *** Dropping 10.9.0.1 (ICMP), port
[ 2102.334535] *** Dropping 10.9.0.1 (TCP), port 23
[ 2103.362271] *** Dropping 10.9.0.1 (TCP), port 23
[ 2105.379589] *** Dropping 10.9.0.1 (TCP), port 23
```

Task2

A

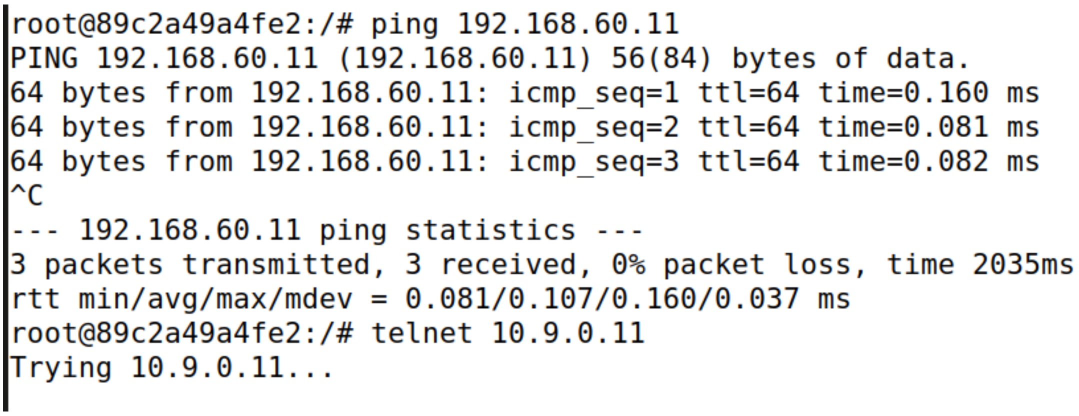
用户主机的IP地址为10.9.0.5，路由器的IP地址为10.9.0.11，内网网段的IP地址192.168.60.0/24。

在路由器上设置以下过滤规则：



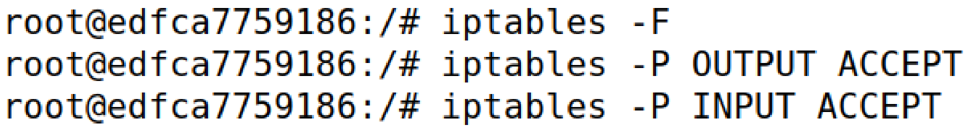
```
root@edfca7759186:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@edfca7759186:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@edfca7759186:/# iptables -P OUTPUT DROP
root@edfca7759186:/# iptables -P INPUT DROP
```

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：

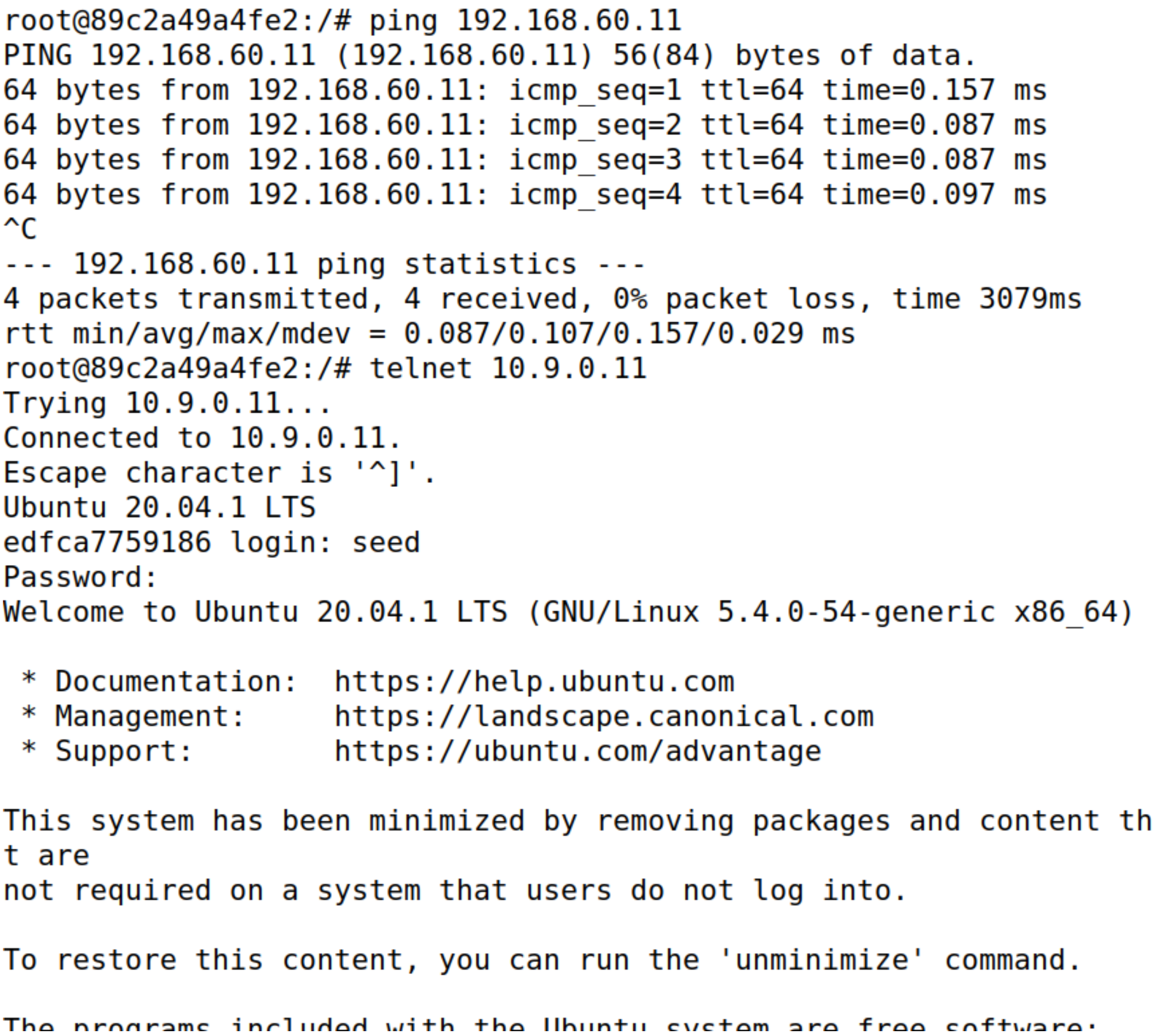


```
root@89c2a49a4fe2:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.160 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.081 ms
64 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.082 ms
^C
--- 192.168.60.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.081/0.107/0.160/0.037 ms
root@89c2a49a4fe2:/# telnet 10.9.0.11
Trying 10.9.0.11...
```

将上述规则取消掉，发现可以ping和telnet



```
root@edfca7759186:/# iptables -F
root@edfca7759186:/# iptables -P OUTPUT ACCEPT
root@edfca7759186:/# iptables -P INPUT ACCEPT
```



```
root@89c2a49a4fe2:/# 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=0.157 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.087 ms
64 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.087 ms
64 bytes from 192.168.60.11: icmp_seq=4 ttl=64 time=0.097 ms
^C
--- 192.168.60.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.087/0.107/0.157/0.029 ms
root@89c2a49a4fe2:/# telnet 10.9.0.11
Trying 10.9.0.11...
Connected to 10.9.0.11.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
edfca7759186 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 th
t are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
```

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。

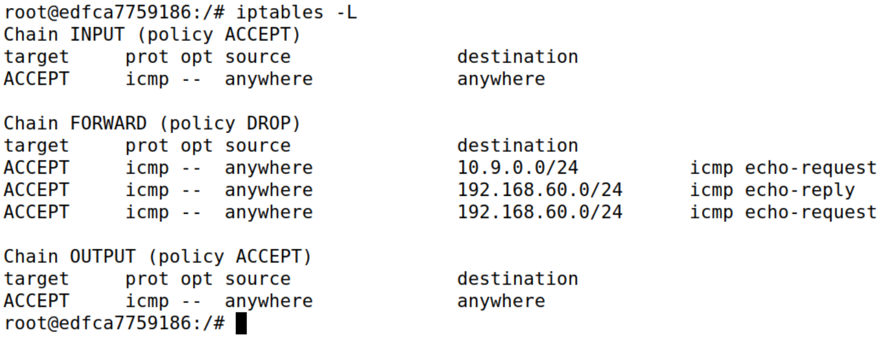
设置了iptables -P OUTPUT DROP后，二者无法ping通，表示丢弃所有外出的包

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。

B

配置：

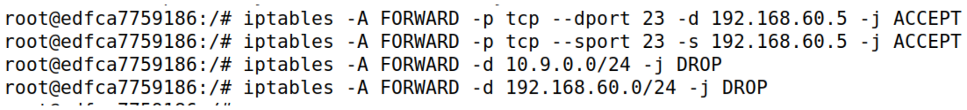
```
root@edfca7759186:/# iptables -A FORWARD -p icmp --icmp-type echo-request -d 10.9.0.5/24 -j ACCEPT
root@edfca7759186:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168.60.0/24 -j ACCEPT
root@edfca7759186:/# iptables -A FORWARD -p icmp --icmp-type echo-request -d 192.168.60.0/24 -j ACCEPT
root@edfca7759186:/# iptables -A INPUT -p icmp -j ACCEPT
root@edfca7759186:/# iptables -A OUTPUT -p icmp -j ACCEPT
root@edfca7759186:/# iptables -P FORWARD DROP
root@edfca7759186:/#
```



```
root@edfca7759186:/# 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
ACCEPT     icmp --  anywhere             192.168.60.0/24      icmp echo-request

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere
root@edfca7759186:/#
```

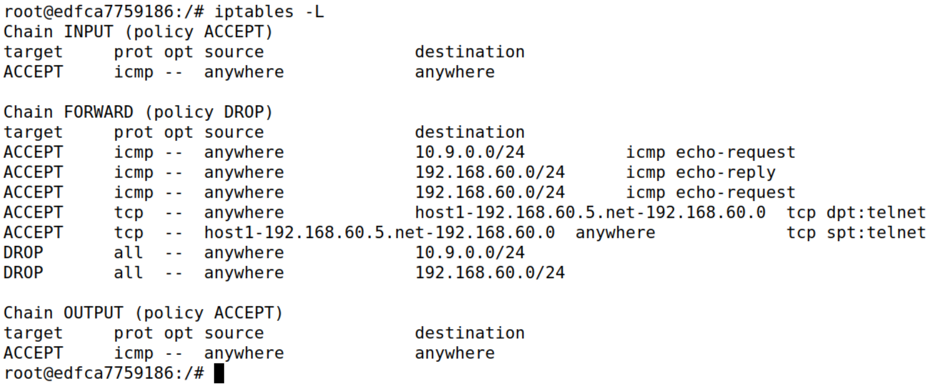
从外部主机ping 路由器，可以ping 通； ping 内部主机不通； telnet 内部主机不通。
内部主机ping 外部主机，可以ping 通； telnet 外部主机不通。

C



```
root@edfca7759186:/# iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 -j ACCEPT
root@edfca7759186:/# iptables -A FORWARD -p tcp --sport 23 -s 192.168.60.5 -j ACCEPT
root@edfca7759186:/# iptables -A FORWARD -d 10.9.0.0/24 -j DROP
root@edfca7759186:/# iptables -A FORWARD -d 192.168.60.0/24 -j DROP
```

查看配置：



```
root@edfca7759186:/# 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
ACCEPT     icmp --  anywhere             192.168.60.0/24      icmp echo-request
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
ACCEPT     icmp --  anywhere             anywhere
root@edfca7759186:/#
```

结果：

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

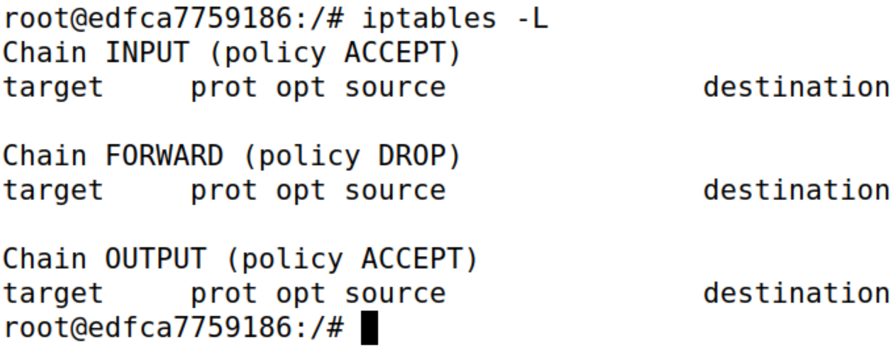
从外部主机(10.9.0.5)telnet 192.168.60.6， 无法连接。

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。

Task3

清空iptables配置

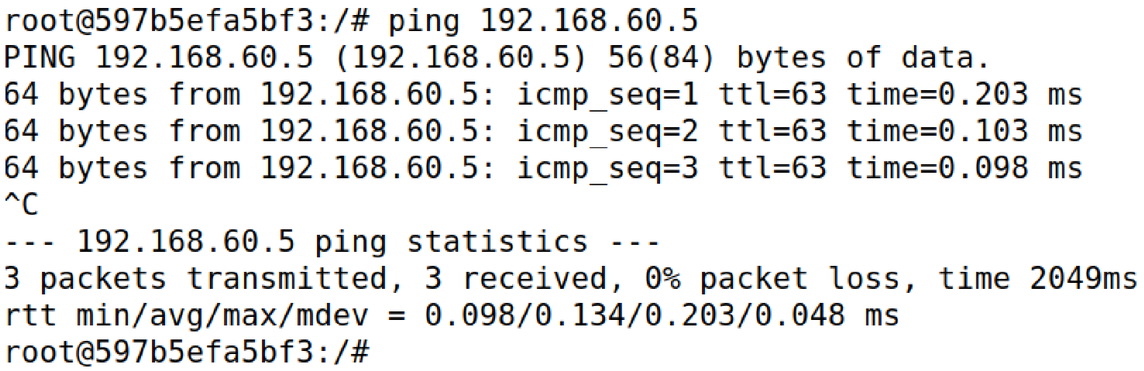


```
root@edfca7759186:/# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@edfca7759186:/# 
```

A

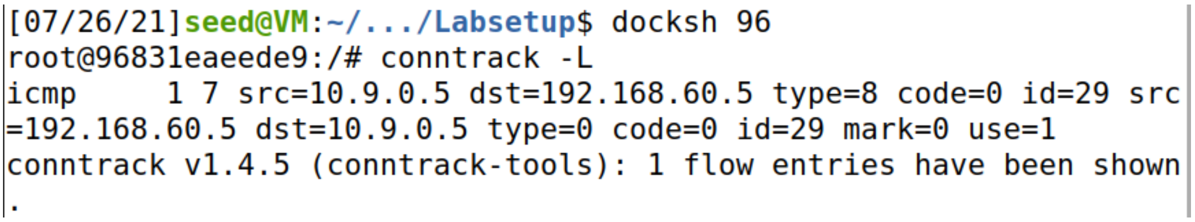
ICMP

在用户主机上ping内网主机192.168.60.5



```
root@597b5efa5bf3:/# 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=0.203 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.103 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.098 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.098/0.134/0.203/0.048 ms
root@597b5efa5bf3:/#
```

在路由器上利用conntrack -L命令实现连接跟踪，得到结果如下

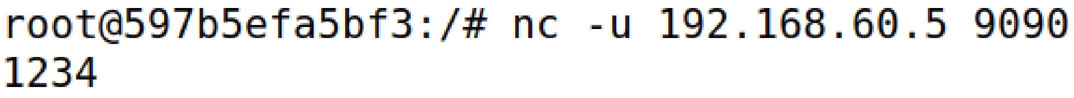


```
[07/26/21]seed@VM:~/.../Labsetup$ docksh 96
root@96831eaeede9:/# conntrack -L
icmp     1 7 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=29 src
=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=29 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
```

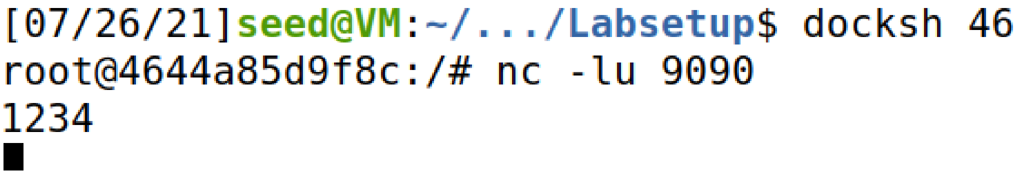
一个 ICMP 连接持续时间为 30s

UDP

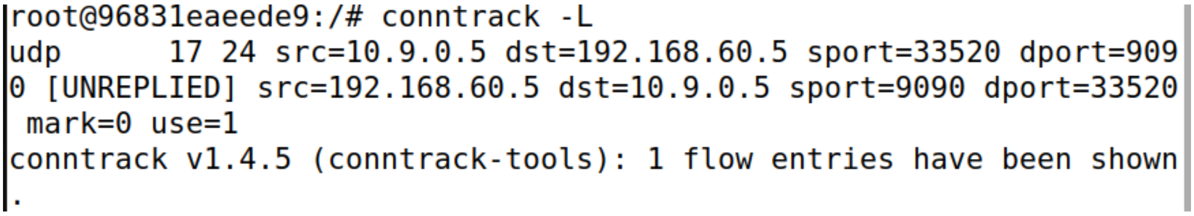
在用户主机上利用UDP远程连接IP地址为192.168.60.5的内网主机9090端口，并发送消息如下



```
root@597b5efa5bf3:/# nc -u 192.168.60.5 9090
1234
```



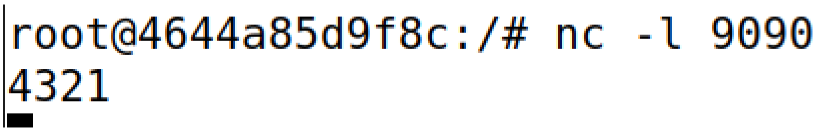
```
[07/26/21]seed@VM:~/.../Labsetup$ docksh 46
root@4644a85d9f8c:/# nc -lu 9090
1234
```



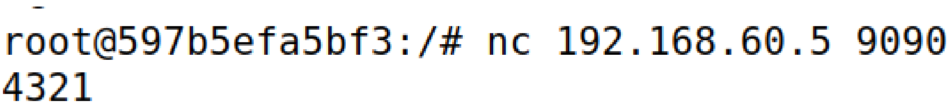
```
root@96831eaeede9:/# conntrack -L
udp      17 24 src=10.9.0.5 dst=192.168.60.5 sport=33520 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=33520 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

udp也是30s左右

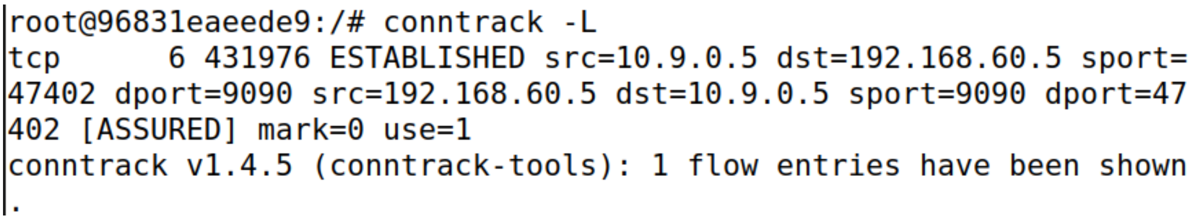
TCP



```
root@4644a85d9f8c:/# nc -l 9090
4321
```



```
root@597b5efa5bf3:/# nc 192.168.60.5 9090
4321
```

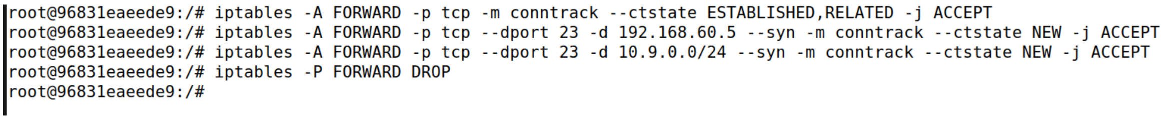


```
root@96831eaeede9:/# conntrack -L
tcp      6 431976 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=47402 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47402 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

tcp是432000s左右

B

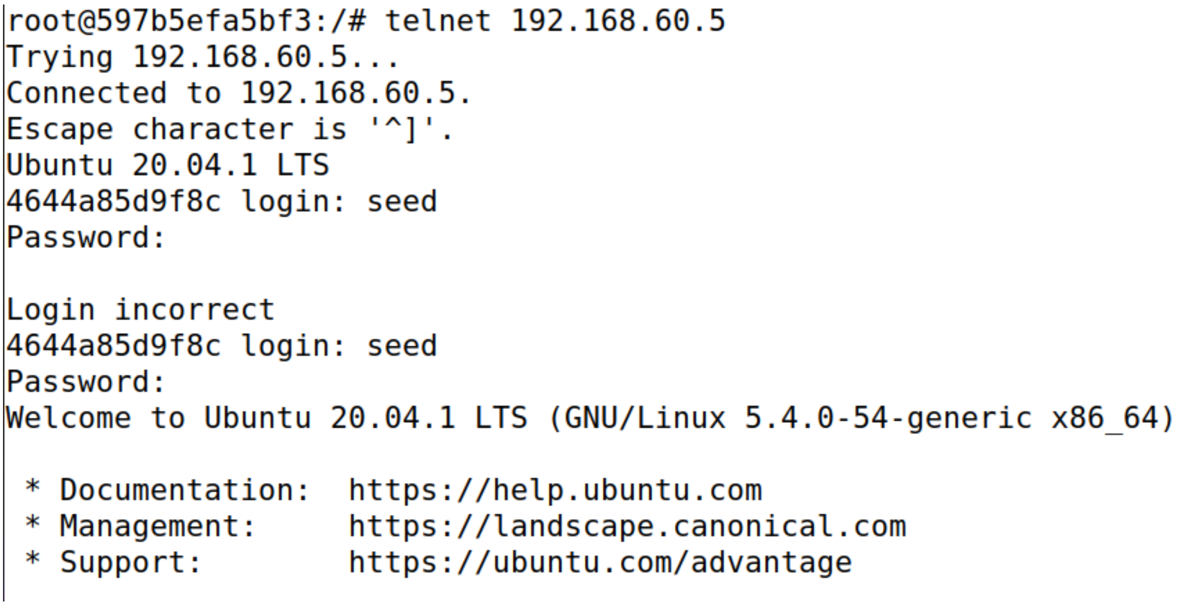
在路由器上利用iptables命令和连接跟踪机制，创建过滤规则如下：



```
root@96831eaeede9:/# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
root@96831eaeede9:/# iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 --syn -m conntrack --ctstate NEW -j ACCEPT
root@96831eaeede9:/# iptables -A FORWARD -p tcp --dport 23 -d 10.9.0.0/24 --syn -m conntrack --ctstate NEW -j ACCEPT
root@96831eaeede9:/# iptables -P FORWARD DROP
root@96831eaeede9:/#
```

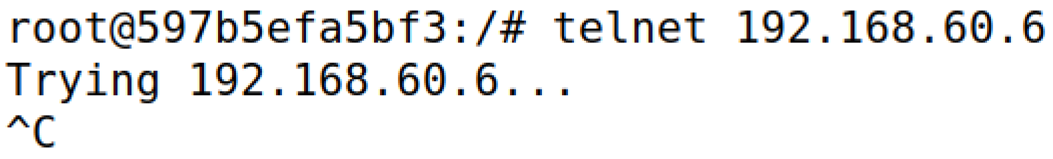
```
root@96831eaeede9:/# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
root@96831eaeede9:/# iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 --syn -m conntrack --ctstate NEW -j ACCEPT
root@96831eaeede9:/# iptables -A FORWARD -p tcp --dport 23 -d 10.9.0.0/24 --syn -m conntrack --ctstate NEW -j ACCEPT
root@96831eaeede9:/# iptables -P FORWARD DROP

```

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



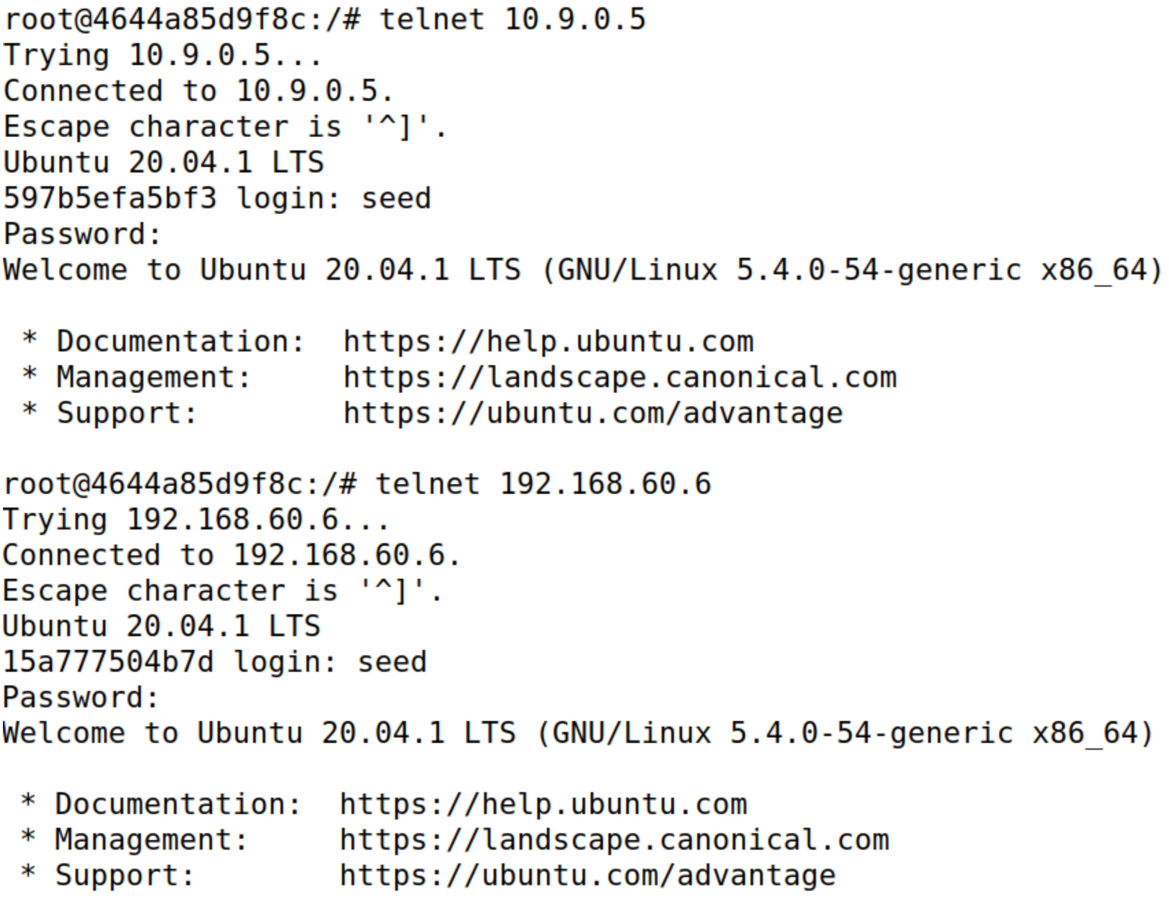
```
root@597b5efa5bf3:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4644a85d9f8c login: seed
Password:

Login incorrect
4644a85d9f8c 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@597b5efa5bf3:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
```

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



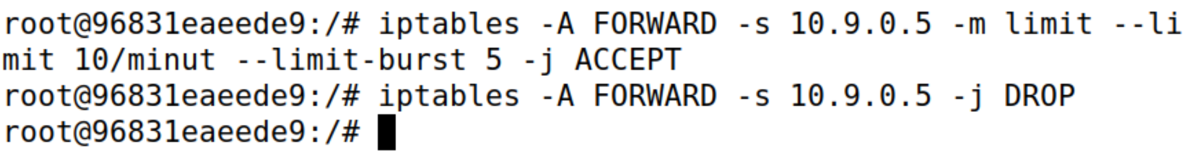
```
root@4644a85d9f8c:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
597b5efa5bf3 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

root@4644a85d9f8c:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
15a777504b7d 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
```

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。

Task4

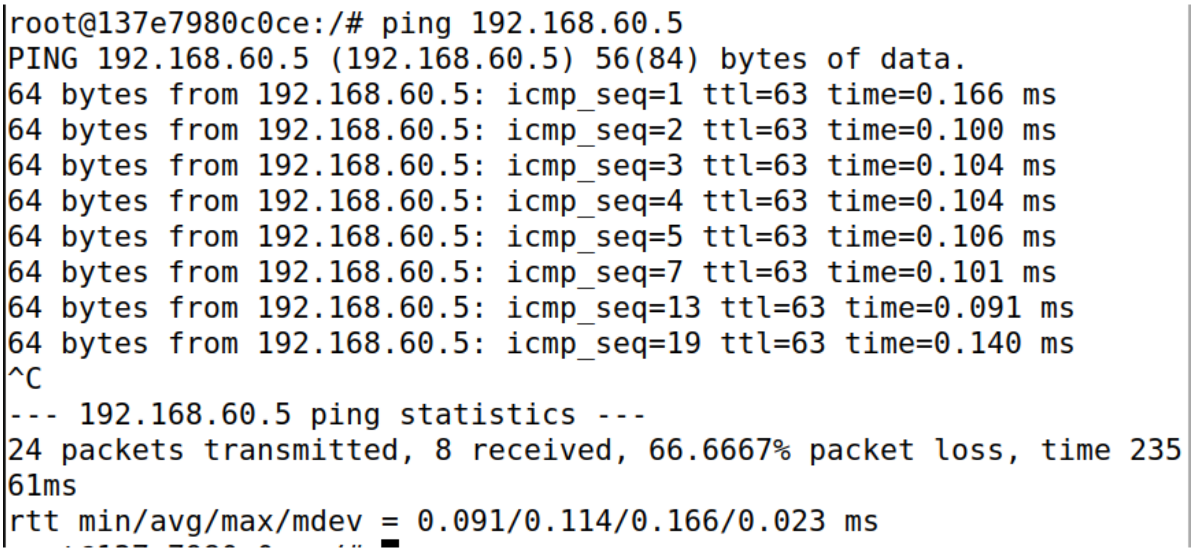
先`iptables -F`清空路由器配置

在路由器上利用iptables命令，创建流量限制规则如下:



```
root@96831eaeede9:/# iptables -A FORWARD -s 10.9.0.5 -m limit --li
mit 10/minut --limit-burst 5 -j ACCEPT
root@96831eaeede9:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@96831eaeede9:/# 
```

可以观察到前六个包的速度很快，后面每隔6秒发一个包



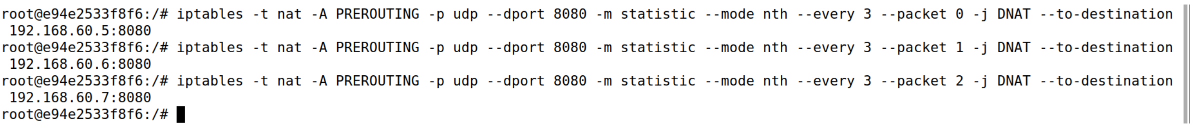
```
root@137e7980c0ce:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.166 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.100 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.104 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.104 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.101 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.091 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.140 ms
^C
--- 192.168.60.5 ping statistics ---
24 packets transmitted, 8 received, 66.6667% packet loss, time 235
61ms
rtt min/avg/max/mdev = 0.091/0.114/0.166/0.023 ms
```

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。

Task5

配置如下:

```
root@e94e2533f8f6:/# 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@e94e2533f8f6:/# 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
root@e94e2533f8f6:/# 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

```



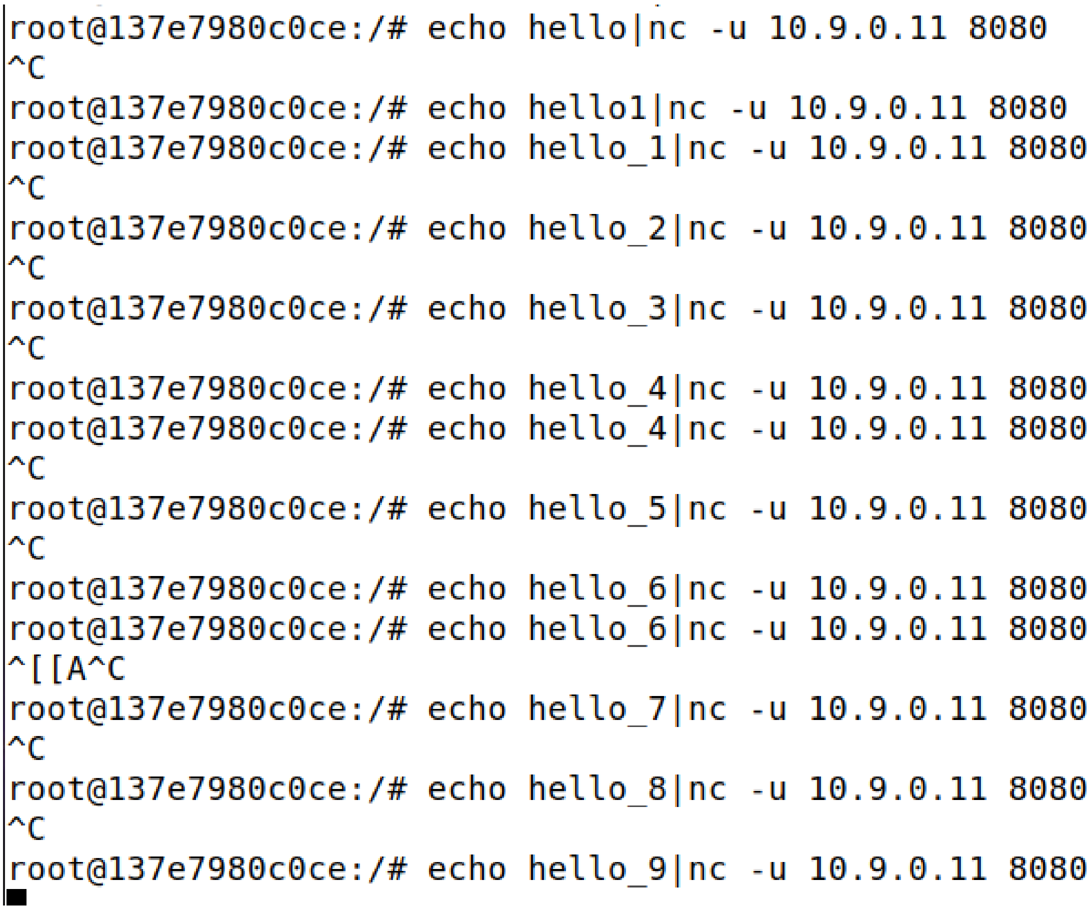
```
root@e94e2533f8f6:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination
 192.168.60.5:8080
root@e94e2533f8f6:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 1 -j DNAT --to-destination
 192.168.60.6:8080
root@e94e2533f8f6:/# 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
root@e94e2533f8f6:/# 
```

三个host上开启监听udp 8080端口：nc -luk 8080

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

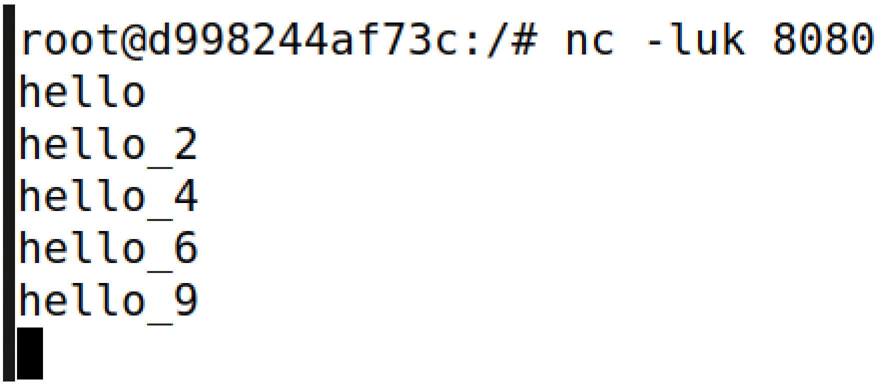
三个主机负载均衡:

主机上:



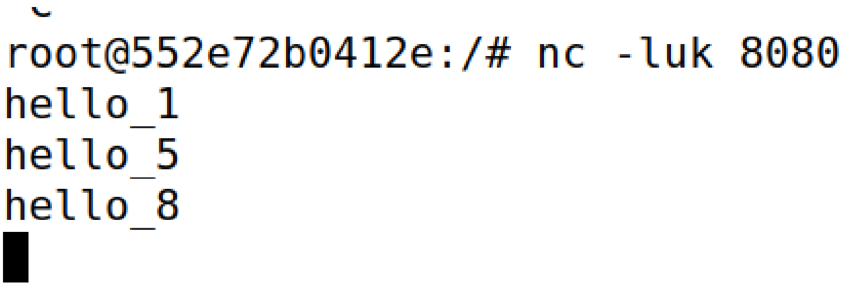
```
root@137e7980c0ce:/# echo hello|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello1|nc -u 10.9.0.11 8080
root@137e7980c0ce:/# echo hello_1|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_2|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_3|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_4|nc -u 10.9.0.11 8080
root@137e7980c0ce:/# echo hello_4|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_5|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_6|nc -u 10.9.0.11 8080
root@137e7980c0ce:/# echo hello_6|nc -u 10.9.0.11 8080
^[[A^C
root@137e7980c0ce:/# echo hello_7|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_8|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_9|nc -u 10.9.0.11 8080
```

在服务器192.168.60.5上监听8080端口，得到结果如下：



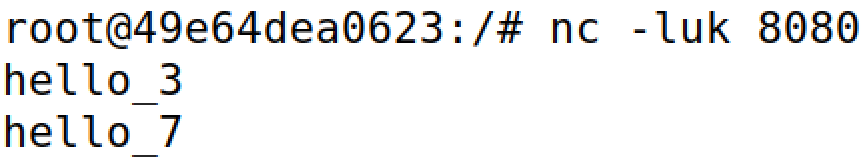
```
root@d998244af73c:/# nc -luk 8080
hello
hello_2
hello_4
hello_6
hello_9
```

在服务器192.168.60.6上监听8080端口，得到结果如下：



```
root@552e72b0412e:/# nc -luk 8080
hello_1
hello_5
hello_8
```

在服务器192.168.60.7上监听8080端口，得到结果如下：



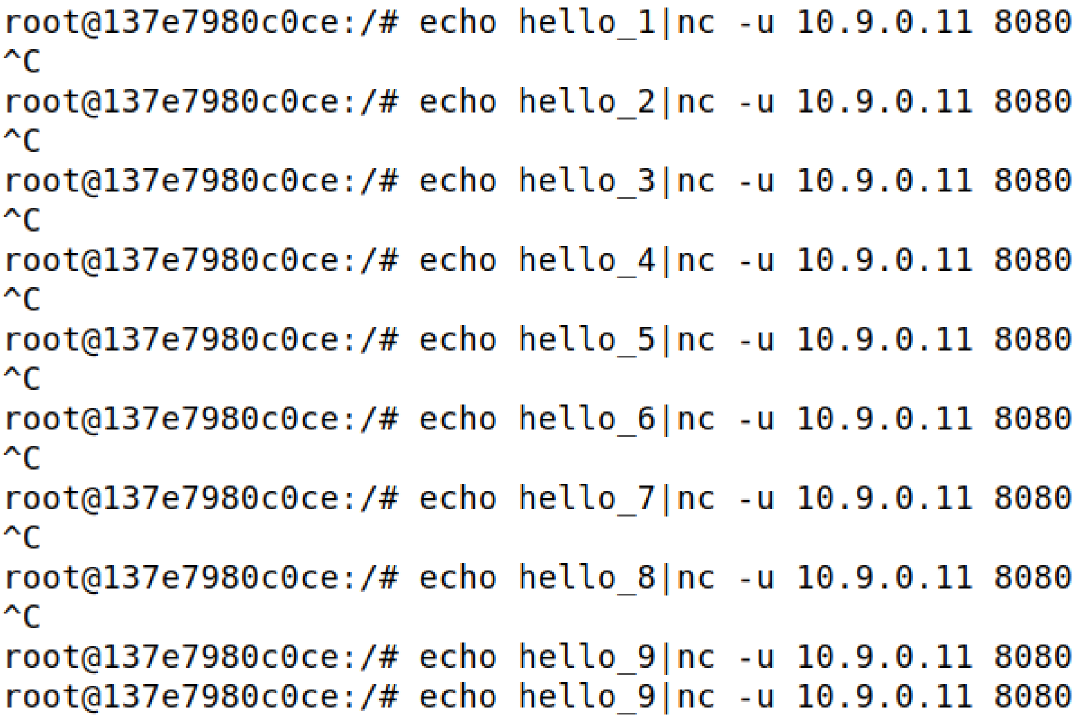
```
root@49e64dea0623:/# nc -luk 8080
hello_3
hello_7
```

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：

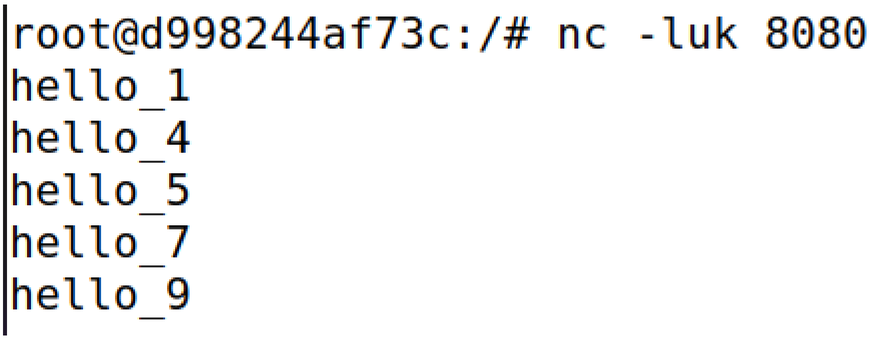
```
root@e94e2533f8f6:/# iptables -F
root@e94e2533f8f6:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.5:8080
root@e94e2533f8f6:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.6:8080
root@e94e2533f8f6:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.7:8080

```

结果如下：

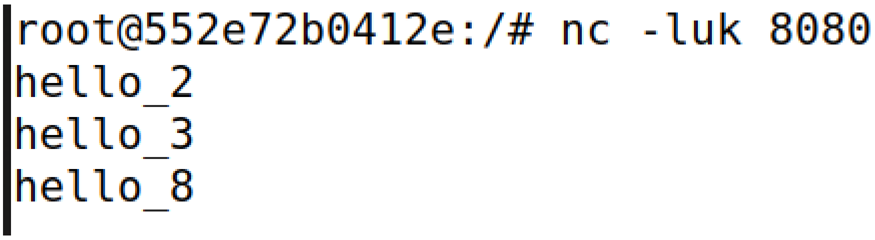
HostA上发送：



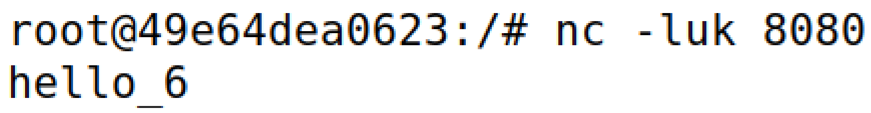
```
root@137e7980c0ce:/# echo hello_1|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_2|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_3|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_4|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_5|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_6|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_7|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_8|nc -u 10.9.0.11 8080
^C
root@137e7980c0ce:/# echo hello_9|nc -u 10.9.0.11 8080
root@137e7980c0ce:/# echo hello_9|nc -u 10.9.0.11 8080
```



```
root@d998244af73c:/# nc -luk 8080
hello_1
hello_4
hello_5
hello_7
hello_9
```



```
root@552e72b0412e:/# nc -luk 8080
hello_2
hello_3
hello_8
```



```
root@49e64dea0623:/# nc -luk 8080
hello_6
```

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。