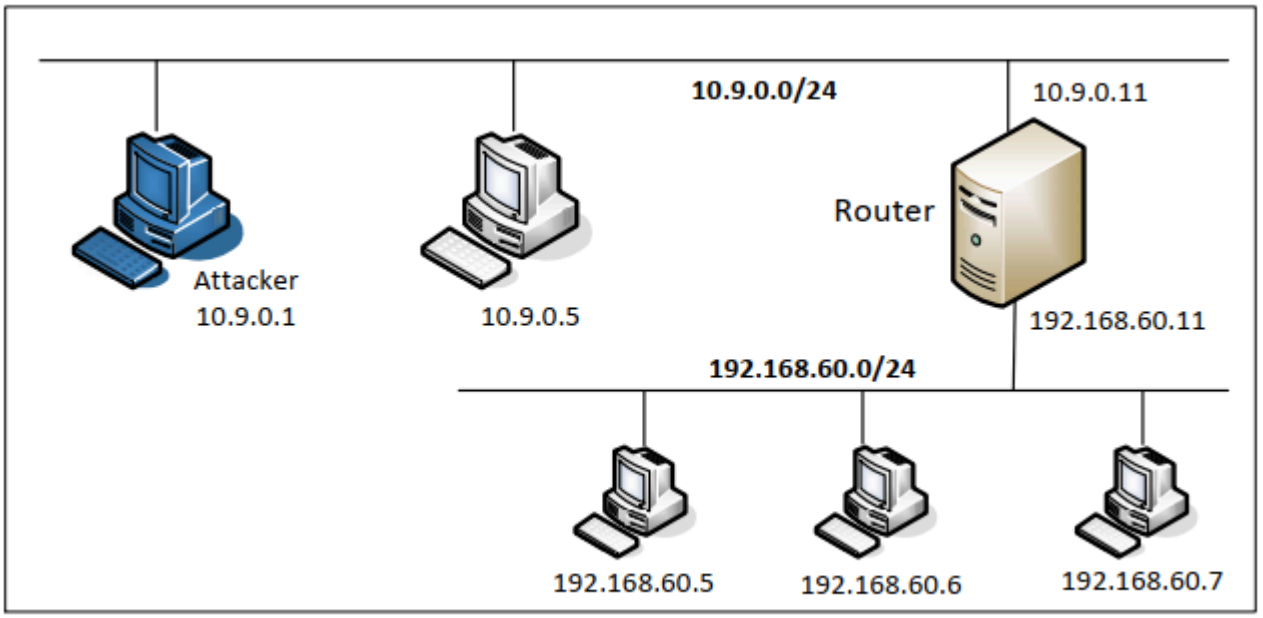
Firewall Lab

Before

环境如下图



记录一下ID，不过按实验手册所说，主要还是在虚拟机上完成

```
[08/07/25]seed@VM:~/.../Firewall Lab$ dockps
abcf40598ebd  hostA-10.9.0.5
23d813b42dab  seed-router
d967d797002d  host1-192.168.60.5
0467b853f0a9  host3-192.168.60.7
2d6785d94a4c  host2-192.168.60.6
```

值得一提的是，当我部署环境时，起初出现了无法解析域名的问题，难不成DNS实验的时候不小心改动了什么？最后靠网上的一篇blog解决的，简单讲就是设置一个临时的DNS服务器，从而让过本地的DNS故障，链接如下[Ubuntu 16.04:Temporary failure resolving 'security.ubuntu.com'*51CTO博客*Ubuntu 16.04]

Task1：实现一个简单的防火墙

Task1.A:实现一个简单的内核模块

本任务的文件均已包含在/Files/kernel_modules中，按照实验手册的要求运行`make`

```
[08/08/25]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Firewall_Lab/Fi
les/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Firewall_Lab/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Firewall_Lab/Files/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/Firewall_Lab/Files/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

插入模块，输入`sudo insmod hello.ko`

接下来让我们先再开一个窗口监控内核，输入指令`sudo dmesg`

```
754 585 929 697 967 000 ns fSetTimeLastLoop=true )
[ 4195.721168] hello: module verification failed: signature and/or required key
missing - tainting kernel
[ 4195.726163] Hello World!
```

可以看到内核信息的输出，再输入`lsmod | grep hello`列出模块

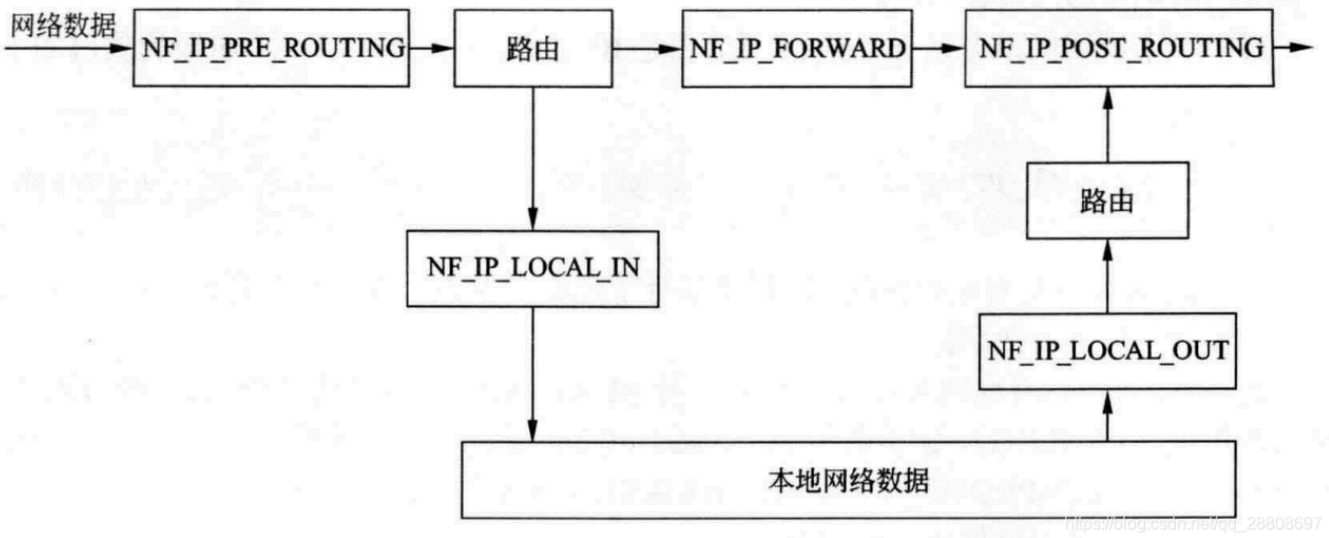
```
[08/08/25]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
```

最后再移除模块并查看信息

```
$ sudo rmmod hello （移除模块）
$ dmesg （查看信息）
$ sudo dmesg （在 Ubuntu 22.04 中，此命令需要超级用户权限）
```

```
[ 4195.721168] hello: module verification failed: signature and/or required key
missing - tainting kernel
[ 4195.726163] Hello World!
[ 4368.505310] Bye-bye World!.
```

Task1.B:使用Netfilter实现一个简单的防火墙



实验前先找张图，简单看看hook的位置

任务1.

打开packet_filter文件夹，和上一个任务相似的操作，先`make`

```
[08/08/25]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Firewall_Lab/Fi
les/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Firewall_Lab/Files/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Firewall_Lab/Files/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Firewall_Lab/Files/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

接下来让我们测试一下能否生成到8.8.8.8的UDP数据包

```
[08/08/25]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: 12864
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        149     IN      CNAME   www.example.com-v4.edgesuite.net
.
www.example.com-v4.edgesuite.net. 17592 IN CNAME a1422.dscr.akamai.net.
a1422.dscr.akamai.net.  20      IN      A       23.218.94.19
a1422.dscr.akamai.net.  20      IN      A       23.218.94.16

;; Query time: 76 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
:: WHEN: Fri Aug 08 04:37:18 EDT 2025
```

嗯，目前没有问题，让我们讲程序加载到内核中

```
[08/08/25]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[08/08/25]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

让我们再次测试

```
[08/08/25]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
```

再查看一下信息

```
[ 5332.054671] *** Dropping 8.8.8.8 (UDP), port 53
[ 5337.051814] *** LOCAL_OUT
[ 5337.051820]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 5337.051847] *** Dropping 8.8.8.8 (UDP), port 53
[ 5342.052187] *** LOCAL_OUT
[ 5342.052191]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 5342.052209] *** Dropping 8.8.8.8 (UDP), port 53
```

要求的数据包都丢失了，可见防火墙工作成功

任务2.

记下来要将printInfo函数挂载到所有hook上，让我们修改代码,

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


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


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 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 = printInfo;
   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_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_IN;
    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);

    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");
```

再重新挂载上去，ping一下其他IP试试，可以看到数据包的路径

```
[ 5931.114973] *** LOCAL_OUT
[ 5931.114976]     10.0.2.15  --> 8.8.8.8 (ICMP)
[ 5931.114988] *** POST_ROUTING
[ 5931.114989]     10.0.2.15  --> 8.8.8.8 (ICMP)
[ 5931.203590] *** PRE_ROUTING
[ 5931.203898]     8.8.8.8  --> 10.0.2.15 (ICMP)
[ 5931.204152] *** LOCAL_IN
[ 5931.204281]     8.8.8.8  --> 10.0.2.15 (ICMP)
[ 5932.117385] *** LOCAL_OUT
[ 5932.117389]     10.0.2.15  --> 8.8.8.8 (ICMP)
[ 5932.117400] *** POST_ROUTING
[ 5932.117402]     10.0.2.15  --> 8.8.8.8 (ICMP)
[ 5932.203875] *** PRE_ROUTING
[ 5932.204098]     8.8.8.8  --> 10.0.2.15 (ICMP)
[ 5932.204266] *** LOCAL_IN
[ 5932.204416]     8.8.8.8  --> 10.0.2.15 (ICMP)
```

各个钩子函数具体的执行过程是

任务3

实现另外两个钩子以实现以下功能： (1) 阻止其他计算机 ping 虚拟机， (2) 阻止其他计算机 telnet到虚拟机。

修改代码

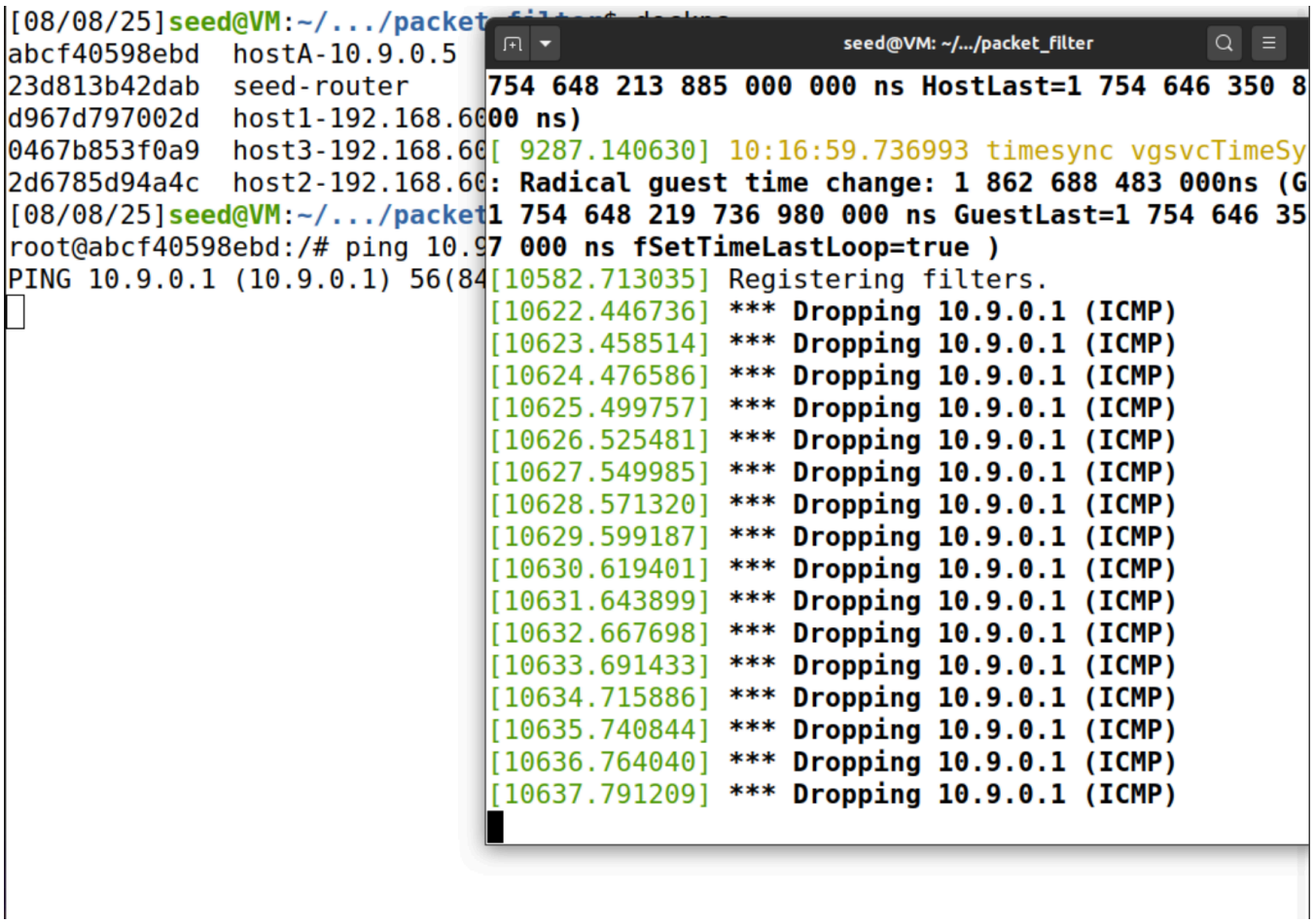
```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/icmp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>
```

```c
static struct nf_hook_ops hook1, hook2;


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

   //u16  port   = 53;
   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 && icmph->type==ICMP_ECHO){
            printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n", &(iph->daddr));
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}

unsigned int preventTelnet(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 (Telnet)\n", &(iph->daddr));
            return NF_DROP;
        }
   }
```

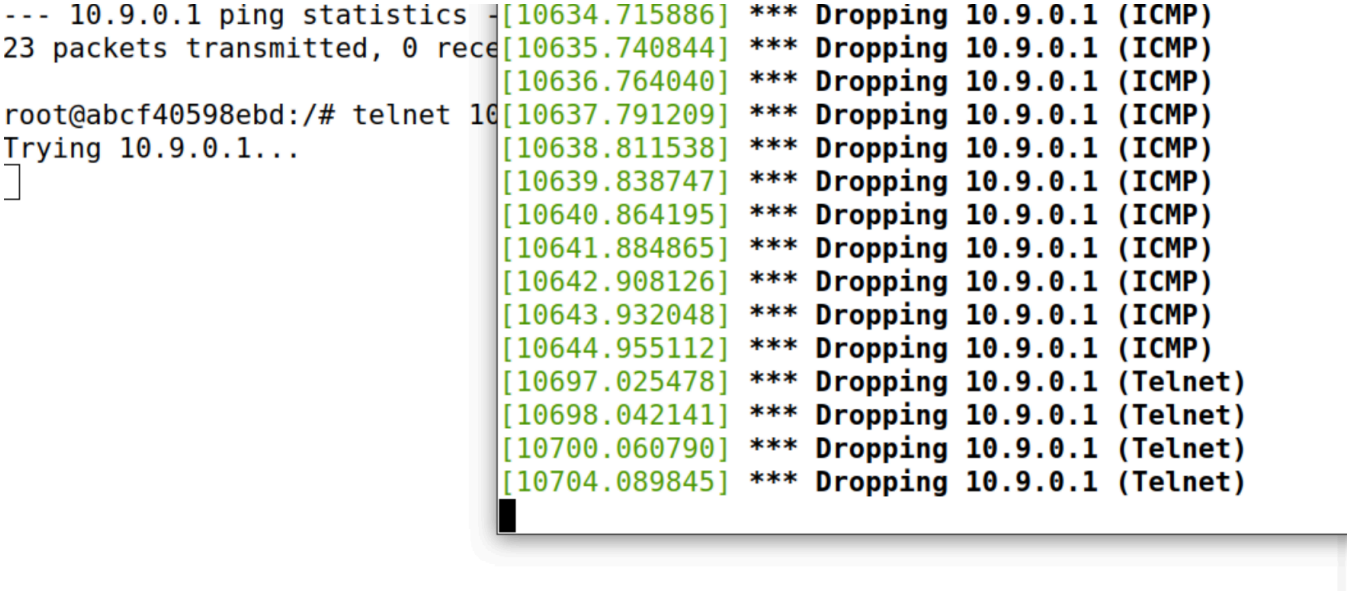
```
    return NF_ACCEPT;
}

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

    hook1.hook = preventPing;
    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 = preventTelnet;
    hook2.hooknum = NF_INET_PRE_ROUTING;
    hook2.pf = PF_INET;
    hook2.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook2);


    return 0;
}

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

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

两个钩子函数都注册到PRE_ROUTING上，以便对外来的数据包进行第一时间的筛查

在容器中ping一下虚拟机



再用telnet尝试一下



可见操作均被阻止，实验成功

Task2：防火墙规则的实验

表 1: iptables 表和链

表	链	功能
filter	INPUT FORWARD OUTPUT	数据包过滤
nat	PREROUTING INPUT OUTPUT POSTROUTING	修改源或目的网络地址
mangle	PREROUTING INPUT FORWARD OUTPUT POSTROUTING	数据包修改

根据实验手册，命令的结构一般如下：

```
iptables -t <table> -<operation> <chain> <rule> -j <target>
---------- -------------------- ------- -----------
              表                    链      规则       动作
```

总之，理清iptables的命令是这一块的难点

Task2.A：保护路由器

路由器执行以下命令

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

再用容器A访问路由器

```
root@abcf40598ebd:/# 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=0.292 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.074 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.066 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.090 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.054 ms
64 bytes from 10.9.0.11: icmp_seq=7 ttl=64 time=0.445 ms
^C
--- 10.9.0.11 ping statistics ---
```

```
root@abcf40598ebd:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

可以看到ping可以，但是telnet不行

最后还要清理一下

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

Task2.B:保护内部网络

本实验的要求是

1. 外部主机不能 ping 内部主机
2. 外部主机可以 ping 路由器
3. 内部主机可以 ping 外部主机
4. 内外部网络之间的其他数据包应当被阻止

```
iptables -A FORWARD  -i eth0 -p icmp --icmp-type echo-request -j DROP
iptables -A FORWARD  -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
iptables -A FORWARD  -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
iptables -P FORWARD DROP
```

让我们先进入内部主机host1-192.168.60.5

先ping一下外部主机

```
[08/08/25]seed@VM:~/.../packet_filter$ docksh d9
root@d967d797002d:/# 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=0.291 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.052 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.087 ms
^C
```

可行，再在外部主机hostA-10.9.0.5上ping一下内部主机和路由器

```
root@abcf40598ebd:/# 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 ---
7 packets transmitted, 0 received, 100% packet loss, time 61
34ms
```

```
root@abcf40598ebd:/# 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=0.100 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.108 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.128 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2036
```

一个不可行，一个可行

再telnet一下试试

```
root@abcf40598ebd:/# telnet 192.168.60.5
Trying 192.168.60.5...
telnet: Unable to connect to remote host: Connection timed o
ut
```

显然其他数据包被阻止了

最后清除一下规则

```
iptables -F
iptables -P FORWARD ACCEPT
```

Task2.C:保护内部服务器

要求如下：

1. 所有内部主机都运行了一个 telnet 服务器（监听端口 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
2. 外部主机不能访问其他内部服务器
3. 内部主机可以访问所有内部服务器
4. 内部主机不能访问外部服务器
5. 在本任务中，不允许使用连接追踪机制，它将在后续任务中使用

```
iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 --sport 23 -j ACCEPT
iptables -P FORWARD DROP
```

在10.9.0.5上telnet 192.168.60.5

```
root@abcf40598ebd:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
d967d797002d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Fri Aug  8 13:28:59 UTC 2025 from www.SeedLabSQLInjection.com on pts/2
seed@d967d797002d:~$
```

可以连接，但是尝试其他机子

```
root@abcf40598ebd:/# telnet 192.168.60.6
Trying 192.168.60.6...
telnet: Unable to connect to remote host: Connection timed o
ut
```

嗯，失败了，再看看内部机子，可以互联，但是不能连外界的机子

```
root@d967d797002d:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2d6785d94a4c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

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

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@2d6785d94a4c:~$
```

```
seed@2d6785d94a4c:~$ exit
logout
Connection closed by foreign host.
root@d967d797002d:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

最后还是要清除一下

Task3：连接追踪与有状态防火墙

Task3.A：连接追踪实验

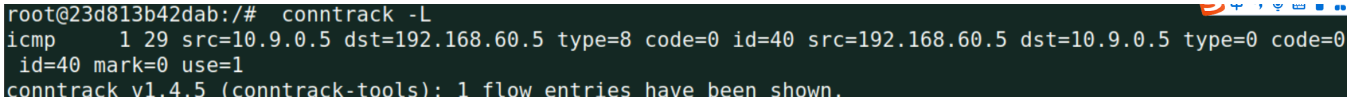
通过命令`conntrack -L`进行操作

- ICMP实验

运行命令

```
// 在 10.9.0.5 上，发送 ICMP 数据包
# ping 192.168.60.5
```

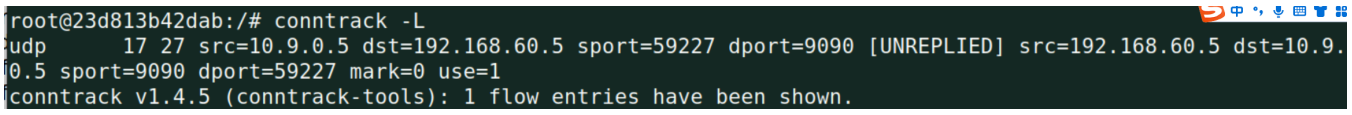
在路由器上查看



可见ICMP连接的持续时间为半分钟

- UDP实验

```
// 在 192.168.60.5 上，启动一个 netcat UDP 服务器
# nc -lu 9090
// 在 10.9.0.5 上，发送 UDP 数据包
# nc -u 192.168.60.5 9090
<输入一些内容，然后回车>
```



UDP连接的时间也在半分钟左右

- TCP实验

```
// 在 192.168.60.5 上，启动一个 netcat TCP 服务器
# nc -l 9090
// 在 10.9.0.5 上，发送 TCP 数据包
# nc 192.168.60.5 9090
<输入一些内容，然后回车>
```

```
root@23d813b42dab:/# conntrack -L
tcp      6 431994 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=44526 dport=9090 src=192.168.60.5 dst=10
.9.0.5 sport=9090 dport=44526 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@23d813b42dab:/#
```

可以看到TCP连接时间还有431994s,约5天的样子

Task3.B：设置有状态防火墙

我们要重新写task2.C中的规则，并添加一条**允许内部主机访问任何外部服务器**的规则

总结：

1. 所有内部主机都运行了一个 telnet 服务器（监听端口 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
2. 外部主机不能访问其他内部服务器
3. 内部主机可以访问所有内部服务器
4. 允许内部主机访问任何外部服务器

```
iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate
NEW -j ACCEPT
iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT
iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p tcp -j DROP
iptables -P FORWARD ACCEPT
```

规则1、2满足

```
root@abcf40598ebd:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
d967d797002d login:
```

```
root@abcf40598ebd:/# telnet 192.168.60.6
Trying 192.168.60.6...
telnet: Unable to connect to remote host: Connection timed o
ut
```

规则3满足

```
root@d967d797002d:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2d6785d94a4c login:
```

规则4满足

```
root@d967d797002d:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
abcf40598ebd login:
```

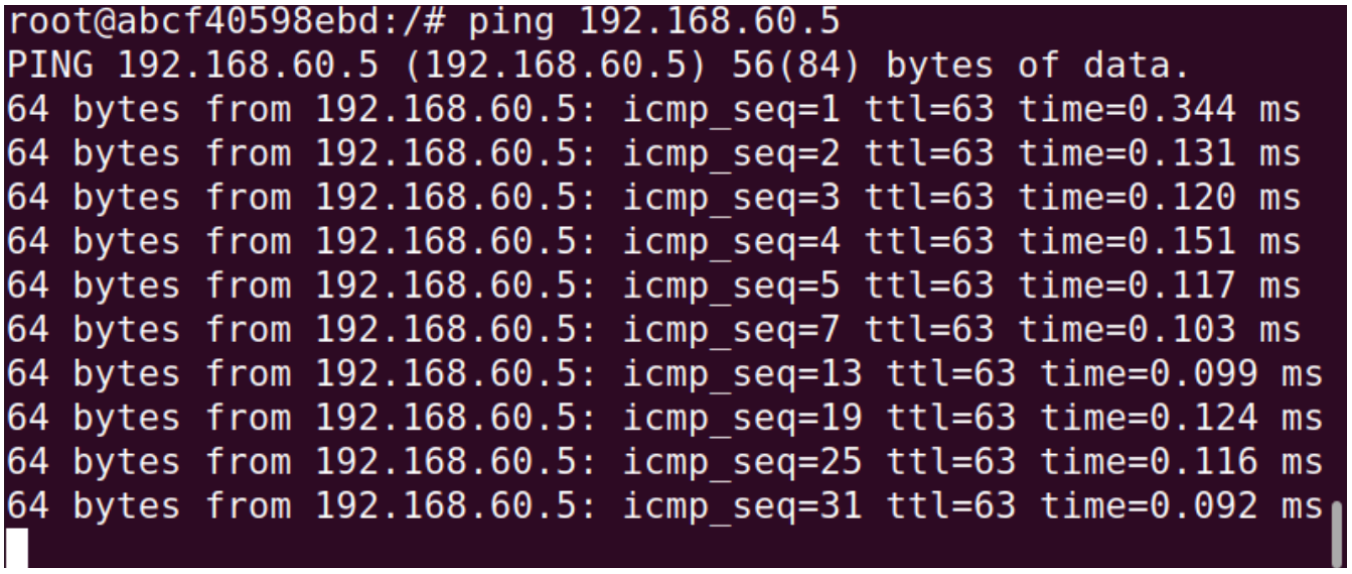
最后清除一下

Task4：限制网络流量

样例代码

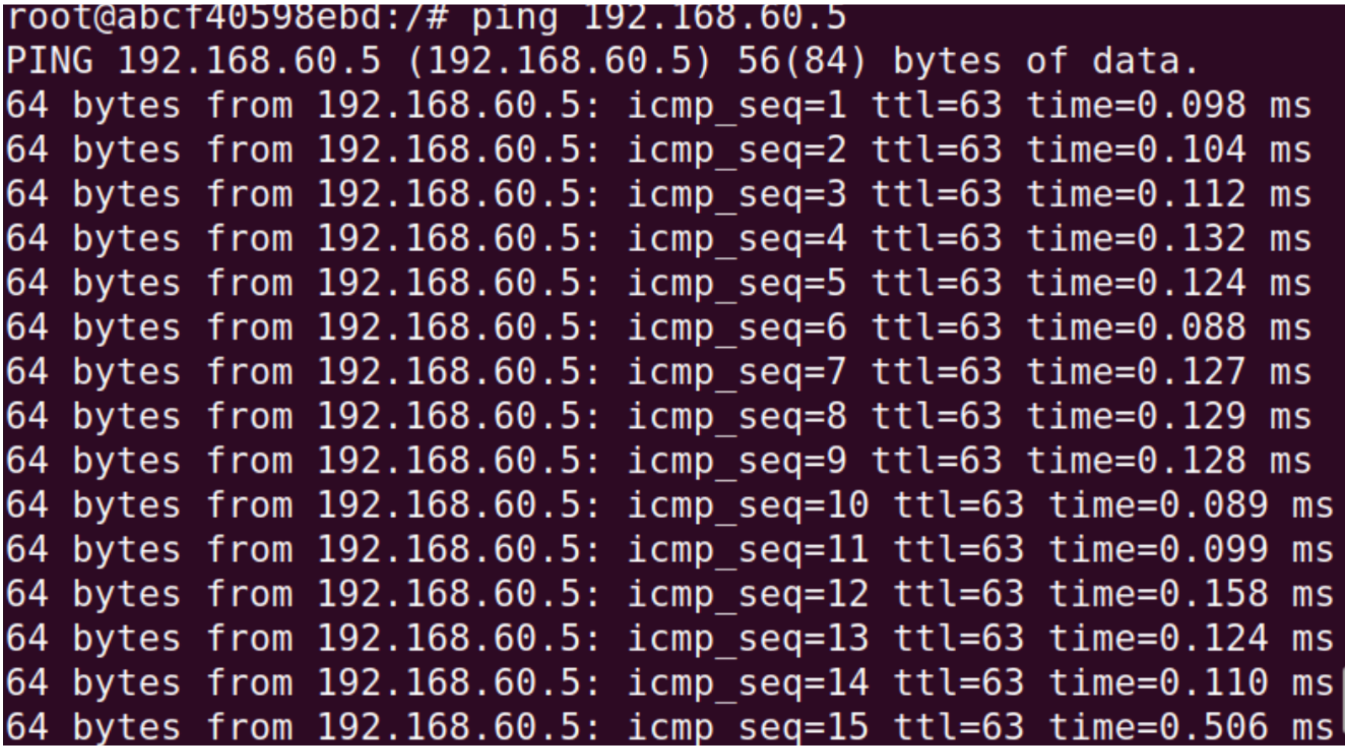
```
iptables -A FORWARD -s 10.9.0.5 -m limit \
--limit 10/minute --limit-burst 5 -j ACCEPT
iptables -A FORWARD -s 10.9.0.5 -j DROP
```

在路由器上运行后，尝试从10.9.0.5 ping 192.168.60.5



一开始速度还挺快，后来变慢，icmp_seq也变得不连续

尝试把第二条命令去掉再次尝试



感觉和正常情况下差不多，说明如果没有第二条命令，超过限制的数据包并不会丢失，所以第二条命令是必要的

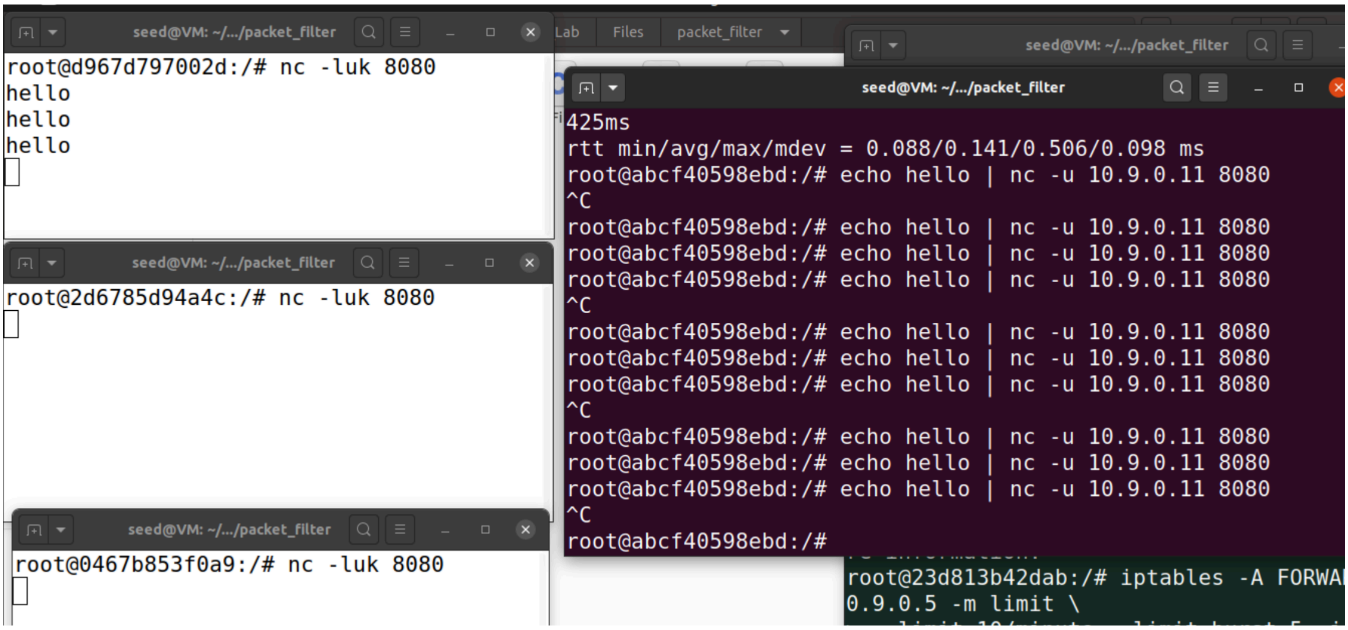
Task5：负载均衡

让我们使用两个机制来负载均衡

nth模式(轮询)

在路由器设置以下规则

```
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
```

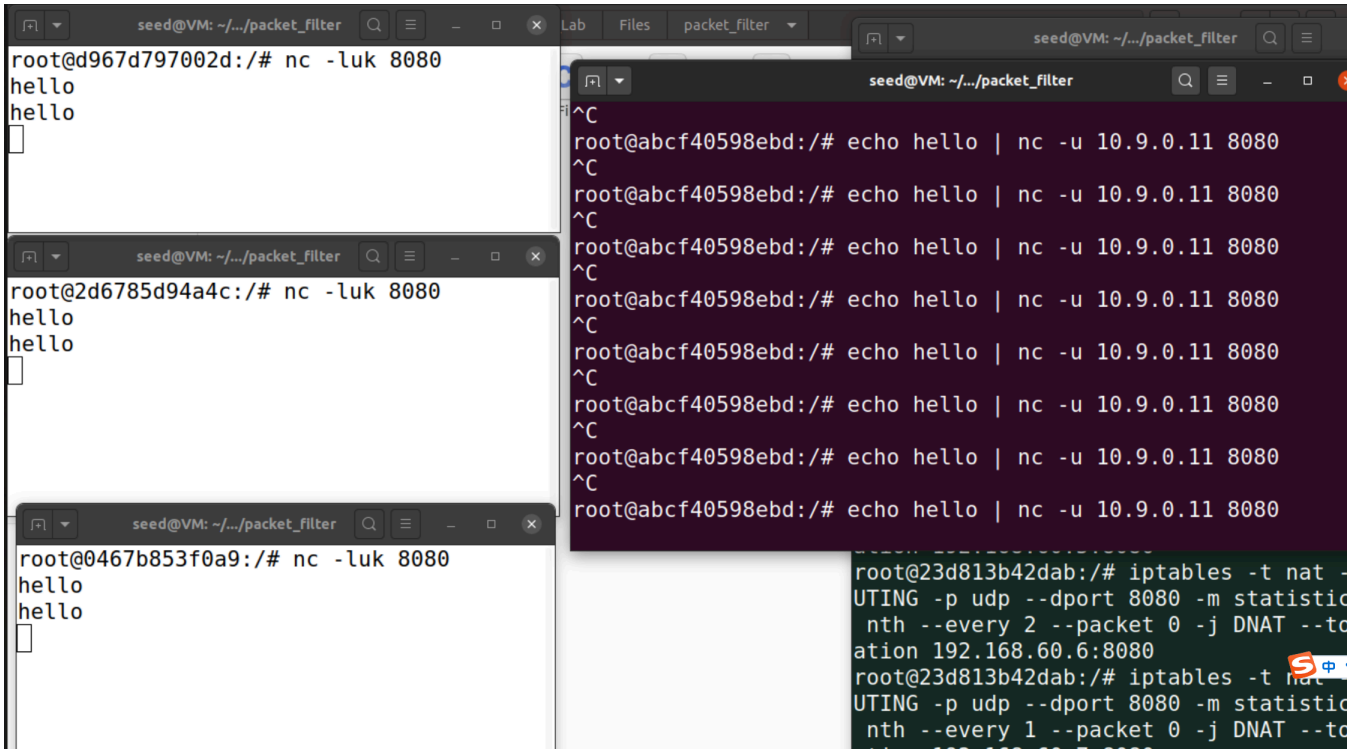


会发现每三个数据包才会发送一个

接下来修改规则

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

具体实现就是先三取一，在未接收的两个包裹里再二取一，剩下来的全给另一个，这里的要点就是规则的顺序会对规则的具体实现产生影响



可见三个主机收到相同数量的数据包

记得清楚规则

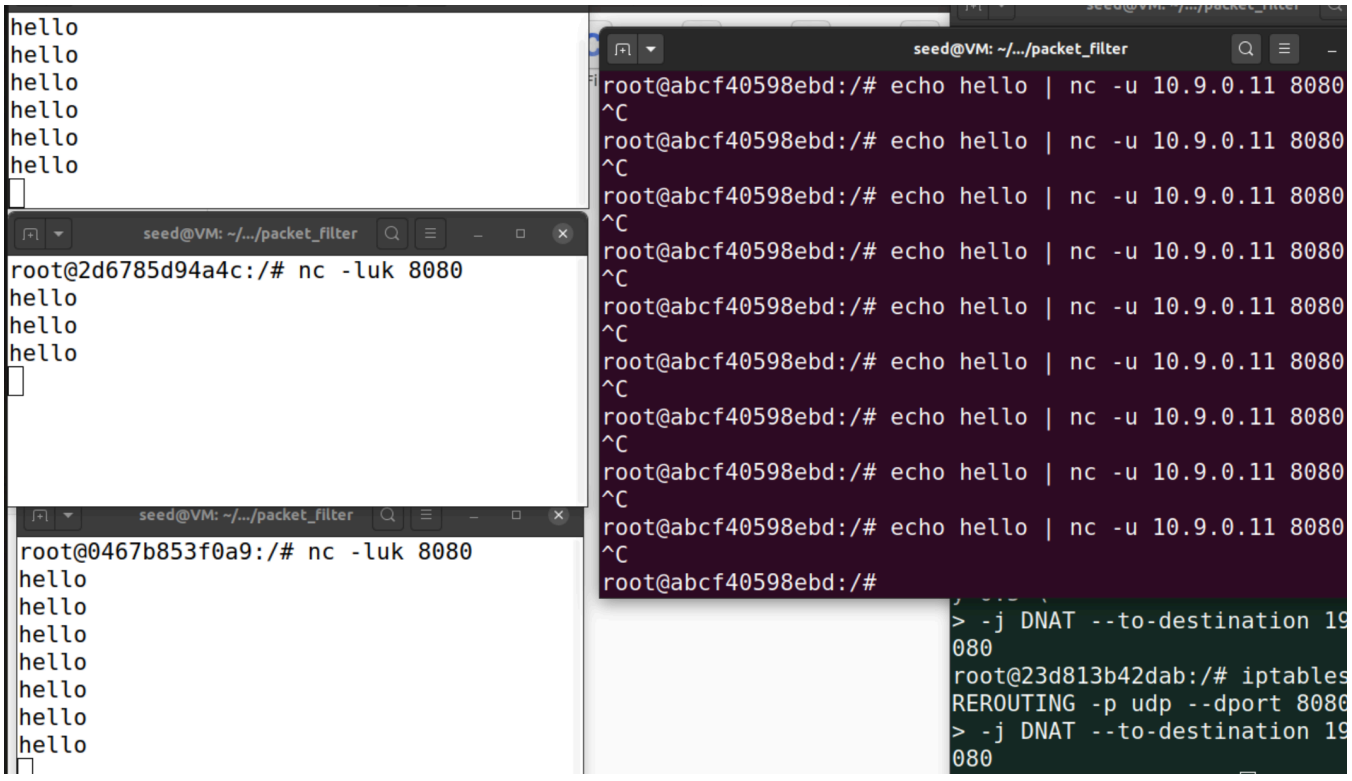
```
iptables -t nat -F PREROUTING
```

random模式

编写规则

```
iptables -t nat -A PREROUTING -p udp --dport 8080 \
-m statistic --mode random --probability 0.3333 \
-j DNAT --to-destination 192.168.60.5:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 \
-m statistic --mode random --probability 0.5 \
-j DNAT --to-destination 192.168.60.6:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 \
-j DNAT --to-destination 192.168.60.7:8080
```

需要注意的是这里仍然是链式规则，下面的规则要在上面未分配的数据包中计算



暂时来看第二台机子分到的明显要少，不知道样本再大一些效果如何。

本实验的难点一方面是C语言编程，还有就是对防火墙规则的理解，整体做下来对这些有了较为清晰的了解，但熟练还远远谈不上。

Over！！！