NAME: YASH SNEHAL SHETIYA

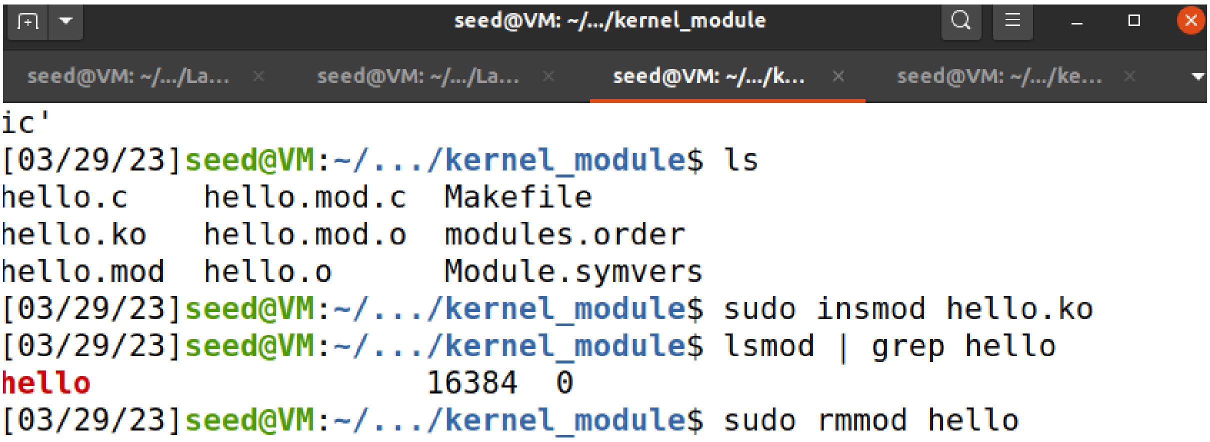
SUID: 9276568741

FIREWALL EXPLORATION LAB

Task 1: Implementing a Simple Firewall

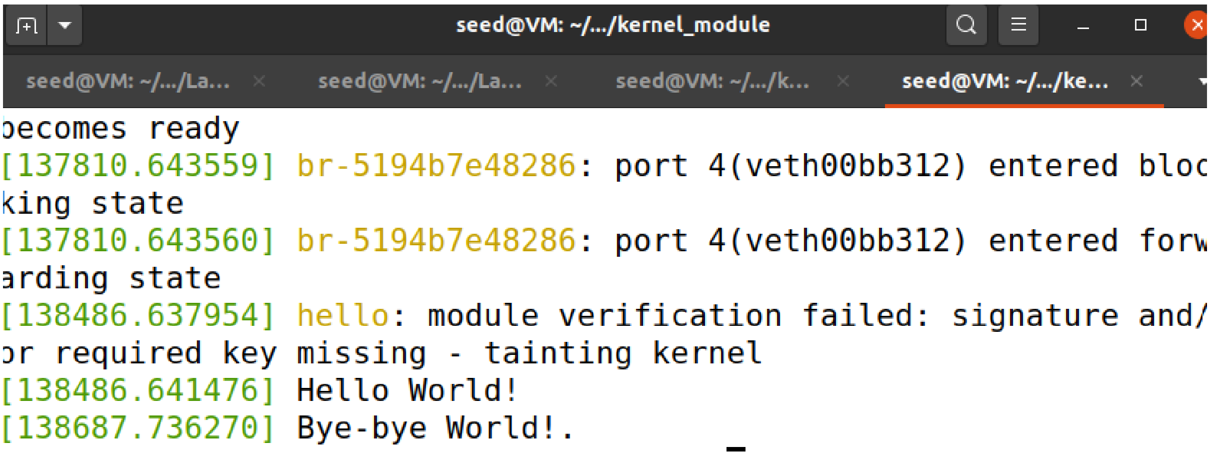
1A

First, we browse the kernel module folder and make the files into a loadable kernel module using the make command.



```
ic'
[03/29/23]seed@VM:~/.../kernel_module$ ls
hello.c    hello.mod.c  Makefile
hello.ko   hello.mod.o  modules.order
hello.mod  hello.o      Module.symvers
[03/29/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[03/29/23]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[03/29/23]seed@VM:~/.../kernel_module$ sudo rmmod hello
```

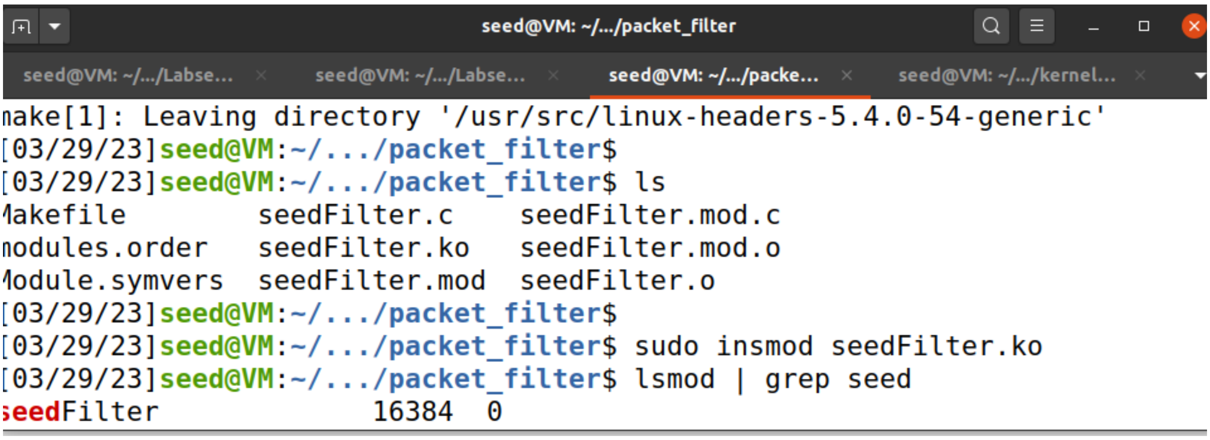
We have also inserted the 'hello.ko' kernel module and have listed it as seen above.



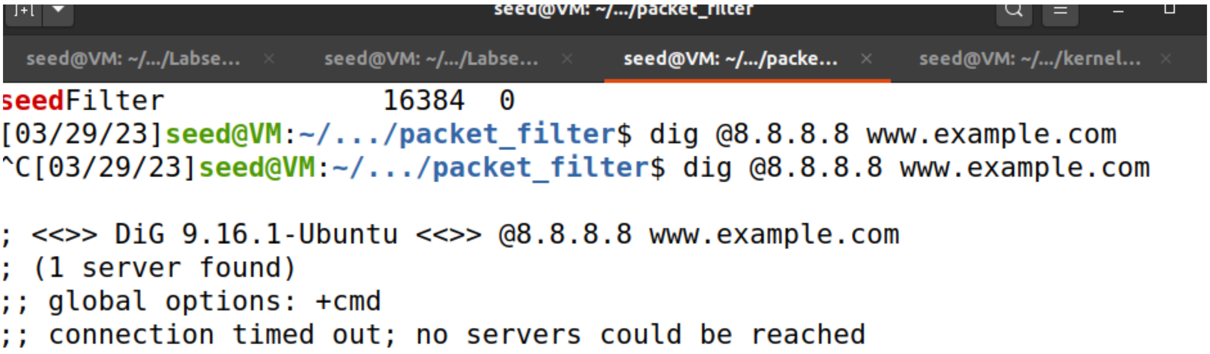
```
becomes ready
[137810.643559] br-5194b7e48286: port 4(veth00bb312) entered bloc
king state
[137810.643560] br-5194b7e48286: port 4(veth00bb312) entered forw
arding state
[138486.637954] hello: module verification failed: signature and/
or required key missing - tainting kernel
[138486.641476] Hello World!
[138687.736270] Bye-bye World!.
```

1B

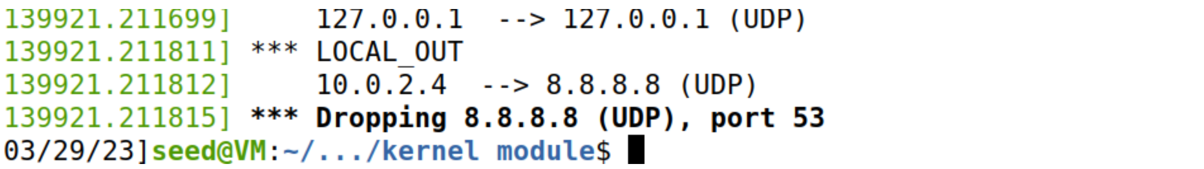
Firstly compiled the seedFilter.c file using the make command. Before making the kernel module we are able to send UDP packets to Google’s DNS server i.e 8.8.8.8 but now that we have inserted the kernel module, the following has been achieved.



```
nake[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/29/23]seed@VM:~/.../packet_filter$
[03/29/23]seed@VM:~/.../packet_filter$ ls
1akefile        seedFilter.c    seedFilter.mod.c
nodules.order   seedFilter.ko   seedFilter.mod.o
1odule.symvers  seedFilter.mod  seedFilter.o
[03/29/23]seed@VM:~/.../packet_filter$
[03/29/23]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[03/29/23]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
```



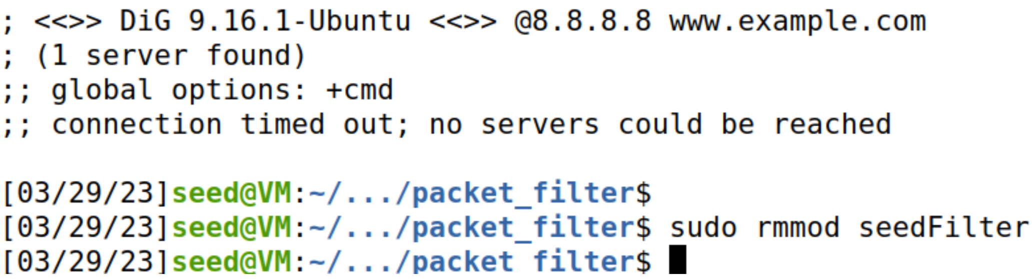
```
seedFilter             16384  0
[03/29/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com
^C[03/29/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> 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
```



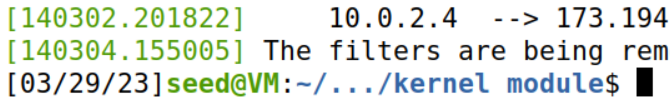
```
139921.211699]     127.0.0.1  --> 127.0.0.1 (UDP)
139921.211811] *** LOCAL_OUT
139921.211812]     10.0.2.4  --> 8.8.8.8 (UDP)
139921.211815] *** Dropping 8.8.8.8 (UDP), port 53
03/29/23]seed@VM:~/.../kernel module$
```

Now, that we can observe when we try to reach 8.8.8.8 its unreachable which means it is working as we want. It can be verified when the UDP packets are attempted to get picked thrice before getting dropped.

Filters being removed using rmmod command



```
; <<>> 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

[03/29/23]seed@VM:~/.../packet_filter$
[03/29/23]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
[03/29/23]seed@VM:~/.../packet_filter$
```

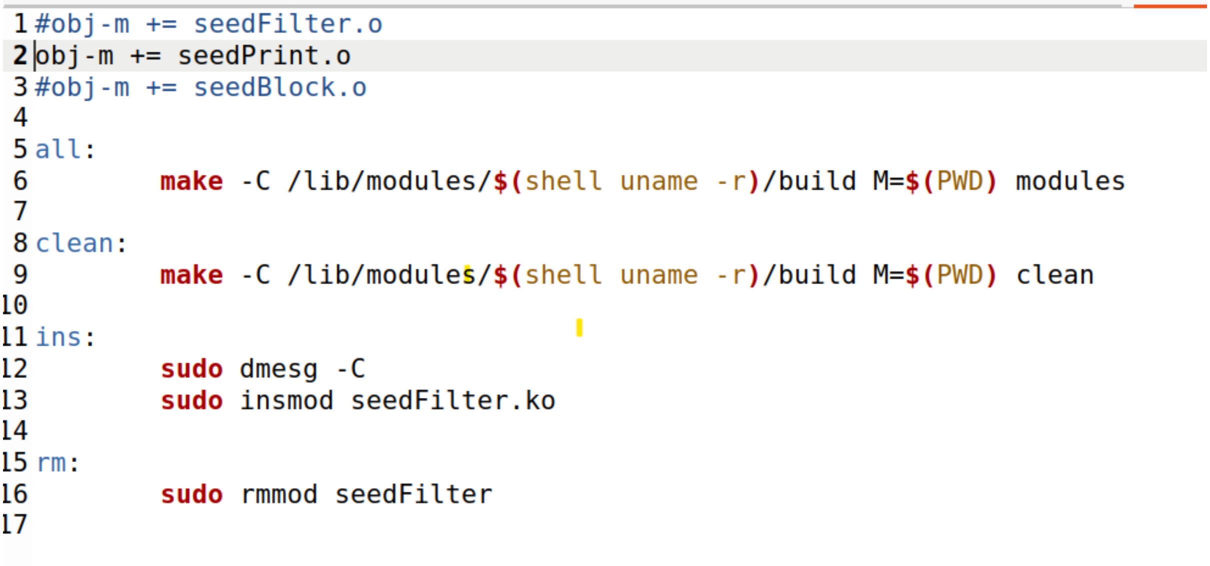


```
[140302.201822]     10.0.2.4  --> 173.194.31.198 (TCP)
[140304.155005] The filters are being removed.
[03/29/23]seed@VM:~/.../kernel module$ 
```

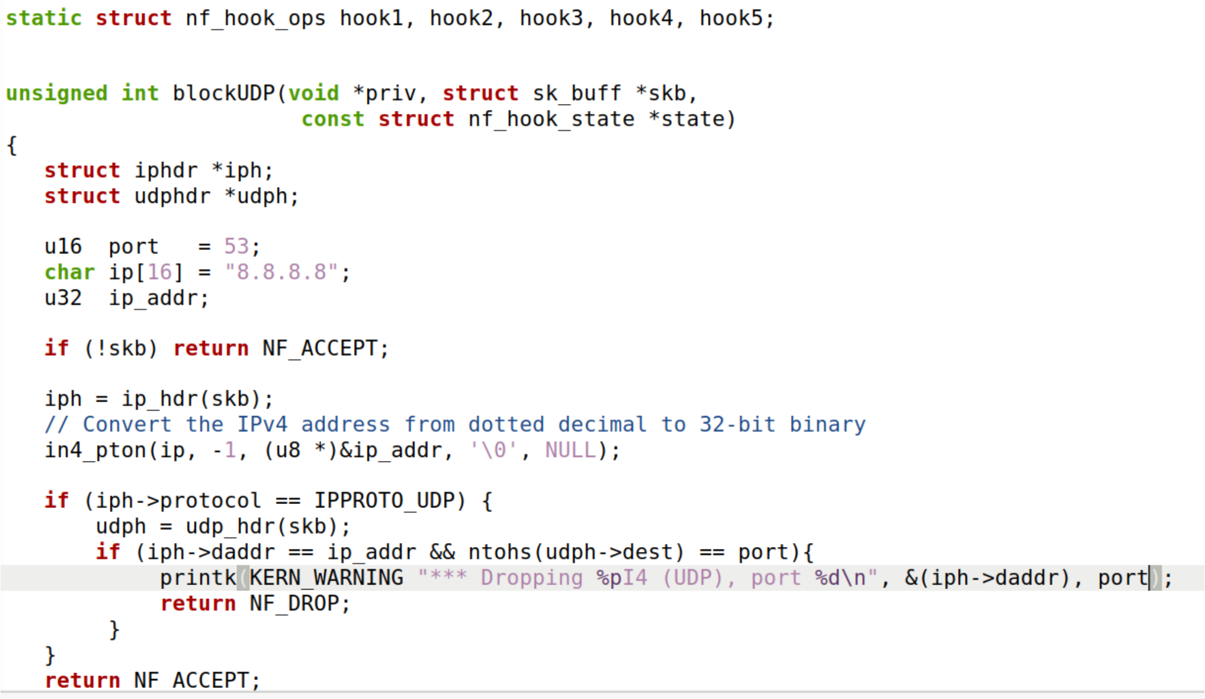
seedPrint

We created a new file named seedPrint and added its executable to the kernel make file.

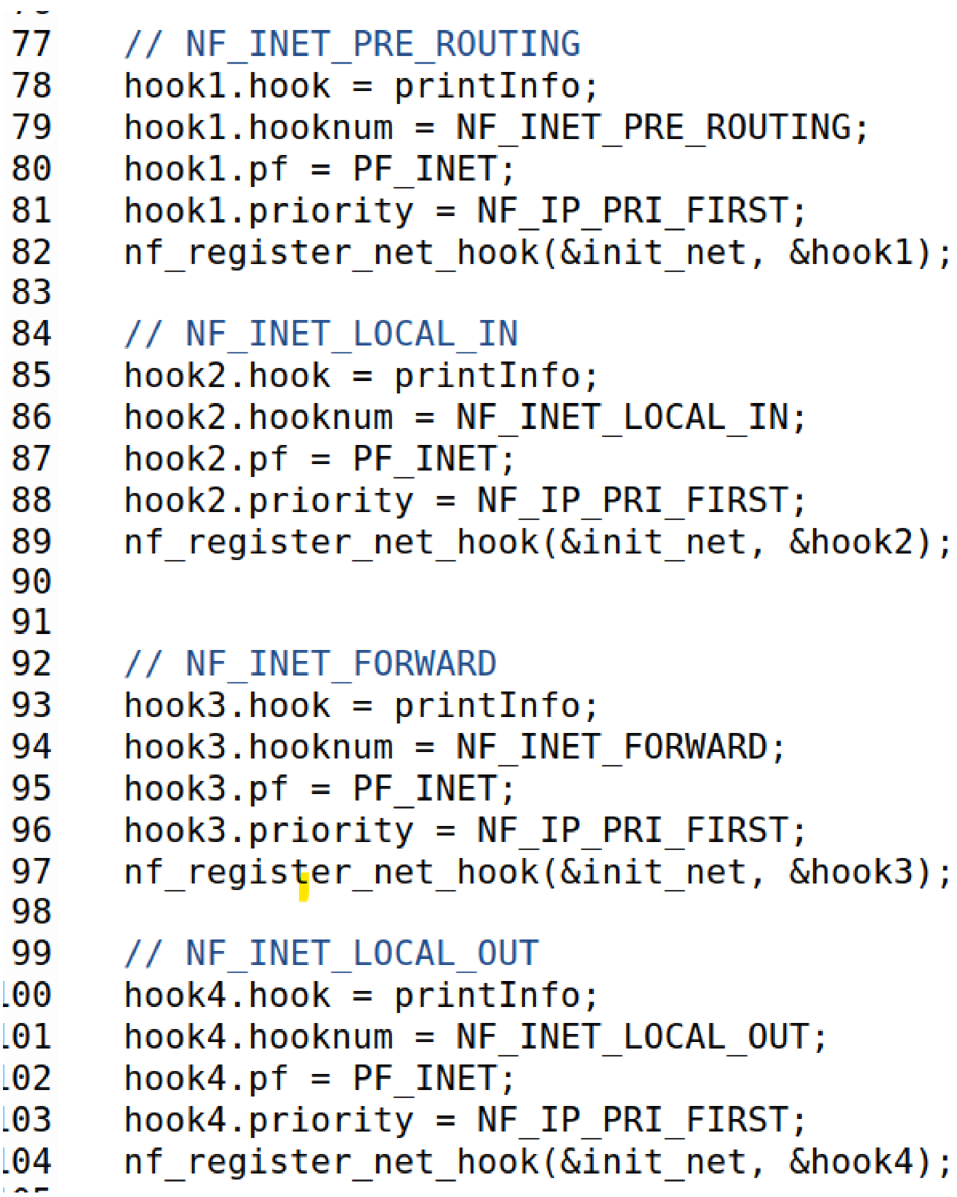
We make specific changes to the existing code, such as adding the number of hooks and also add the commands to unregister the hooks once we exit the module.



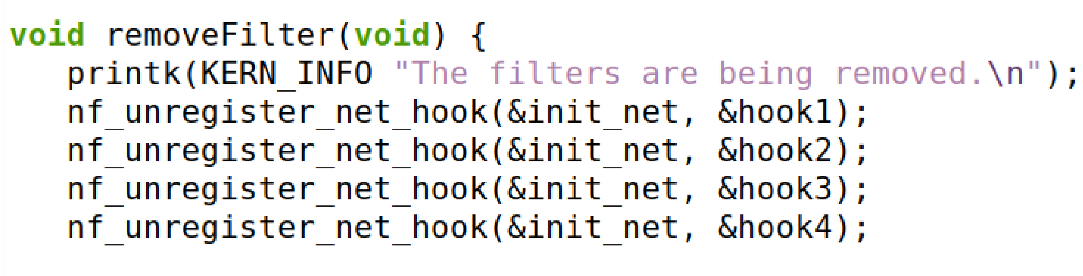
```
#obj-m += seedFilter.o
obj-m += seedPrint.o
#obj-m += seedBlock.o

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

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

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter

```



```
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;
```

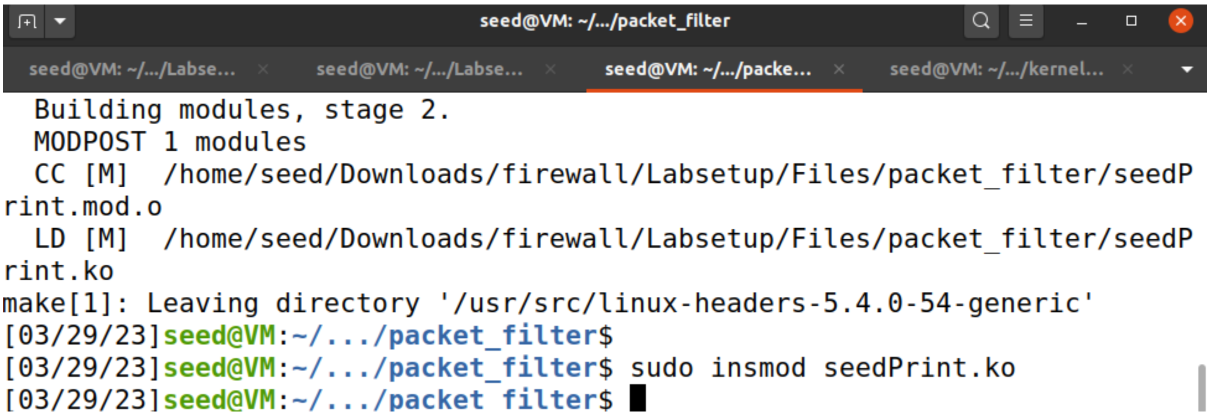


```
    // NF_INET_PRE_ROUTING
    hook1.hook = printInfo;
    hook1.hooknum = NF_INET_PRE_ROUTING;
    hook1.pf = PF_INET;
    hook1.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook1);

    // NF_INET_LOCAL_IN
    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);


    // NF_INET_FORWARD
    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);

    // NF_INET_LOCAL_OUT
    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);
```



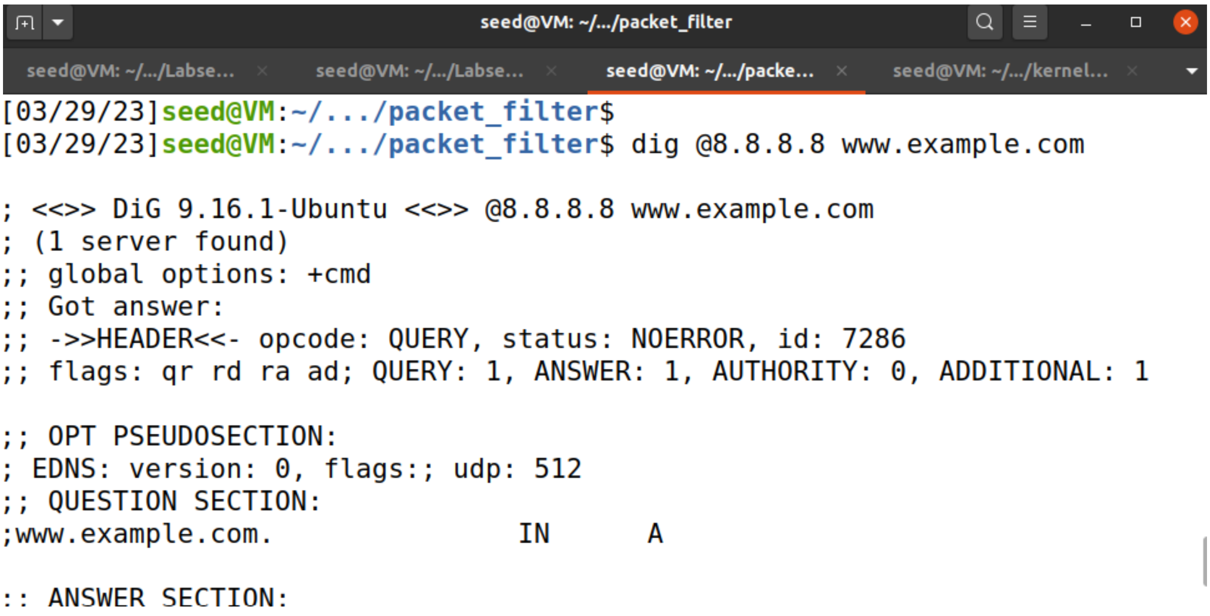
```
void removeFilter(void) {
   printk(KERN_INFO "The filters are being removed.\n");
   nf_unregister_net_hook(&init_net, &hook1);
   nf_unregister_net_hook(&init_net, &hook2);
   nf_unregister_net_hook(&init_net, &hook3);
   nf_unregister_net_hook(&init_net, &hook4);
```

we compile the fine into a kernel module using make command and insert the seedPrint kernel module as follows



```
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/firewall/Labsetup/Files/packet_filter/seedP
rint.mod.o
  LD [M]  /home/seed/Downloads/firewall/Labsetup/Files/packet_filter/seedP
rint.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/29/23]seed@VM:~/.../packet_filter$
[03/29/23]seed@VM:~/.../packet_filter$ sudo insmod seedPrint.ko
[03/29/23]seed@VM:~/.../packet filter$ 
```

We make the use of dig command again to check the UDP packets which are generated and the different functions that are being invoked.



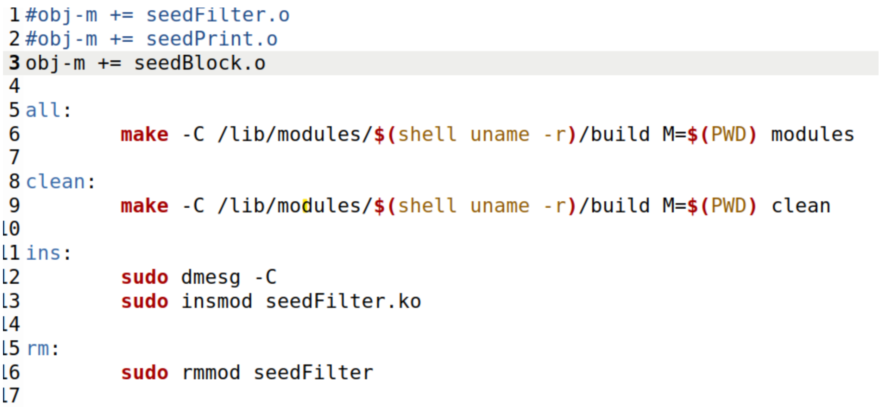
```
[03/29/23]seed@VM:~/.../packet_filter$
[03/29/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> 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: 7286
;; 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:
```

When the dmesg command is applied we can view that the LOCAL_OUT, LOCAL_IN, POST_ROUTING and PRE_ROUTING functions were invoked as the UDP packets were generated.

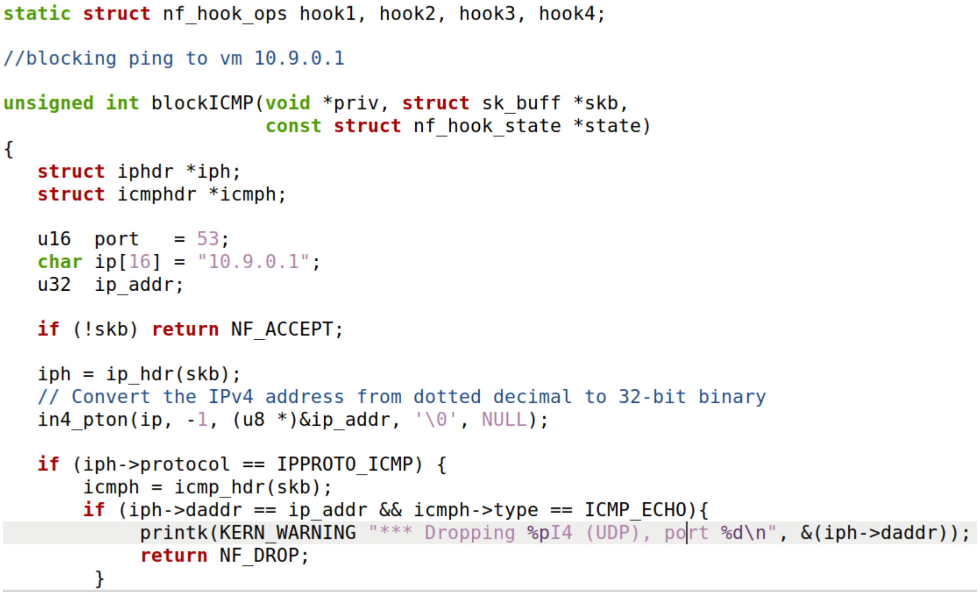
After completing the task we unhook the hooks.

seedBlock

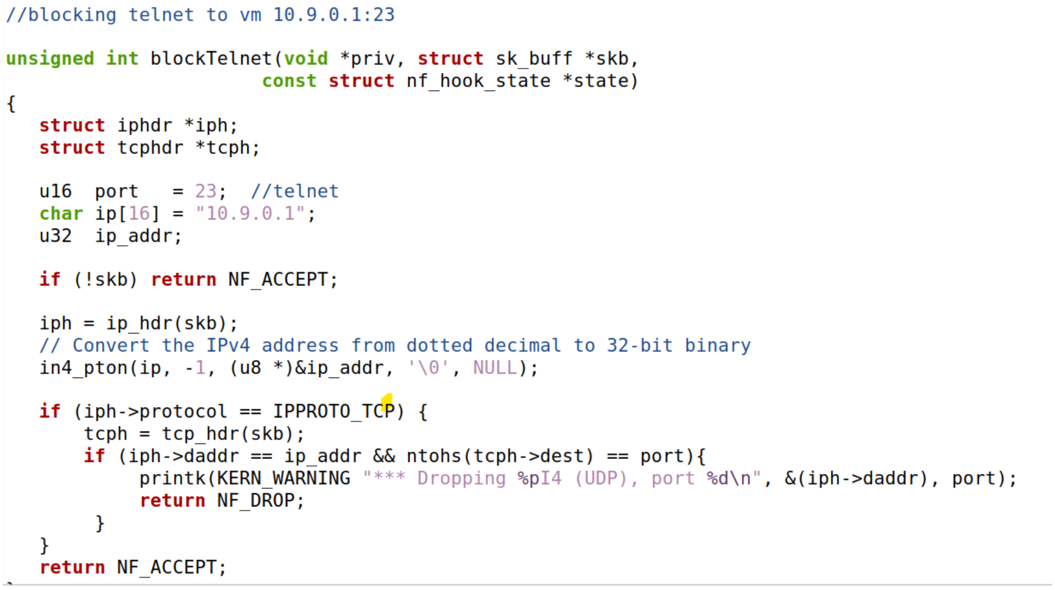
For this task we create a new file named seedBlock.ko. We have 2 separate functions to be done using this, firstly prevent other computers to ping the Vm and prevent other computers to telnet the VM.



```
#obj-m += seedFilter.o
#obj-m += seedPrint.o
obj-m += seedBlock.o

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

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

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter

```



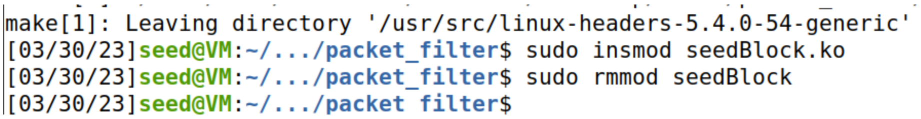
```
static struct nf_hook_ops hook1, hook2, hook3, hook4;

//blocking ping to vm 10.9.0.1

unsigned int blockICMP(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 (UDP), port %d\n", &(iph->daddr));
            return NF_DROP;
        }
```



```
//blocking telnet to vm 10.9.0.1:23

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

   u16  port   = 23;  //telnet
   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 (UDP), port %d\n", &(iph->daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
```

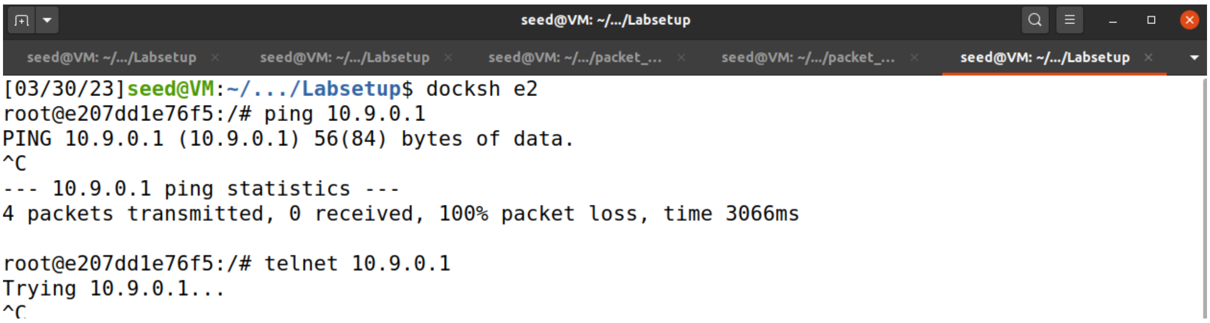
Similarly we add the other modifications too.

We make our module and insert the kernel module as follows:

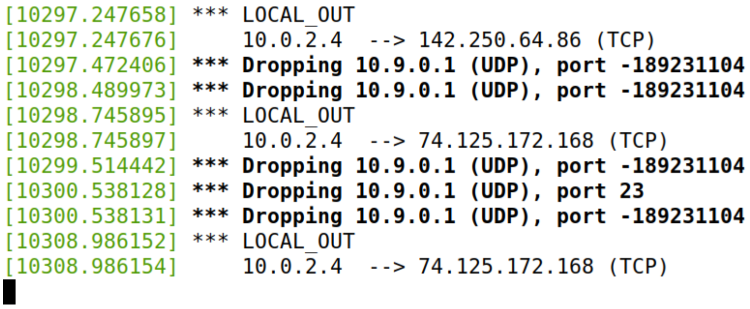


```
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/30/23]seed@VM:~/.../packet_filter$ sudo insmod seedBlock.ko
[03/30/23]seed@VM:~/.../packet_filter$ sudo rmmod seedBlock
[03/30/23]seed@VM:~/.../packet_filter$
```

When we try to ping the VM (ping 10.9.0.1) , we observed that the UDP packets are being dropped as follows:

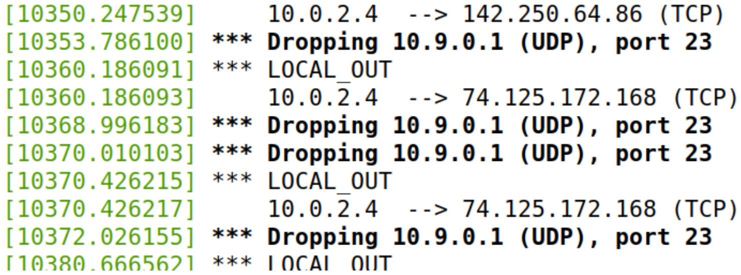


```
[03/30/23]seed@VM:~/.../Labsetup$ docksh e2
root@e207dd1e76f5:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3066ms

root@e207dd1e76f5:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
```



```
[10297.247658] *** LOCAL_OUT
[10297.247676]     10.0.2.4  --> 142.250.64.86 (TCP)
[10297.472406] *** Dropping 10.9.0.1 (UDP), port -189231104
[10298.489973] *** Dropping 10.9.0.1 (UDP), port -189231104
[10298.745895] *** LOCAL_OUT
[10298.745897]     10.0.2.4  --> 74.125.172.168 (TCP)
[10299.514442] *** Dropping 10.9.0.1 (UDP), port -189231104
[10300.538128] *** Dropping 10.9.0.1 (UDP), port 23
[10300.538131] *** Dropping 10.9.0.1 (UDP), port -189231104
[10308.986152] *** LOCAL_OUT
[10308.986154]     10.0.2.4  --> 74.125.172.168 (TCP)
```

Now, we try to telnet the VM:



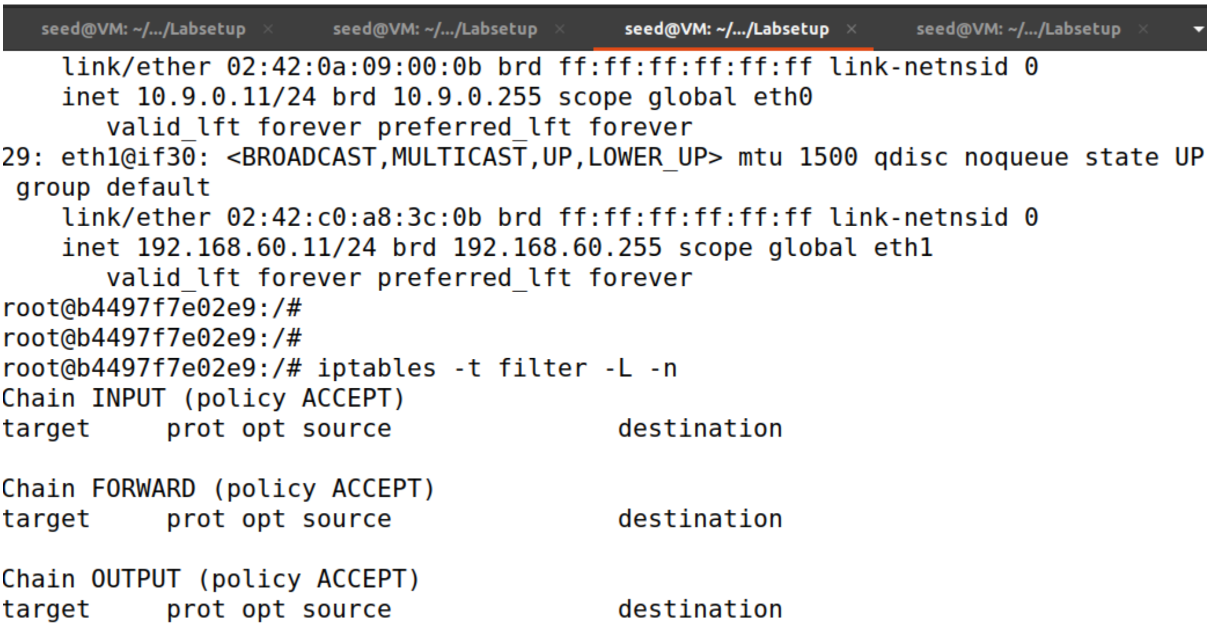
```
[10350.247539]     10.0.2.4  --> 142.250.64.86 (TCP)
[10353.786100] *** Dropping 10.9.0.1 (UDP), port 23
[10360.186091] *** LOCAL_OUT
[10360.186093]     10.0.2.4  --> 74.125.172.168 (TCP)
[10368.996183] *** Dropping 10.9.0.1 (UDP), port 23
[10370.010103] *** Dropping 10.9.0.1 (UDP), port 23
[10370.426215] *** LOCAL_OUT
[10370.426217]     10.0.2.4  --> 74.125.172.168 (TCP)
[10372.026155] *** Dropping 10.9.0.1 (UDP), port 23
[10380.666562] *** LOCAL_OUT
```

Task 2: Experimenting with Stateless Firewall Rules

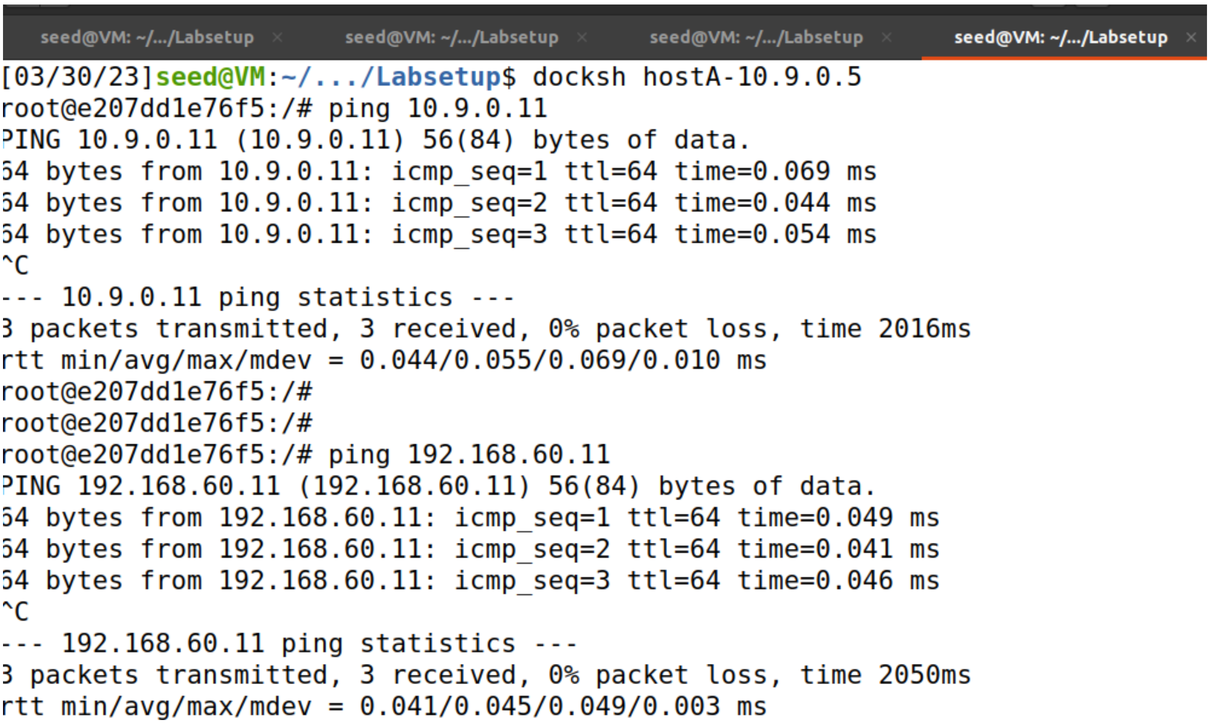
2A

When we hit the ‘ ip addr’ command, we can see that there are two interfaces Eth0 and Eth1 with IP’s 10.9.0.11 and 192.168.60.11 respectively.

Before applying the rules we check the iptables status and ping the required two IP’s:

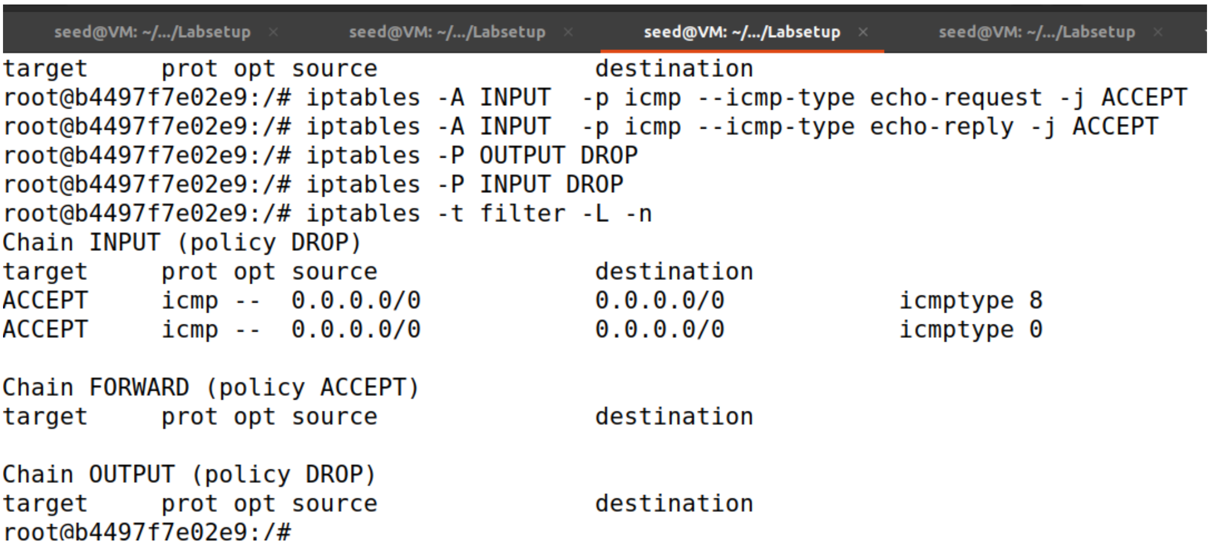


```
    link/ether 02:42:0a:09:00:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.11/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
29: eth1@if30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default
    link/ether 02:42:c0:a8:3c:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.60.11/24 brd 192.168.60.255 scope global eth1
       valid_lft forever preferred_lft forever
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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



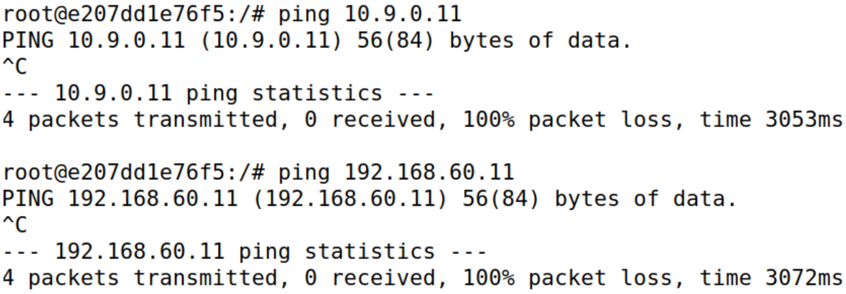
```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@e207dd1e76f5:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
54 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.069 ms
54 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.044 ms
54 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.054 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 0.044/0.055/0.069/0.010 ms
root@e207dd1e76f5:/#
root@e207dd1e76f5:/#
root@e207dd1e76f5:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
54 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.049 ms
54 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.041 ms
54 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.046 ms
^C
--- 192.168.60.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.041/0.045/0.049/0.003 ms
```

Applying the Required rules:



```
target     prot opt source               destination
root@b4497f7e02e9:/# iptables -A INPUT  -p icmp --icmp-type echo-request -j ACCEPT
root@b4497f7e02e9:/# iptables -A INPUT  -p icmp --icmp-type echo-reply -j ACCEPT
root@b4497f7e02e9:/# iptables -P OUTPUT DROP
root@b4497f7e02e9:/# iptables -P INPUT DROP
root@b4497f7e02e9:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
root@b4497f7e02e9:/#
```

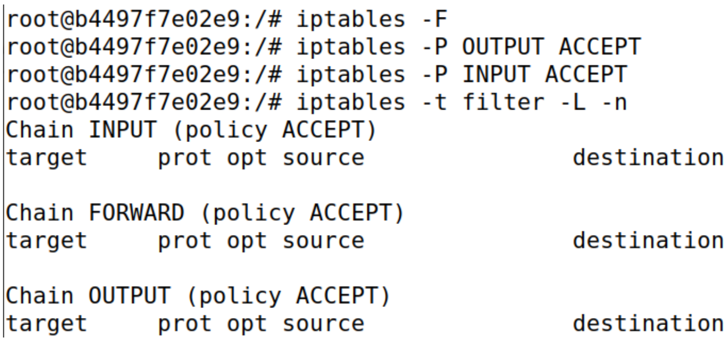
- iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT : The following command accepts the echo requests as ping requests from ping and block all other machines.
- iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT For this command, we also accept the ping reply to the user.
- ‘iptables -P OUTPUT DROP’, ‘iptables -P INPUT DROP’ : e following two commands are set to default for Output and Input which means other protocols will not have any access to the router.



```
root@e207dd1e76f5:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3053ms

root@e207dd1e76f5:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
^C
--- 192.168.60.11 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3072ms
```

As seen above, we were not able receive back anything as we did before applying the rules, hence we can say it was successful.

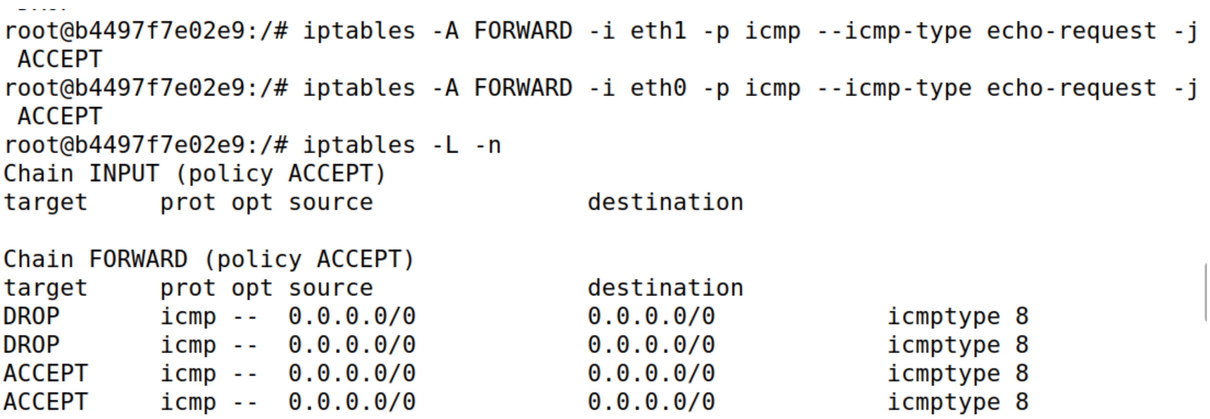
Restoring the iptables to its original state:



```
root@b4497f7e02e9:/# iptables -F
root@b4497f7e02e9:/# iptables -P OUTPUT ACCEPT
root@b4497f7e02e9:/# iptables -P INPUT ACCEPT
root@b4497f7e02e9:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

2B

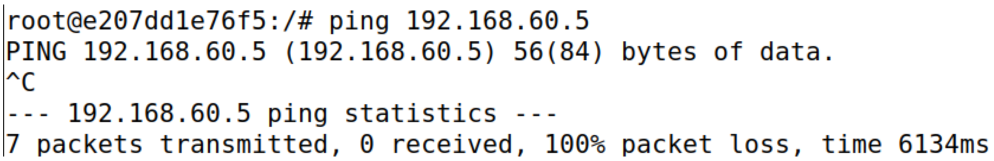
Applying the required rules and checking the ip table:



```
root@b4497f7e02e9:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j
 ACCEPT
root@b4497f7e02e9:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j
 ACCEPT
root@b4497f7e02e9:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

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

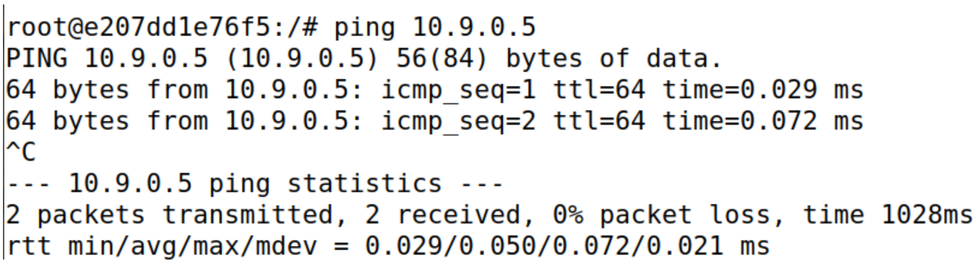
we can see that the outside hosts will not be able to ping the inside hosts because there will be a drop of the packets and a rule which accepts the packets sent outside from the internal host and vice versa.

Outside host cannot ping internal host:



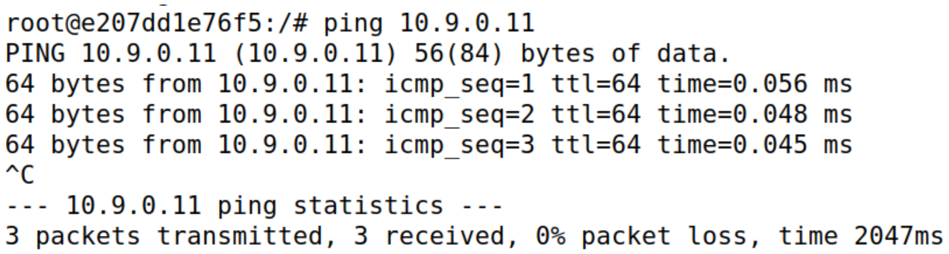
```
root@e207dd1e76f5:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6134ms
```

Internal host can ping the outside host:



```
root@e207dd1e76f5:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.072 ms
^C
--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1028ms
rtt min/avg/max/mdev = 0.029/0.050/0.072/0.021 ms
```

Outside host can ping the router:

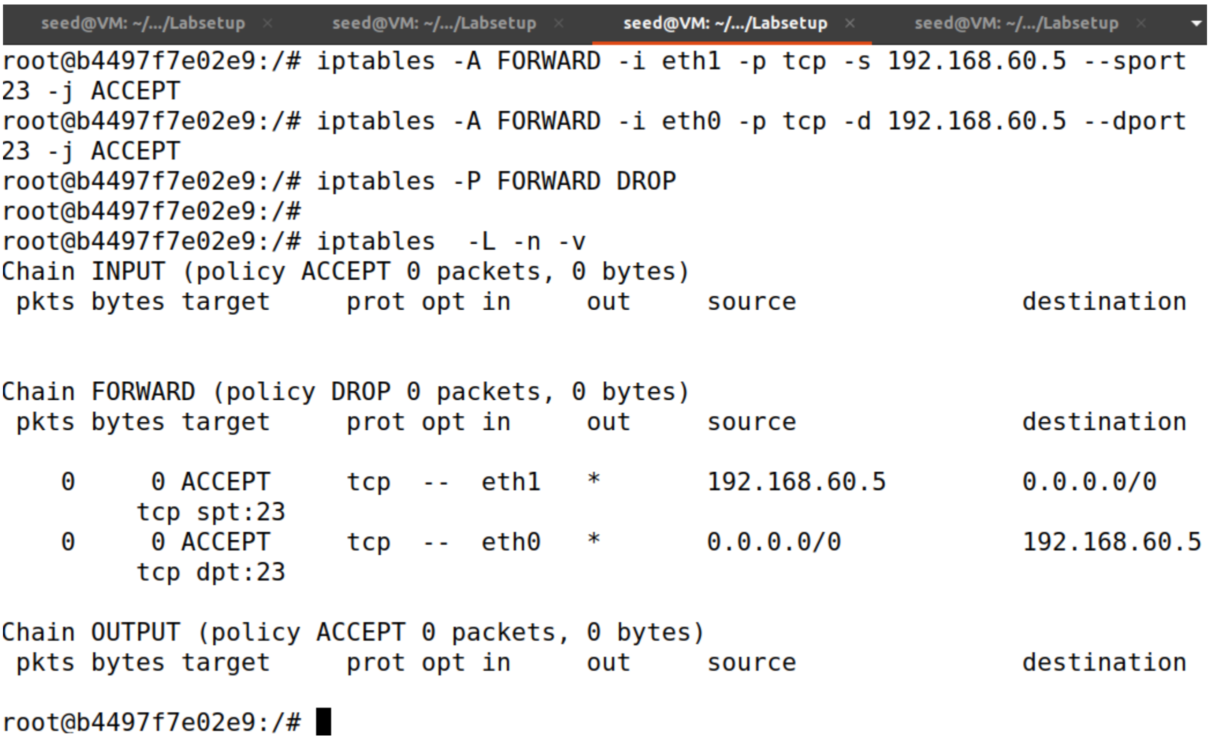


```
root@e207dd1e76f5:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.045 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2047ms
```

Any other packets between the internal and external networks are blocked too.

2C Protecting internal servers

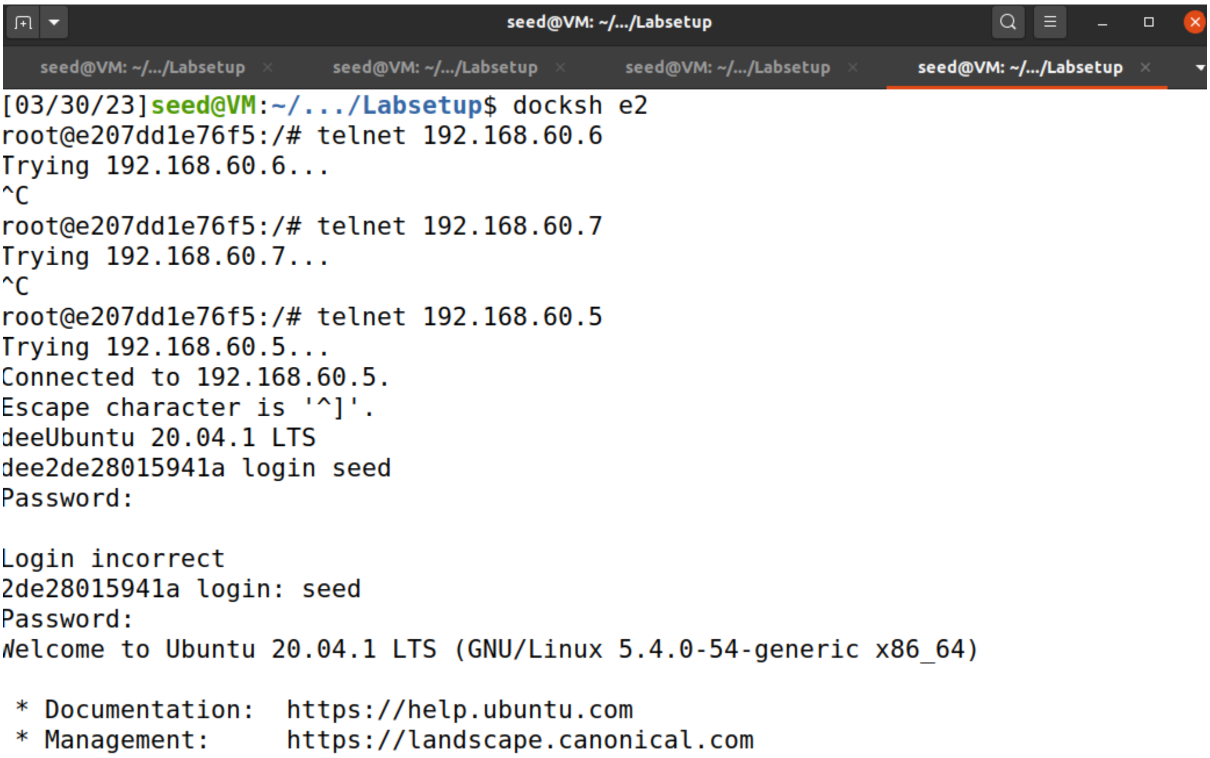
Applying the rules:



```
root@b4497f7e02e9:/# iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 --sport
23 -j ACCEPT
root@b4497f7e02e9:/# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport
23 -j ACCEPT
root@b4497f7e02e9:/# iptables -P FORWARD DROP
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables  -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 ACCEPT     tcp  --  eth1   *       192.168.60.5         0.0.0.0/0
         tcp spt:23
    0     0 ACCEPT     tcp  --  eth0   *       0.0.0.0/0            192.168.60.5
         tcp dpt:23

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

root@b4497f7e02e9:/# █
```

Only requests on the dest IP 192.168.60.5 and dport 23 are accepted by Eth0, whereas Eth1 accepts requests from inside to outside.

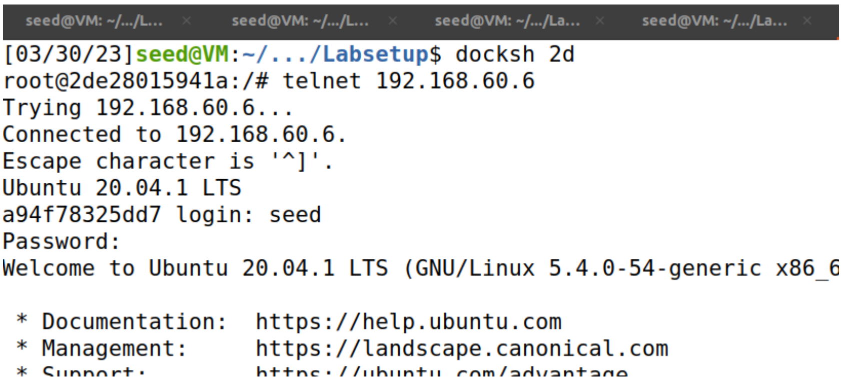
The third rule specifies that we drop any other packets not intended to the IP or port specified in the above rules.



```
[03/30/23]seed@VM:~/.../Labsetup$ docksh e2
root@e207dd1e76f5:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@e207dd1e76f5:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@e207dd1e76f5:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
deeUbuntu 20.04.1 LTS
dee2de28015941a login seed
Password:

Login incorrect
2de28015941a 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
```

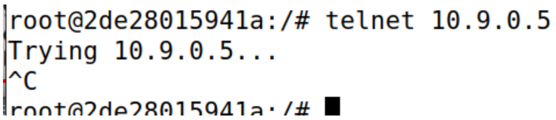
We try to telnet external network servers (0.6,0.7,0.5) As seen above only connection to 192.168.60.5 is successful, the other two wont work as specified by our rules.

When tried to connect an internal server from an internal server it should be working as follows:



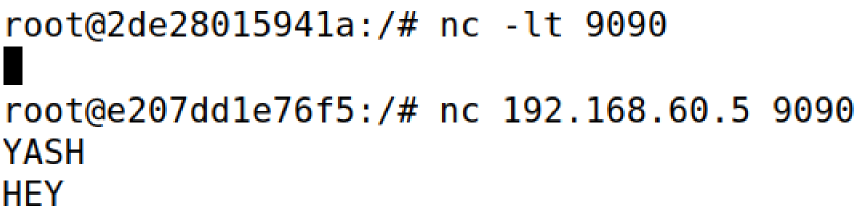
```
[03/30/23]seed@VM:~/.../Labsetup$ docksh 2d
root@2de28015941a:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a94f78325dd7 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_6

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

But when we try to telnet a external server from an internal server ot wont work:



```
root@2de28015941a:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@2de28015941a:/# 
```

Trying to connect external host and internal host on a port number and we see that's also not possible according to the rules:



```
root@2de28015941a:/# nc -lt 9090
█
root@e207dd1e76f5:/# nc 192.168.60.5 9090
YASH
HEY
```

Restoring the iptables to the original state:

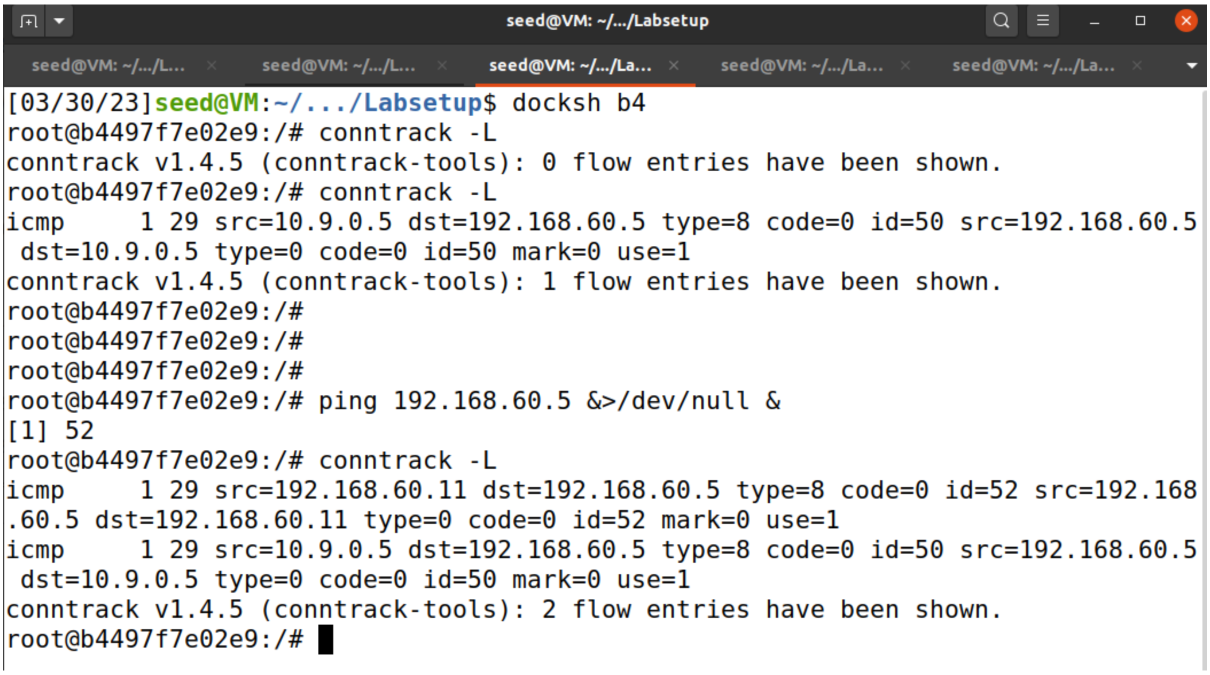
```
root@b4497f7e02e9:/# iptables -F
root@b4497f7e02e9:/# iptables -P FORWARD ACCEPT
root@b4497f7e02e9:/# iptables  -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Task 3: Connection Tracking and Stateful Firewall

3A

Experiment with connection tracking:

ICMP EXPERIMENT:

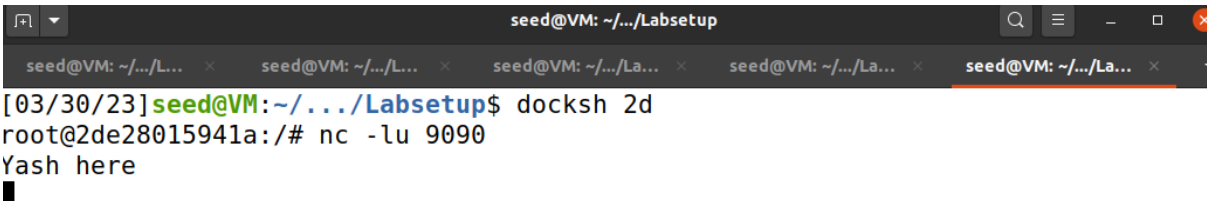


```
[03/30/23]seed@VM:~/.../Labsetup$ docksh b4
root@b4497f7e02e9:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@b4497f7e02e9:/# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5
 dst=10.9.0.5 type=0 code=0 id=50 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# ping 192.168.60.5 &>/dev/null &
[1] 52
root@b4497f7e02e9:/# conntrack -L
icmp     1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=52 src=192.168
.60.5 dst=192.168.60.11 type=0 code=0 id=52 mark=0 use=1
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5
 dst=10.9.0.5 type=0 code=0 id=50 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
root@b4497f7e02e9:/# █
```

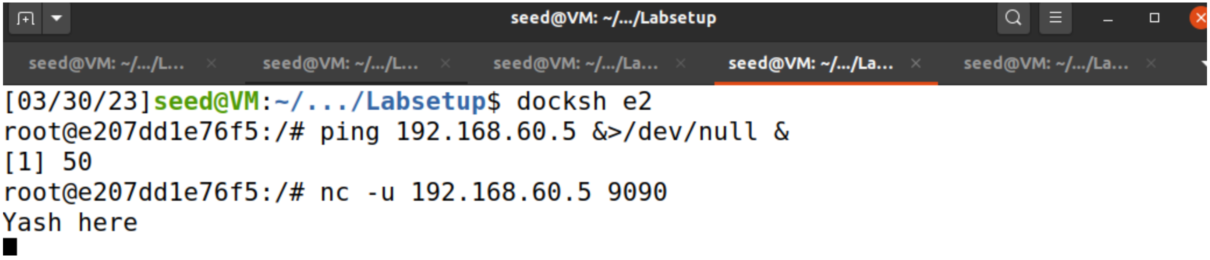
On the server router we can use the command specified in the text to gain connection tracking information. For ICMP the connection stays for about 4-5 seconds for each packet snet.

UDP EXPERIMENT:

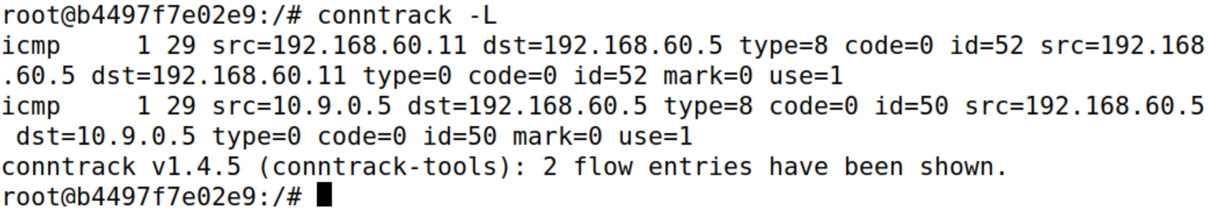
For the UDP, we need to create a netcat UDP server:



```
[03/30/23]seed@VM:~/.../Labsetup$ docksh 2d
root@2de28015941a:/# nc -lu 9090
Yash here
```



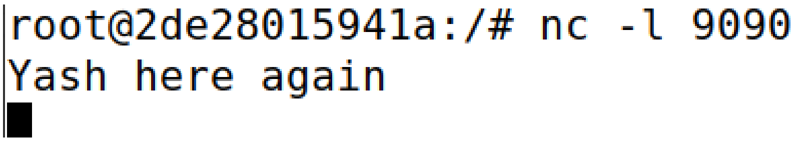
```
[03/30/23]seed@VM:~/.../Labsetup$ docksh e2
root@e207dd1e76f5:/# ping 192.168.60.5 &>/dev/null &
[1] 50
root@e207dd1e76f5:/# nc -u 192.168.60.5 9090
Yash here
```



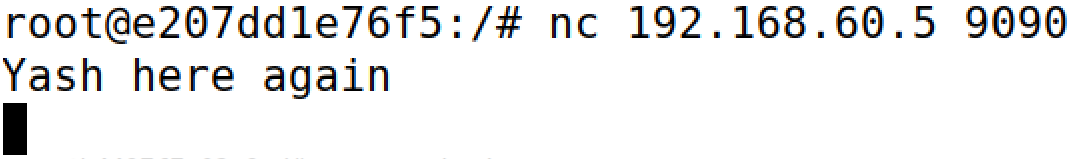
```
root@b4497f7e02e9:/# conntrack -L
icmp     1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=52 src=192.168
.60.5 dst=192.168.60.11 type=0 code=0 id=52 mark=0 use=1
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5
 dst=10.9.0.5 type=0 code=0 id=50 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
root@b4497f7e02e9:/# 
```

We keep on tab with the conntrack-L command. And by observation we can say that the UDP connections stays for around 7-8 seconds.

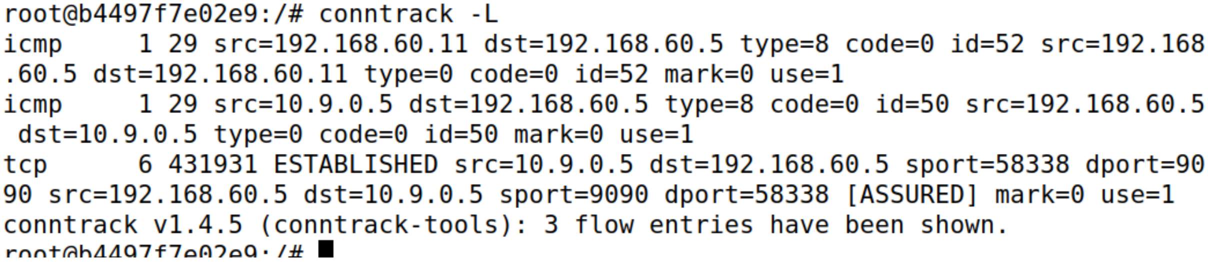
TCP EXPERIMENT:



```
root@2de28015941a:/# nc -l 9090
Yash here again
```



```
root@e207dd1e76f5:/# nc 192.168.60.5 9090
Yash here again
```

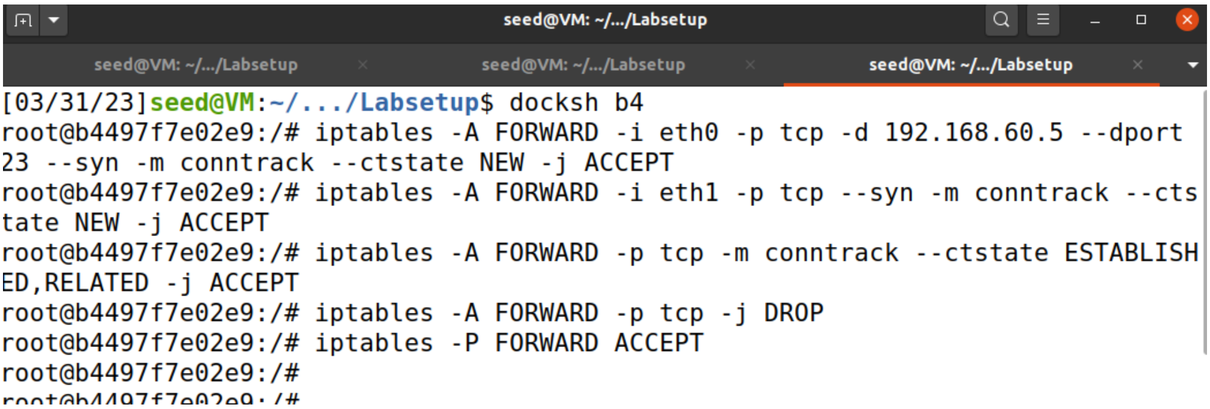


```
root@b4497f7e02e9:/# conntrack -L
icmp     1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=52 src=192.168
.60.5 dst=192.168.60.11 type=0 code=0 id=52 mark=0 use=1
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5
 dst=10.9.0.5 type=0 code=0 id=50 mark=0 use=1
tcp      6 431931 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=58338 dport=90
90 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=58338 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 3 flow entries have been shown.
root@b4497f7e02e9:/#
```

We continue this command and we can note that the connection stays for more than a minute.

Task 3.B: Setting Up a Stateful Firewall

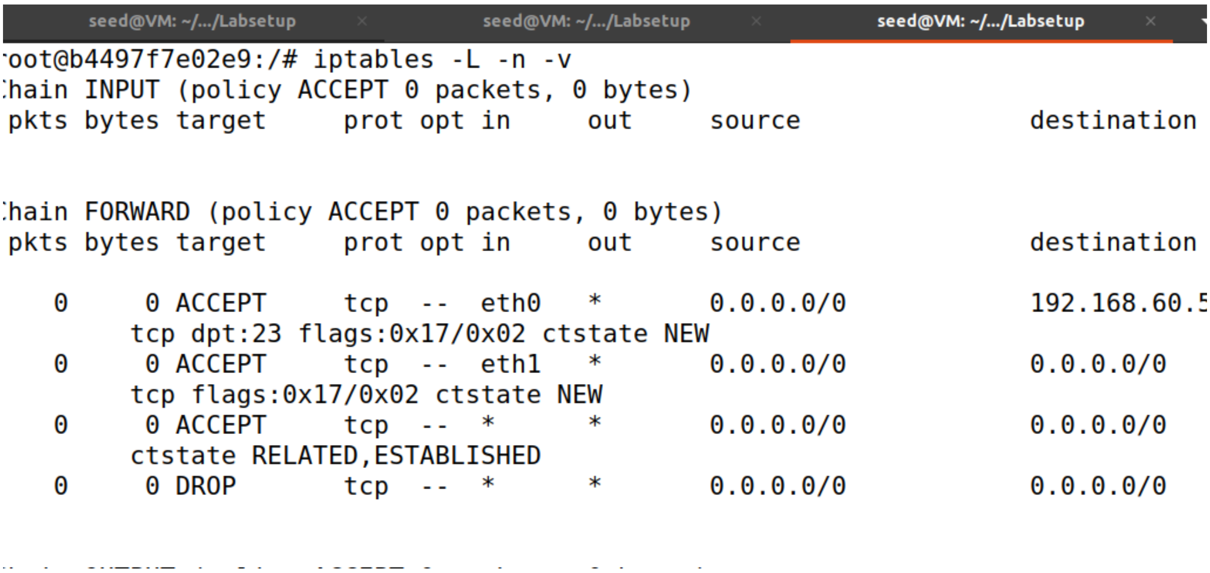
Applying the rules:



```
[03/31/23]seed@VM:~/.../Labsetup$ docksh b4
root@b4497f7e02e9:/# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport
23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@b4497f7e02e9:/# iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --cts
tate NEW -j ACCEPT
root@b4497f7e02e9:/# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISH
ED,RELATED -j ACCEPT
root@b4497f7e02e9:/# iptables -A FORWARD -p tcp -j DROP
root@b4497f7e02e9:/# iptables -P FORWARD ACCEPT
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
```

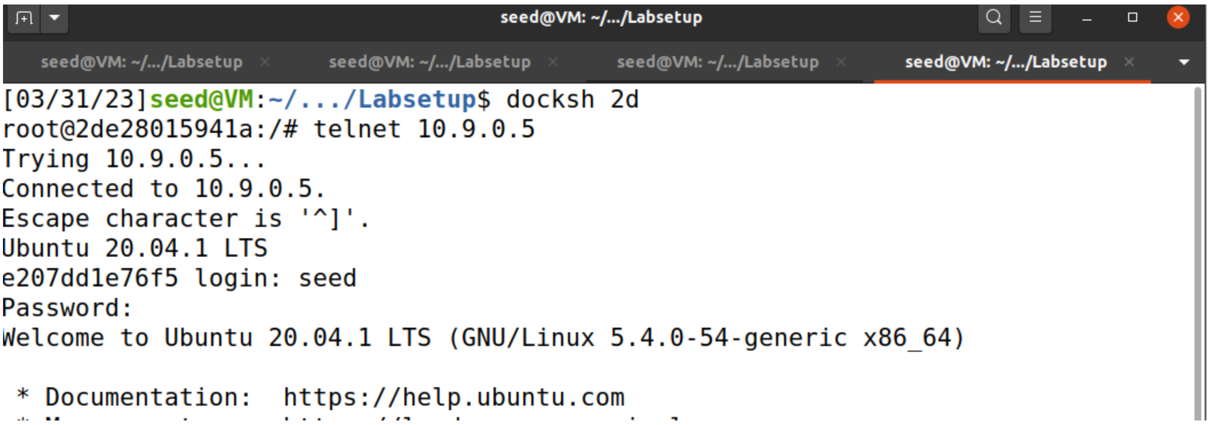
The new sync packet that was used to connect to TCP will be accepted by us. No matter which connection they originate from, as long as they adhere to the established and related rule, WE want to accept the TCP packets that belong to the established and associated connections. Any additional connections will be lost.

The iptable after this:



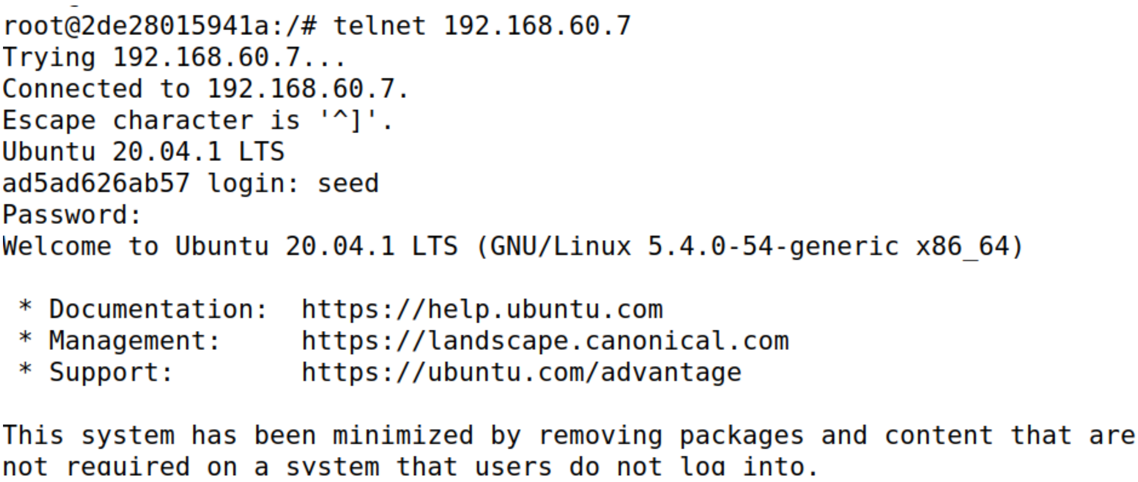
```
oot@b4497f7e02e9:/# iptables -L -n -v
hain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination


hain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

   0     0 ACCEPT     tcp  --  eth0   *       0.0.0.0/0            192.168.60.5
        tcp dpt:23 flags:0x17/0x02 ctstate NEW
   0     0 ACCEPT     tcp  --  eth1   *       0.0.0.0/0            0.0.0.0/0
        tcp flags:0x17/0x02 ctstate NEW
   0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
        ctstate RELATED,ESTABLISHED
   0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
```

Trying to telnet external host 10.9.0.5:



```
[03/31/23]seed@VM:~/.../Labsetup$ docksh 2d
root@2de28015941a:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e207dd1e76f5 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
```

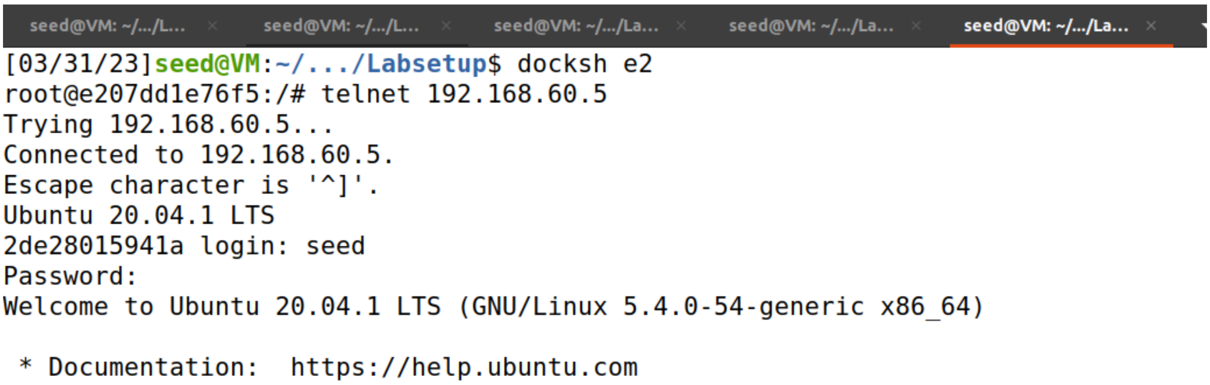
We can see that a successful connection has been established.



```
root@2de28015941a:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
ad5ad626ab57 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.
```

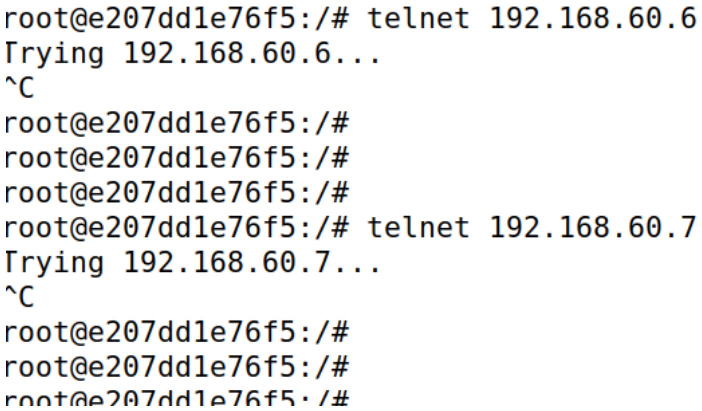
We can see this connection also gets established successfully.

From an external host to an internal host



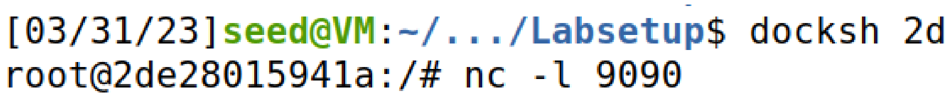
```
[03/31/23]seed@VM:~/.../Labsetup$ docksh e2
root@e207dd1e76f5:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2de28015941a login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
```

We try to telnet internal server 192.168.60.6 and 192.168.60.7 and as in the rules it isn't allowed:

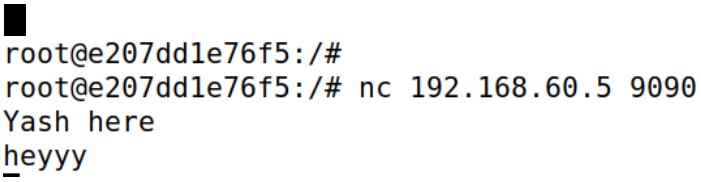


```
root@e207dd1e76f5:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@e207dd1e76f5:/#
root@e207dd1e76f5:/#
root@e207dd1e76f5:/#
root@e207dd1e76f5:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@e207dd1e76f5:/#
root@e207dd1e76f5:/#
root@e207dd1e76f5:/#
```

Trying to connect, On the outside host to connect on the internal server through the netcap TCP server, which is not allowed:



```
[03/31/23]seed@VM:~/.../Labsetup$ docksh 2d
root@2de28015941a:/# nc -l 9090
```



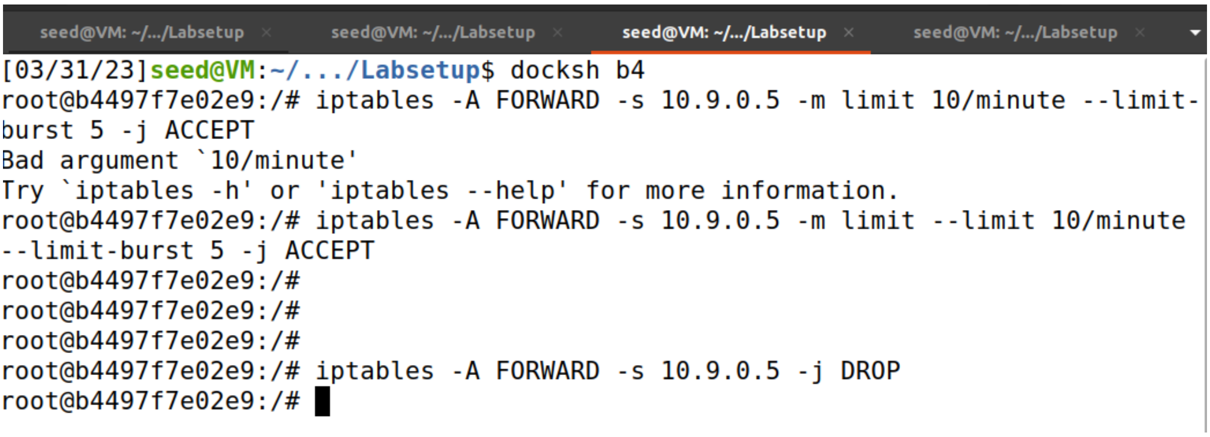
```
root@e207dd1e76f5:/#
root@e207dd1e76f5:/# nc 192.168.60.5 9090
Yash here
heyyy
```

According to a set of rules with and without a connection tracking mechanism, the connection-based system has the drawback of using more resources because the connection must be maintained, but this is not the case for the rules without a connection tracking mechanism.

The constraints are less stringent for the mechanism without connection tracking, making it less secure than the one with connection tracking. This is not the case with connection tracking-based systems since, as we have seen above, the criteria are rigorous and only TCP connections with established and associated connections receive replies.

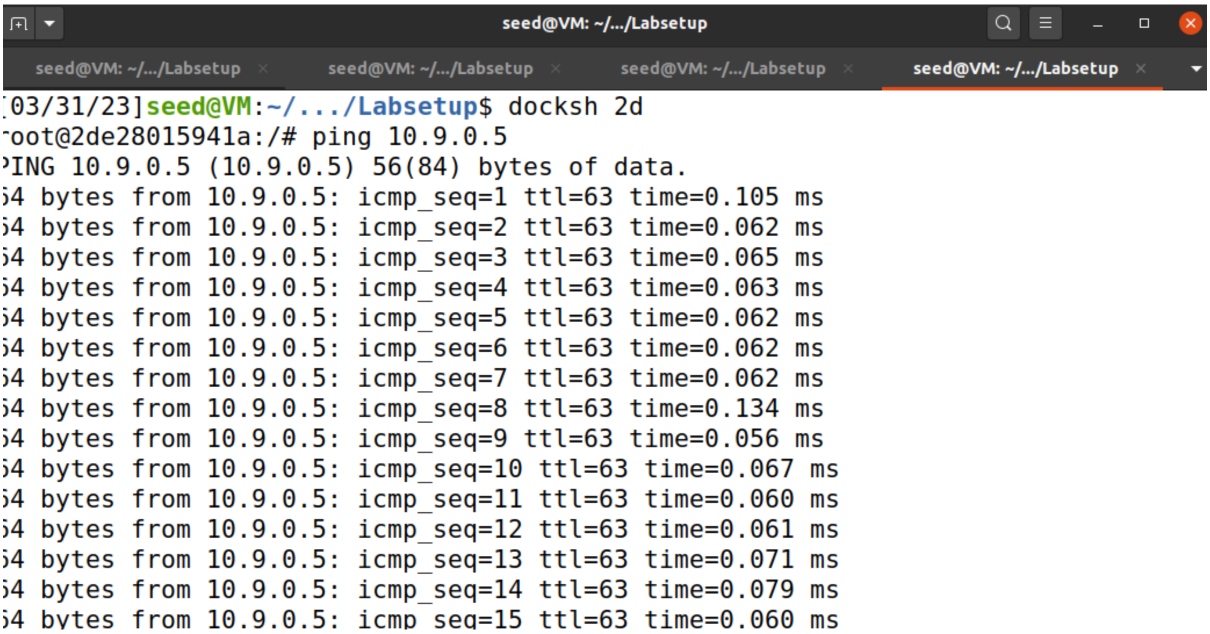
Task 4: Limiting Network Traffic

Applying the required rule:



```
[03/31/23]seed@VM:~/.../Labsetup$ docksh b4
root@b4497f7e02e9:/# iptables -A FORWARD -s 10.9.0.5 -m limit 10/minute --limit-
burst 5 -j ACCEPT
Bad argument `10/minute'
Try `iptables -h' or 'iptables --help' for more information.
root@b4497f7e02e9:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute
--limit-burst 5 -j ACCEPT
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@b4497f7e02e9:/# 
```

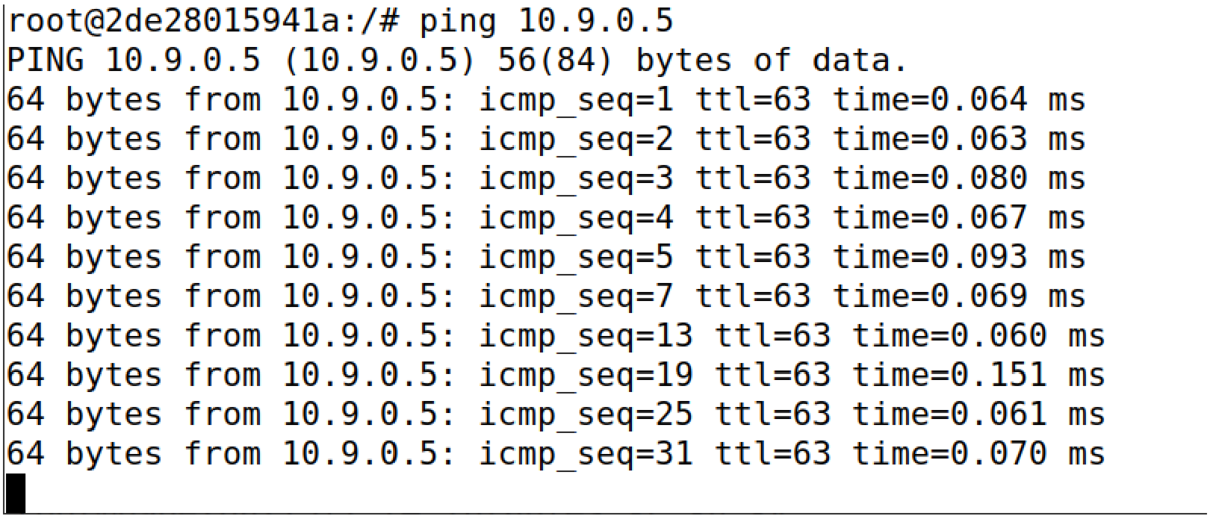
We ping 10.9.0.5 from 192.168.60.5



```
[03/31/23]seed@VM:~/.../Labsetup$ docksh 2d
root@2de28015941a:/# 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=0.105 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.062 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.063 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.062 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.062 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.062 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.134 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=63 time=0.056 ms
64 bytes from 10.9.0.5: icmp_seq=10 ttl=63 time=0.067 ms
64 bytes from 10.9.0.5: icmp_seq=11 ttl=63 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=12 ttl=63 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=63 time=0.071 ms
64 bytes from 10.9.0.5: icmp_seq=14 ttl=63 time=0.079 ms
64 bytes from 10.9.0.5: icmp_seq=15 ttl=63 time=0.060 ms
```

//This screenshot is before we add the "_A FORWARD -s 10.9.0.5 -j DROP"

After adding this command, we could observe that the count of packets transmitted and packets received is different and can be seen using the icmp_seq. If we compare it to the screenshot above the difference is noticeable



```
root@2de28015941a:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.064 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.063 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.080 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.067 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.093 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.069 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=63 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=19 ttl=63 time=0.151 ms
64 bytes from 10.9.0.5: icmp_seq=25 ttl=63 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=31 ttl=63 time=0.070 ms
```

(MY MISTAKE NOT TO TAKE THE FULL SCREENSHOT SHOWING THE PACKETS TRANSMITTED AND PCKETS RECEIVED)

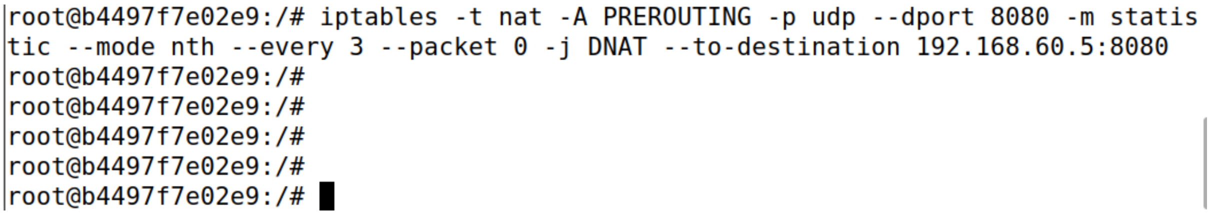
There is a time condition on the connection in the rule which is 10/ minute i.e 6 seconds for each packet to be sent.

Task 5: Load Balancing

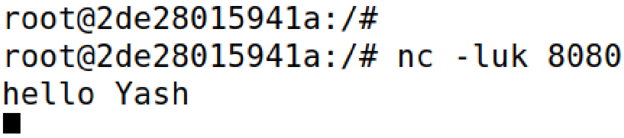
We have servers set up on external servers 192.168.60.5, 192.168.60.6 and 192.168.60.7

Applying the rules:

When we use the following command to send a UDP packet from an external host to the router's 8080 port, we can see that only one out of every three packets reach 192.168.60.5

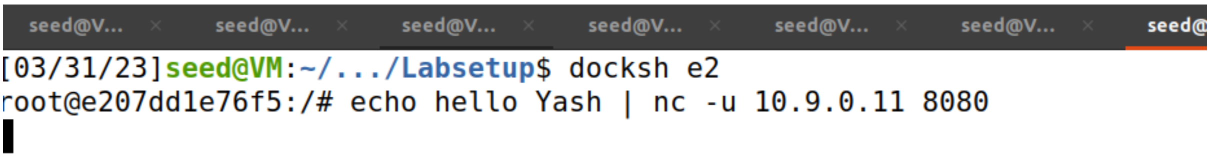


```
root@b4497f7e02e9:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
```



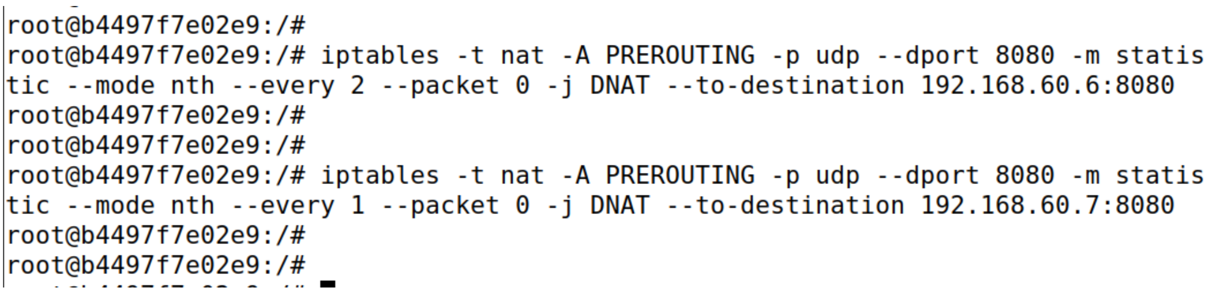
```
root@2de28015941a:/#
root@2de28015941a:/# nc -luk 8080
hello Yash
```

It dispatches the packets coming to 8080 to the inner hosts.



```
[03/31/23]seed@VM:~/.../Labsetup$ docksh e2
root@e207dd1e76f5:/# echo hello Yash | nc -u 10.9.0.11 8080
```

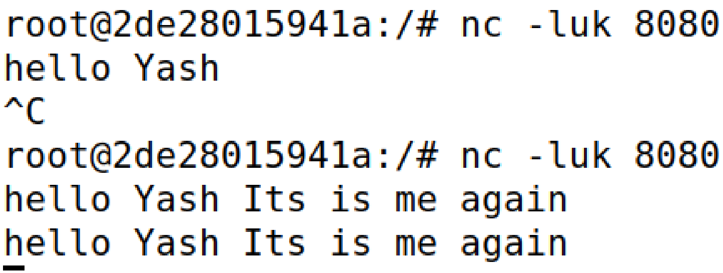
The other rules for the rest of the servers are also being added:



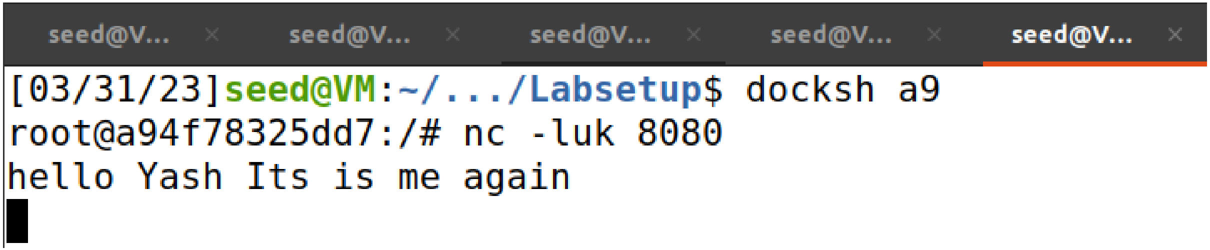
```
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 1 --packet 0 -j DNAT --to-destination 192.168.60.7:8080
root@b4497f7e02e9:/#
root@b4497f7e02e9:/#
```

These rules are added so that all internal servers get equal number of packets.

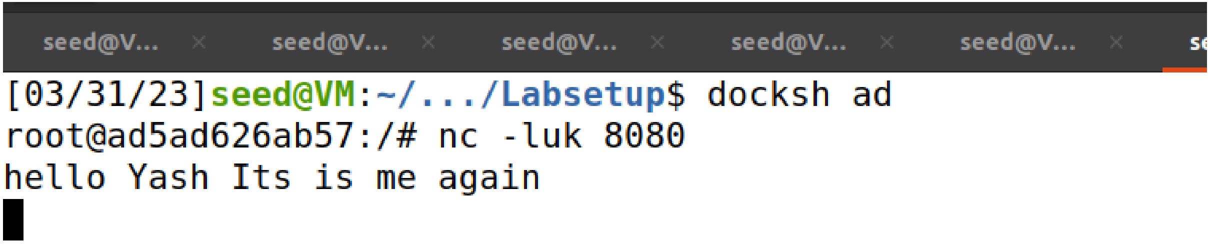
Echoing the message “hello Yash Its me again” from the server router



```
root@2de28015941a:/# nc -luk 8080
hello Yash
^C
root@2de28015941a:/# nc -luk 8080
hello Yash Its is me again
hello Yash Its is me again
```



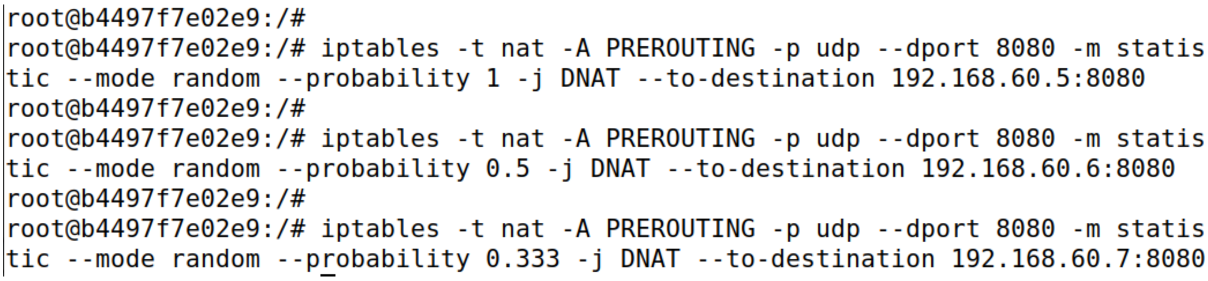
```
[03/31/23]seed@VM:~/.../Labsetup$ docksh a9
root@a94f78325dd7:/# nc -luk 8080
hello Yash Its is me again
```



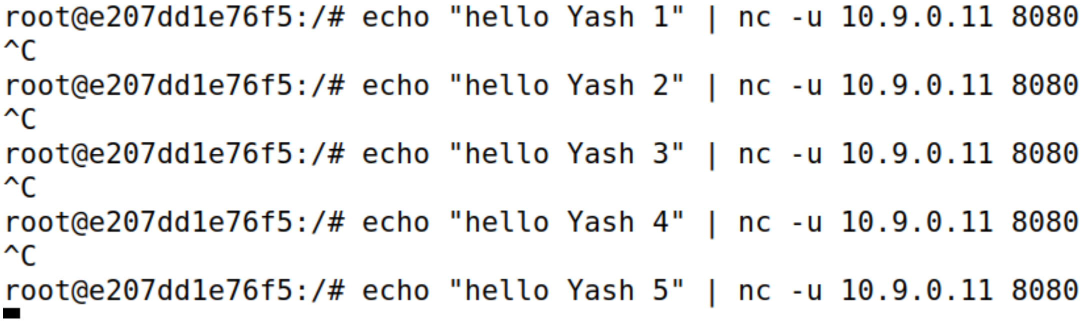
```
[03/31/23]seed@VM:~/.../Labsetup$ docksh ad
root@ad5ad626ab57:/# nc -luk 8080
hello Yash Its is me again
```

USING THE RANDOM MODE:

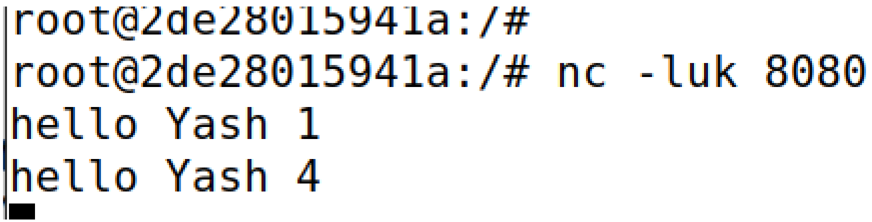
We have changed the rules according to the c=given condition for the random mode for achieving load balancing with the random probabilities assigned to the packets for each server.



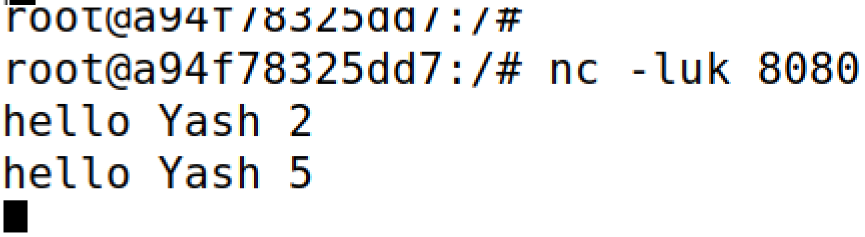
```
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 1 -j DNAT --to-destination 192.168.60.5:8080
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.5 -j DNAT --to-destination 192.168.60.6:8080
root@b4497f7e02e9:/#
root@b4497f7e02e9:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.333 -j DNAT --to-destination 192.168.60.7:8080
```



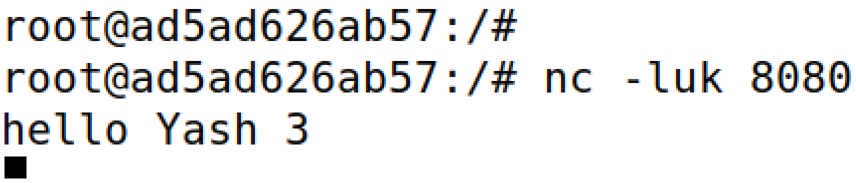
```
root@e207dd1e76f5:/# echo "hello Yash 1" | nc -u 10.9.0.11 8080
^C
root@e207dd1e76f5:/# echo "hello Yash 2" | nc -u 10.9.0.11 8080
^C
root@e207dd1e76f5:/# echo "hello Yash 3" | nc -u 10.9.0.11 8080
^C
root@e207dd1e76f5:/# echo "hello Yash 4" | nc -u 10.9.0.11 8080
^C
root@e207dd1e76f5:/# echo "hello Yash 5" | nc -u 10.9.0.11 8080
```



```
root@2de28015941a:/#
root@2de28015941a:/# nc -luk 8080
hello Yash 1
hello Yash 4
```



```
root@a94f78325dd7:/#
root@a94f78325dd7:/# nc -luk 8080
hello Yash 2
hello Yash 5
```



```
root@ad5ad626ab57:/#
root@ad5ad626ab57:/# nc -luk 8080
hello Yash 3
```

Based on the probabilities packets have been sent to the respective hosts.