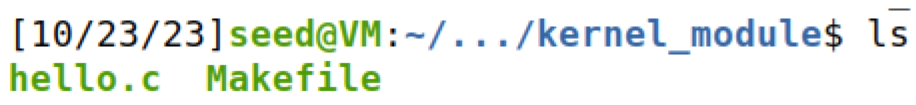
Lab 5

Student ID: 1005288

Task 1

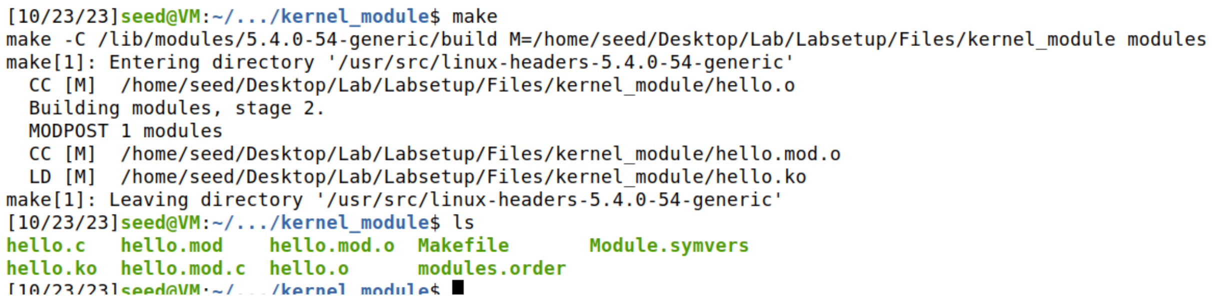
Part A

We basically get a .c file and a Makefile as our starting point



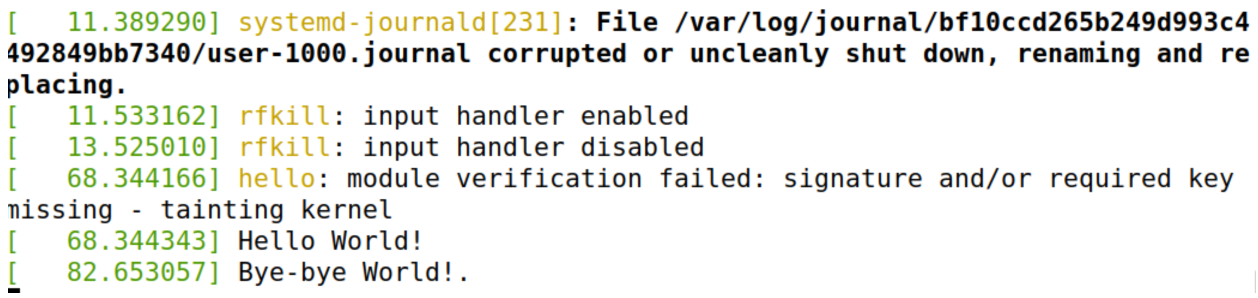
```
[10/23/23]seed@VM:~/.../kernel_module$ ls
hello.c  Makefile
```

I ran make to get the files, looking out for a hello.ko file, that file will be loaded into the kernel



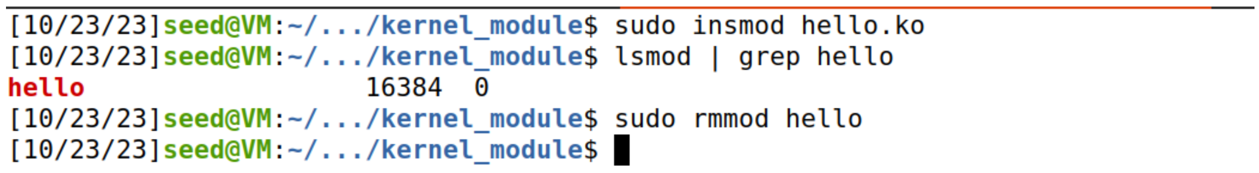
```
[10/23/23]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Lab/Labsetup/Files/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Lab/Labsetup/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Lab/Labsetup/Files/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/Lab/Labsetup/Files/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[10/23/23]seed@VM:~/.../kernel_module$ ls
hello.c   hello.mod    hello.mod.o  Makefile       Module.symvers
hello.ko  hello.mod.c  hello.o      modules.order
[10/23/23]seed@VM:~/.../kernel_module$
```

I used dmesg -w on a terminal window to get the kernel buffer to wait for new messages
After running sudo insmod hello.ko, we can see the message "Hello World!"
sudo rmmod hello removes the module, and displays the message "Bye-Bye world" in the kernel buffer



```
[   11.389290] systemd-journald[231]: File /var/log/journal/bf10ccd265b249d993c4
492849bb7340/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[   11.533162] rfkill: input handler enabled
[   13.525010] rfkill: input handler disabled
[   68.344166] hello: module verification failed: signature and/or required key
missing - tainting kernel
[   68.344343] Hello World!
[   82.653057] Bye-bye World!.
```

lsmod | grep hello shows modules with the name hello, can see that 1 module is loaded



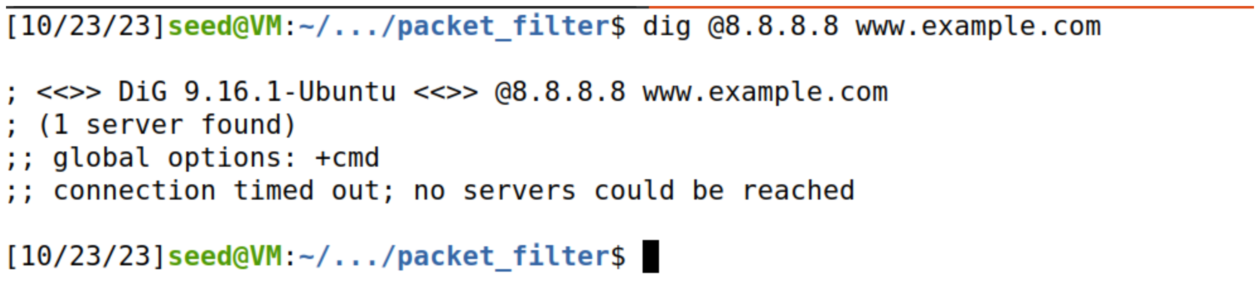
```
[10/23/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[10/23/23]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[10/23/23]seed@VM:~/.../kernel_module$ sudo rmmod hello
[10/23/23]seed@VM:~/.../kernel_module$ 
```

Having both the register and unregister print statements in the buffer implies that modules were loaded and unloaded successfully.

Part B

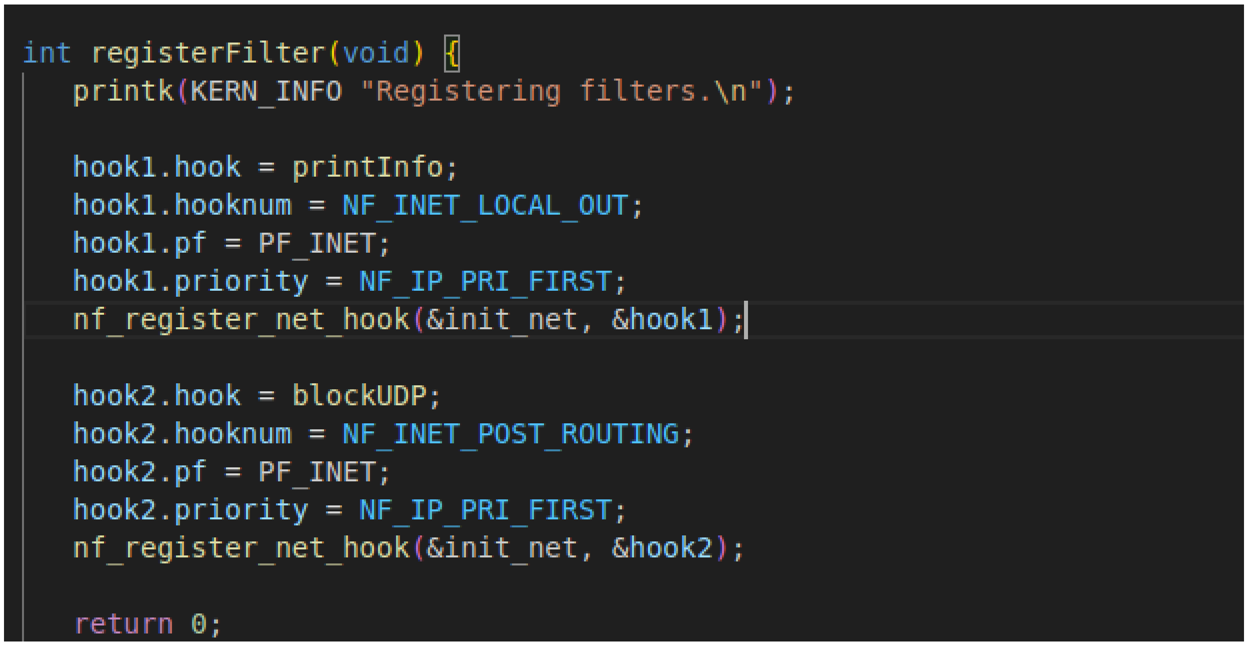
Part 1

I tested out the seedFilter.c file, the filter was working as seen that there is no route to www.example.com from 8.8.8.8 when we run the dig command.



```
[10/23/23]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

[10/23/23]seed@VM:~/.../packet_filter$ 
```

The function blockUDP was used on the NF_INET_POST_ROUTING hook, implying that the UDP packet was dropped after it left the user machine.



```
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);

   return 0;
```

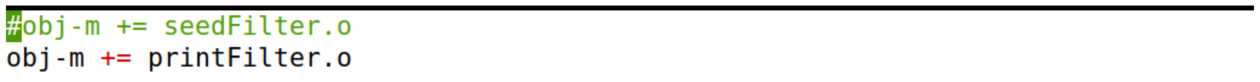
Hence the UDP packet did not reach 8.8.8.8, and not prompting a reply from www.example.com

Part 2

Using seedFilter.c as our foundation, I created another file called printFilter.c which

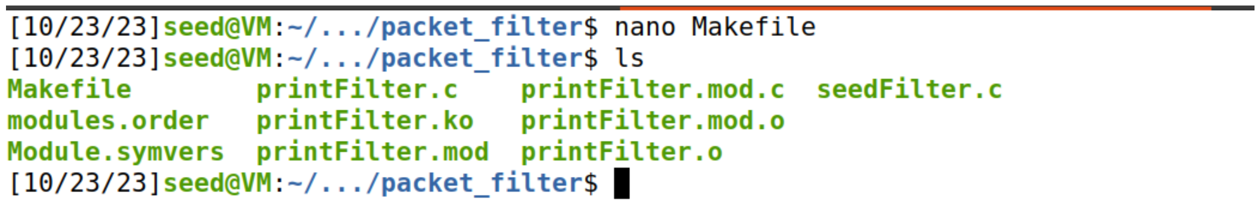
- instantiates the different hooks
- hooks the printInfo function to all instantiated hooks
- includes the hooks in the removal of filters

After the file is done, change the first line of Makefile to set printFilter.c as the target for make.



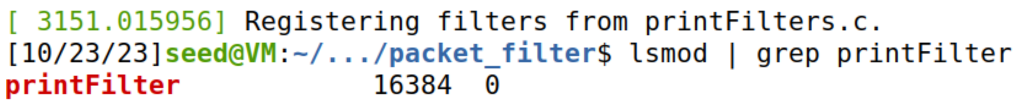
```
#obj-m += seedFilter.o
obj-m += printFilter.o
```

After running make, the printFilter.ko file was created.



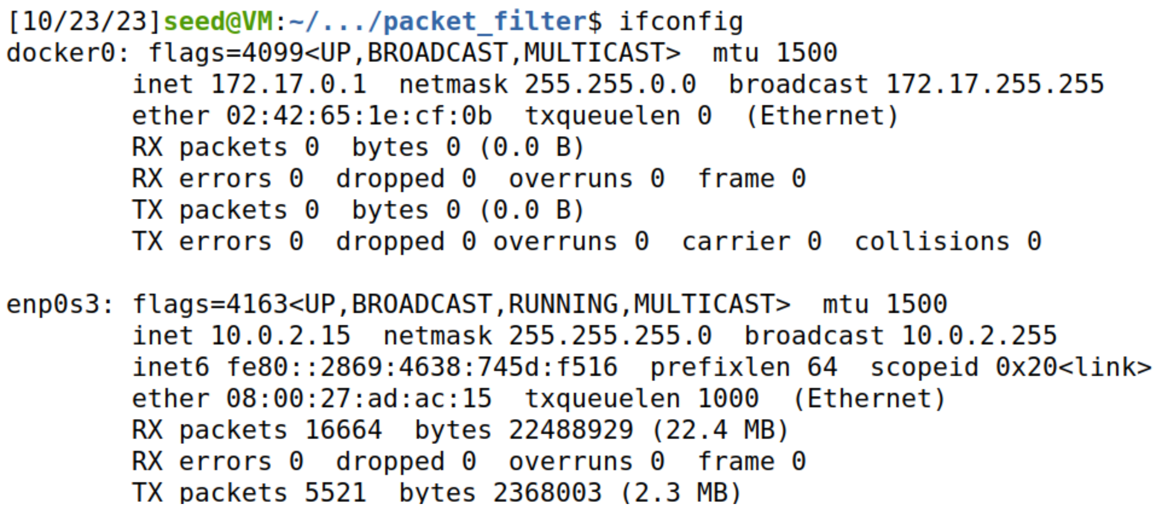
```
[10/23/23]seed@VM:~/.../packet_filter$ nano Makefile
[10/23/23]seed@VM:~/.../packet_filter$ ls
Makefile        printFilter.c    printFilter.mod.c  seedFilter.c
modules.order   printFilter.ko   printFilter.mod.o
Module.symvers  printFilter.mod  printFilter.o
[10/23/23]seed@VM:~/.../packet_filter$ 
```

After that, load printFilter.ko LKM into the kernel and check the lsmod output
We can see that the printFilter module was loaded



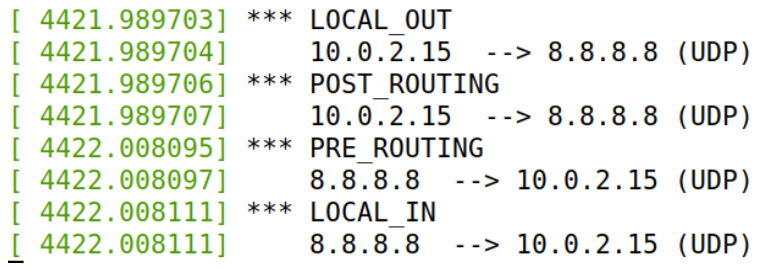
```
[ 3151.015956] Registering filters from printFilters.c.
[10/23/23]seed@VM:~/.../packet_filter$ lsmod | grep printFilter
printFilter            16384  0
```

Using ifconfig, we know that our machine is 10.0.2.15, which will be useful when we try to trigger the FORWARD hook.



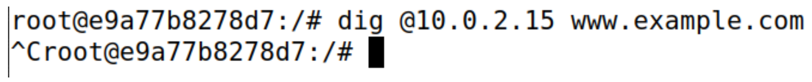
```
[10/23/23]seed@VM:~/.../packet_filter$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:65:1e:cf:0b  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::2869:4638:745d:f516  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:ad:ac:15  txqueuelen 1000  (Ethernet)
        RX packets 16664  bytes 22488929 (22.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5521  bytes 2368003 (2.3 MB)
```

NF_INET_PRE_ROUTING: any routing / modifications done before the packet reaches the input chain, which explains why this hook is before LOCAL_IN
NF_INET_LOCAL_IN: packets incoming to machine
NF_INET_LOCAL_OUT: packets outgoing from machine
NF_INET_POST_ROUTING: any routing / modifications done after the packet leaves the output chain, which is why the hook is after LOCAL_OUT

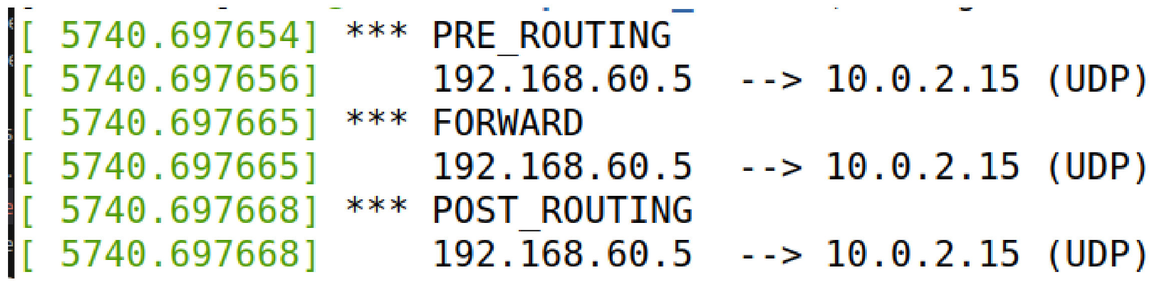


```
[ 4421.989703] *** LOCAL_OUT
[ 4421.989704]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 4421.989706] *** POST_ROUTING
[ 4421.989707]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 4422.008095] *** PRE_ROUTING
[ 4422.008097]     8.8.8.8  --> 10.0.2.15 (UDP)
[ 4422.008111] *** LOCAL_IN
[ 4422.008111]     8.8.8.8  --> 10.0.2.15 (UDP)
```

NF_INET_FORWARD: triggered when the machine is a forwarder instead, as shown when I ran dig to www.example.com through the user (10.0.2.15), we see a FORWARD hook being triggered.



```
root@e9a77b8278d7:/# dig @10.0.2.15 www.example.com
^Croot@e9a77b8278d7:/# 
```



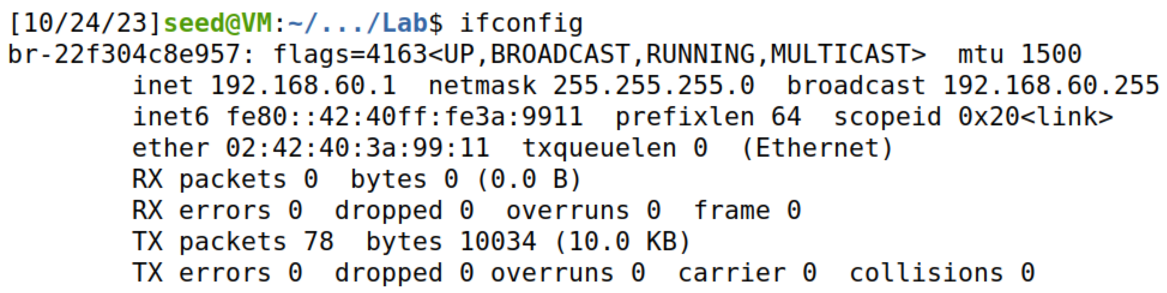
```
[ 5740.697654] *** PRE_ROUTING
[ 5740.697656]     192.168.60.5  --> 10.0.2.15 (UDP)
[ 5740.697665] *** FORWARD
[ 5740.697665]     192.168.60.5  --> 10.0.2.15 (UDP)
[ 5740.697668] *** POST_ROUTING
[ 5740.697668]     192.168.60.5  --> 10.0.2.15 (UDP)
```

Part 3

Preparation:

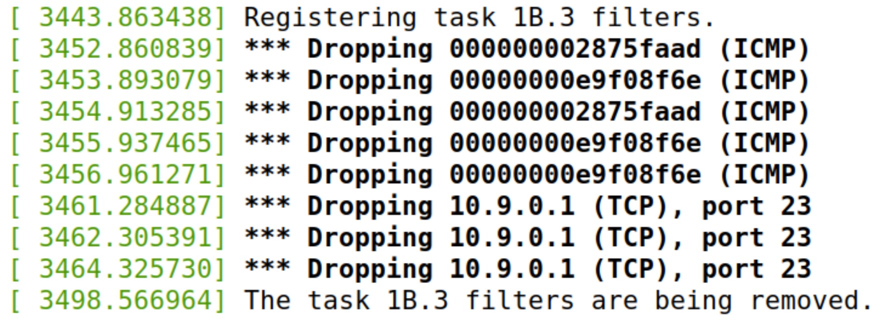
HostA was used as a container name in lab1, so there was an error building the container. I needed to do a docker container prune to remove the previous image.

After running dcup, the machine (10.9.0.1) was part of a subnet with hostA (10.9.0.5), since ping uses ICMP and telnet uses port 23, we will need to modify the code to filter out any ICMP and telnet packets. Hooks: LOCAL_IN, dropping ICMP and TCP packets on LOCAL_IN hook, saves trouble to generate a reply and handshake. (implementation is in prevent_ping_telnet.c)

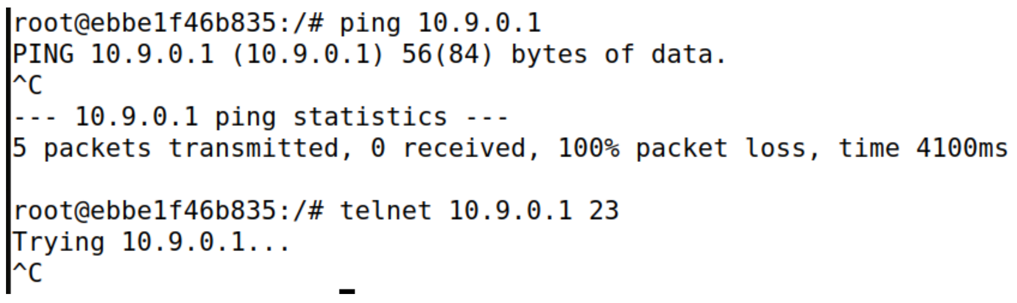


```
[10/24/23]seed@VM:~/.../Lab$ ifconfig
br-22f304c8e957: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.1  netmask 255.255.255.0  broadcast 192.168.60.255
        inet6 fe80::42:40ff:fe3a:9911  prefixlen 64  scopeid 0x20<link>
        ether 02:42:40:3a:99:11  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 78  bytes 10034 (10.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-d381a4e4b88a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:26ff:fe35:74a6  prefixlen 64  scopeid 0x20<link>
        ether 02:42:26:35:74:a6  txqueuelen 0  (Ethernet)
        RX packets 199  bytes 11685 (11.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 232  bytes 22532 (22.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

I followed the same procedure to generate a .ko file by changing the Makefile to set prevent_ping_telnet.c as the target for make
Then after loading into the kernel, we can see that hostA tried to ping 10.9.0.1 and telnet 10.9.0.1 in port 23 but failed.
As seen below, all packets were dropped by the module and hostA did not receive any replies from 10.9.0.1. Hence, the blocking was a success



```
[ 3443.863438] Registering task 1B.3 filters.
[ 3452.860839] *** Dropping 000000002875faad (ICMP)
[ 3453.893079] *** Dropping 00000000e9f08f6e (ICMP)
[ 3454.913285] *** Dropping 000000002875faad (ICMP)
[ 3455.937465] *** Dropping 00000000e9f08f6e (ICMP)
[ 3456.961271] *** Dropping 00000000e9f08f6e (ICMP)
[ 3461.284887] *** Dropping 10.9.0.1 (TCP), port 23
[ 3462.305391] *** Dropping 10.9.0.1 (TCP), port 23
[ 3464.325730] *** Dropping 10.9.0.1 (TCP), port 23
[ 3498.566964] The task 1B.3 filters are being removed.
```

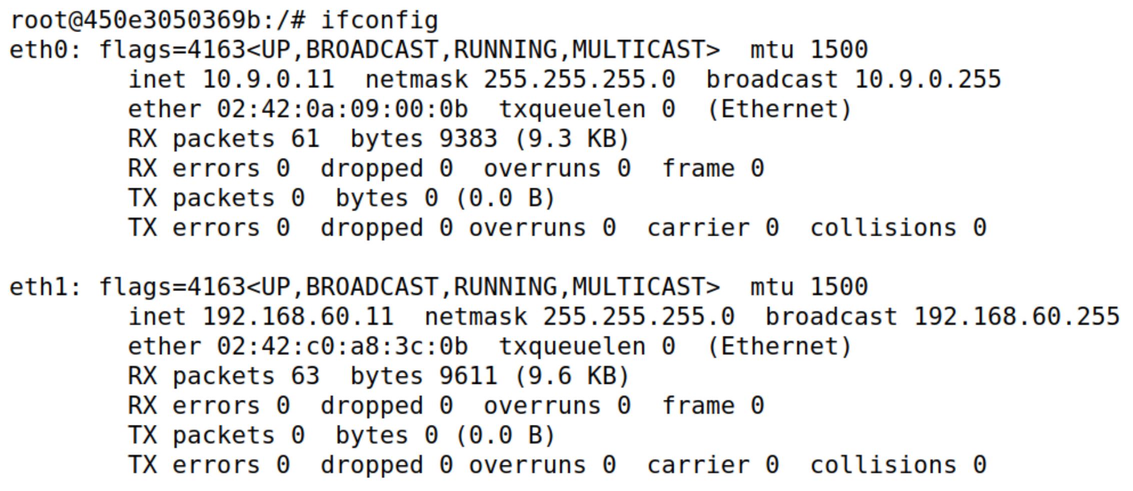


```
root@ebbe1f46b835:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4100ms

root@ebbe1f46b835:/# telnet 10.9.0.1 23
Trying 10.9.0.1...
^C
```

Task 2

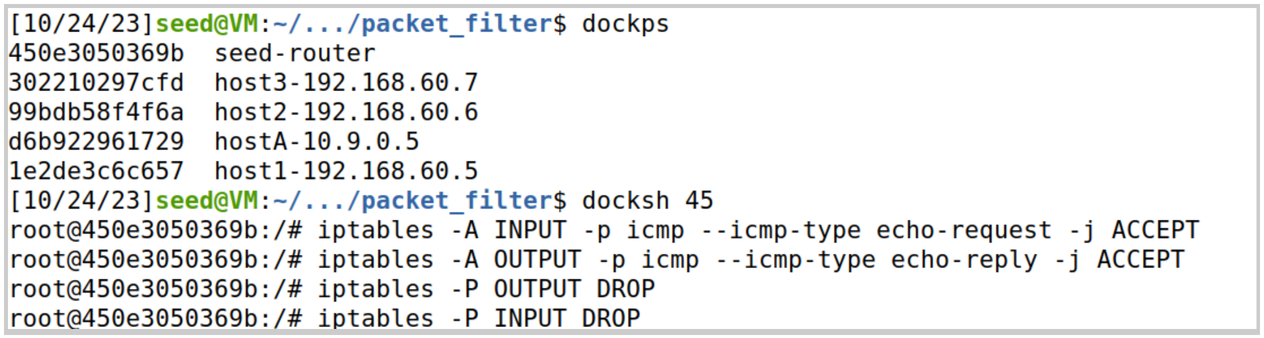
Part A

run ifconfig in the router container to know what IP it is



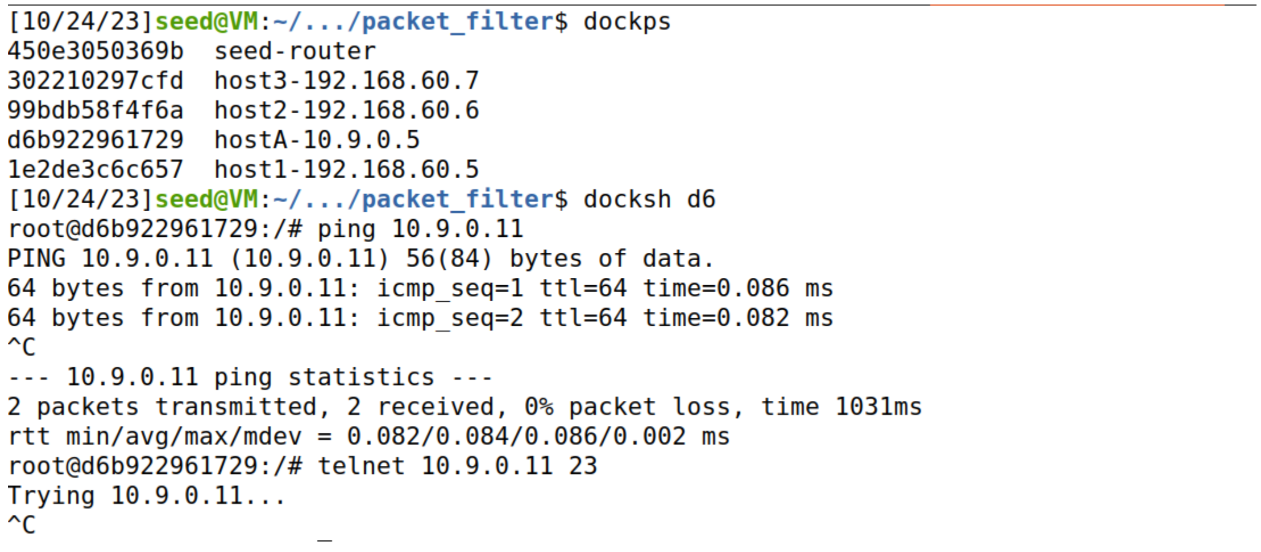
```
root@450e3050369b:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 61  bytes 9383 (9.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.11  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:0b  txqueuelen 0  (Ethernet)
        RX packets 63  bytes 9611 (9.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

run the given commands



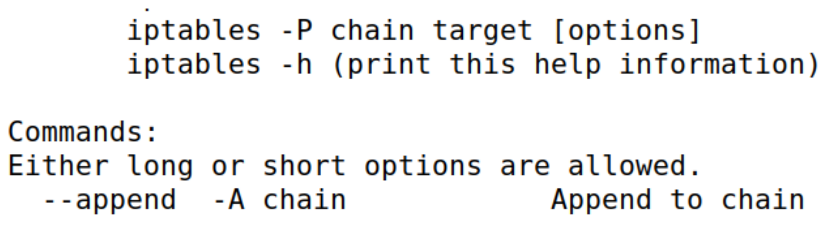
```
[10/24/23]seed@VM:~/.../packet_filter$ dockps
450e3050369b  seed-router
302210297cfd  host3-192.168.60.7
99bdb58f4f6a  host2-192.168.60.6
d6b922961729  hostA-10.9.0.5
1e2de3c6c657  host1-192.168.60.5
[10/24/23]seed@VM:~/.../packet_filter$ docksh 45
root@450e3050369b:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@450e3050369b:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@450e3050369b:/# iptables -P OUTPUT DROP
root@450e3050369b:/# iptables -P INPUT DROP
```

After running, I was still able to ping but unable to telnet into 10.9.0.11



```
[10/24/23]seed@VM:~/.../packet_filter$ dockps
450e3050369b  seed-router
302210297cfd  host3-192.168.60.7
99bdb58f4f6a  host2-192.168.60.6
d6b922961729  hostA-10.9.0.5
1e2de3c6c657  host1-192.168.60.5
[10/24/23]seed@VM:~/.../packet_filter$ docksh d6
root@d6b922961729:/# 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=0.086 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.082 ms
^C
--- 10.9.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.082/0.084/0.086/0.002 ms
root@d6b922961729:/# telnet 10.9.0.11 23
Trying 10.9.0.11...
^C
```

I used iptables -h to find out the different flags and their usage, that was how I understood the flags in the above commands.



```
        iptables -P chain target [options]
        iptables -h (print this help information)

Commands:
Either long or short options are allowed.
  --append  -A chain            Append to chain
```

The first two lines are to append a chain in the iptable to allow the router to accept icmp packets which is what ping uses, essentially to accept ping request and reply packets.
The next two drop all packets from input and output chains by default other than the ones appended in the above 2 lines.

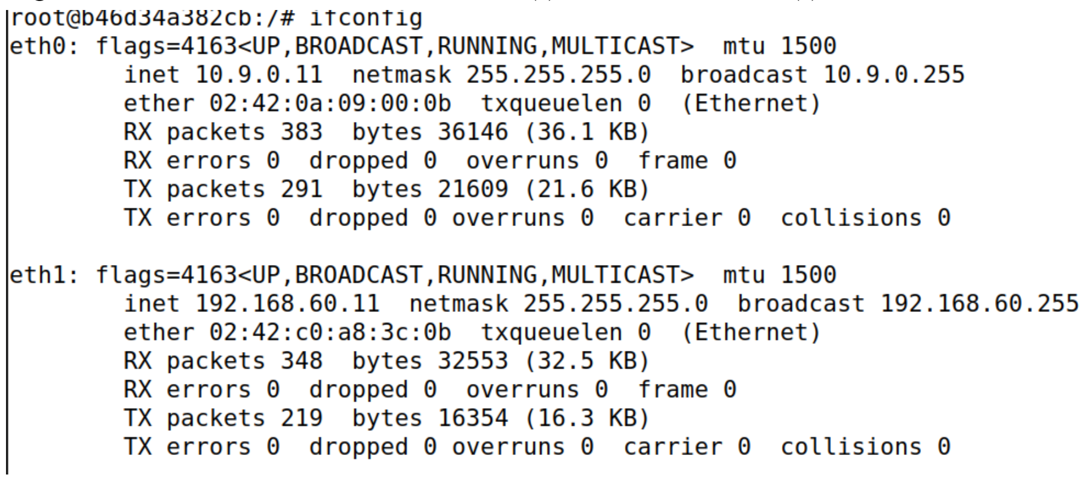
Part B

Preparation:

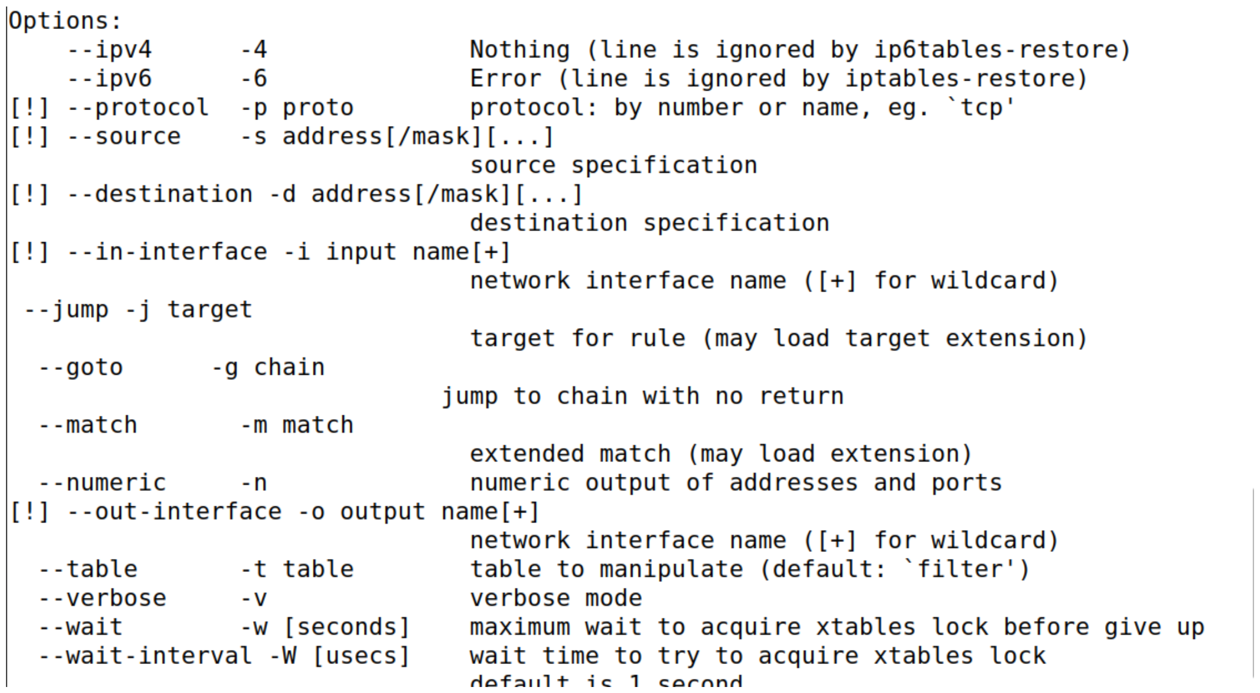
```
iptables -F
iptables -P OUTPUT ACCEPT
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
```

These commands will reset the tables to ACCEPT all packets on all chains
iptables -L -v -n to check for the policies in the iptable, making the tables verbose with numbers

To be able to ping internal hosts from an external host, the router will receive the ping request first, thus the FORWARD chain has to be edited.
I used ifconfig to find out the IP of the router (192.168.60.11), and used iptables -h to see what useful flags there were. I found -d to set destination(s) and -s to set source(s).

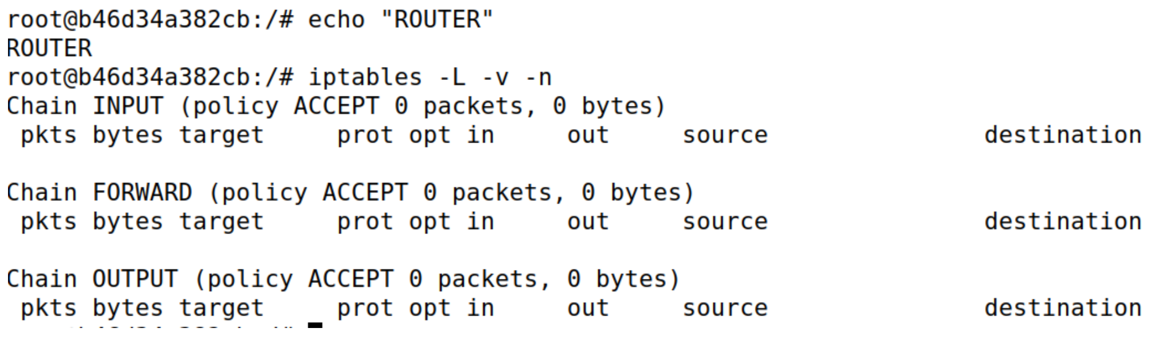


```
root@b46d34a382cb:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 383  bytes 36146 (36.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 291  bytes 21609 (21.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.11  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:0b  txqueuelen 0  (Ethernet)
        RX packets 348  bytes 32553 (32.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 219  bytes 16354 (16.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



```
Options:
    --ipv4      -4              Nothing (line is ignored by ip6tables-restore)
    --ipv6      -6              Error (line is ignored by iptables-restore)
[!] --protocol  -p proto        protocol: by number or name, eg. `tcp'
[!] --source    -s address[/mask][...]
                                source specification
[!] --destination -d address[/mask][...]
                                destination specification
[!] --in-interface -i input name[+]
                                network interface name ([+] for wildcard)
 --jump -j target
                                target for rule (may load target extension)
  --goto      -g chain
                              jump to chain with no return
  --match       -m match
                                extended match (may load extension)
  --numeric     -n              numeric output of addresses and ports
[!] --out-interface -o output name[+]
                                network interface name ([+] for wildcard)
  --table       -t table        table to manipulate (default: `filter')
  --verbose     -v              verbose mode
  --wait        -w [seconds]    maximum wait to acquire xtables lock before give up
  --wait-interval -W [usecs]    wait time to try to acquire xtables lock
                                default is 1 second
```

The initial state of the iptable is a fresh state



```
root@b46d34a382cb:/# echo "ROUTER"
ROUTER
root@b46d34a382cb:/# iptables -L -v -n
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
```

commands:

iptables -A FORWARD -p icmp --icmp-type echo-request -d 192.168.60.11 -j ACCEPT

→ external hosts cannot ping internal hosts, only can ping router

iptables -A FORWARD -p icmp --icmp-type echo-request -s 192.168.60.0/24 -j ACCEPT

→ internal hosts can ping outside hosts

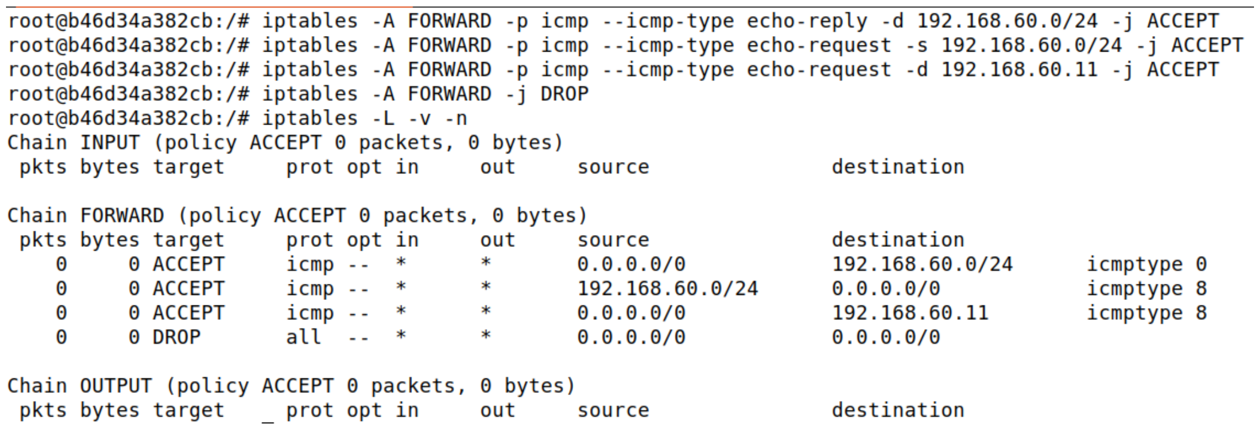
iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168.60.0/24 -j ACCEPT

→ accepts ping reply packets to all hosts after a request is sent

iptables -A FORWARD -j DROP → drop packets that do not fall into these categories

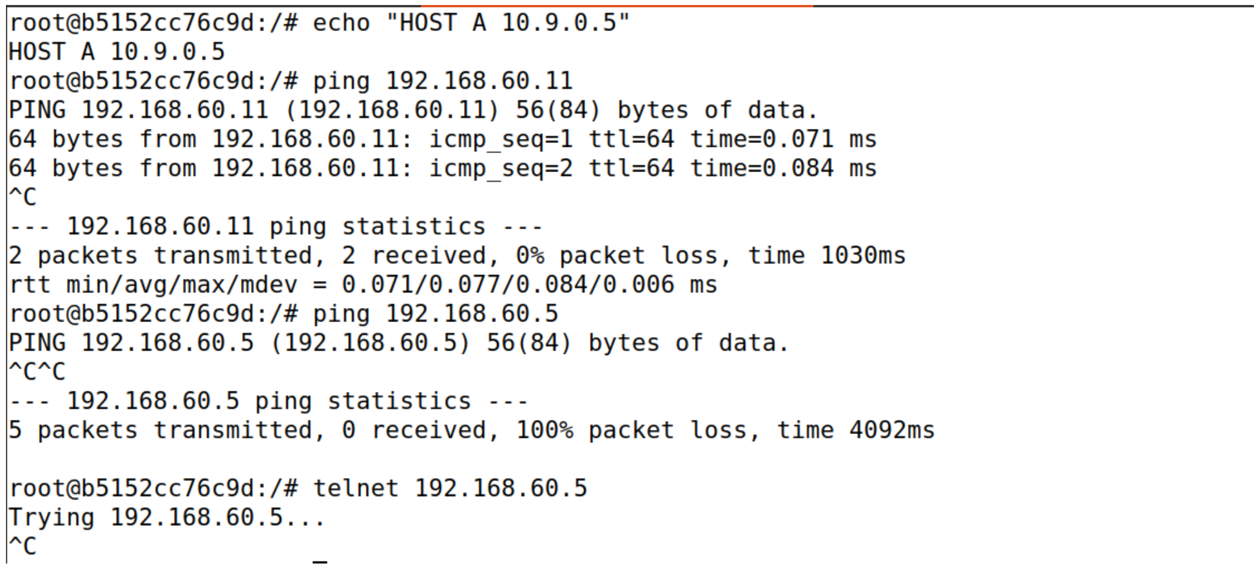
Essentially, append to the FORWARD chain to conditionally accept packets with an ICMP protocol and from specified destination and source, drop packets otherwise

The iptable had this final state



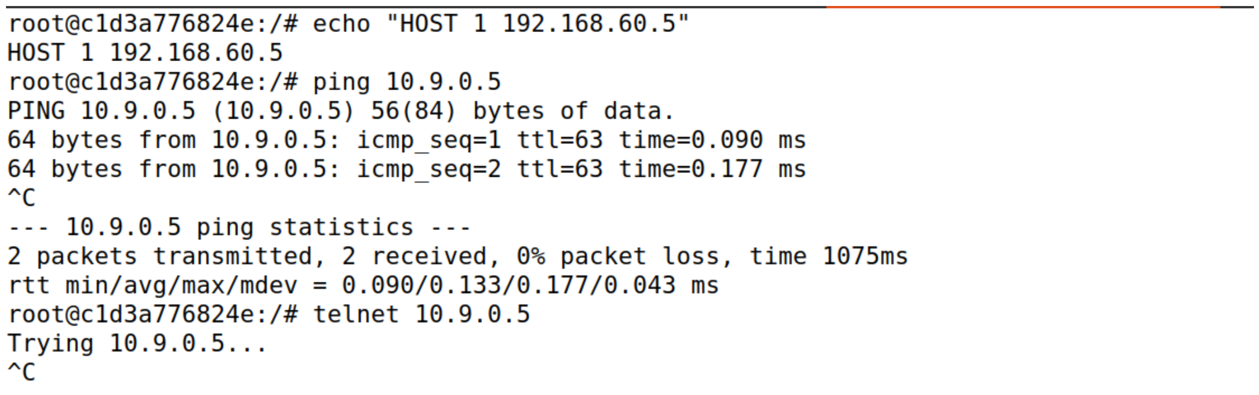
```
root@b46d34a382cb:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168.60.0/24 -j ACCEPT
root@b46d34a382cb:/# iptables -A FORWARD -p icmp --icmp-type echo-request -s 192.168.60.0/24 -j ACCEPT
root@b46d34a382cb:/# iptables -A FORWARD -p icmp --icmp-type echo-request -d 192.168.60.11 -j ACCEPT
root@b46d34a382cb:/# iptables -A FORWARD -j DROP
root@b46d34a382cb:/# iptables -L -v -n
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
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            192.168.60.0/24      icmptype 0
    0     0 ACCEPT     icmp --  *      *       192.168.60.0/24      0.0.0.0/0            icmptype 8
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            192.168.60.11        icmptype 8
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0

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

Results:
hostA (10.9.0.5) was unable to ping host1 (192.168.60.5) but was able to ping router (192.168.60.11)
hostA was also unable to telnet into host1



```
root@b5152cc76c9d:/# echo "HOST A 10.9.0.5"
HOST A 10.9.0.5
root@b5152cc76c9d:/# 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=0.071 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.084 ms
^C
--- 192.168.60.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1030ms
rtt min/avg/max/mdev = 0.071/0.077/0.084/0.006 ms
root@b5152cc76c9d:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4092ms

root@b5152cc76c9d:/# telnet 192.168.60.5
Trying 192.168.60.5...
^C
```

host1 able to ping hostA but unable to telnet into hostA



```
root@c1d3a776824e:/# echo "HOST 1 192.168.60.5"
HOST 1 192.168.60.5
root@c1d3a776824e:/# 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=0.090 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.177 ms
^C
--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1075ms
rtt min/avg/max/mdev = 0.090/0.133/0.177/0.043 ms
root@c1d3a776824e:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
```

Part C

Preparation:

same as in 2B

We will still work on the FORWARD chain and building on the flags to set destinations and sources in iptables.
We will set dport to be 23 and protocol to be tcp for all conditions.
The ifconfig shows IP 192.168.60.0/24 is in eth1

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.11  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:0b  txqueuelen 0  (Ethernet)
        RX packets 283  bytes 25264 (25.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 166  bytes 11166 (11.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

commands:

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

→ internal hosts can access internal servers

```
iptables -A FORWARD -p tcp --dport 23 -d 192.168.60.5 -j ACCEPT
```

→ external hosts can only access 192.168.60.5

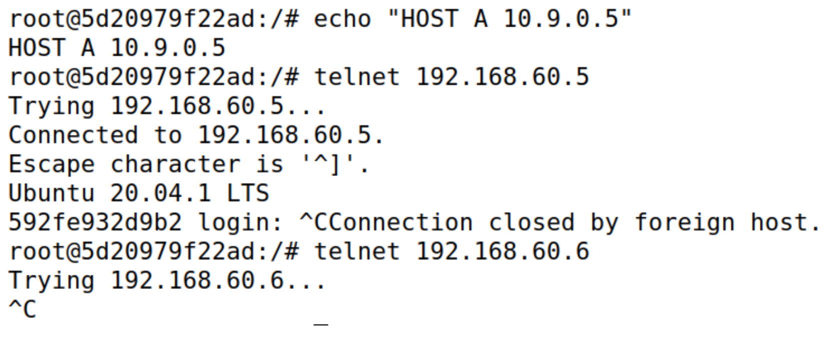
```
iptables -A FORWARD -p tcp -o eth1 -j DROP
```

→ drop outgoing packets from eth1 → internal hosts cannot reach external hosts

The final state of iptable

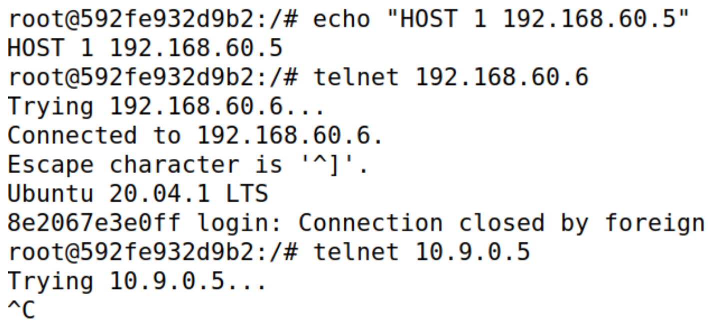
```
root@eeff6dec88f7:/# iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


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

    0     0 ACCEPT     tcp  --  *      *       192.168.60.0/24      192.168.60.0/24
   tcp dpt:23
    0     0 ACCEPT     tcp  --  *      *       10.9.0.5             192.168.60.5
   tcp dpt:23
    0     0 DROP       tcp  --  *      eth1    0.0.0.0/0            0.0.0.0/0


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

Results:
External hostA (10.9.0.5) was able to access host1 (192.168.60.5), but was unable to access host2 (192.168.60.6)



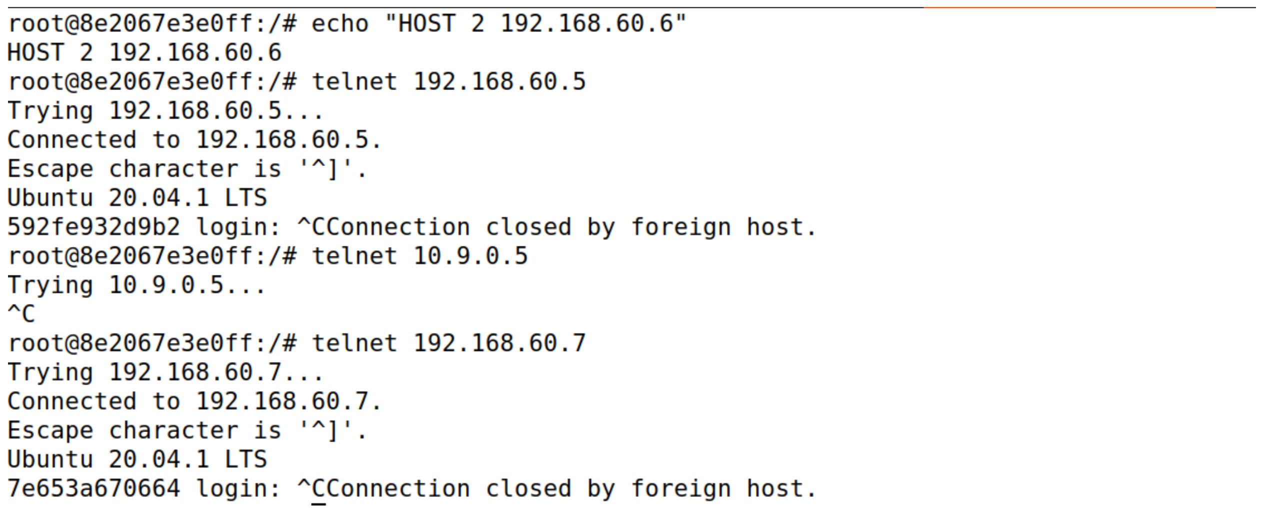
```
root@5d20979f22ad:/# echo "HOST A 10.9.0.5"
HOST A 10.9.0.5
root@5d20979f22ad:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
592fe932d9b2 login: ^CConnection closed by foreign host.
root@5d20979f22ad:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
```

Internal host1 was able to access host2, but was unable to access hostA



```
root@592fe932d9b2:/# echo "HOST 1 192.168.60.5"
HOST 1 192.168.60.5
root@592fe932d9b2:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
8e2067e3e0ff login: Connection closed by foreign host.
root@592fe932d9b2:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
```

Internal host2 was able to access host3 (192.168.60.7) and host1, but was unable to access the external hostA

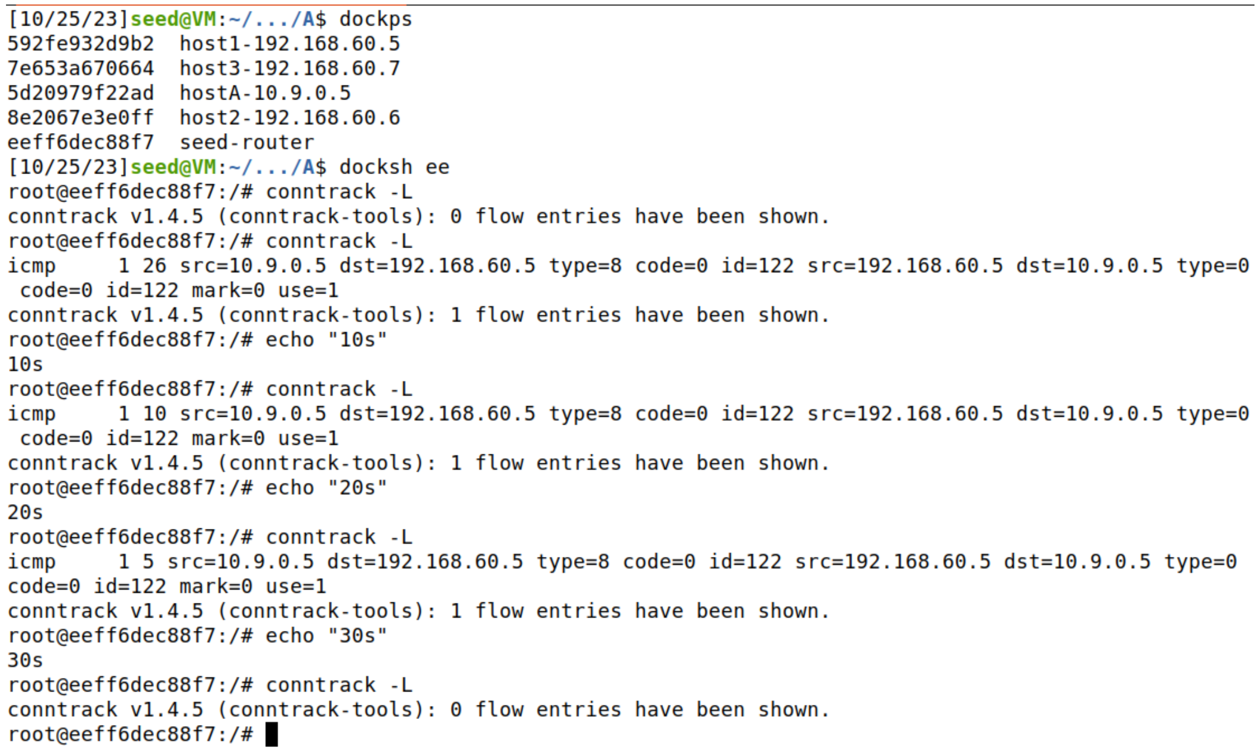


```
root@8e2067e3e0ff:/# echo "HOST 2 192.168.60.6"
HOST 2 192.168.60.6
root@8e2067e3e0ff:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
592fe932d9b2 login: ^CConnection closed by foreign host.
root@8e2067e3e0ff:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@8e2067e3e0ff:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7e653a670664 login: ^CConnection closed by foreign host.
```

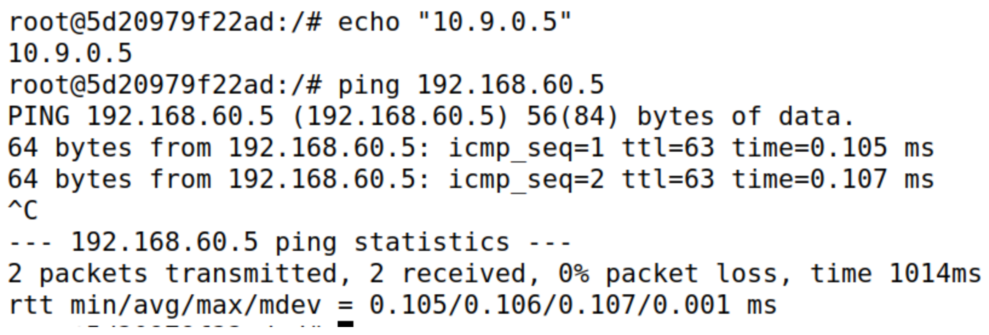
Task 3

Part A

The icmp connection state was kept for approximately 30 seconds, after which conntrack -L shows 0 entries. This is due to icmp packets being connectionless and a ping packet does not need such a long waiting time.

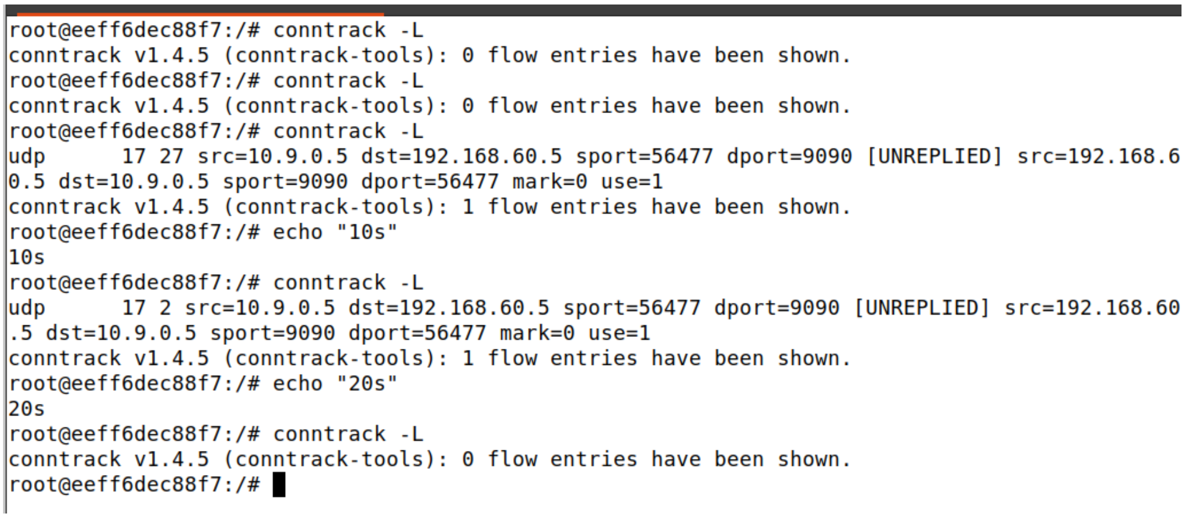


```
[10/25/23]seed@VM:~/.../A$ dockps
592fe932d9b2  host1-192.168.60.5
7e653a670664  host3-192.168.60.7
5d20979f22ad  hostA-10.9.0.5
8e2067e3e0ff  host2-192.168.60.6
eeff6dec88f7  seed-router
[10/25/23]seed@VM:~/.../A$ docksh ee
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
icmp     1 26 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=122 src=192.168.60.5 dst=10.9.0.5 type=0
 code=0 id=122 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# echo "10s"
10s
root@eeff6dec88f7:/# conntrack -L
icmp     1 10 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=122 src=192.168.60.5 dst=10.9.0.5 type=0
 code=0 id=122 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# echo "20s"
20s
root@eeff6dec88f7:/# conntrack -L
icmp     1 5 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=122 src=192.168.60.5 dst=10.9.0.5 type=0
code=0 id=122 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# echo "30s"
30s
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/#
```



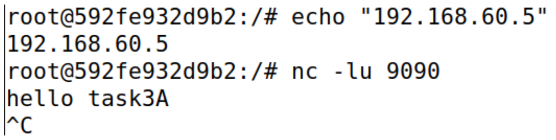
```
root@5d20979f22ad:/# echo "10.9.0.5"
10.9.0.5
root@5d20979f22ad:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.105 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.107 ms
^C
--- 192.168.60.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1014ms
rtt min/avg/max/mdev = 0.105/0.106/0.107/0.001 ms
```

The UDP connection state is kept for about 20 seconds, and the router will only keep the state once hostA(10.9.0.5) starts sending a packet over to host1 (192.168.60.5), otherwise the connection information remains empty. The same logic applies here as UDP is also a connectionless protocol, thus a long waiting time is not needed.

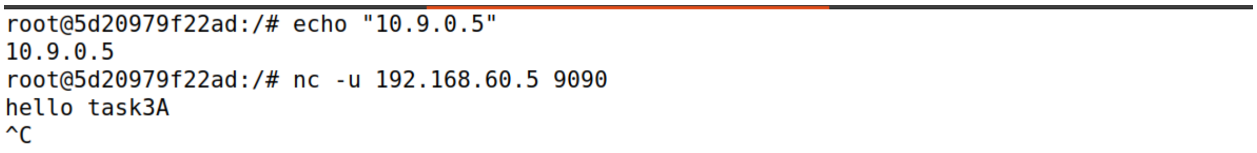


```
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
udp      17 27 src=10.9.0.5 dst=192.168.60.5 sport=56477 dport=9090 [UNREPLIED] src=192.168.6
0.5 dst=10.9.0.5 sport=9090 dport=56477 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# echo "10s"
10s
root@eeff6dec88f7:/# conntrack -L
udp      17 2 src=10.9.0.5 dst=192.168.60.5 sport=56477 dport=9090 [UNREPLIED] src=192.168.60
.5 dst=10.9.0.5 sport=9090 dport=56477 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# echo "20s"
20s
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/# 
```

The UDP connection was established as shown below.

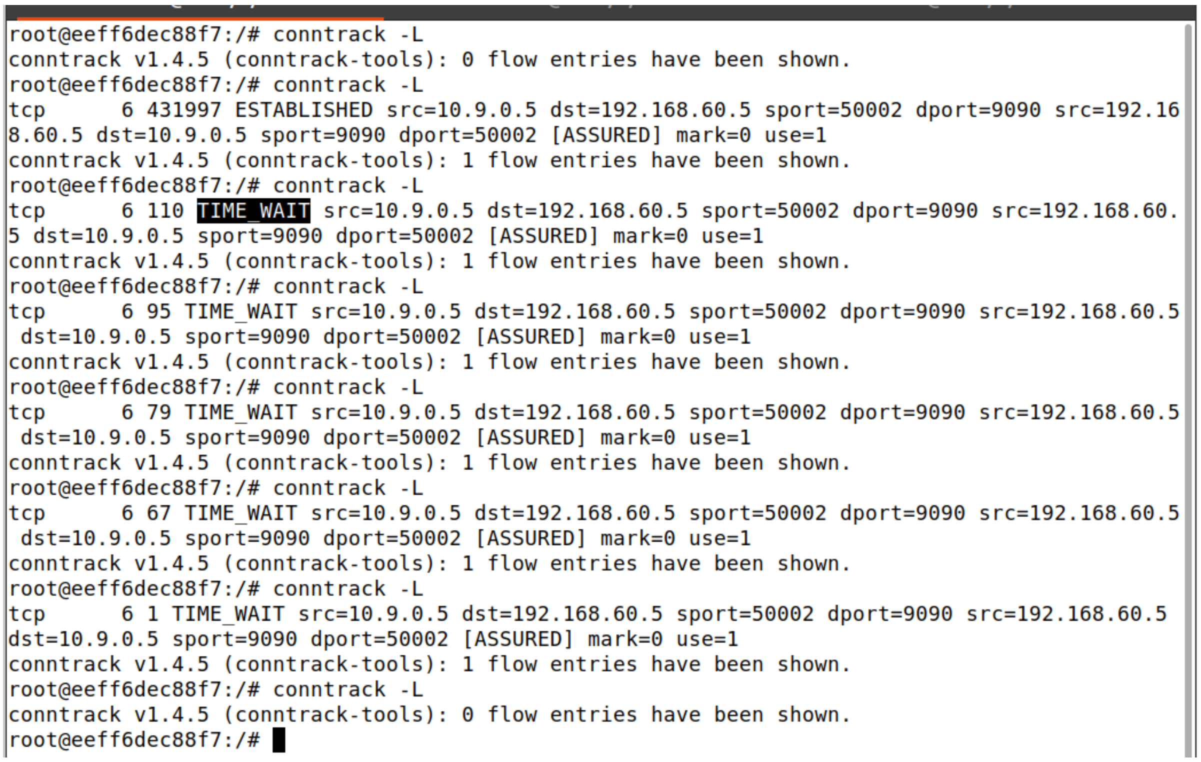


```
root@592fe932d9b2:/# echo "192.168.60.5"
192.168.60.5
root@592fe932d9b2:/# nc -lu 9090
hello task3A
^C
```



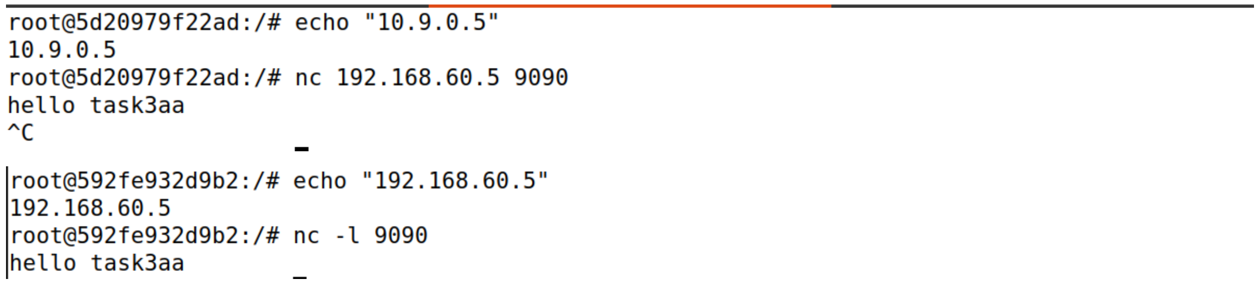
```
root@5d20979f22ad:/# echo "10.9.0.5"
10.9.0.5
root@5d20979f22ad:/# nc -u 192.168.60.5 9090
hello task3A
^C
```

The TCP connection state is kept for about 110s as seen from the TIME_WAIT field, the timer started when the connection was closed. The 110s wait is for TCP to handle problems due to unreliable or delayed packet delivery, waiting for the unreceived packet upon closing the connection.



```
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
tcp      6 431997 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=50002 dport=9090 src=192.16
8.60.5 dst=10.9.0.5 sport=9090 dport=50002 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
tcp      6 110 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50002 dport=9090 src=192.168.60.
5 dst=10.9.0.5 sport=9090 dport=50002 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
tcp      6 95 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50002 dport=9090 src=192.168.60.5
 dst=10.9.0.5 sport=9090 dport=50002 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
tcp      6 79 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50002 dport=9090 src=192.168.60.5
 dst=10.9.0.5 sport=9090 dport=50002 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
tcp      6 67 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50002 dport=9090 src=192.168.60.5
 dst=10.9.0.5 sport=9090 dport=50002 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
tcp      6 1 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50002 dport=9090 src=192.168.60.5
dst=10.9.0.5 sport=9090 dport=50002 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@eeff6dec88f7:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@eeff6dec88f7:/#
```

The TCP connection was established as shown below



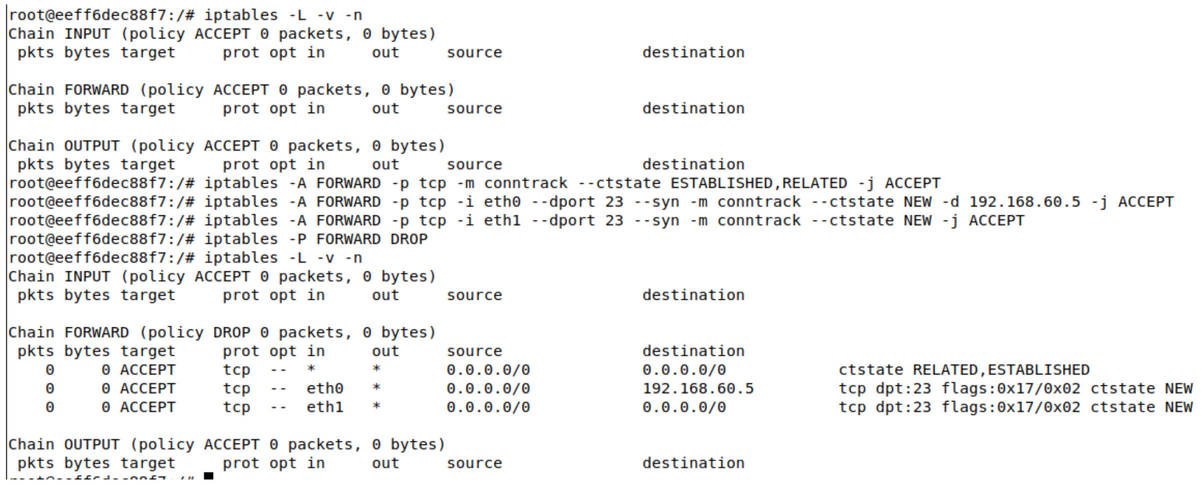
```
root@5d20979f22ad:/# echo "10.9.0.5"
10.9.0.5
root@5d20979f22ad:/# nc 192.168.60.5 9090
hello task3aa
^C
```

```
root@592fe932d9b2:/# echo "192.168.60.5"
192.168.60.5
root@592fe932d9b2:/# nc -l 9090
hello task3aa
```

Part B

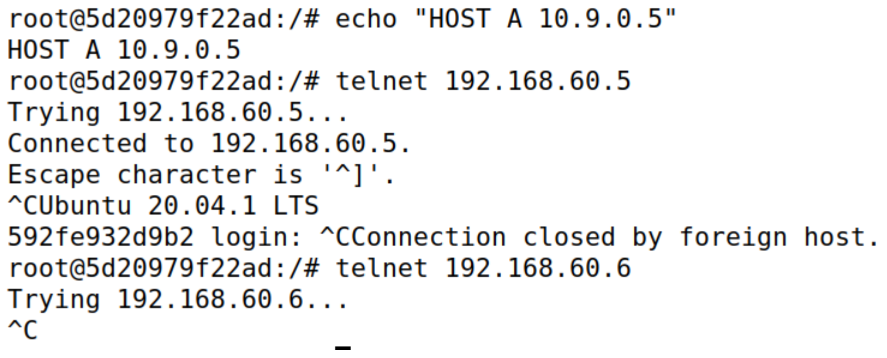
iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
→ accepts packets from established connections or any related to it
iptables -A FORWARD -p tcp -i eth0 --dport 23 --syn -m conntrack --ctstate NEW -d 192.168.60.5 -j ACCEPT
→ accepts SYN packets going into external hosts
iptables -A FORWARD -p tcp -i eth1 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
→ accepts SYN packets going into internal hosts
iptables -P FORWARD DROP
→ drop everything else that does not belong to the above conditions

The below shows the initial and final state of the iptable



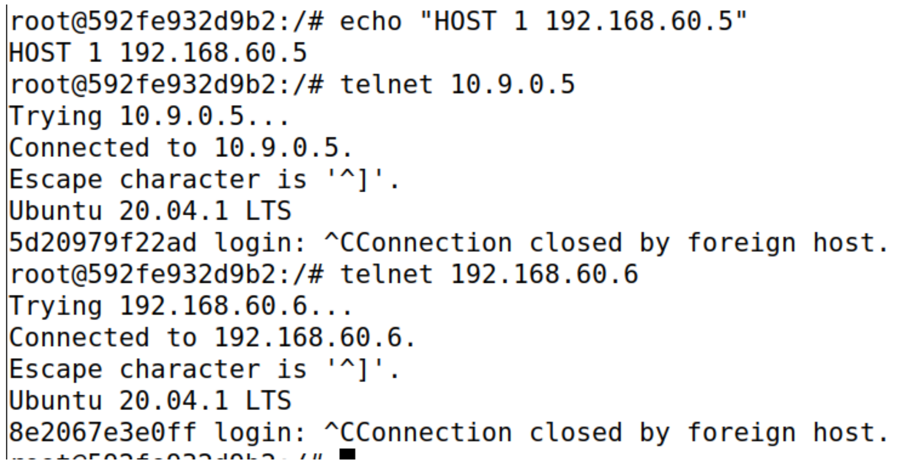
```
root@eeff6dec88f7:/# iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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
root@eeff6dec88f7:/# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
root@eeff6dec88f7:/# iptables -A FORWARD -p tcp -i eth0 --dport 23 --syn -m conntrack --ctstate NEW -d 192.168.60.5 -j ACCEPT
root@eeff6dec88f7:/# iptables -A FORWARD -p tcp -i eth1 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@eeff6dec88f7:/# iptables -P FORWARD DROP
root@eeff6dec88f7:/# iptables -L -v -n
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  --  *      *       0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
    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 dpt:23 flags:0x17/0x02 ctstate NEW

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

Result:
External hostA (10.9.0.5) was able to access internal host1 (192.168.60.5), but was unable to access host2 (192.168.60.6)



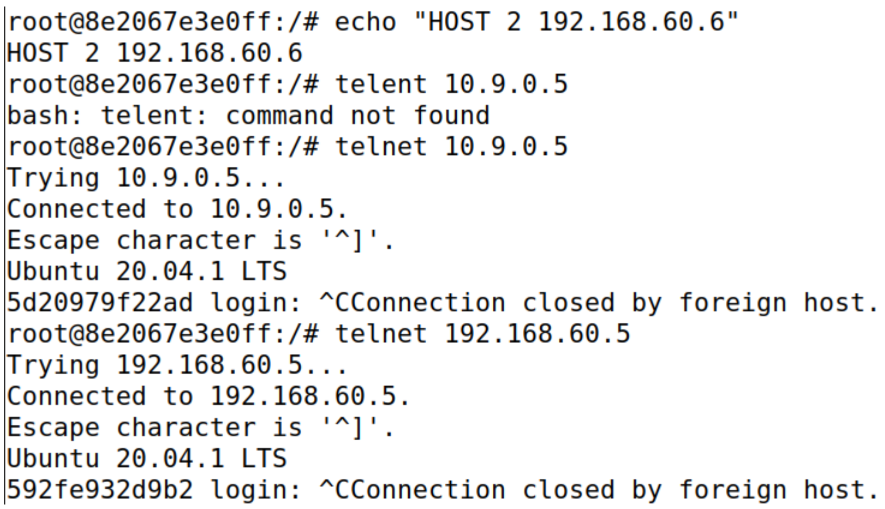
```
root@5d20979f22ad:/# echo "HOST A 10.9.0.5"
HOST A 10.9.0.5
root@5d20979f22ad:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
^CUbuntu 20.04.1 LTS
592fe932d9b2 login: ^CConnection closed by foreign host.
root@5d20979f22ad:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
```

Internal host1 was able to access host2 and hostA



```
root@592fe932d9b2:/# echo "HOST 1 192.168.60.5"
HOST 1 192.168.60.5
root@592fe932d9b2:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5d20979f22ad login: ^CConnection closed by foreign host.
root@592fe932d9b2:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
8e2067e3e0ff login: ^CConnection closed by foreign host.
```

Internal host2 was able to access host1 and hostA

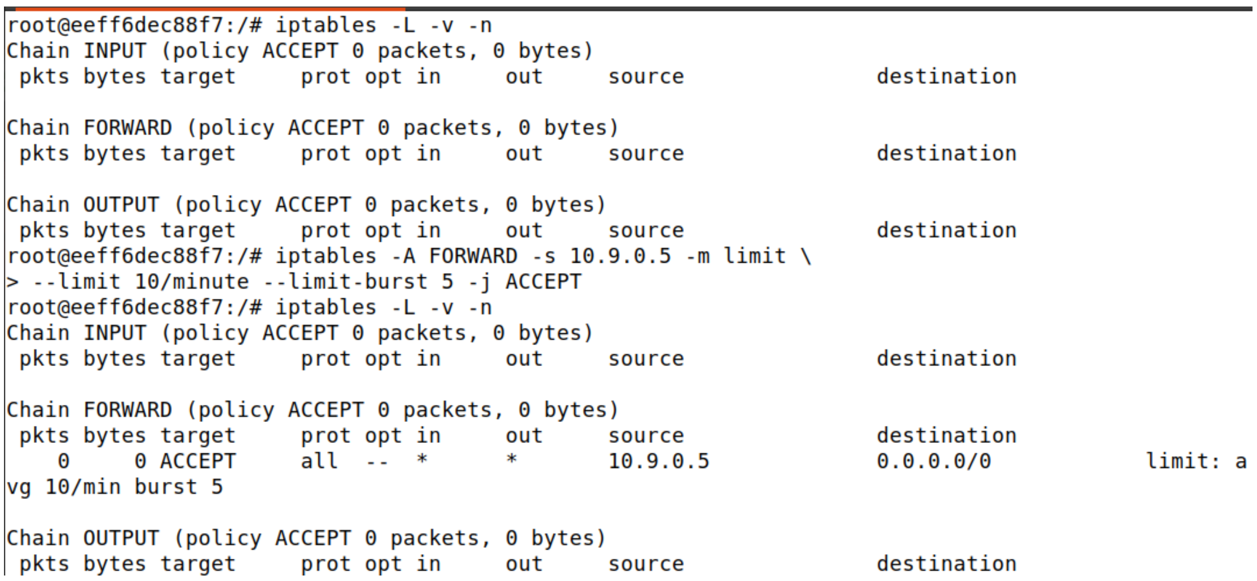


```
root@8e2067e3e0ff:/# echo "HOST 2 192.168.60.6"
HOST 2 192.168.60.6
root@8e2067e3e0ff:/# telent 10.9.0.5
bash: telent: command not found
root@8e2067e3e0ff:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5d20979f22ad login: ^CConnection closed by foreign host.
root@8e2067e3e0ff:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
592fe932d9b2 login: ^CConnection closed by foreign host.
```

The difference would be due to the nature of the firewall (stateful or stateless), and having a stateless firewall would mean that any rules on connections should be strict and concise as they will take effect before any connection is established.
A stateful firewall would mean that the router will have to constantly check for connection states (ESTABLISHED, NEW, CLOSING) and their rules before deciding what to do with the packets

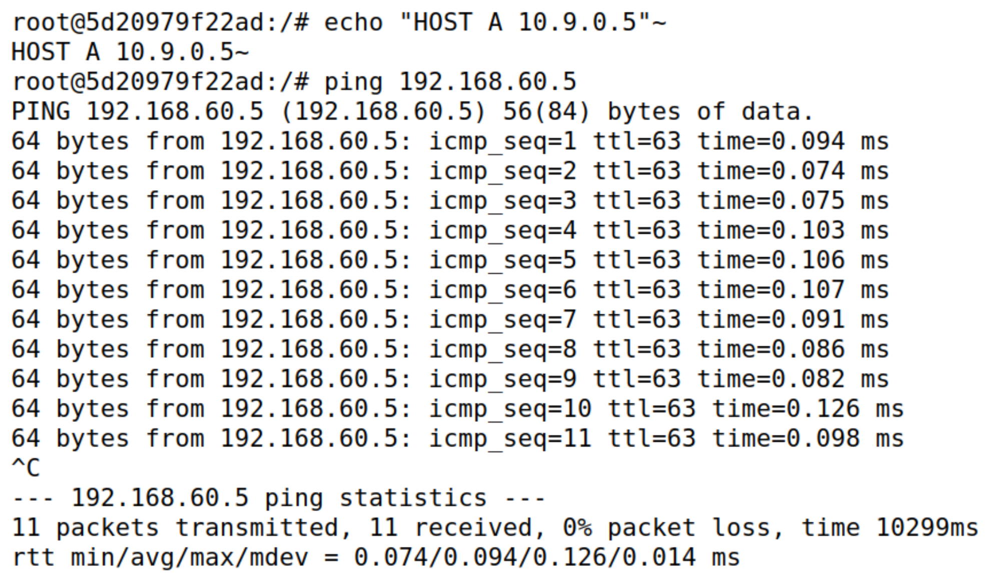
Task 4

This is the state of the iptable after loading only the first rule



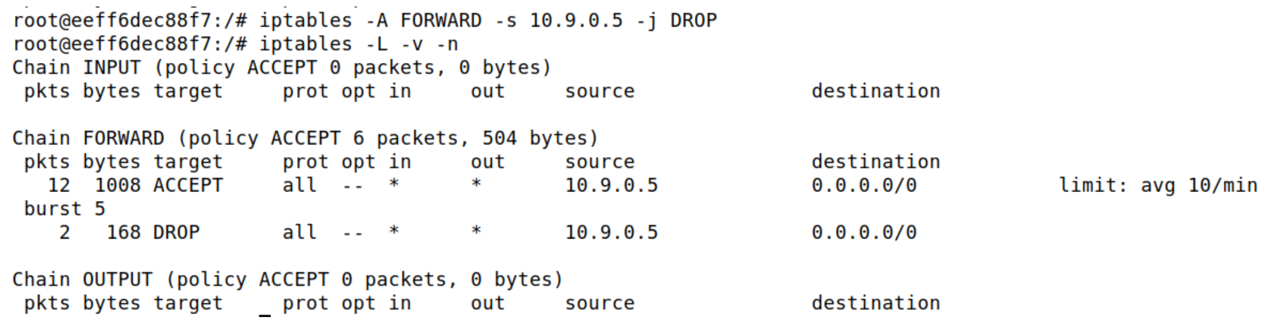
```
root@eeff6dec88f7:/# iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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
root@eeff6dec88f7:/# iptables -A FORWARD -s 10.9.0.5 -m limit \
> --limit 10/minute --limit-burst 5 -j ACCEPT
root@eeff6dec88f7:/# iptables -L -v -n
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
    0     0 ACCEPT     all  --  *      *       10.9.0.5             0.0.0.0/0            limit: a
vg 10/min burst 5

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

The limit of 5 did not take effect as packets were not dropped, hence 11 ping packets were transmitted and received.



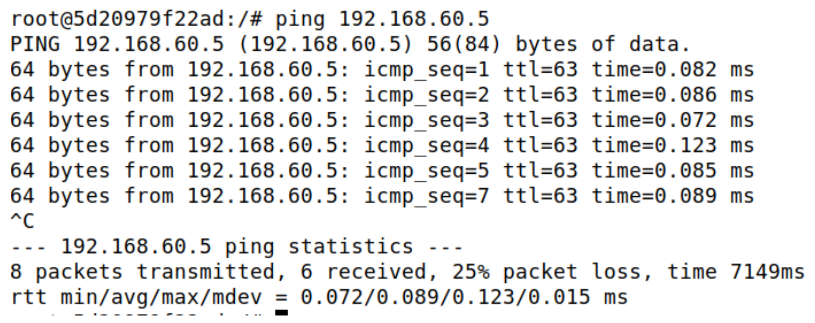
```
root@5d20979f22ad:/# echo "HOST A 10.9.0.5"~
HOST A 10.9.0.5~
root@5d20979f22ad:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.094 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.074 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.075 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.103 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=6 ttl=63 time=0.107 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.091 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.086 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.082 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.126 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.098 ms
^C
--- 192.168.60.5 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10299ms
rtt min/avg/max/mdev = 0.074/0.094/0.126/0.014 ms
```

This is the state of the iptable after both rules were added



```
root@eeff6dec88f7:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@eeff6dec88f7:/# iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 6 packets, 504 bytes)
 pkts bytes target     prot opt in     out     source               destination
   12  1008 ACCEPT     all  --  *      *       10.9.0.5             0.0.0.0/0            limit: avg 10/min
 burst 5
    2   168 DROP       all  --  *      *       10.9.0.5             0.0.0.0/0

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

With the limit set, no more packets were received after the 6th one.



```
root@5d20979f22ad:/# 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=0.082 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.086 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.072 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.123 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.085 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.089 ms
^C
--- 192.168.60.5 ping statistics ---
8 packets transmitted, 6 received, 25% packet loss, time 7149ms
rtt min/avg/max/mdev = 0.072/0.089/0.123/0.015 ms
```

The second rule was essential to enforce the limit by dropping packets, changing the FORWARD chain policy to DROP does not work as the chain should not drop packets by default.