Firewall Exploration Lab

Submitted by: Mahika Gupta

SRN: PES1UG20CS243

DATE: 27/10/2022

Contents

LAB SETUP 1

LAB OVERVIEW 2

TASK 1: IMPLEMENTING A SIMPLE FIREWALL 3

TASK 2: EXPERIMENTING WITH STATELESS FIREWALL RULES 11

TASK 3: CONNECTION TRACKING AND STATEFUL FIREWALL 18

TASK 4: LIMITING NETWORK TRAFFIC 23

TASK 5: LOAD BALANCING 25

Lab Setup

Please download the Labsetup.zip file from the below link to your VM, unzip it, enter the Labsetup folder, and use the docker-compose.yml file to set up the lab environment.

https://seedsecuritylabs.org/Labs_20.04/Files/Firewall/Labsetup.zip

In this lab, we need to use multiple machines. Their setup is depicted in Figure 1. We will use containers to set up this lab environment..

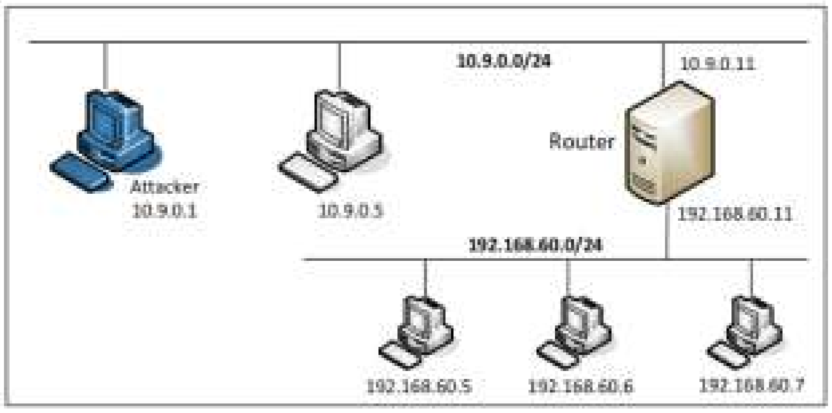


Figure 1: Lab Setup

Lab Overview

The learning objective of this lab is two-fold: learning how firewalls work, and setting up a simple firewall for a network. Students will first implement a simple stateless packet-filtering firewall, which inspects packets, and decides whether to drop or forward a packet based on firewall rules.

Through this implementation task, students can get basic ideas on how a firewall works. Actually, Linux already has a built-in firewall, also based on netfilter. This firewall is called iptables. Students will be given simple network topology, and are asked to use iptables to set up firewall rules to protect the network. Students will also be exposed to several other interesting applications of iptables.

This lab covers the following topics:

• Firewall
• Netfilter
• Loadable kernel module
• Using iptables to set up firewall rules
• Various applications of iptables

Task 1: Implementing a Simple Firewall

In this task, we will implement a simple packet filtering type of firewall, which inspects each incoming and outgoing packet, and enforces the firewall policies set by the administrator. Since the packet processing is done within the kernel, the filtering must also be done within the kernel. Therefore, it seems that implementing such a firewall requires us to modify the Linux kernel. In the past, this had to be done by modifying and rebuilding the kernel. The modern Linux operating systems provide several new mechanisms to facilitate the manipulation of packets without rebuilding the kernel image. These two mechanisms are Loadable Kernel Module (LKM) and Netfilter.

Notes about containers - Since all the containers share the same kernel, kernel modules are global. Therefore, if we set a kernel model from a container, it affects all the containers and the host. For this reason, it does not matter where you set the kernel module. We will just set the kernel module from the host VM in this lab**.**

Another thing to keep in mind is that containers' IP addresses are virtual. Packets going to these virtual IP addresses may not traverse the same path as what is described in the Netfilter document.

Therefore, in this task, to avoid confusion, we will try to avoid using those virtual addresses. We do most tasks on the host VM. The containers are mainly for other tasks.

Task 1.A: Implement a Simple Kernel Module

LKM allows us to add a new module to the kernel at runtime. This new module enables us to extend the functionalities of the kernel, without rebuilding the kernel or even rebooting the computer. The packet filtering part of a firewall can be implemented as an LKM. In this task, we will get familiar with LKM.
The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded; when the module is removed from the kernel, it prints out "Bye-bye World!".

The messages are not printed out on the screen; they are actually printed into the /var/log/syslog file. You can use "dmesg" to view the messages.

hello. c - included in the lab setup files

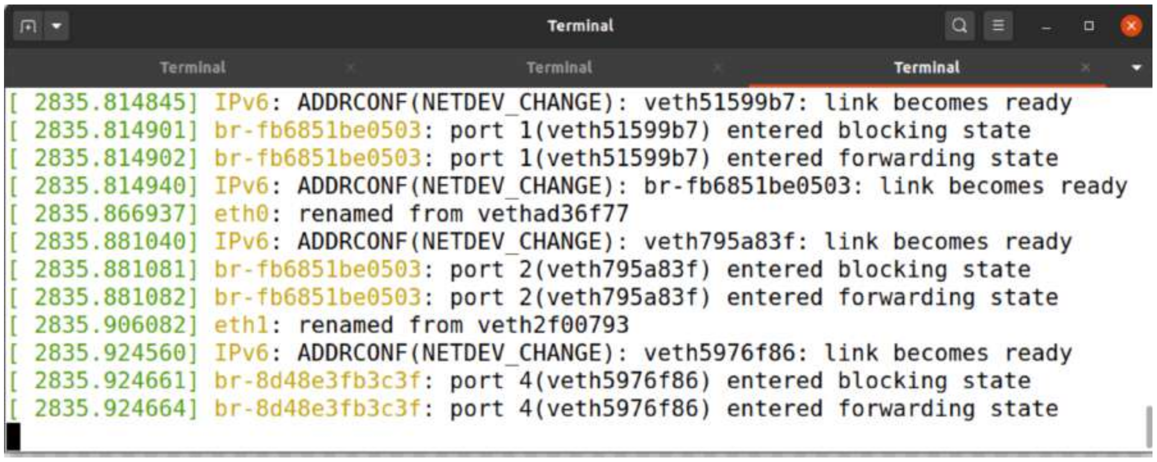
```
#include <linux/module.h>
#include <linux/kernel.h>
int initialization(void)
{
    printk(KERN_INFO "Hello World!\n");
    return 0;
}
void cleanup(void)
{
    printk(KERN_INFO "Bye-bye World!.\n");
}
module_init(initialization);
module_exit(cleanup);
MODULE_LICENSE("GPL");
```

On the VM - Open two Terminal Tabs, one to load the module and the other to view the messages.

- Use one terminal window to view the messages

Command:

- $ sudo dmesg -k -w



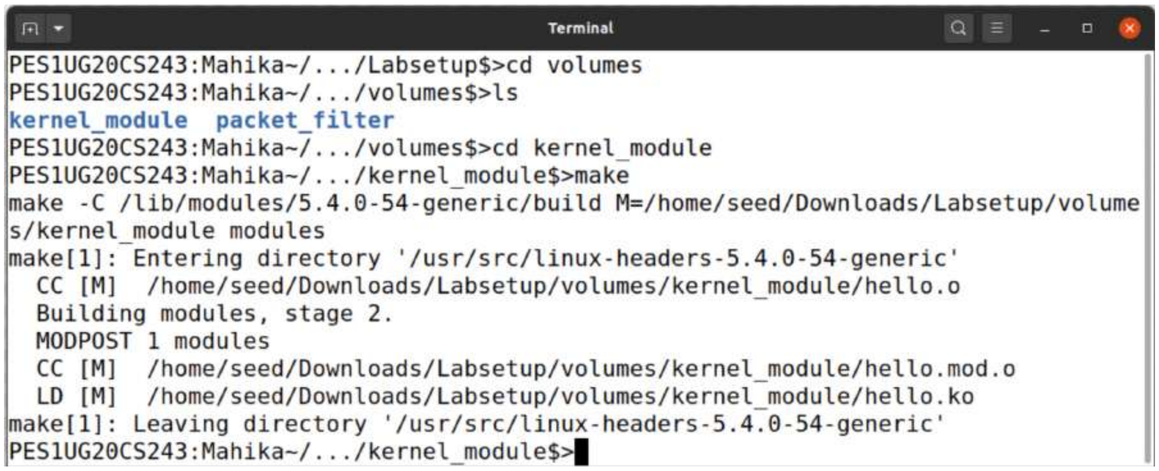
```
[ 2835.814845] IPv6: ADDRCONF(NETDEV_CHANGE): veth51599b7: link becomes ready
[ 2835.814901] br-fb6851be0503: port 1(veth51599b7) entered blocking state
[ 2835.814902] br-fb6851be0503: port 1(veth51599b7) entered forwarding state
[ 2835.814940] IPv6: ADDRCONF(NETDEV_CHANGE): br-fb6851be0503: link becomes ready
[ 2835.866937] eth0: renamed from vethad36f77
[ 2835.881040] IPv6: ADDRCONF(NETDEV_CHANGE): veth795a83f: link becomes ready
[ 2835.881081] br-fb6851be0503: port 2(veth795a83f) entered blocking state
[ 2835.881082] br-fb6851be0503: port 2(veth795a83f) entered forwarding state
[ 2835.906082] eth1: renamed from veth2f00793
[ 2835.924560] IPv6: ADDRCONF(NETDEV_CHANGE): veth5976f86: link becomes ready
[ 2835.924661] br-8d48e3fb3c3f: port 4(veth5976f86) entered blocking state
[ 2835.924664] br-8d48e3fb3c3f: port 4(veth5976f86) entered forwarding state
```

This displays the kernel buffer.

- The other to Load and Remove the Kernel

Command:

- $ make

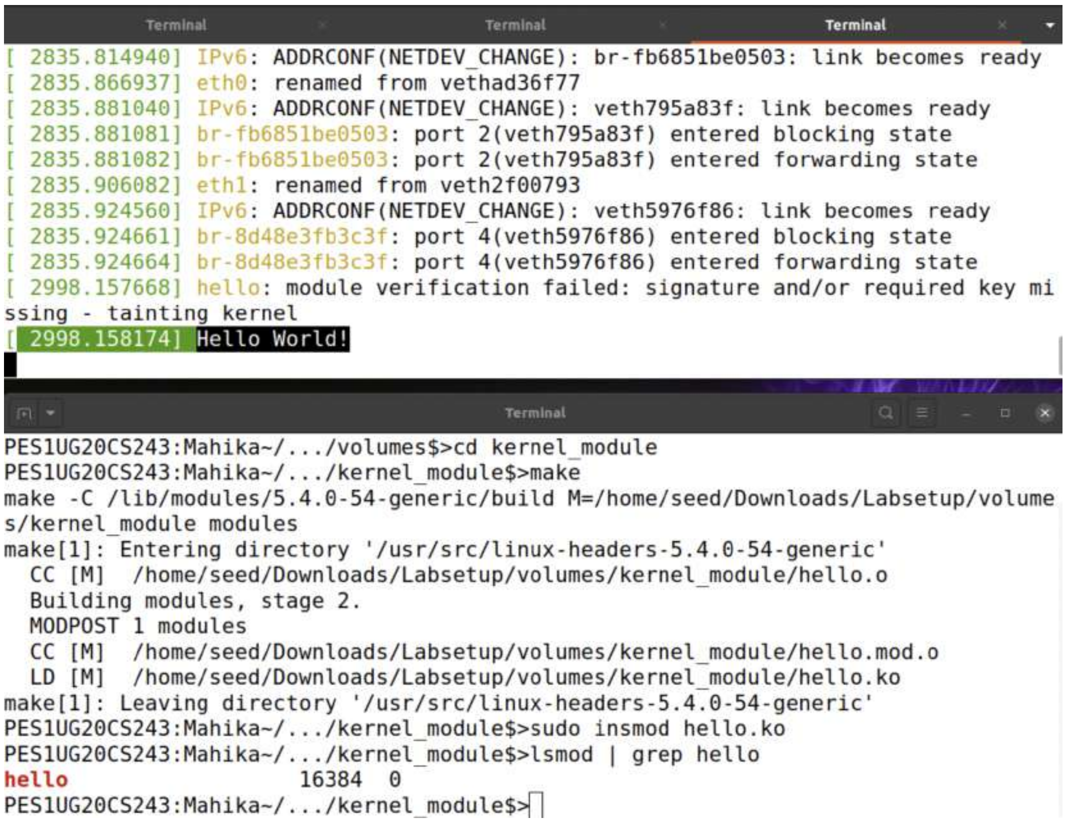


```
PES1UG20CS243:Mahika~/.../Labsetup$>cd volumes
PES1UG20CS243:Mahika~/.../volumes$>ls
kernel_module  packet_filter
PES1UG20CS243:Mahika~/.../volumes$>cd kernel_module
PES1UG20CS243:Mahika~/.../kernel_module$>make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/volume
s/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/volumes/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/volumes/kernel_module/hello.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/volumes/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG20CS243:Mahika~/.../kernel_module$>
```

$ sudo insmod hello.ko (inserting a module)

$ lsmod | grep hello (list modules)

$ sudo rmmod hello

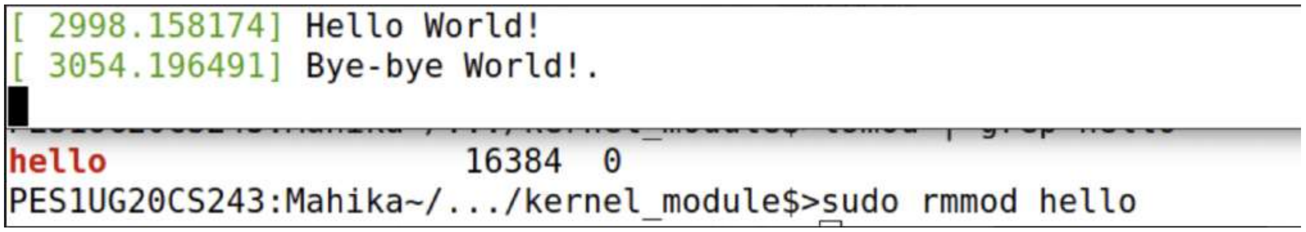


```
[ 2835.814940] IPv6: ADDRCONF(NETDEV_CHANGE): br-fb6851be0503: link becomes ready
[ 2835.866937] eth0: renamed from vethad36f77
[ 2835.881040] IPv6: ADDRCONF(NETDEV_CHANGE): veth795a83f: link becomes ready
[ 2835.881081] br-fb6851be0503: port 2(veth795a83f) entered blocking state
[ 2835.881082] br-fb6851be0503: port 2(veth795a83f) entered forwarding state
[ 2835.906082] eth1: renamed from veth2f00793
[ 2835.924560] IPv6: ADDRCONF(NETDEV_CHANGE): veth5976f86: link becomes ready
[ 2835.924661] br-8d48e3fb3c3f: port 4(veth5976f86) entered blocking state
[ 2835.924664] br-8d48e3fb3c3f: port 4(veth5976f86) entered forwarding state
[ 2998.157668] hello: module verification failed: signature and/or required key mi
ssing - tainting kernel
[ 2998.158174] Hello World!
```

```
PES1UG20CS243:Mahika~/.../volumes$>cd kernel_module
PES1UG20CS243:Mahika~/.../kernel_module$>make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/volume
s/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/volumes/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/volumes/kernel_module/hello.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/volumes/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG20CS243:Mahika~/.../kernel_module$>sudo insmod hello.ko
PES1UG20CS243:Mahika~/.../kernel_module$>lsmod | grep hello
hello                  16384  0
PES1UG20CS243:Mahika~/.../kernel_module$>
```

As we can see, on using the command insmod hello.ko, the hello module is inserted into the buffer, which displays the message.

lsmod shows which module is currently loaded in the buffer along with its size in bytes. Here we can see hello module is loaded and its size is 16384 bytes.



```
[ 2998.158174] Hello World!
[ 3054.196491] Bye-bye World!.
```

```
hello                  16384  0
PES1UG20CS243:Mahika~/.../kernel_module$>sudo rmmod hello
```

The command rmmod is used to remove the currently loaded module from the buffer. The message on removing the module is displayed in the buffer.

Task 1.B: Implement a Simple Firewall Using Netfilter

In this task, we will write our packet filtering program as an LKM, and then insert it into the packet processing path inside the kernel. This cannot be easily done in the past before netfilter was introduced into Linux.

Netfilter is designed to facilitate the manipulation of packets by authorized users. It achieves this goal by implementing a number of hooks in the Linux kernel. These hooks are inserted into various places, including the packet's incoming and outgoing paths. If we want to manipulate the incoming packets, we simply need to connect our own programs (within LKM) to the corresponding hooks.

Once an incoming packet arrives, our program will be invoked. Our program can decide whether this packet should be blocked or not; moreover, we can also modify the packets in the program.

In this task, you need to use LKM and Netfilter to implement a packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. We would like students to focus on the filtering part, the core of firewalls, so students are allowed to hardcode firewall policies in the program.

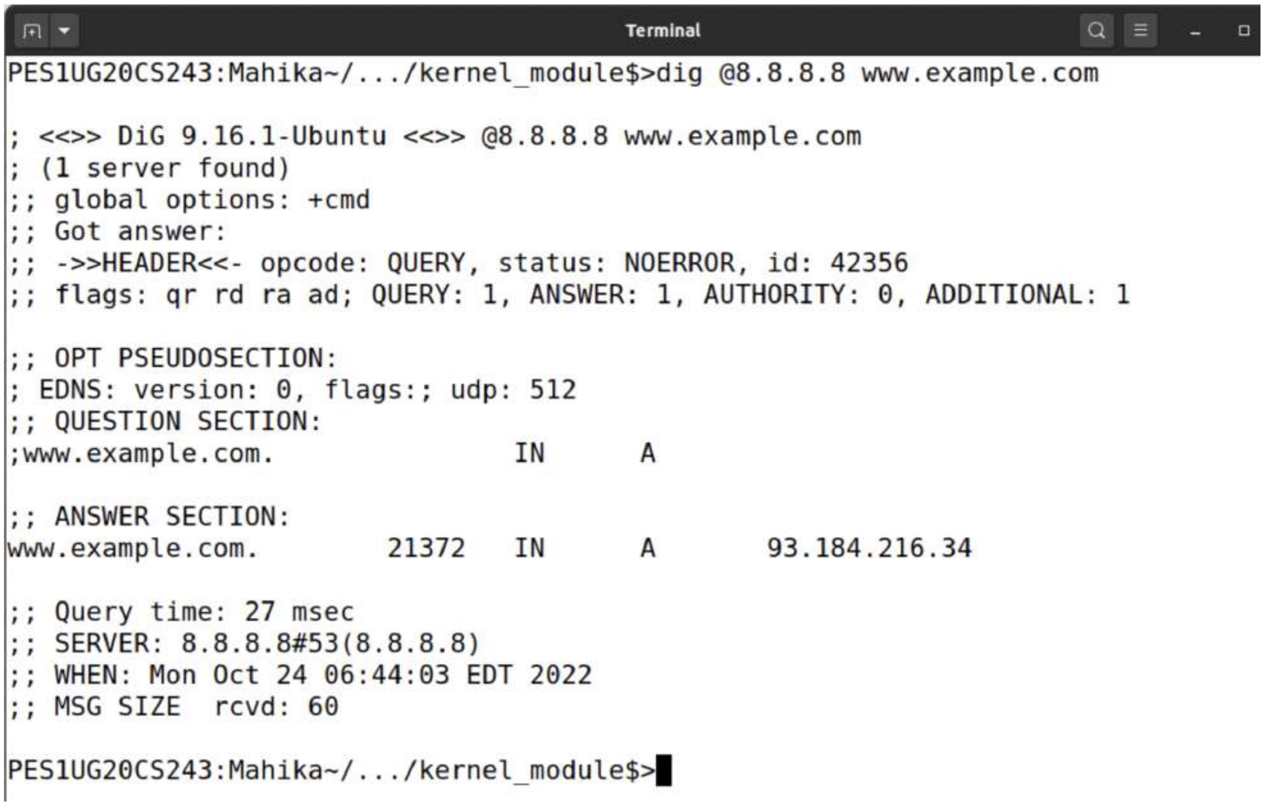
Tasks

1. Compile the code using the provided Makefile. Load it into the kernel, and demonstrate that the firewall is working as expected. You can use the following command to generate UDP packets to 8.8.8.8, which is Google's DNS server. If your firewall works, your request will be blocked; otherwise, you will get a response.

On the VM Terminal type the following command to make sure www.example.com is reachable, if it is not consider changing 8.8.8.8 to 8.8.4.4

Command:

$ **dig @8.8.8.8 www.example.com**



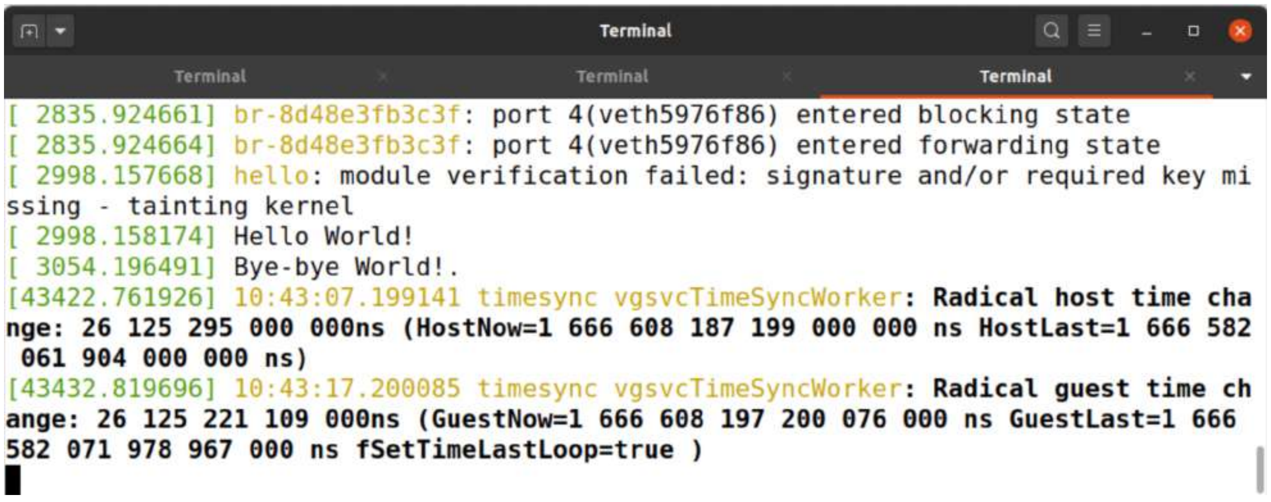
```
PES1UG20CS243:Mahika~/.../kernel_module$>dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42356
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        21372   IN      A       93.184.216.34

;; Query time: 27 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Oct 24 06:44:03 EDT 2022
;; MSG SIZE  rcvd: 60

PES1UG20CS243:Mahika~/.../kernel_module$>
```

On using dig for googles's DNS server, we can see that it generates a response, before the firewall is implemented.

Open a new terminal to view kernel messages and execute the following -

Command:

$ sudo dmesg -k -w



```
[ 2835.924661] br-8d48e3fb3c3f: port 4(veth5976f86) entered blocking state
[ 2835.924664] br-8d48e3fb3c3f: port 4(veth5976f86) entered forwarding state
[ 2998.157668] hello: module verification failed: signature and/or required key mi
ssing - tainting kernel
[ 2998.158174] Hello World!
[ 3054.196491] Bye-bye World!.
[43422.761926] 10:43:07.199141 timesync vgsvcTimeSyncWorker: Radical host time cha
nge: 26 125 295 000 000ns (HostNow=1 666 608 187 199 000 000 ns HostLast=1 666 582
 061 904 000 000 ns)
[43432.819696] 10:43:17.200085 timesync vgsvcTimeSyncWorker: Radical guest time ch
ange: 26 125 221 109 000ns (GuestNow=1 666 608 197 200 076 000 ns GuestLast=1 666
582 071 978 967 000 ns fSetTimeLastLoop=true )
```

This is the current kernel buffer.

Now on the earlier Terminal Window execute the following to insert the kernel object.

Note - Please change the make file, 'uncomment' the respective names (seedFilter, seedPrint, seedBlock) based on the task

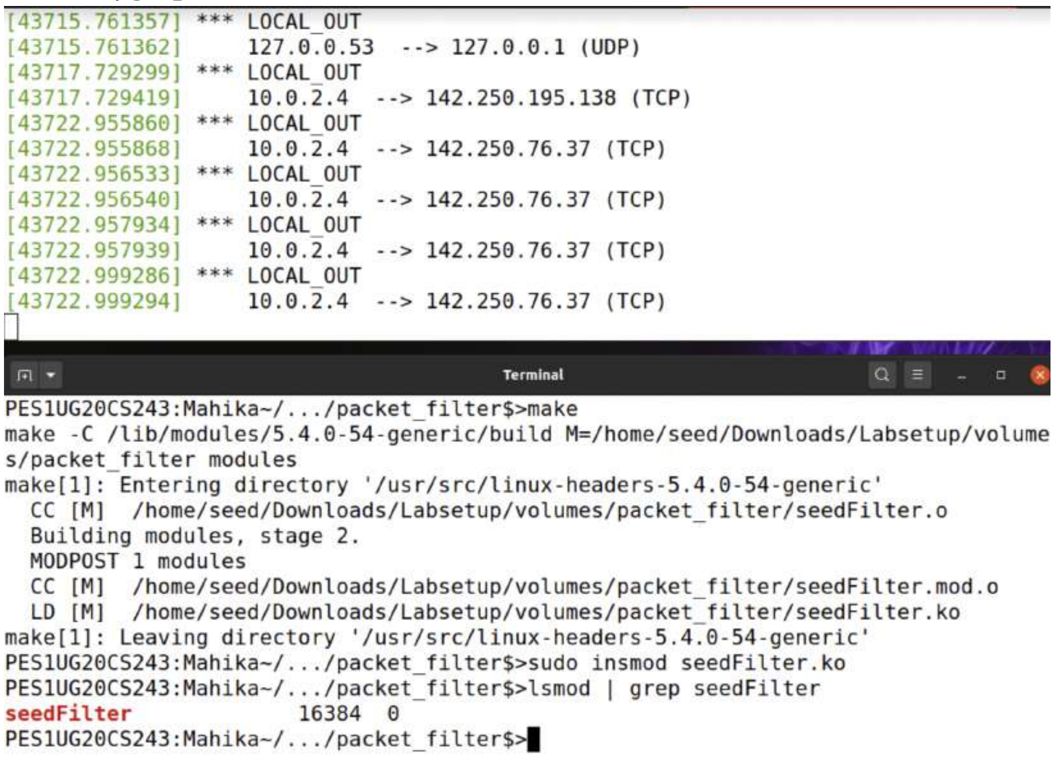
Uncomment - 'obj-m += seedFilter.o' and comment the other two.

Command:

$ make

$ sudo insmod seedFilter.ko

$ lsmod | grep seedFilter



```
[43715.761357] *** LOCAL_OUT
[43715.761362]     127.0.0.53  --> 127.0.0.1 (UDP)
[43717.729299] *** LOCAL_OUT
[43717.729419]     10.0.2.4  --> 142.250.195.138 (TCP)
[43722.955860] *** LOCAL_OUT
[43722.955868]     10.0.2.4  --> 142.250.76.37 (TCP)
[43722.956533] *** LOCAL_OUT
[43722.956540]     10.0.2.4  --> 142.250.76.37 (TCP)
[43722.957934] *** LOCAL_OUT
[43722.957939]     10.0.2.4  --> 142.250.76.37 (TCP)
[43722.999286] *** LOCAL_OUT
[43722.999294]     10.0.2.4  --> 142.250.76.37 (TCP)
```

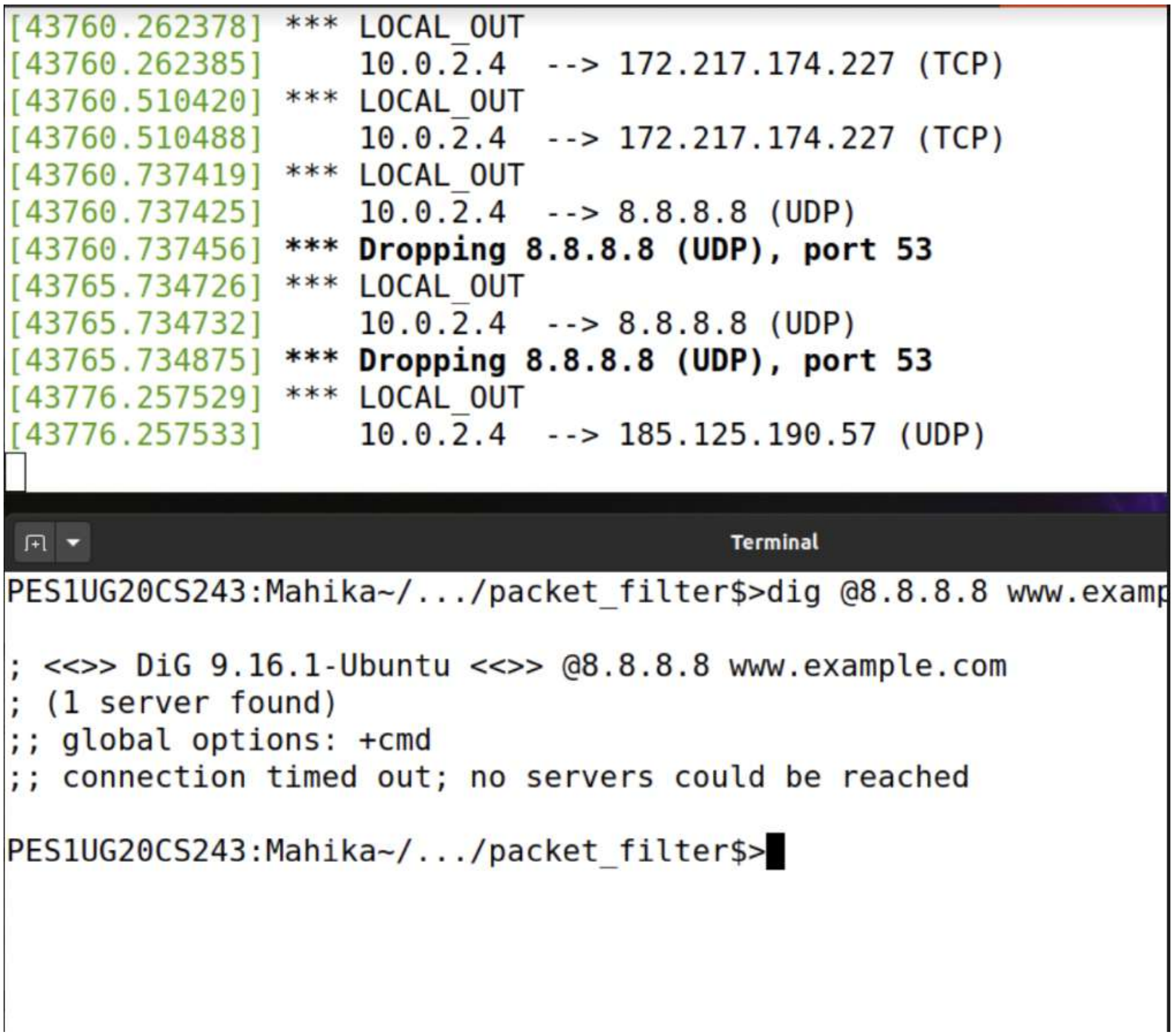
```
PES1UG20CS243:Mahika~/.../packet_filter$>make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/volume
s/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG20CS243:Mahika~/.../packet_filter$>sudo insmod seedFilter.ko
PES1UG20CS243:Mahika~/.../packet_filter$>lsmod | grep seedFilter
seedFilter             16384  0
PES1UG20CS243:Mahika~/.../packet_filter$>
```

On inserting the seedFilter.ko module, in the kernel buffer displays the message which tells us that our requests are directed to the LOCAL_OUT hook.

After inserting execute the below and notice the difference -

Command:

$ **dig @8.8.8.8 www.example.com**



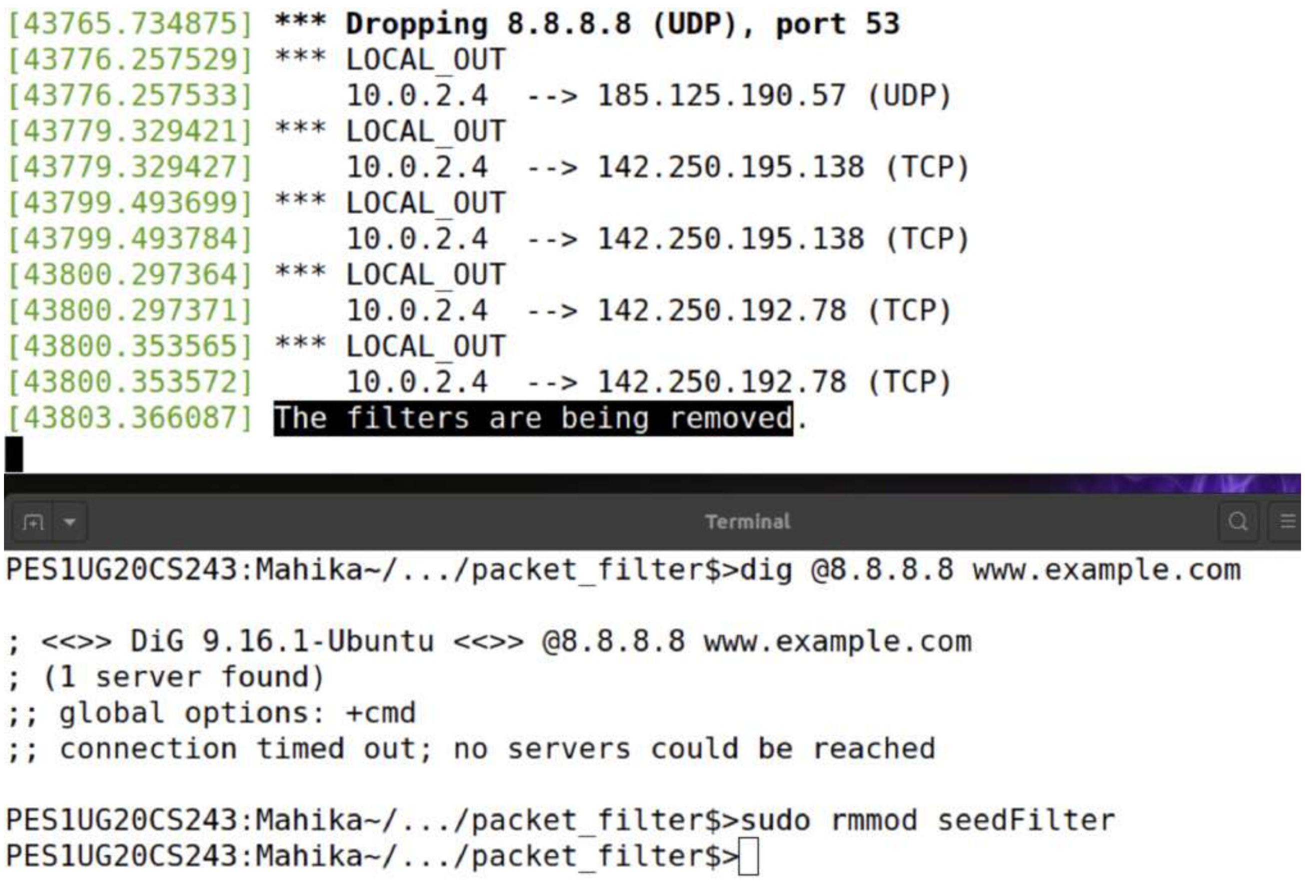
```
[43760.262378] *** LOCAL_OUT
[43760.262385]      10.0.2.4  --> 172.217.174.227 (TCP)
[43760.510420] *** LOCAL_OUT
[43760.510488]      10.0.2.4  --> 172.217.174.227 (TCP)
[43760.737419] *** LOCAL_OUT
[43760.737425]      10.0.2.4  --> 8.8.8.8 (UDP)
[43760.737456] *** Dropping 8.8.8.8 (UDP), port 53
[43765.734726] *** LOCAL_OUT
[43765.734732]      10.0.2.4  --> 8.8.8.8 (UDP)
[43765.734875] *** Dropping 8.8.8.8 (UDP), port 53
[43776.257529] *** LOCAL_OUT
[43776.257533]      10.0.2.4  --> 185.125.190.57 (UDP)
```

```
PES1UG20CS243:Mahika~/.../packet_filter$>dig @8.8.8.8 www.examp

; <<>> 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

PES1UG20CS243:Mahika~/.../packet_filter$>
```

On using the dig command for google's server now, we can see that we dont receive a response, and in the kernel buffer we can see that our request packets are dropped at the LOCAL_OUT hook. This means that the firewall is effective.

Remove the module by executing **-**

$ sudo rmmod seedFilter



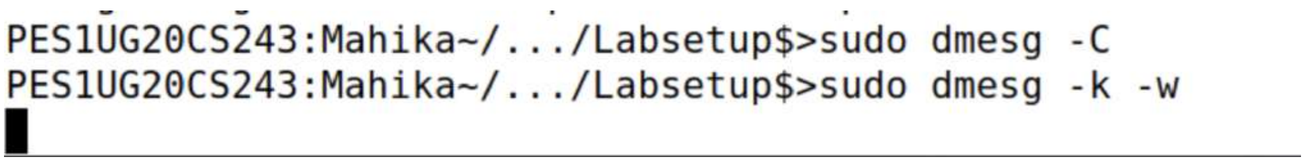
```
[43765.734875] *** Dropping 8.8.8.8 (UDP), port 53
[43776.257529] *** LOCAL_OUT
[43776.257533]     10.0.2.4  --> 185.125.190.57 (UDP)
[43779.329421] *** LOCAL_OUT
[43779.329427]     10.0.2.4  --> 142.250.195.138 (TCP)
[43799.493699] *** LOCAL_OUT
[43799.493784]     10.0.2.4  --> 142.250.195.138 (TCP)
[43800.297364] *** LOCAL_OUT
[43800.297371]     10.0.2.4  --> 142.250.192.78 (TCP)
[43800.353565] *** LOCAL_OUT
[43800.353572]     10.0.2.4  --> 142.250.192.78 (TCP)
[43803.366087] The filters are being removed.
```

```
PES1UG20CS243:Mahika~/.../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

PES1UG20CS243:Mahika~/.../packet_filter$>sudo rmmod seedFilter
PES1UG20CS243:Mahika~/.../packet_filter$>
```

On using rmmod the module is removed. The filters are also removed.

To clear the kernel messages execute -

$ dmesg -C



```
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -C
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -k -w
```

2. Hook the printInfo function to all of the netfilter hooks. Here are the macros of the hook numbers. Using your experiment results to help explain at what condition each of the hook functions be invoked.

   NF_INET_PRE_ROUTING

   NF_INET_LOCAL_IN

   NF_INET_FORWARD

   NF_INET_LOCAL_OUT

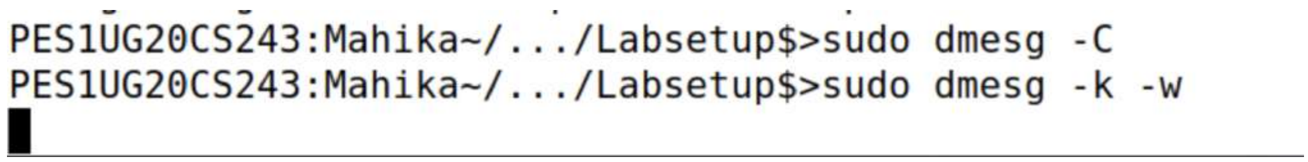
   NF_INET_POST_ROUTING

Similar to the previous task, open two terminal windows on your VM Host - one for viewing the kernel messages and the other for inserting the module.

On one window execute the following to view kernel messages -

Command:

$ sudo dmesg -k -w



```
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -C
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -k -w
```

Change the makefile as previously mentioned -

Uncomment - ‘obj-m += seedPrint.o’ and comment the other two.

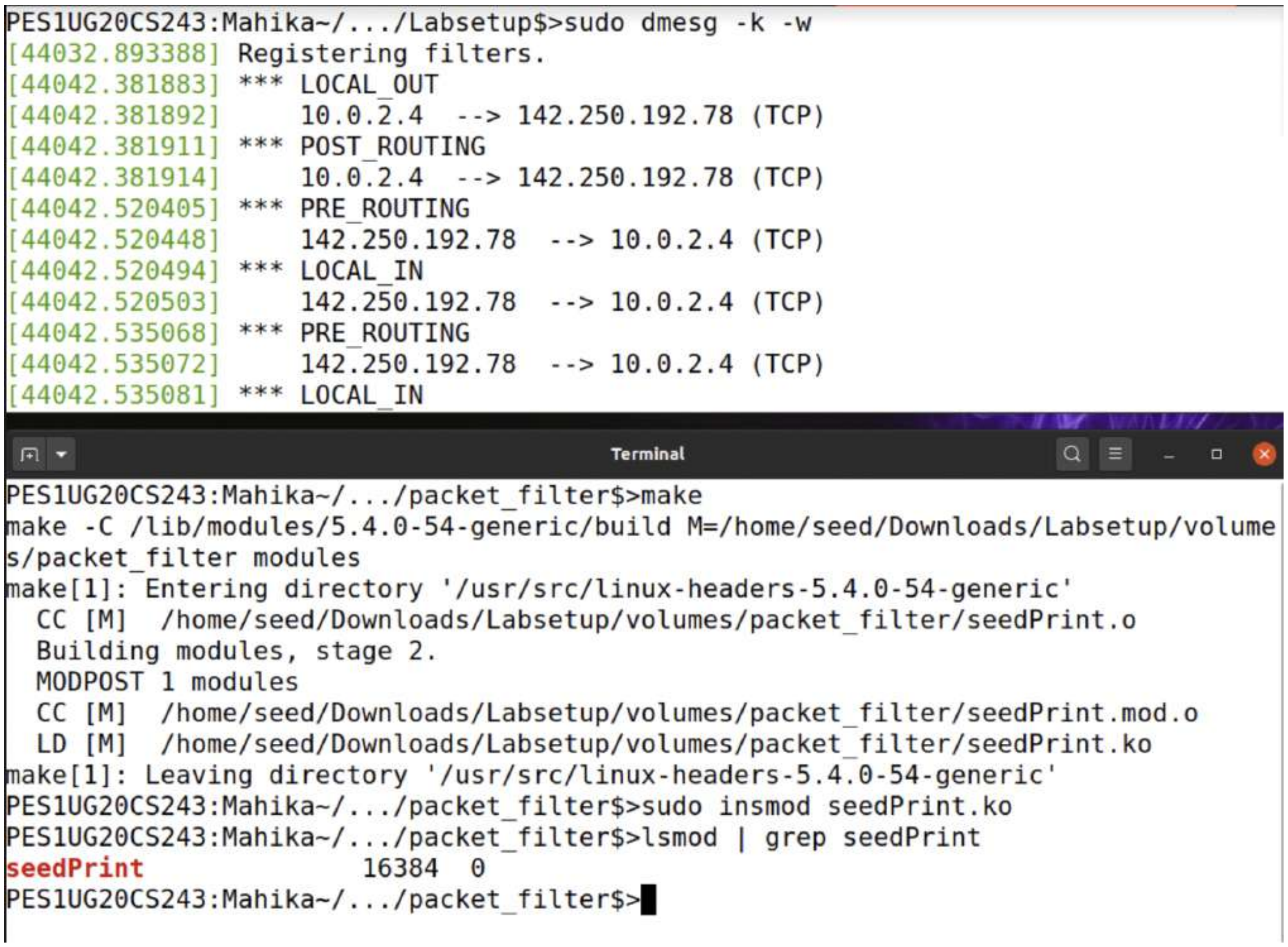
Now on the other terminal window, insert the kernel module -

Command:

$ make

$ sudo insmod seedPrint.ko

$ lsmod | grep seedPrint



```
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -k -w
[44032.893388] Registering filters.
[44042.381883] *** LOCAL_OUT
[44042.381892]      10.0.2.4  --> 142.250.192.78 (TCP)
[44042.381911] *** POST_ROUTING
[44042.381914]      10.0.2.4  --> 142.250.192.78 (TCP)
[44042.520405] *** PRE_ROUTING
[44042.520448]      142.250.192.78  --> 10.0.2.4 (TCP)
[44042.520494] *** LOCAL_IN
[44042.520503]      142.250.192.78  --> 10.0.2.4 (TCP)
[44042.535068] *** PRE_ROUTING
[44042.535072]      142.250.192.78  --> 10.0.2.4 (TCP)
[44042.535081] *** LOCAL_IN
```

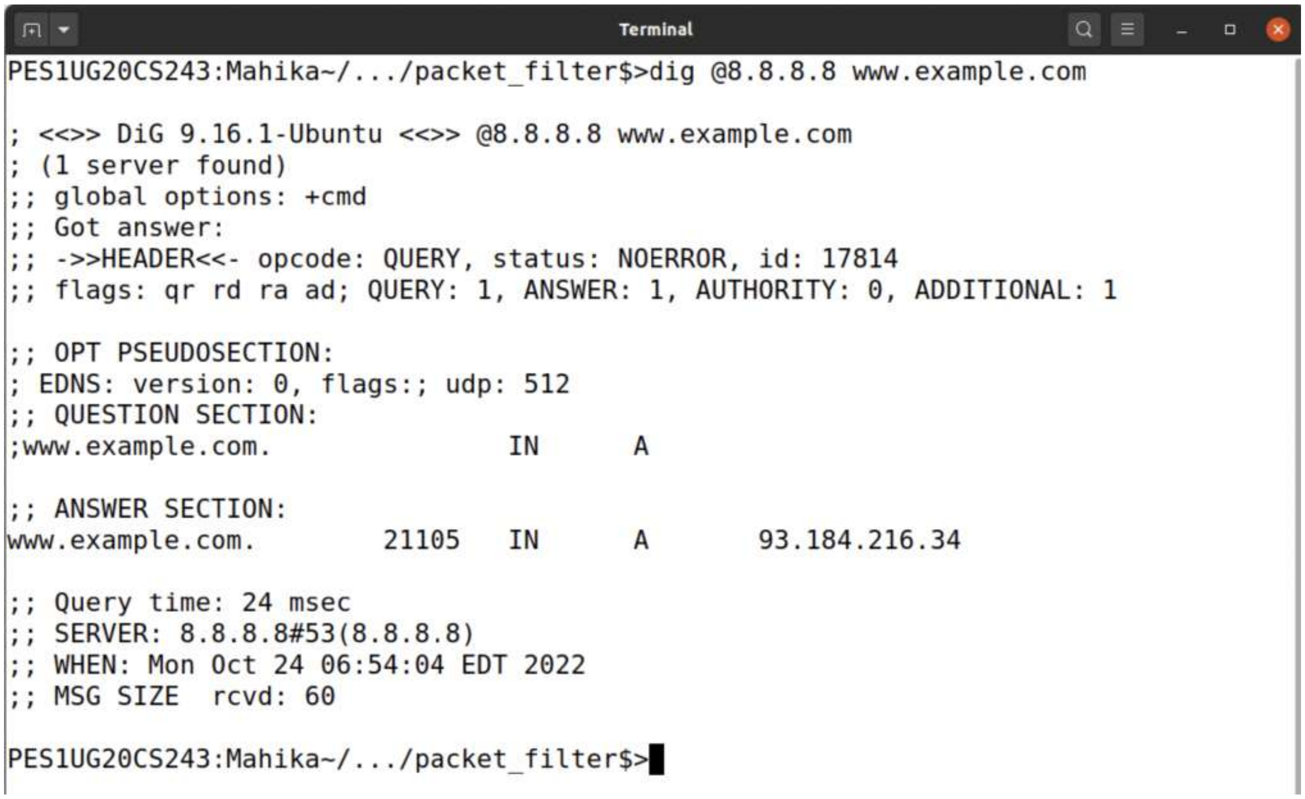
```
PES1UG20CS243:Mahika~/.../packet_filter$>make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/volume
s/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedPrint.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedPrint.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedPrint.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG20CS243:Mahika~/.../packet_filter$>sudo insmod seedPrint.ko
PES1UG20CS243:Mahika~/.../packet_filter$>lsmod | grep seedPrint
seedPrint              16384  0
PES1UG20CS243:Mahika~/.../packet_filter$>
```

On inserting the module, the filters are registered and we can see the hooks displayed along with the packet information. We also see from lsmod command that the size of the inserted module is 16384 bytes.

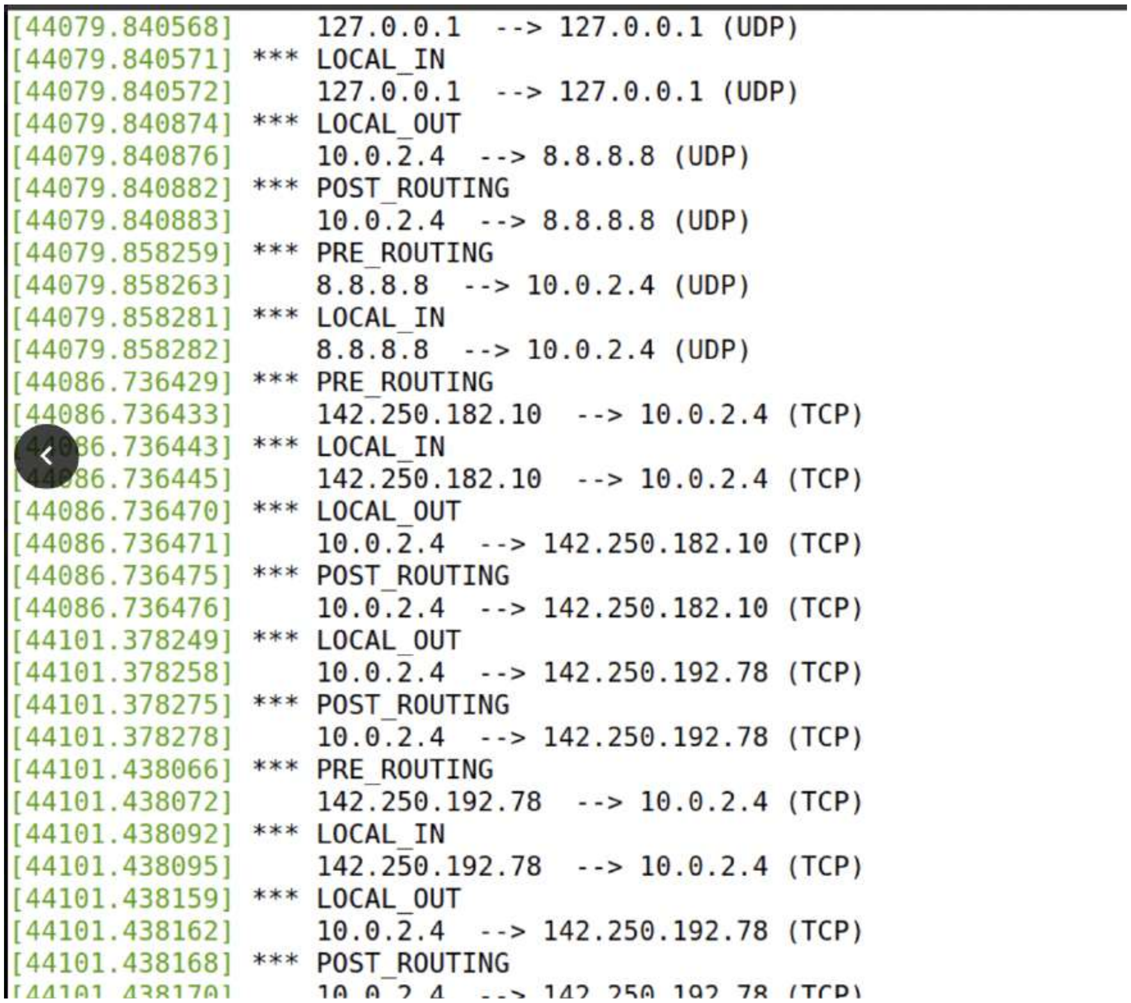
After inserting execute the below and notice the difference -

Command:

$ dig @8.8.8.8 www.example.com



```
PES1UG20CS243:Mahika~/.../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: 17814
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        21105   IN      A       93.184.216.34

;; Query time: 24 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Oct 24 06:54:04 EDT 2022
;; MSG SIZE  rcvd: 60

PES1UG20CS243:Mahika~/.../packet_filter$>
```



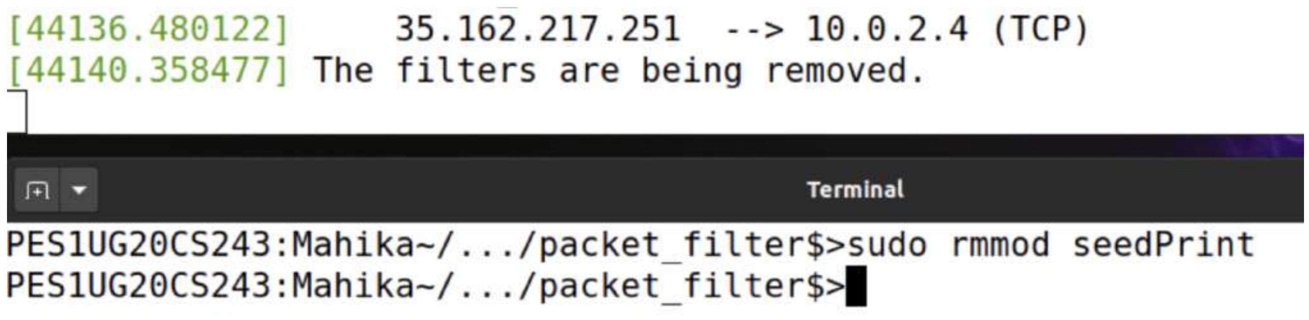
```
[44079.840568]     127.0.0.1  --> 127.0.0.1 (UDP)
[44079.840571] *** LOCAL_IN
[44079.840572]     127.0.0.1  --> 127.0.0.1 (UDP)
[44079.840874] *** LOCAL_OUT
[44079.840876]     10.0.2.4  --> 8.8.8.8 (UDP)
[44079.840882] *** POST_ROUTING
[44079.840883]     10.0.2.4  --> 8.8.8.8 (UDP)
[44079.858259] *** PRE_ROUTING
[44079.858263]     8.8.8.8  --> 10.0.2.4 (UDP)
[44079.858281] *** LOCAL_IN
[44079.858282]     8.8.8.8  --> 10.0.2.4 (UDP)
[44086.736429] *** PRE_ROUTING
[44086.736433]     142.250.182.10  --> 10.0.2.4 (TCP)
[44086.736443] *** LOCAL_IN
[44086.736445]     142.250.182.10  --> 10.0.2.4 (TCP)
[44086.736470] *** LOCAL_OUT
[44086.736471]     10.0.2.4  --> 142.250.182.10 (TCP)
[44086.736475] *** POST_ROUTING
[44086.736476]     10.0.2.4  --> 142.250.182.10 (TCP)
[44101.378249] *** LOCAL_OUT
[44101.378258]     10.0.2.4  --> 142.250.192.78 (TCP)
[44101.378275] *** POST_ROUTING
[44101.378278]     10.0.2.4  --> 142.250.192.78 (TCP)
[44101.438066] *** PRE_ROUTING
[44101.438072]     142.250.192.78  --> 10.0.2.4 (TCP)
[44101.438092] *** LOCAL_IN
[44101.438095]     142.250.192.78  --> 10.0.2.4 (TCP)
[44101.438159] *** LOCAL_OUT
[44101.438162]     10.0.2.4  --> 142.250.192.78 (TCP)
[44101.438168] *** POST_ROUTING
[44101.438170]     10.0.2.4  --> 142.250.192.78 (TCP)
```

On inserting seedPrint module, we can see that on using dig command our packets are sent through all the hooks. We can see the request and response packets travelling through the

PRE_ROUTING, LOCAL_IN, LOCAL_OUT and POST_ROUTING hooks between the google servers and the host machine.

Remove the module by executing -

$ sudo rmmod seedPrint



```
[44136.480122]     35.162.217.251  --> 10.0.2.4 (TCP)
[44140.358477] The filters are being removed.
```

```
PES1UG20CS243:Mahika~/.../packet_filter$>sudo rmmod seedPrint
PES1UG20CS243:Mahika~/.../packet_filter$>
```

3. Implement two more hooks to achieve the following:

   (1) preventing other computers to ping the VM, and

   (2) preventing other computers from telnetting into the VM.

   Please implement two different hook functions, but register them to the same netfilter hook. You should decide what hook to use. Telnet's default port is TCP port 23. To test it, you can start the containers, go to 10.9.0.5, run the following commands (10.9.0.1 is the IP address assigned to the VM; for the sake of simplicity, you can hardcode this IP address in your firewall rules):

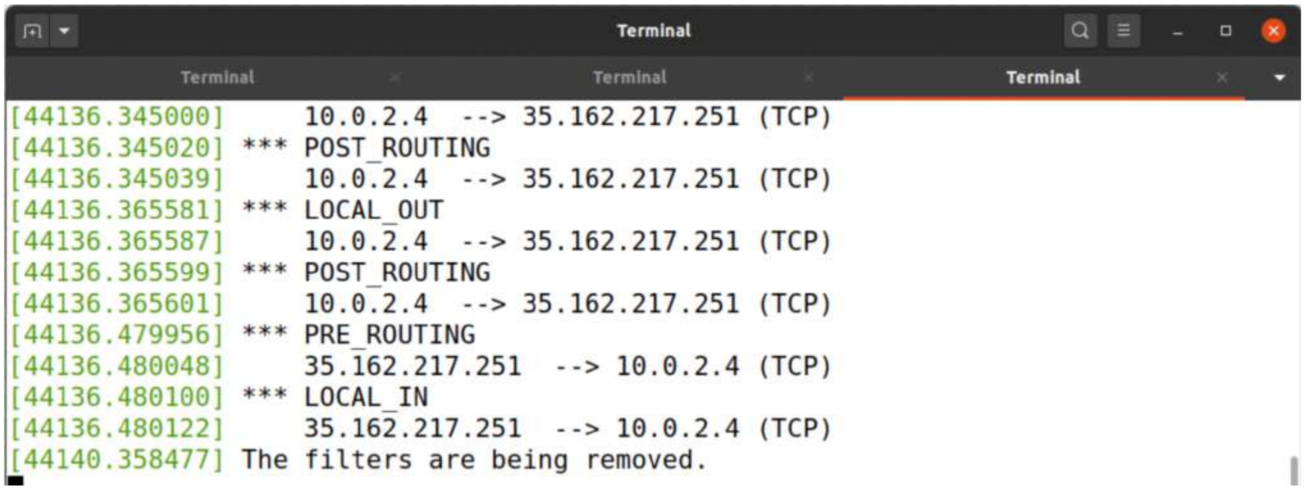
   For this Task you need the docker container - Open the Host 10.9.0.5

Similar to the previous task,this time we need to open three terminal windows on your VM Host - one for viewing the kernel messages, one for inserting the module and the docker container that acts as an external machine.

On one window execute the following to view kernel messages -

Command:

$ sudo dmesg -k -w



```
[44136.345000]      10.0.2.4  --> 35.162.217.251 (TCP)
[44136.345020] *** POST_ROUTING
[44136.345039]      10.0.2.4  --> 35.162.217.251 (TCP)
[44136.365581] *** LOCAL_OUT
[44136.365587]      10.0.2.4  --> 35.162.217.251 (TCP)
[44136.365599] *** POST_ROUTING
[44136.365601]      10.0.2.4  --> 35.162.217.251 (TCP)
[44136.479956] *** PRE_ROUTING
[44136.480048]      35.162.217.251  --> 10.0.2.4 (TCP)
[44136.480100] *** LOCAL_IN
[44136.480122]      35.162.217.251  --> 10.0.2.4 (TCP)
[44140.358477] The filters are being removed.
```

The kernel buffer is displayed

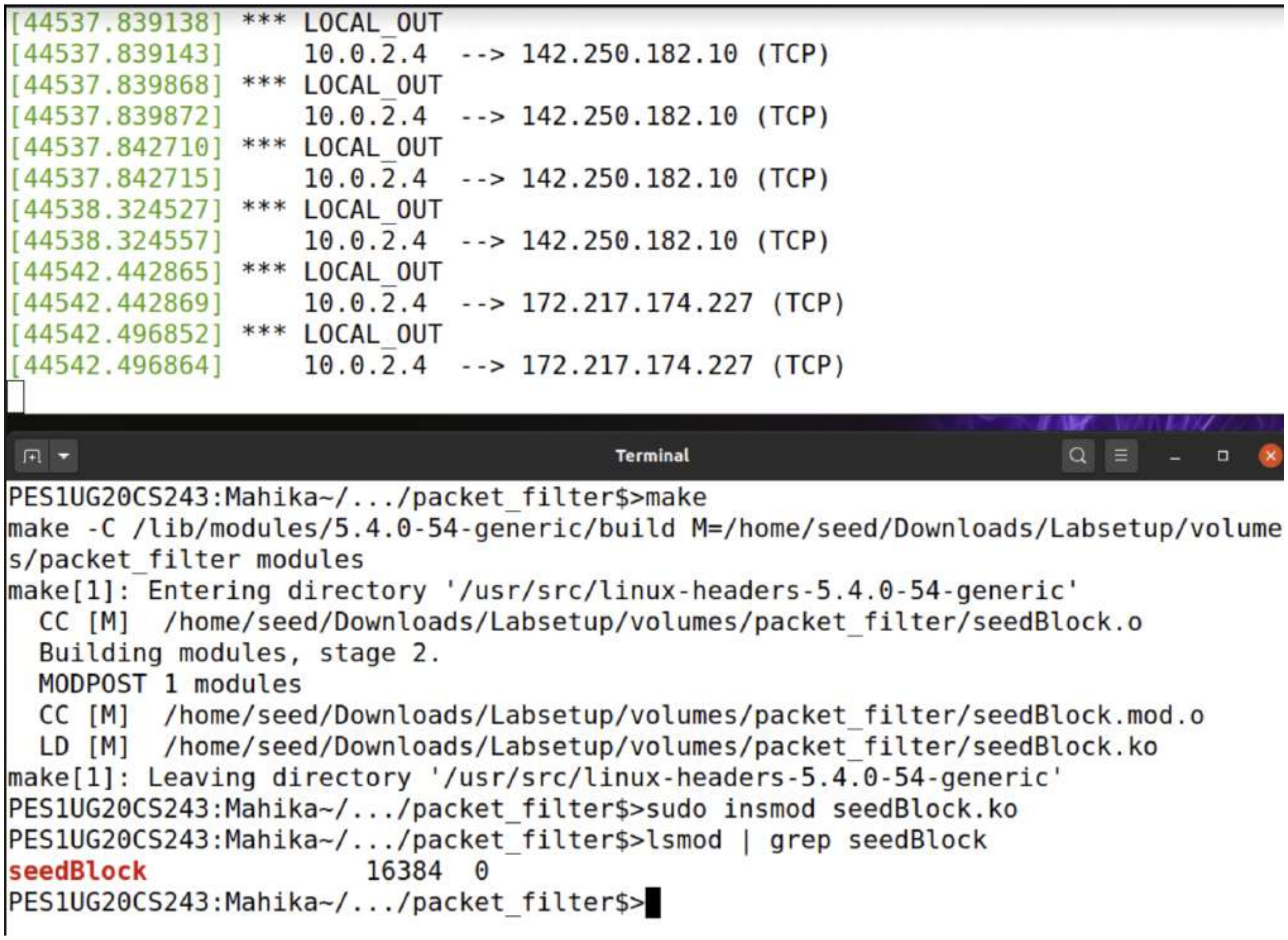
Change the makefile as previously mentioned -

Uncomment - ‘obj-m += seedBlock.o’ and comment the other two.

Now on the other terminal window, insert the kernel module -

Command:

```
$ make
$ sudo insmod seedBlock.ko
$ lsmod | grep seedBlock
```


```
[44537.839138] *** LOCAL_OUT
[44537.839143]     10.0.2.4  --> 142.250.182.10 (TCP)
[44537.839868] *** LOCAL_OUT
[44537.839872]     10.0.2.4  --> 142.250.182.10 (TCP)
[44537.842710] *** LOCAL_OUT
[44537.842715]     10.0.2.4  --> 142.250.182.10 (TCP)
[44538.324527] *** LOCAL_OUT
[44538.324557]     10.0.2.4  --> 142.250.182.10 (TCP)
[44542.442865] *** LOCAL_OUT
[44542.442869]     10.0.2.4  --> 172.217.174.227 (TCP)
[44542.496852] *** LOCAL_OUT
[44542.496864]     10.0.2.4  --> 172.217.174.227 (TCP)
```

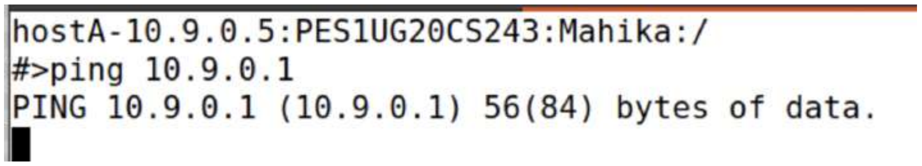
```
PES1UG20CS243:Mahika~/.../packet_filter$>make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/volume
s/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedBlock.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedBlock.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedBlock.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG20CS243:Mahika~/.../packet_filter$>sudo insmod seedBlock.ko
PES1UG20CS243:Mahika~/.../packet_filter$>lsmod | grep seedBlock
seedBlock              16384  0
PES1UG20CS243:Mahika~/.../packet_filter$>
```

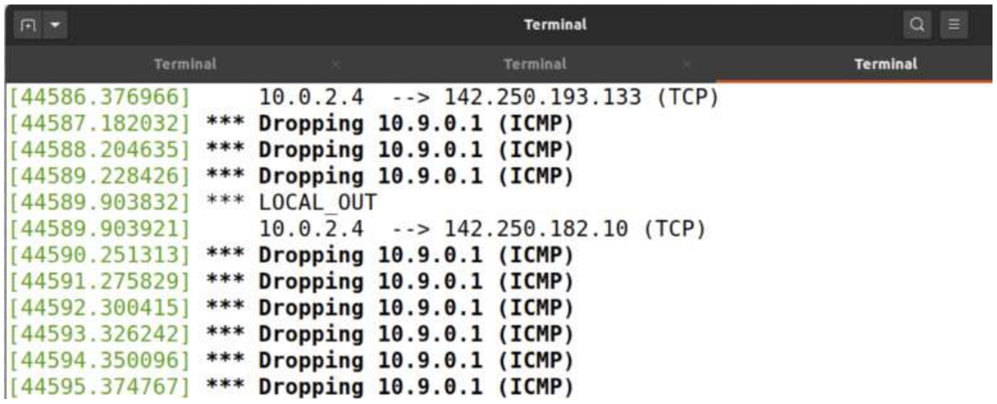
The seedBlock module is inserted and and we can see that the size of the module is 16384 bytes. The kernel buffer shows the LOCAL_OUT hook passing tcp packets from host to an IP address.

On the Host A - 10.9.0.5 terminal try -

Command:

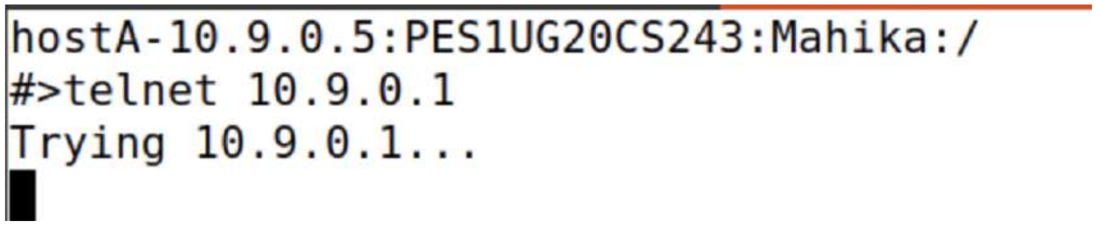
# ping 10.9.0.1


```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
```

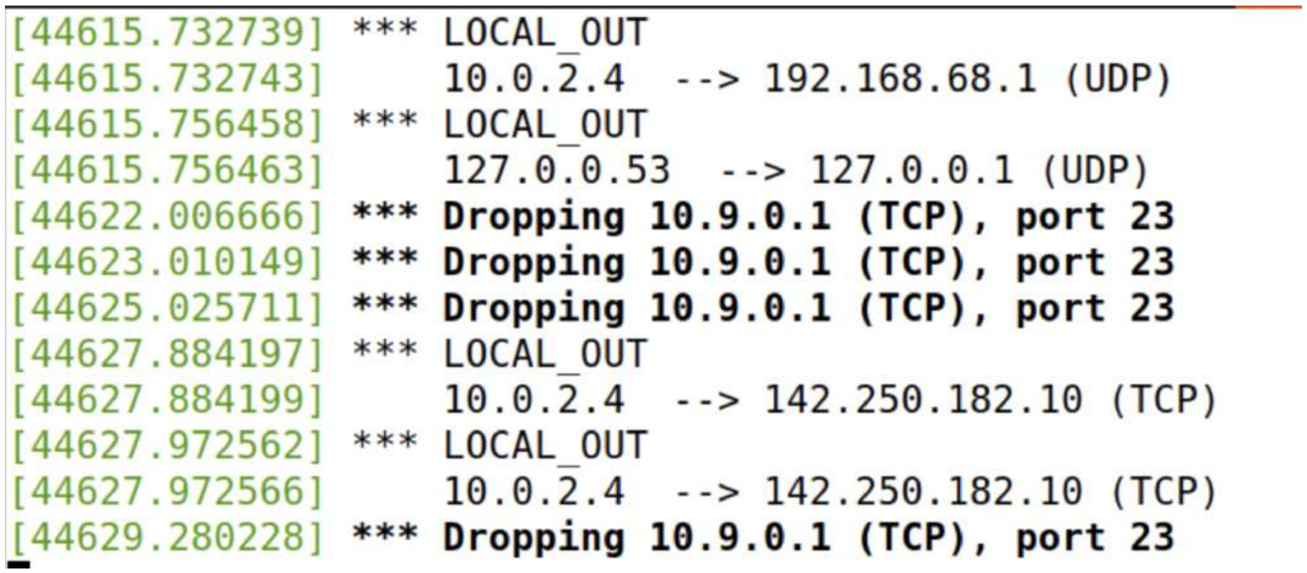

```
[44586.376966]     10.0.2.4  --> 142.250.193.133 (TCP)
[44587.182032] *** Dropping 10.9.0.1 (ICMP)
[44588.204635] *** Dropping 10.9.0.1 (ICMP)
[44589.228426] *** Dropping 10.9.0.1 (ICMP)
[44589.903832] *** LOCAL_OUT
[44589.903921]     10.0.2.4  --> 142.250.182.10 (TCP)
[44590.251313] *** Dropping 10.9.0.1 (ICMP)
[44591.275829] *** Dropping 10.9.0.1 (ICMP)
[44592.300415] *** Dropping 10.9.0.1 (ICMP)
[44593.326242] *** Dropping 10.9.0.1 (ICMP)
[44594.350096] *** Dropping 10.9.0.1 (ICMP)
[44595.374767] *** Dropping 10.9.0.1 (ICMP)
```

We can see that on attempting to ping a 10.9.0.1, the ping request is not sent, as due to the firewall ICMP requests are blocked, and are dropped at the LOCAL_OUT hook.

# telnet 10.9.0.1



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 10.9.0.1
Trying 10.9.0.1...
```

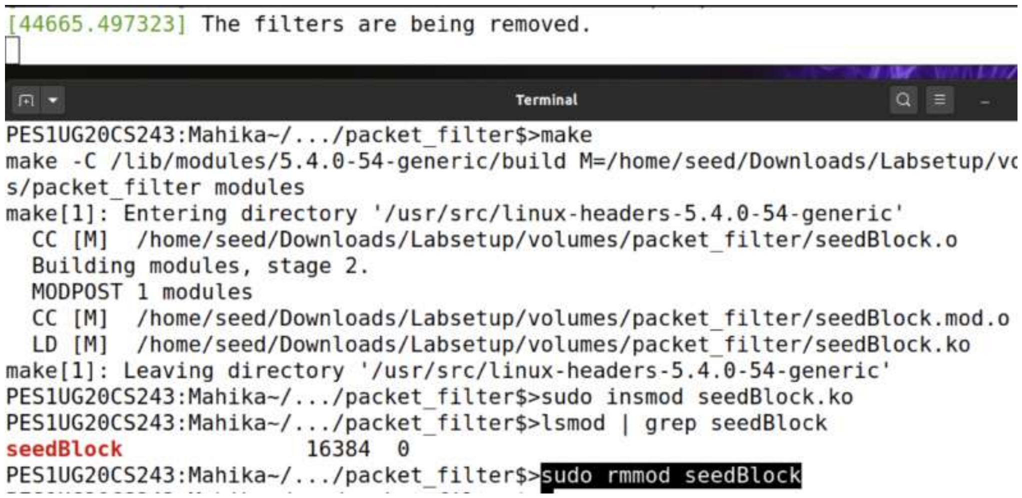


```
[44615.732739] *** LOCAL_OUT
[44615.732743]     10.0.2.4  --> 192.168.68.1 (UDP)
[44615.756458] *** LOCAL_OUT
[44615.756463]     127.0.0.53  --> 127.0.0.1 (UDP)
[44622.006666] *** Dropping 10.9.0.1 (TCP), port 23
[44623.010149] *** Dropping 10.9.0.1 (TCP), port 23
[44625.025711] *** Dropping 10.9.0.1 (TCP), port 23
[44627.884197] *** LOCAL_OUT
[44627.884199]     10.0.2.4  --> 142.250.182.10 (TCP)
[44627.972562] *** LOCAL_OUT
[44627.972566]     10.0.2.4  --> 142.250.182.10 (TCP)
[44629.280228] *** Dropping 10.9.0.1 (TCP), port 23
```

On attempting to telnet 10.9.0.1, due to the seedBlock module firewall, telnet connection to 10.9.0.1 is blocked, and hence all tcp packets to 10.9.0.1 on port 23 are dropped at the LOCAL_OUT hook.

Remove the module by executing **-**

$ sudo rmmod seedBlock



```
[44665.497323] The filters are being removed.
```

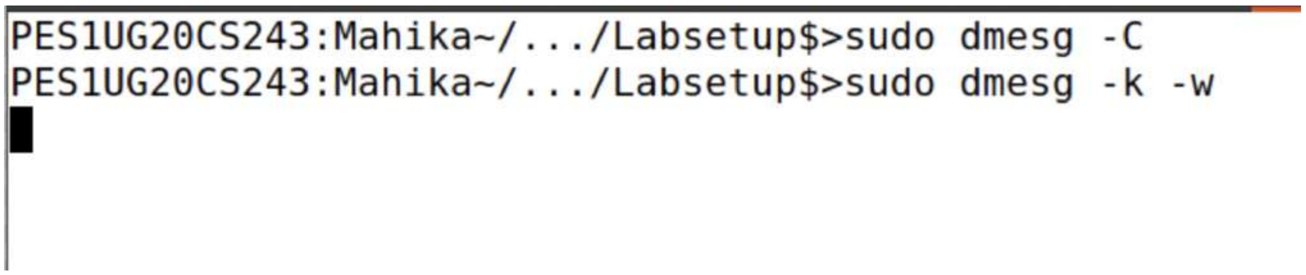
Terminal

```
PES1UG20CS243:Mahika~/.../packet_filter$>make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/vo
s/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedBlock.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedBlock.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/volumes/packet_filter/seedBlock.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG20CS243:Mahika~/.../packet_filter$>sudo insmod seedBlock.ko
PES1UG20CS243:Mahika~/.../packet_filter$>lsmod | grep seedBlock
seedBlock              16384  0
PES1UG20CS243:Mahika~/.../packet_filter$>sudo rmmod seedBlock
```

The seedBlock module is removed along with the filters.

To clear the kernel messages execute -

$ dmesg -C



```
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -C
PES1UG20CS243:Mahika~/.../Labsetup$>sudo dmesg -k -w
```

The kernel buffer is cleared up.

Task 2: Experimenting with Stateless Firewall Rules

In the previous task, we had a chance to build a simple firewall using netfilter. Actually, Linux already has a built-in firewall, also based on netfilter. This firewall is called iptables. Technically, the kernel part implementation of the firewall is called Xtables, while iptables is a user-space program to configure the firewall. However, iptables is often used to refer to both the kernel-part implementation and the user-space program.

Background of iptables

In this task, we will use iptables to set up a firewall. The iptables firewall is designed not only to filter packets, but also to make changes to packets. To help manage these firewall rules for different purposes, iptables organizes all rules using a hierarchical structure: table, chain, and rules. There are several tables, each specifying the main purpose of the rules as shown in Table 1. For example, rules for packet filtering should be placed in the filter table, while rules for making changes to packets should be placed in the nat or mangle tables.

Each table contains several chains, each of which corresponds to a netfilter hook. Basically, each chain indicates where its rules are enforced. For example, rules on the FORWARD chain are enforced at the NF INET FORWARD hook, and rules on the INPUT chain are enforced at the NF INET LOCAL IN hook. Each chain contains a set of firewall rules that will be enforced. When we set up firewalls, we add rules to these chains. For example, if we would like to block all incoming telnet traffic, we would add a rule to the INPUT chain of the filter table. If we would like to redirect all incoming telnet traffic to a different port on a different host, basically doing port forwarding, we can add a rule to the INPUT chain of the mangle table, as we need to make changes to packets

Task 2.A: Protecting the Router

Before proceeding ahead with this task, make sure to **open all the docker containers**.

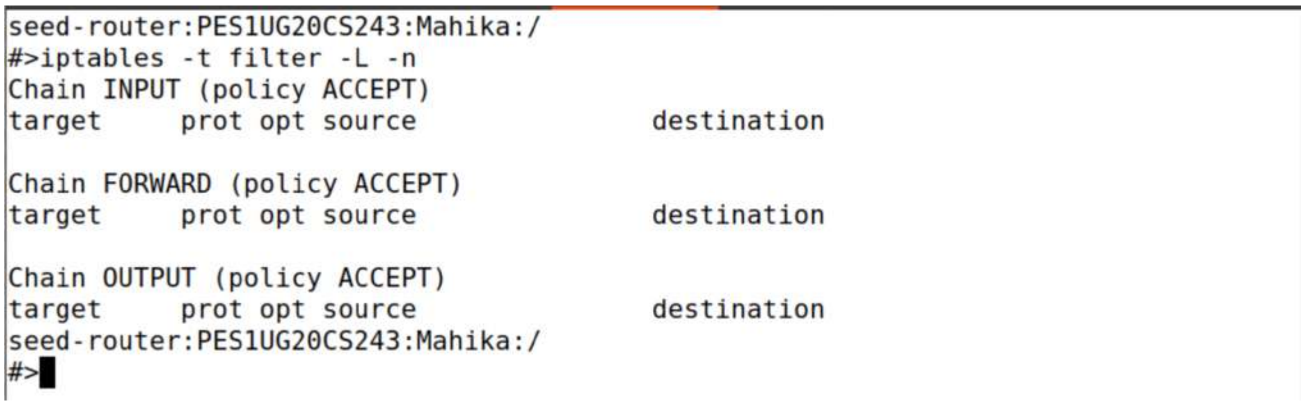
In this task, we will set up rules to prevent outside machines from accessing the router machine, except ping.

On the **Router Container (seed-router),** execute the following Iptables commands.

In order to view the current policies run the below command

Command:

# iptables -t filter -L -n



```
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
#>
```

This command displays the filter table and lists all the rules in the selected chain or table, along with the IP addresses and port numbers.

Now please execute the following iptables command on the router container, and then try to access it from 10.9.0.5.

On seed-router run -

Command:

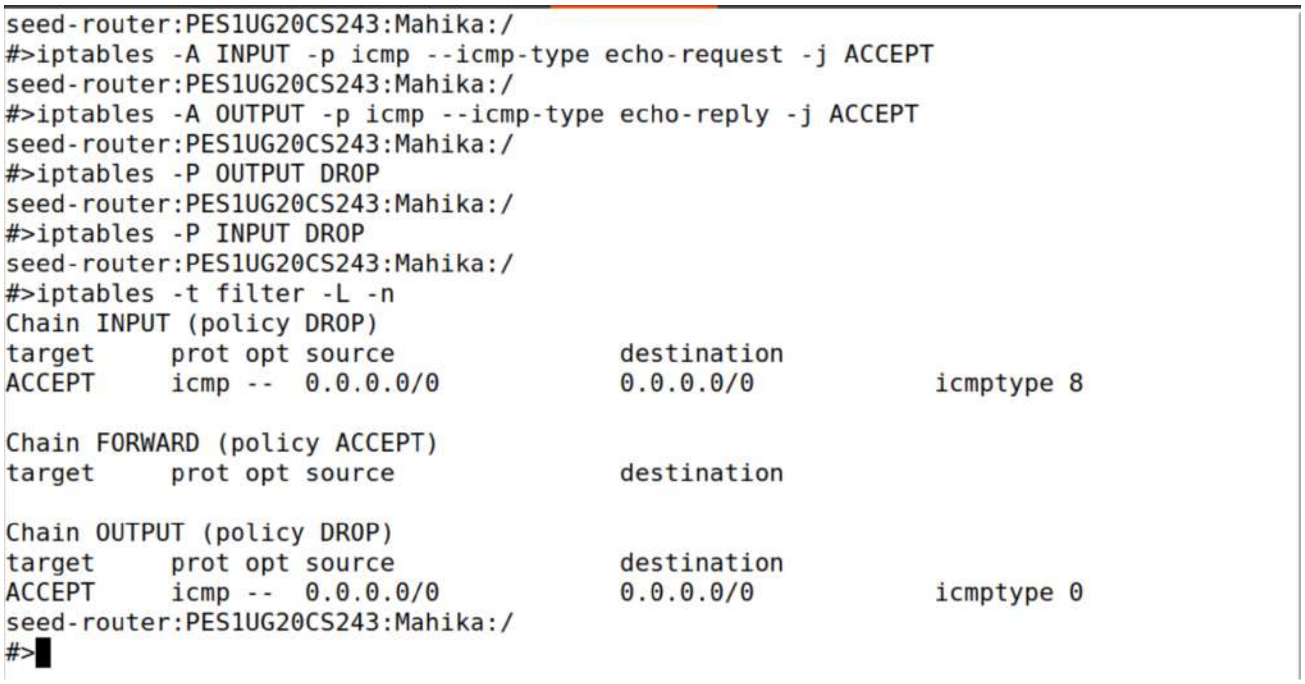
# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

# iptables -P OUTPUT DROP

# iptables -P INPUT DROP

# iptables -t filter -L -n



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P OUTPUT DROP
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P INPUT DROP
seed-router:PES1UG20CS243:Mahika:/
#>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

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
seed-router:PES1UG20CS243:Mahika:/
#>
```

The first rule appended is to allow icmp echo request packets.

The second rule appended is to allow output of icmp echo replies.

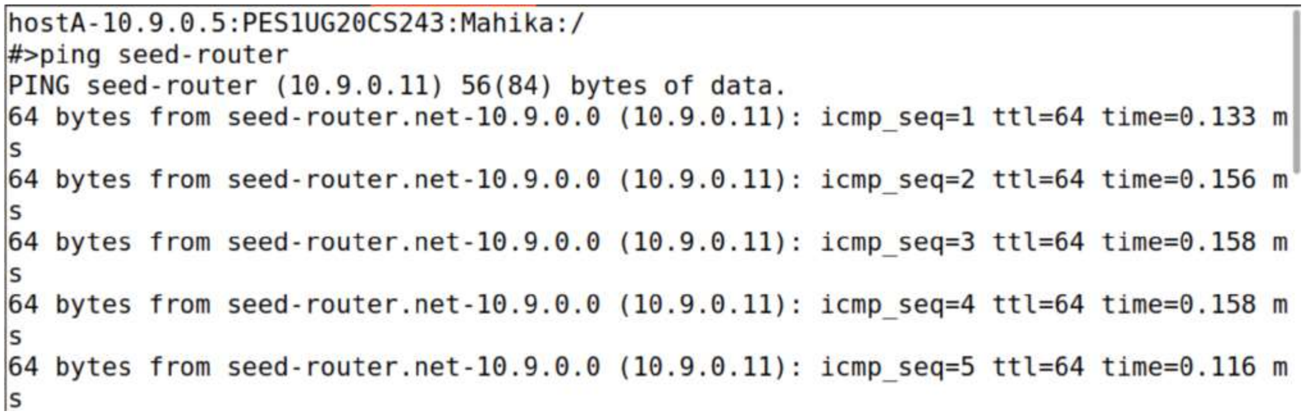
The third and fourth commands are used to update the policy of the chain to keep the default as DROP for every other input and output packets.

The iptable is displayed and we can see the rules updated.

Now try to access (ping and telnet) the router from Host A - 10.9.0.5

Command:

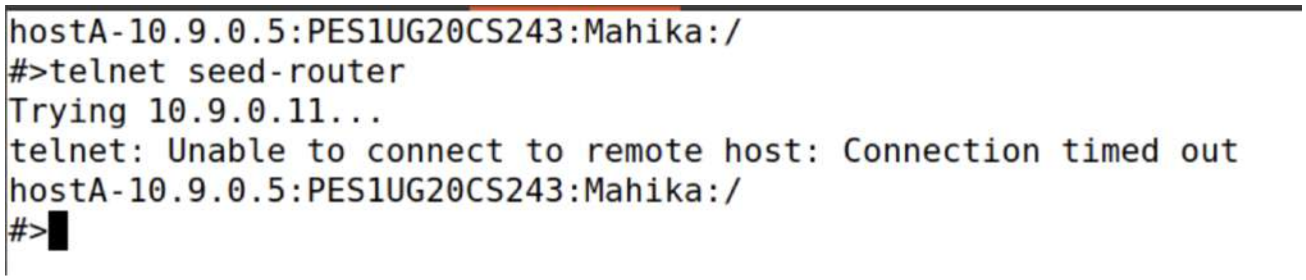
# ping seed-router



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.133 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.156 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.158 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.158 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.116 m
s
```

We can see that ping to the seed-router is possible as stated by the rules.

# telnet seed-router



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet seed-router
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

Telnet to the seed-router is not possible, as according to the rules in the iptable, only icmp packets are allowed, and all other packets are dropped by default.

Questions :

(1) Can you ping the router?

Ans: Yes, the router can be pinged.

(2) Can you telnet into the router (a telnet server is running on all the containers; an account called seed was created on them with a password dees).

Ans: A telnet connection to the router cannot be established as it policies state that all packets that are not icmp should be dropped by default.

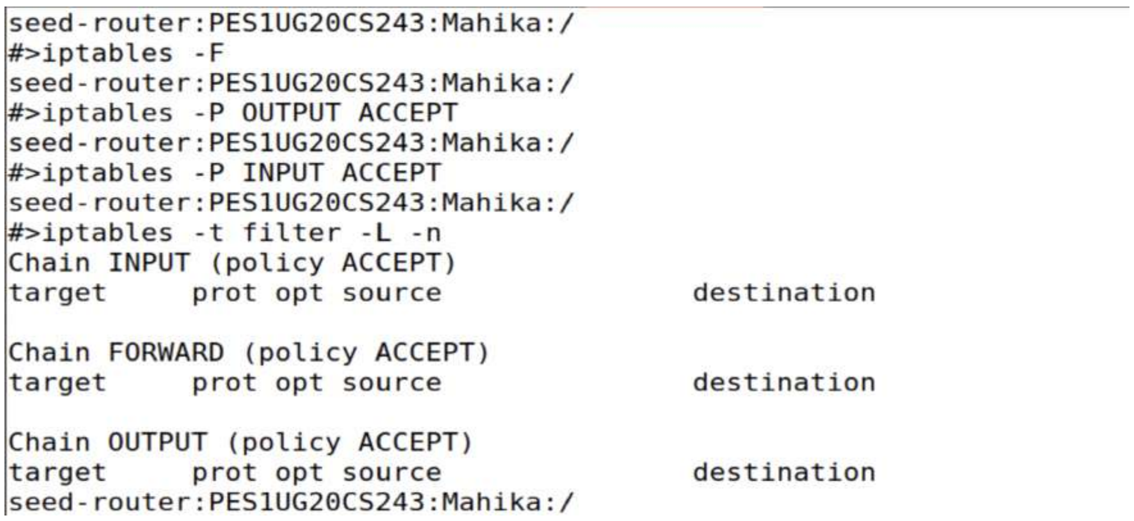
<u>Cleanup</u>

Before moving on to the next task, please restore the filter table to its original state by running the **following commands:**

# iptables -F

# iptables -P OUTPUT ACCEPT

# iptables -P INPUT ACCEPT


```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -F
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P OUTPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P INPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
```

The iptable rules are flushed, which means all the rules are deleted. The output and input policies are set to ACCEPT.

Task 2.B: Protecting the Internal Network

In this task, we will set up firewall rules on the router to protect the internal network 192.168.60.0/24. We need to use the FORWARD chain for this purpose.

The directions of packets in the INPUT and OUTPUT chains are clear: packets are either coming into (for INPUT) or going out (for OUTPUT). This is not true for the FORWARD chain, because it is bi-directional: packets going into the internal network or going out to the external network all go through this chain. To specify the direction, we can add the interface options using "-i xyz" (coming in from the xyz interface) and/or "-o xyz" (going out from the xyz interface). The interfaces for the internal and external networks are different. You can find out the interface names via the "ip addr" command.

In this task, we want to implement a firewall to protect the internal network. More specifically, we need to enforce the following restrictions on the ICMP traffic:

1. Outside hosts cannot ping internal hosts.
2. Outside hosts can ping the router.
3. Internal hosts can ping outside hosts.
4. All other packets between the internal and external networks should be blocked.

Execute the following iptables commands on the **seed-router container** -

Commands:

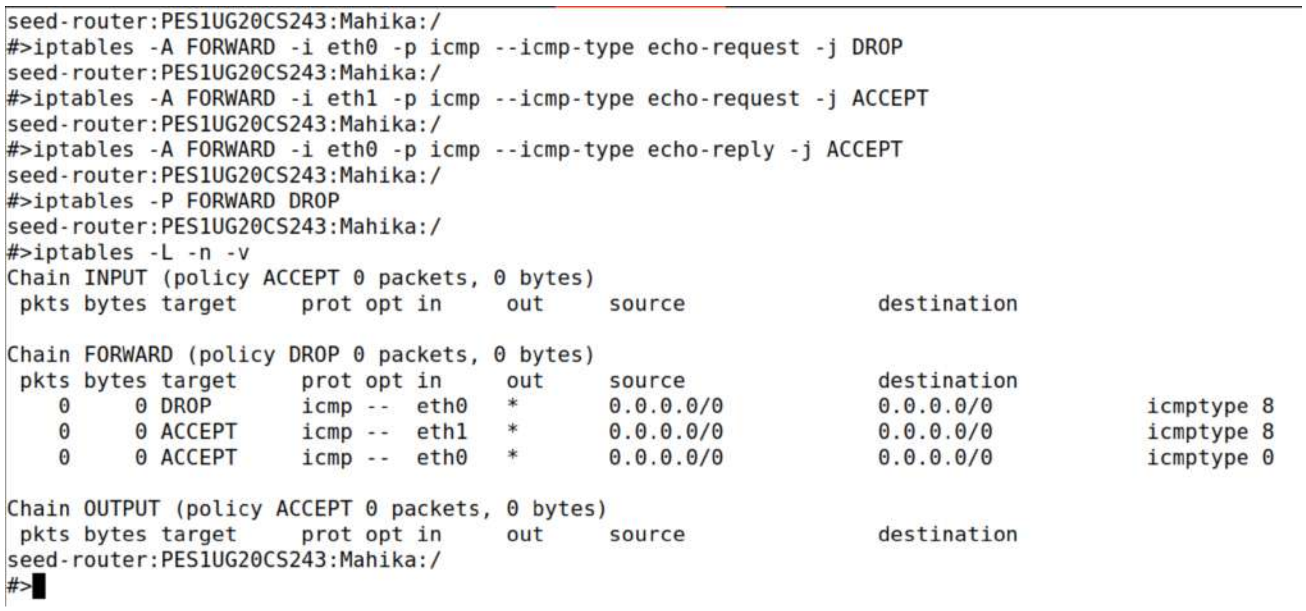
# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP

# 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

# iptables -L -n -v



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P FORWARD DROP
seed-router:PES1UG20CS243:Mahika:/
#>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 DROP       icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0            icmptype 8
    0     0 ACCEPT     icmp --  eth1   *       0.0.0.0/0            0.0.0.0/0            icmptype 8
    0     0 ACCEPT     icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
seed-router:PES1UG20CS243:Mahika:/
#>
```

The iptable rules are updated:
The first rule does not allow forwarding of icmp request packets from the eth0 interface.
In the 2nd rule ip forwarding of icmp request packet through eth1 interface is allowed.
In the 3rd rule ip forwarding of icmp reply packet through eth0 interface is allowed.
In the 4th command, the default forward policy is set to DROP
The iptables are displayed.

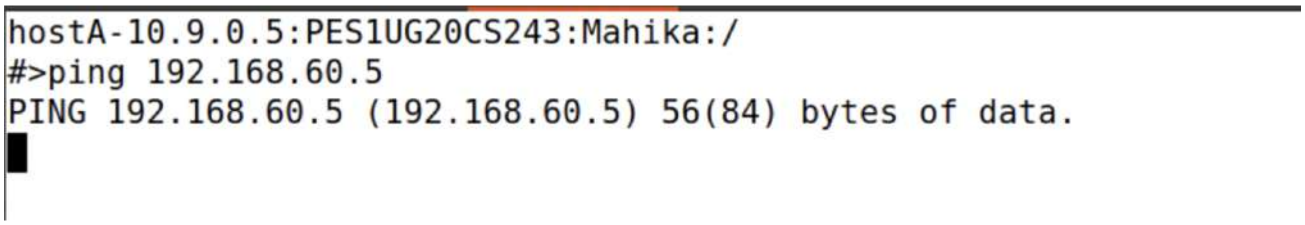
Now we shall see if these **restrictions have been enforced** in the network.

1. Outside hosts cannot ping internal hosts.

On Host A - 10.9.0.5 execute

Command:

# ping 192.168.60.5



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
█
```

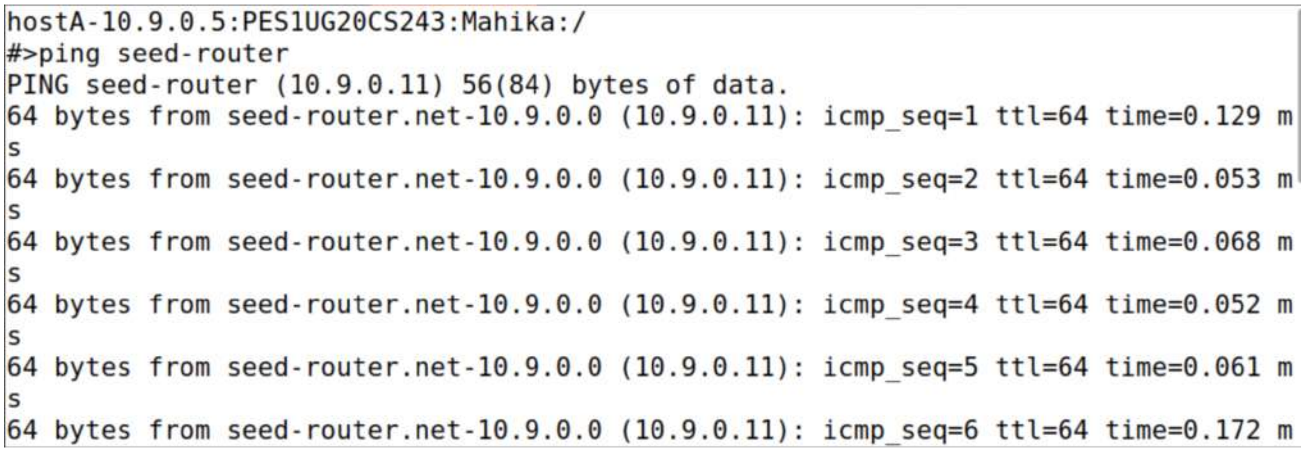
The ping does not go through as icmp request is not allowed on eth0 interface.

2. Outside hosts can ping the router.

On Host A - 10.9.0.5 execute

Command:

# ping seed-router



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.129 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.053 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.068 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.052 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.061 m
s
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=6 ttl=64 time=0.172 m
```

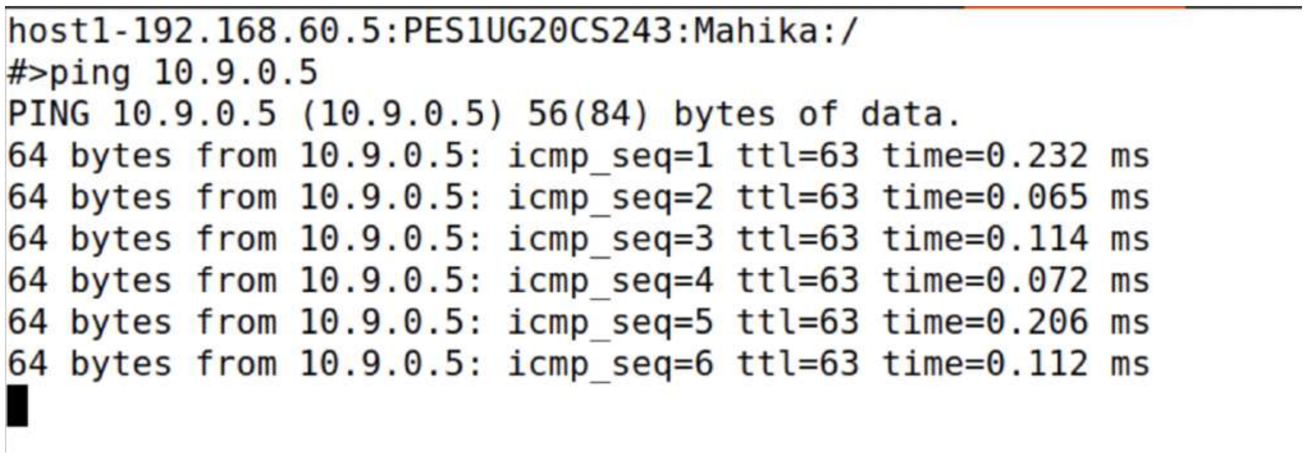
The ping is successful as host and router belong to the same network, and icmp reply is allowed in eth0 interface.

3. Internal hosts can ping Outside Hosts.

On host1-192.168.60.5 execute

Command:

# ping 10.9.0.5



```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>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=0.232 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.114 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.072 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.206 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.112 ms
```

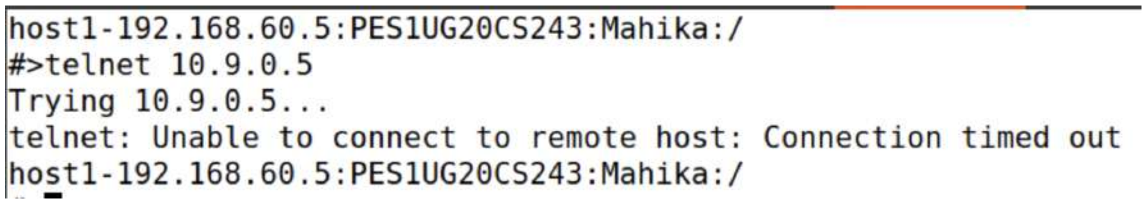
The ping is successful as icmp request packets are allowed to be sent to external hosts on eth1 interface.

4. All other packets between the internal and external networks should be blocked.

On host1-192.168.60.5 execute

Command:

# telnet 10.9.0.5



```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
host1-192.168.60.5:PES1UG20CS243:Mahika:/
```

Telnet connection can not be established as the default policy is DROP for all packets except for the rules which are specified only for icmp packets.

Please report your observation with screenshots and explain the purpose of each rule

Cleanup

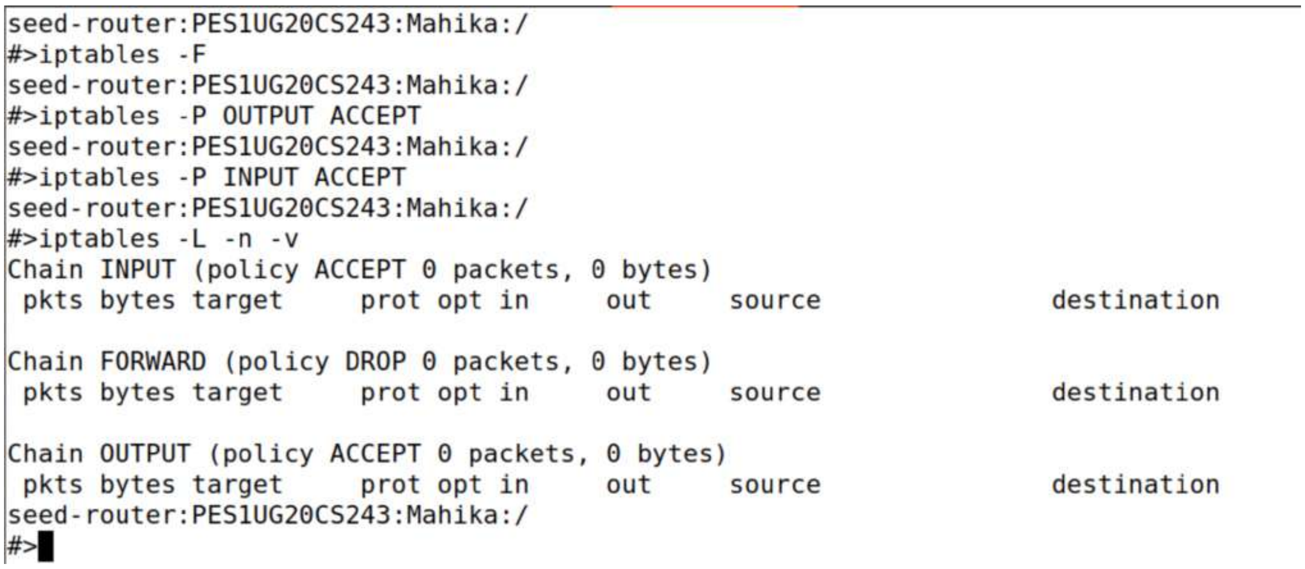
Before moving on to the next task, please restore the filter table to its original state by running the **following commands:**

On seed-router

# iptables -F

# iptables -P OUTPUT ACCEPT

# iptables -P INPUT ACCEPT



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -F
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P OUTPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P INPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
seed-router:PES1UG20CS243:Mahika:/
#>
```

Another way to restore the states of all the tables is to **restart the container.** You can do it using the **following command (you need to find the container's ID first): $ docker restart <container ID>**

Task 2.C: Protecting Internal Servers

In this task, we want to protect the TCP servers inside the internal network (192.168.60.0/24). More specifically, we would like to achieve the following objectives

1. All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.
2. Outside hosts cannot access other internal servers.
3. Internal hosts can access all the internal servers.
4. Internal hosts cannot access external servers.

Execute the following iptables commands on the **seed-router container** -

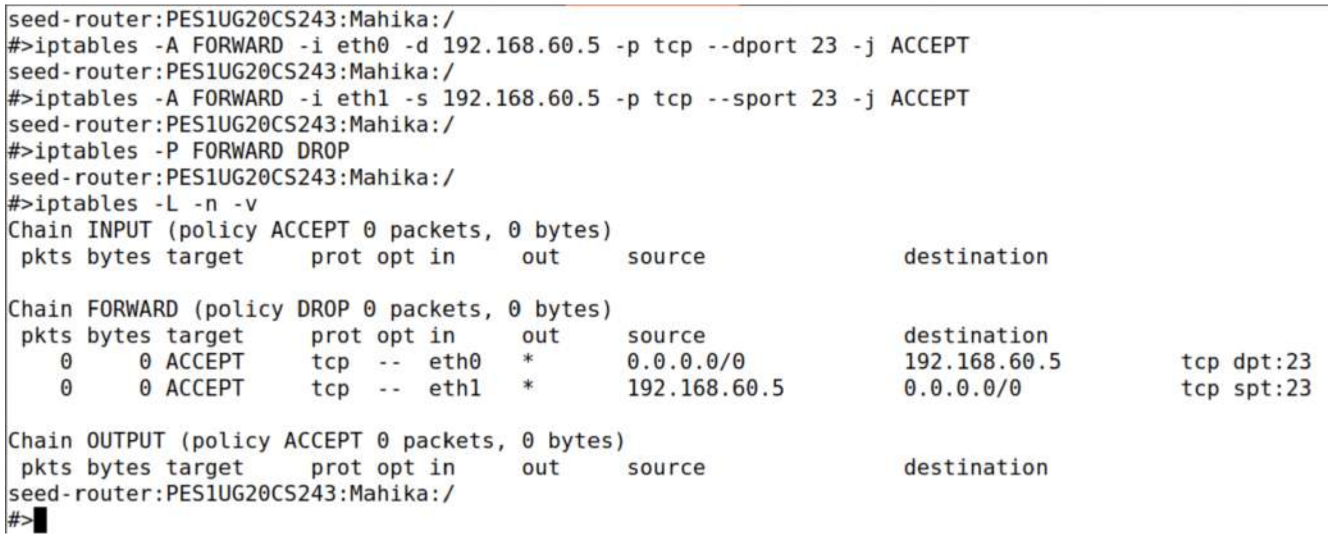
Commands:

# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT

# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT

# iptables -P FORWARD DROP

# iptables -L -n -v



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P FORWARD DROP
seed-router:PES1UG20CS243:Mahika:/
#>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  --  eth0   *       0.0.0.0/0            192.168.60.5         tcp dpt:23
    0     0 ACCEPT     tcp  --  eth1   *       192.168.60.5         0.0.0.0/0            tcp spt:23

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
seed-router:PES1UG20CS243:Mahika:/
#>
```

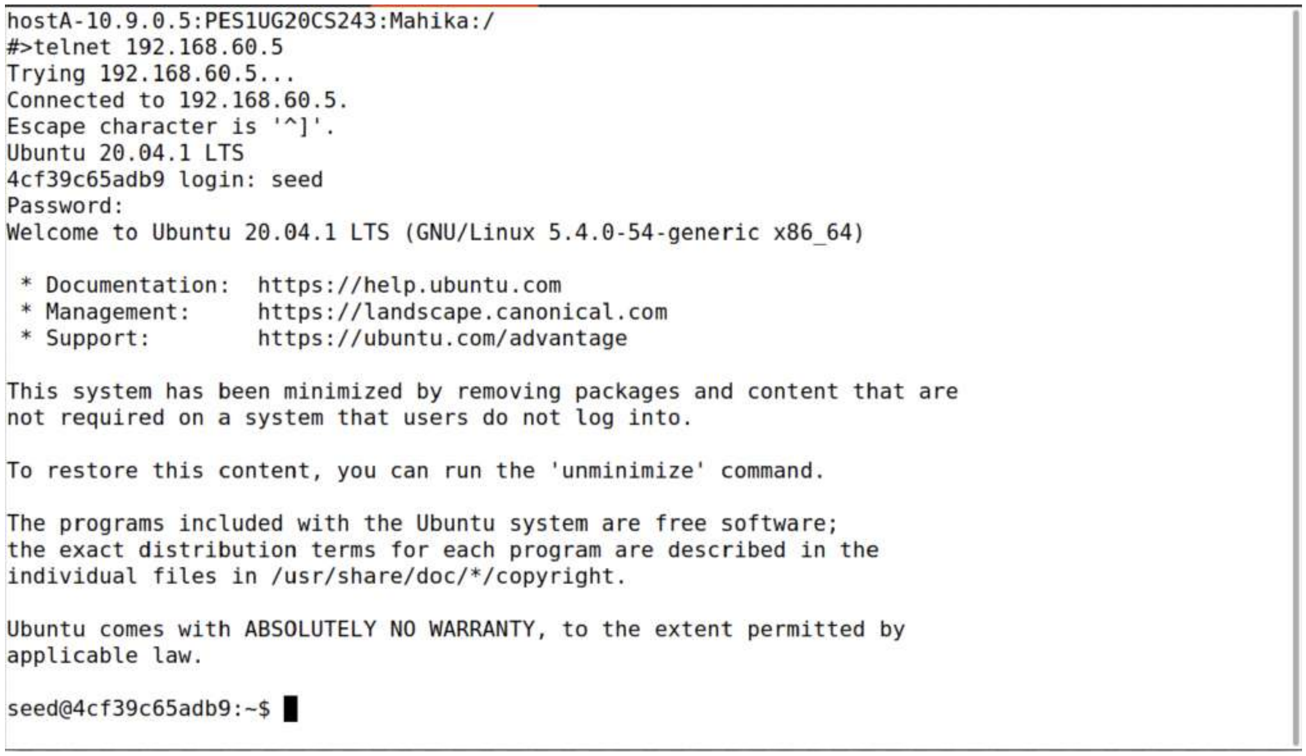
The first rule allows host to let tcp telnet packets at eth0 interface to be forwarded to destination with IP 192.168.60.5 at port 23
The second rule allows host to let tcp telnet packets at eth1 interface to be forwarded from source with IP 192.168.60.5 at port 23
The third command sets the default policy for every other packet to DROP.
The iptable is displayed.

Now we shall see if these **restrictions have been enforced** in the network. 1. All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.

On host A - 10.9.0.5

Command:

# telnet 192.168.60.5



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4cf39c65adb9 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@4cf39c65adb9:~$ █
```

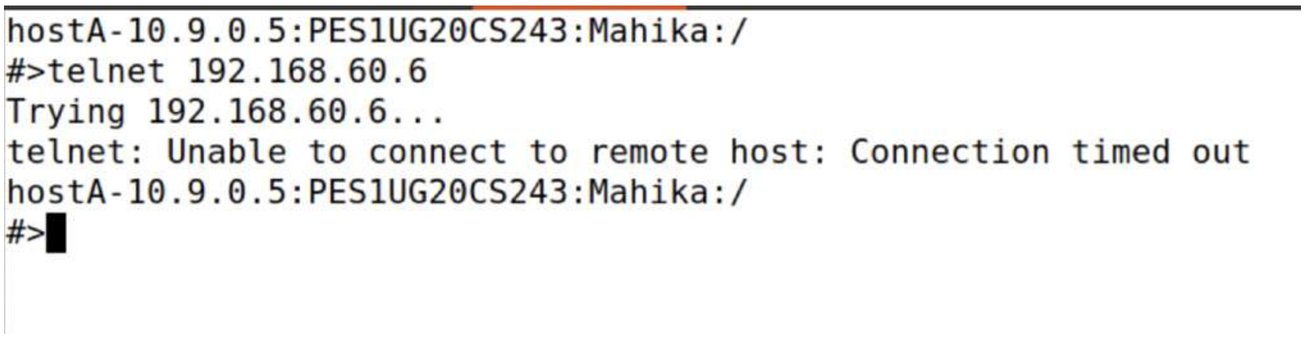
We can see that telnet connection is established with 192.168.60.5 as telent connection forwarding is allowed when destination ip is 192.168.60.5

2. Outside hosts cannot access other internal servers.

On host A - 10.9.0.5

Command:

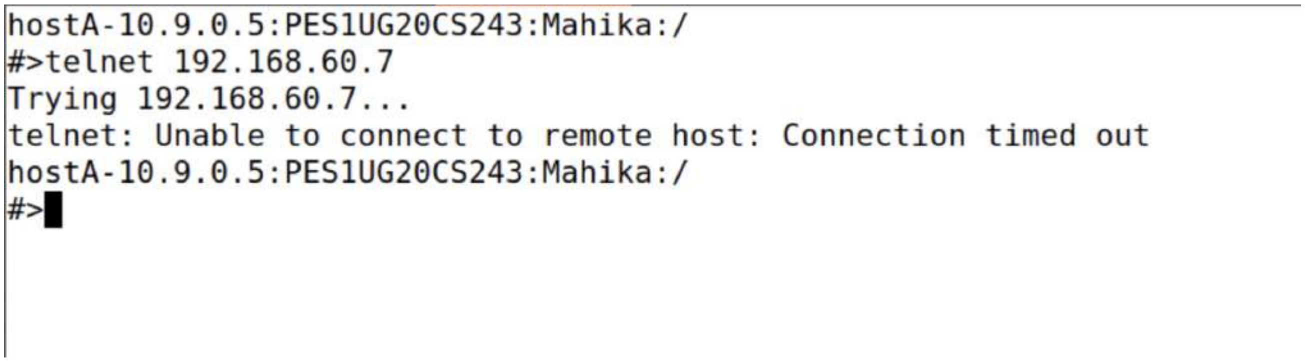
# telnet 192.168.60.6



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.6
Trying 192.168.60.6...
telnet: Unable to connect to remote host: Connection timed out
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

Telnet connection is not established as the default policy is DROP for connections with IP address not equal to 192.168.60.5

# telnet 192.168.60.7



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.7
Trying 192.168.60.7...
telnet: Unable to connect to remote host: Connection timed out
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

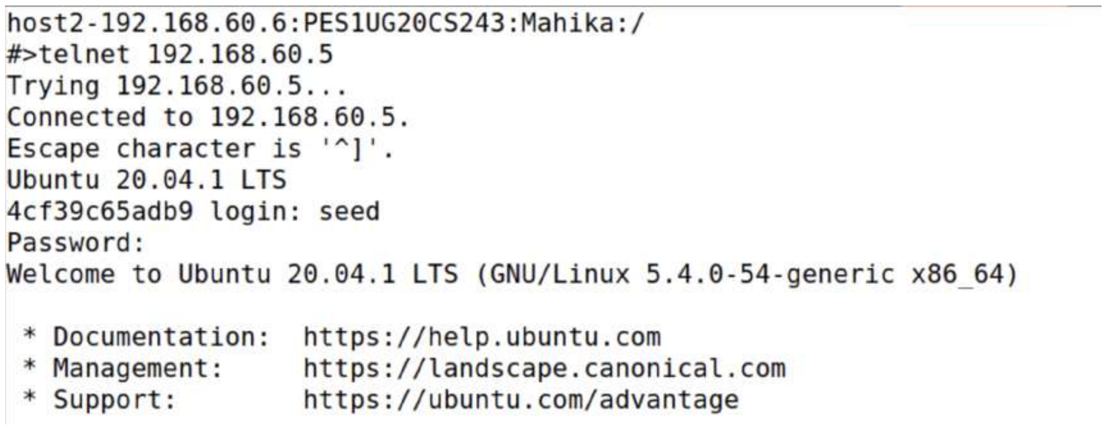
Telnet connection is not established as the default policy is DROP for connections with IP address not equal to 192.168.60.5

3. Internal hosts can access all the internal servers.

On host2 - 192.168.60.6

Command:

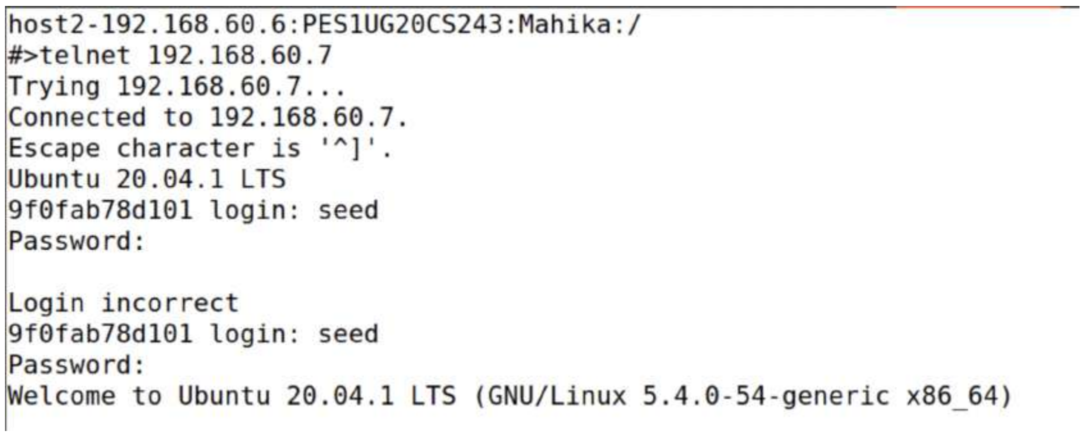
# telnet 192.168.60.5



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4cf39c65adb9 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
```

Telnet connection is established as destination IP address is 192.168.60.5

# telnet 192.168.60.7



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
9f0fab78d101 login: seed
Password:

Login incorrect
9f0fab78d101 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

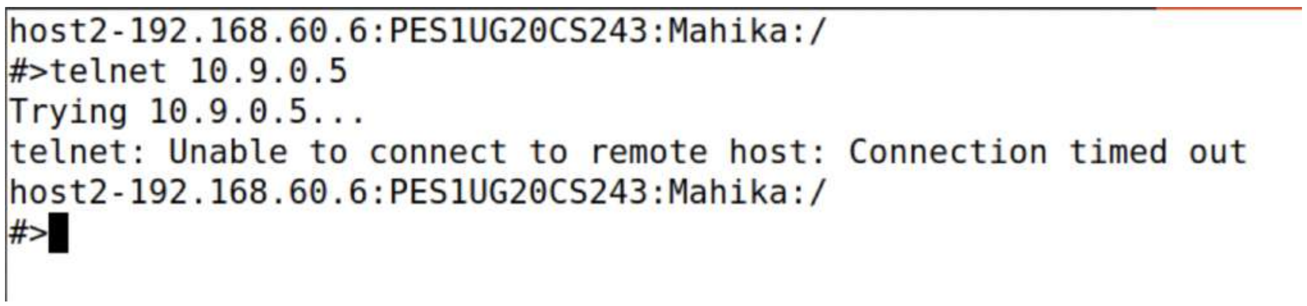
Telnet connection is established as they are on the same network.

4. Internal hosts cannot access external servers.

On host2 - 192.168.60.6

Command:

# telnet 10.9.0.5



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>
```

Telnet connection is not established as external servers cannot be established. The default policy is DROP.

Please report your observation with screenshots and explain the purpose for each rule

Cleanup

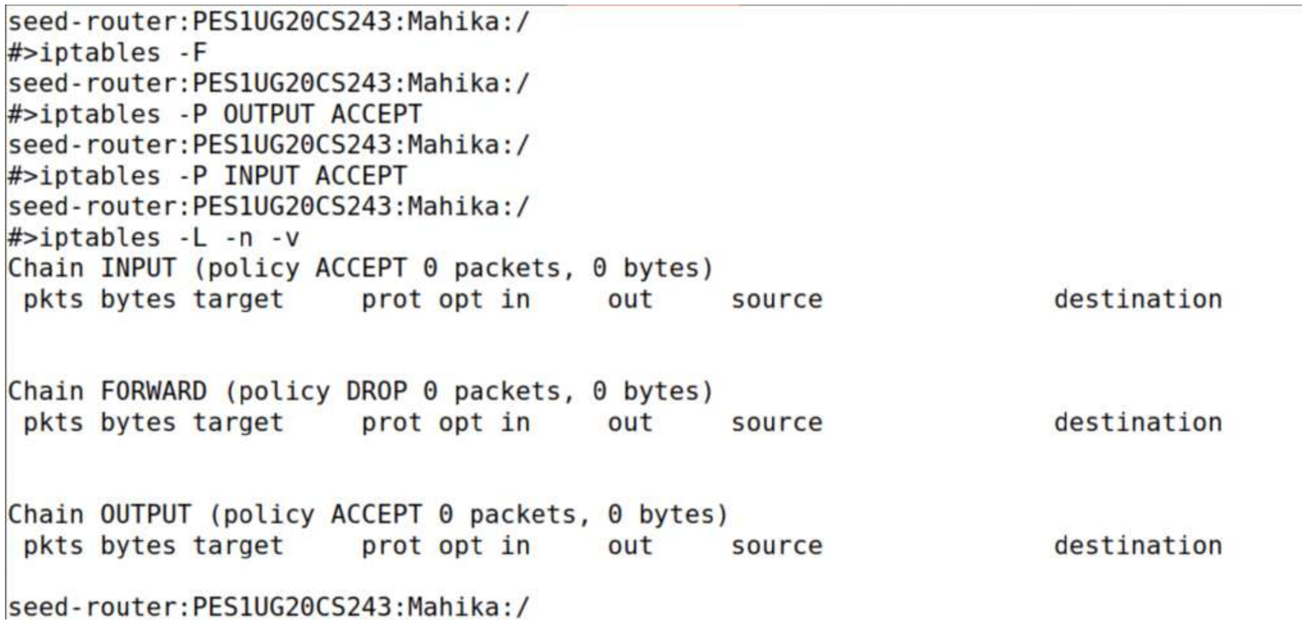
Before moving on to the next task, please restore the filter table to its original state by running the **following commands:**

On seed-router

# iptables -F

# iptables -P OUTPUT ACCEPT

# iptables -P INPUT ACCEPT



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -F
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P OUTPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P INPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>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


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

seed-router:PES1UG20CS243:Mahika:/
```

Another way to restore the states of all the tables is to **restart the container.**

Task 3: Connection Tracking and Stateful Firewall

In the previous task, we have only set up stateless firewalls, which inspect each packet independently. However, packets are usually not independent; they may be part of a TCP connection, or they may be ICMP packets triggered by other packets. Treating them independently does not take into consideration the context of the packets, and can thus lead to inaccurate, unsafe, or complicated firewall rules.

For example, if we would like to allow TCP packets to get into our network only if a connection was made first, we cannot achieve that easily using stateless packet filters, because when the firewall examines each individual TCP packet, it has no idea whether the packet belongs to an existing connection or not, unless the firewall maintains some state information for each connection. If it does that, it becomes a stateful firewall.

Task 3.A: Experiment with the Connection Tracking

To support stateful firewalls, we need to be able to track connections. This is achieved by the conntrack mechanism inside the kernel. In this task, we will conduct experiments related to this module, and get familiar with the connection tracking mechanism. In our experiment, we will check the connection tracking information on the router container.

This can be done using the following command:

```
# conntrack -L
```

The goal of the task is to use a series of experiments to help students understand the connection concept in this tracking mechanism, especially for the ICMP and UDP protocols, because unlike TCP, they do not have connections. Please conduct the following experiments. For each experiment, please describe your observation, along with your explanation.

ICMP experiment:

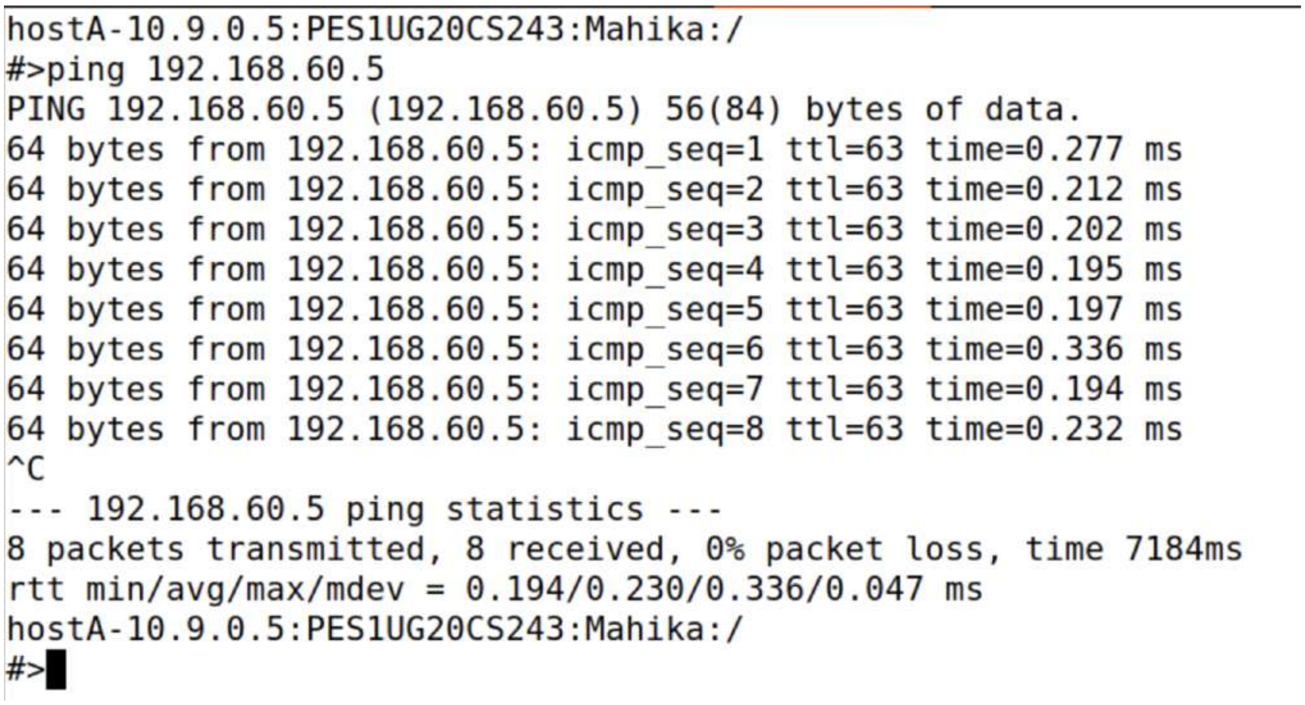
Run the following command and check the connection tracking information on the router. Describe your observation. How long can the ICMP connection state be kept?

On host A - 10.9.0.5

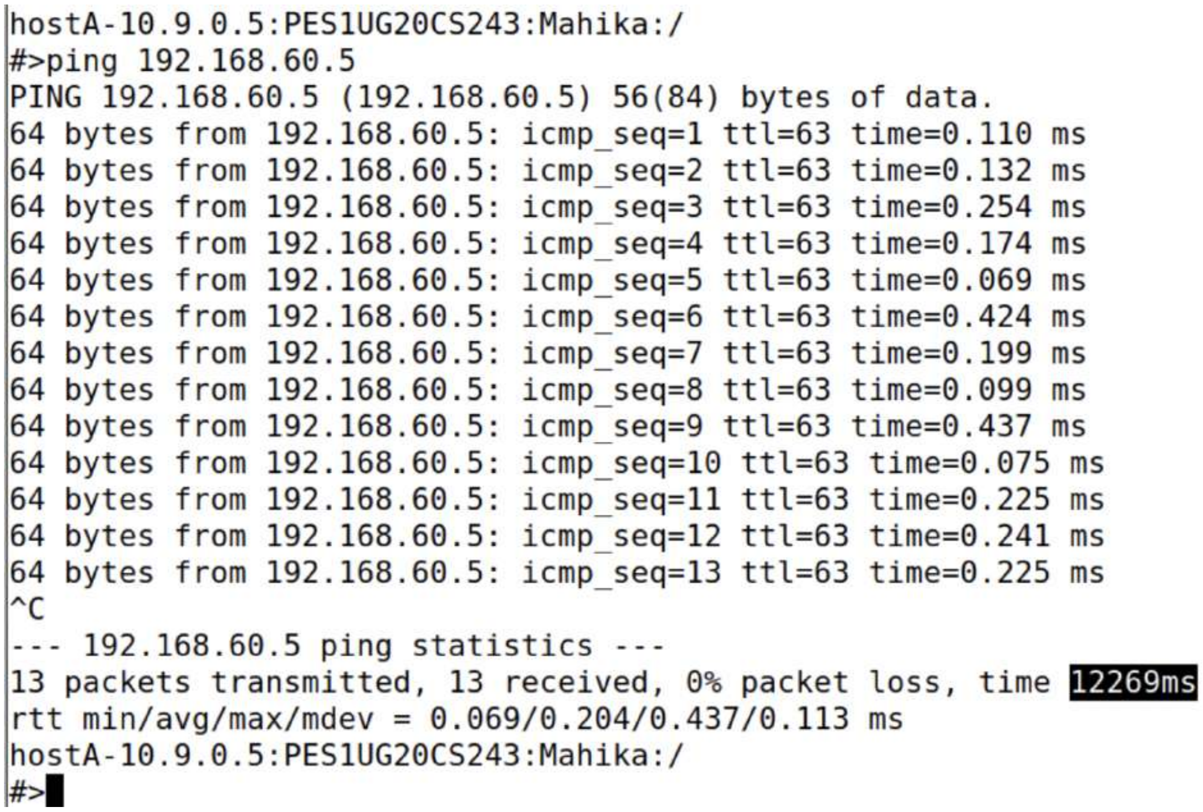
Command:

# ping 192.168.60.5

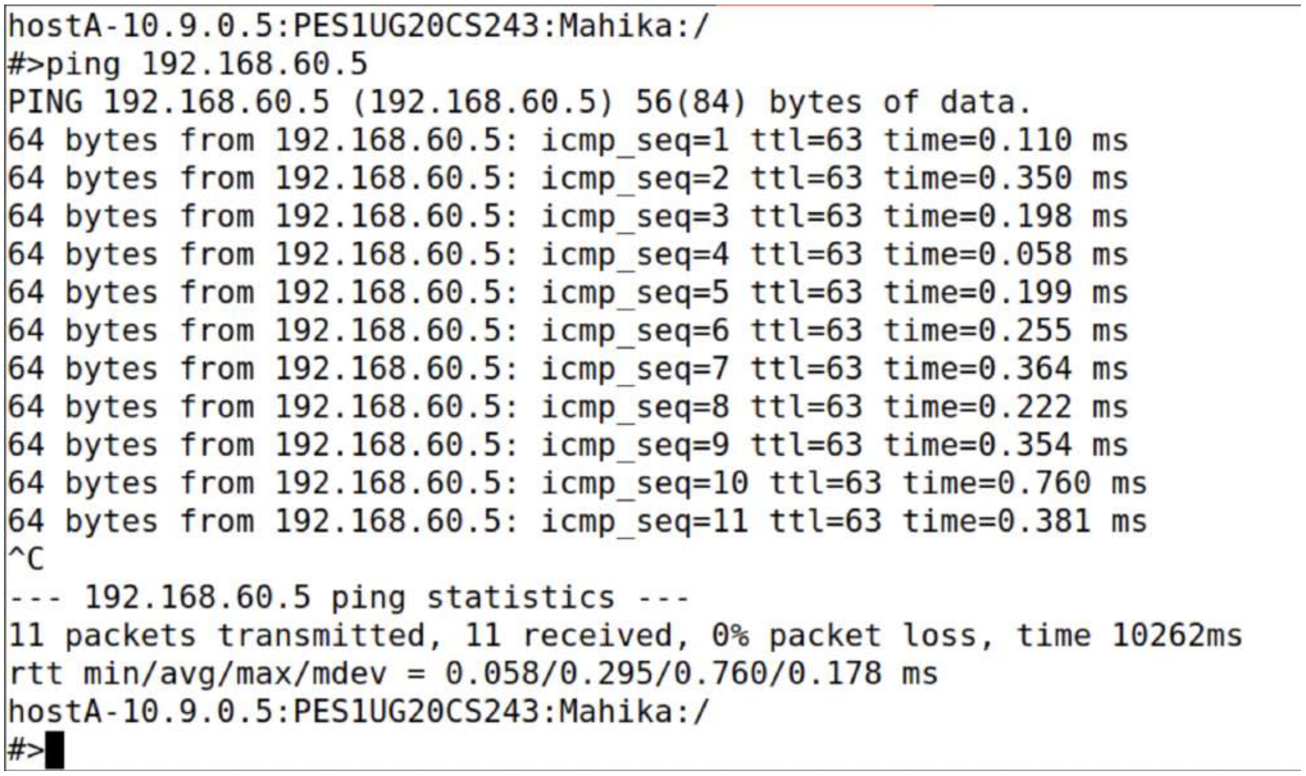
The ping is executed three times:



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>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=0.277 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.212 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.202 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.195 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.197 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.336 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.194 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.232 ms
^C
--- 192.168.60.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7184ms
rtt min/avg/max/mdev = 0.194/0.230/0.336/0.047 ms
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>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=0.110 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.132 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.254 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.174 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.069 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.424 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.199 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.099 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.437 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.075 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.225 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.241 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.225 ms
^C
--- 192.168.60.5 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12269ms
rtt min/avg/max/mdev = 0.069/0.204/0.437/0.113 ms
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>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=0.110 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.350 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.198 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.199 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.255 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.364 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.222 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.354 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.760 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.381 ms
^C
--- 192.168.60.5 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10262ms
rtt min/avg/max/mdev = 0.058/0.295/0.760/0.178 ms
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

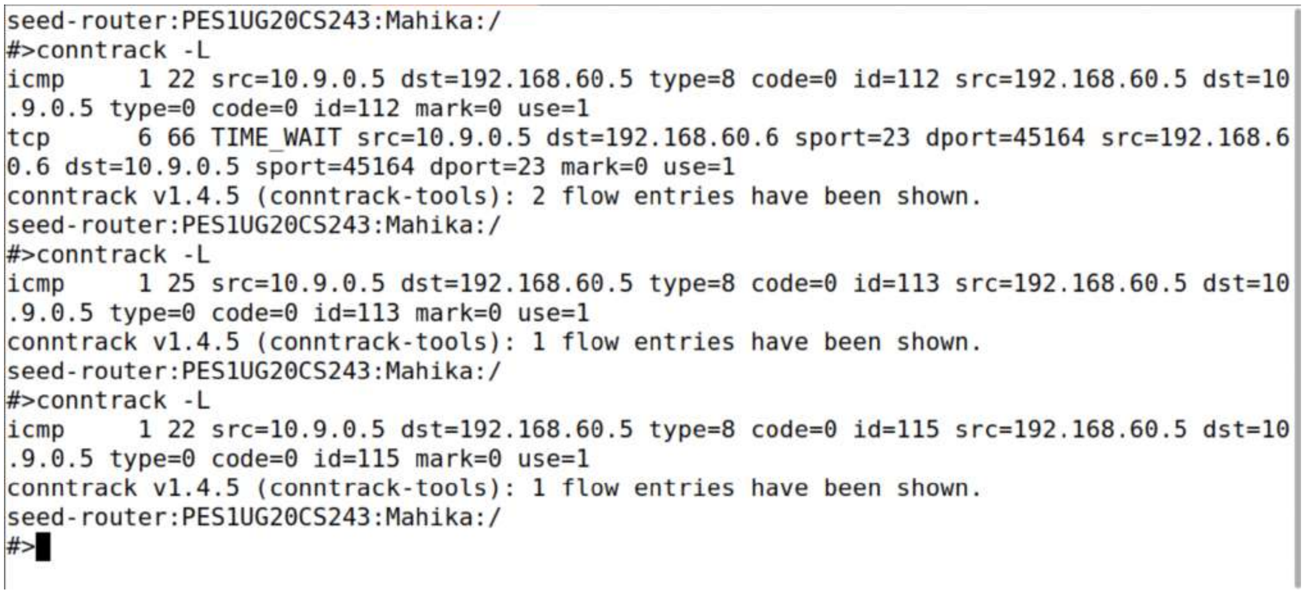
We can see that the time decreases with every ping

Let the ping run for sometime, then stop it (Ctrl + C)

Immediately move to the seed router and run

# conntrack -L

The ping command is executed three times:


```
seed-router:PES1UG20CS243:Mahika:/
#>conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=112 src=192.168.60.5 dst=10
.9.0.5 type=0 code=0 id=112 mark=0 use=1
tcp      6 66 TIME_WAIT src=10.9.0.5 dst=192.168.60.6 sport=23 dport=45164 src=192.168.6
0.6 dst=10.9.0.5 sport=45164 dport=23 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>conntrack -L
icmp     1 25 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=113 src=192.168.60.5 dst=10
.9.0.5 type=0 code=0 id=113 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=115 src=192.168.60.5 dst=10
.9.0.5 type=0 code=0 id=115 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>
```

We can see that with every ping, the time decreases.

Keep executing the above command, and you shall see the timer decreasing (time remaining for the connection)

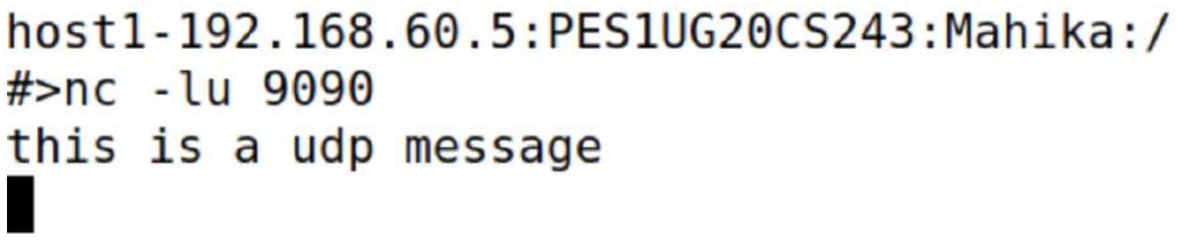
<u>UDP experiment:</u>

Run the following command and check the connection tracking information on the router. Describe your observation. How long can the UDP connection state be kept?

On host 1 - 192.168.60.5

Command:

# nc -lu 9090

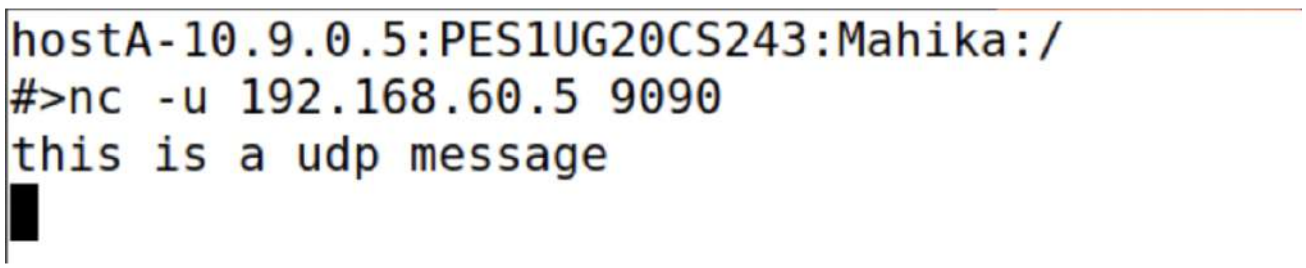

```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>nc -lu 9090
this is a udp message
```

The message typed in host A is displayed in host 1's terminal.

On host A - 10.9.0.5

Command:

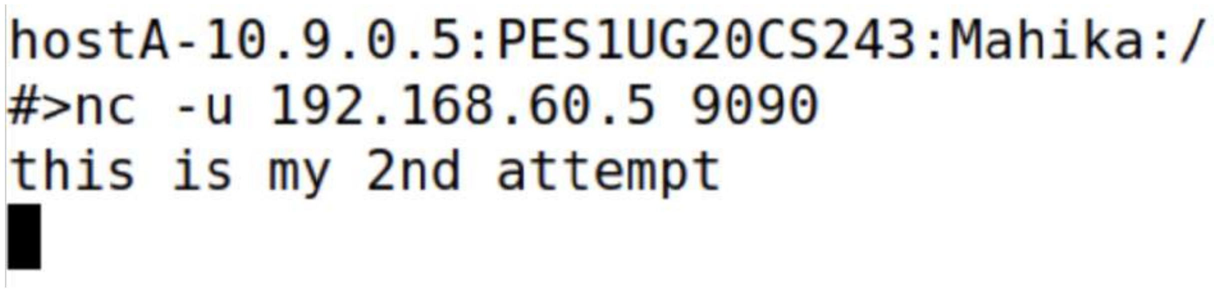
# nc -u 192.168.60.5 9090



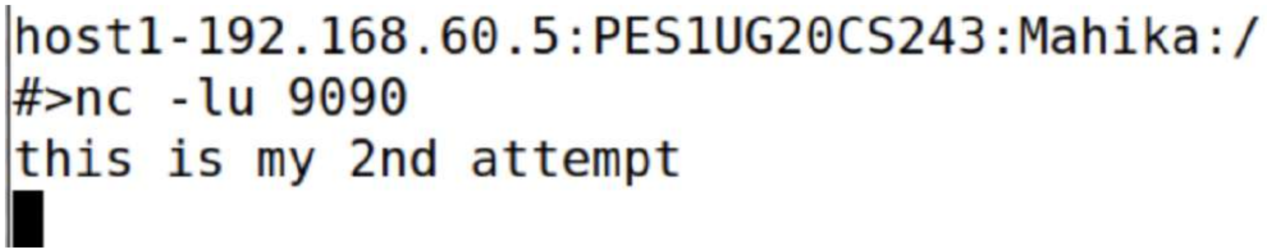
```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>nc -u 192.168.60.5 9090
this is a udp message
```

Message is typed in host A.

UDP message is sent again:

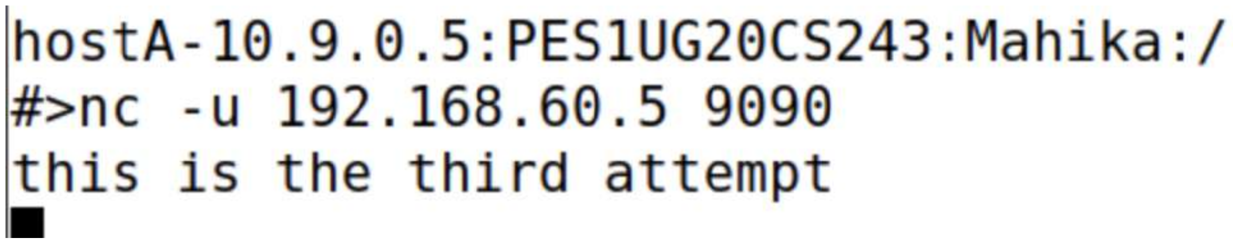


```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>nc -u 192.168.60.5 9090
this is my 2nd attempt
```

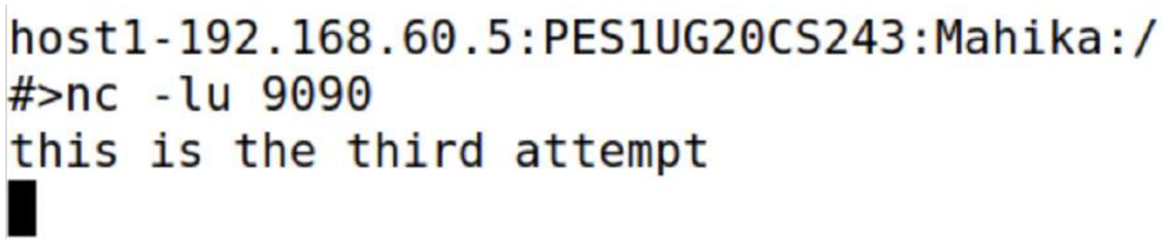


```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>nc -lu 9090
this is my 2nd attempt
```

Message is sent for the third time:



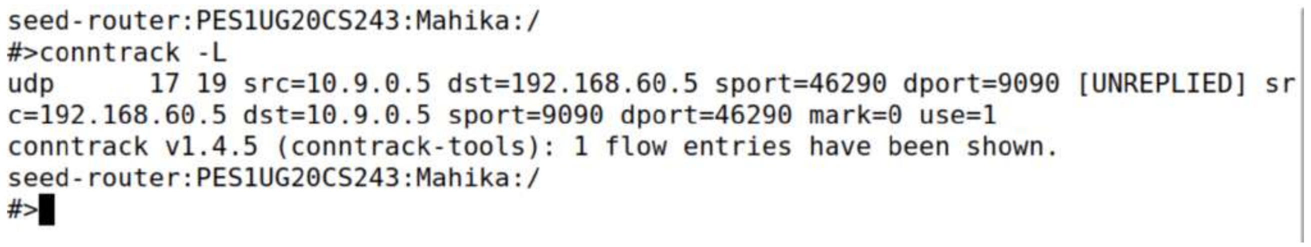
```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>nc -u 192.168.60.5 9090
this is the third attempt
```



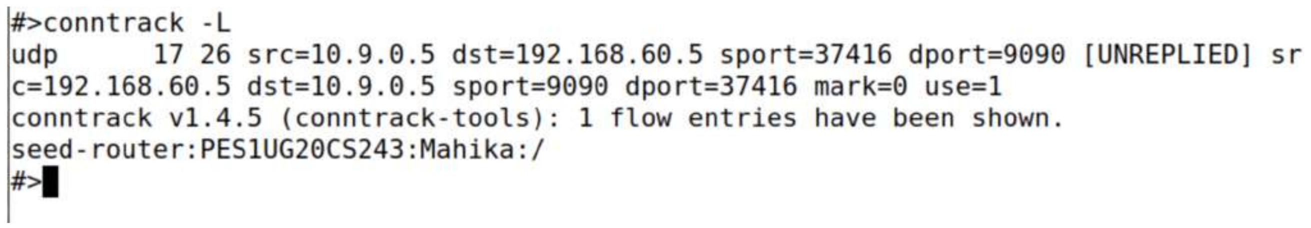
```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>nc -lu 9090
this is the third attempt
```

On seed router

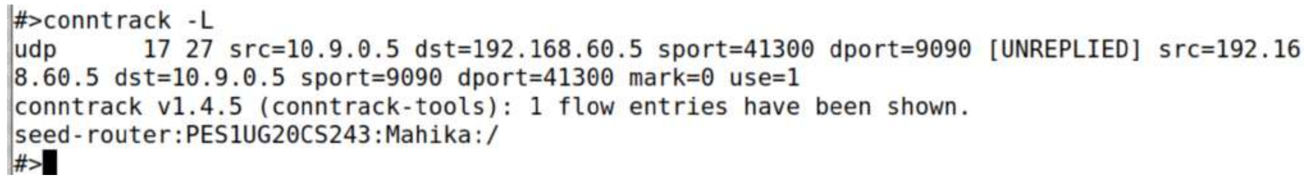
# conntrack -L



```
seed-router:PES1UG20CS243:Mahika:/
#>conntrack -L
udp      17 19 src=10.9.0.5 dst=192.168.60.5 sport=46290 dport=9090 [UNREPLIED] sr
c=192.168.60.5 dst=10.9.0.5 sport=9090 dport=46290 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>
```



```
#>conntrack -L
udp      17 26 src=10.9.0.5 dst=192.168.60.5 sport=37416 dport=9090 [UNREPLIED] sr
c=192.168.60.5 dst=10.9.0.5 sport=9090 dport=37416 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>
```



```
#>conntrack -L
udp      17 27 src=10.9.0.5 dst=192.168.60.5 sport=41300 dport=9090 [UNREPLIED] src=192.16
8.60.5 dst=10.9.0.5 sport=9090 dport=41300 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>
```

After all three UDP connections, we can see that the timer increases from 19 to 26 to 27.

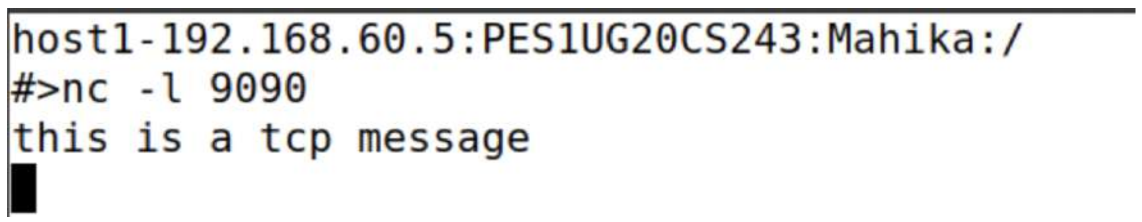
TCP experiment:

Run the following command and check the connection tracking information on the router. Describe your observation. How long can the TCP connection state be kept?

On host 1 - 192.168.60.5

Command:

# nc -l 9090

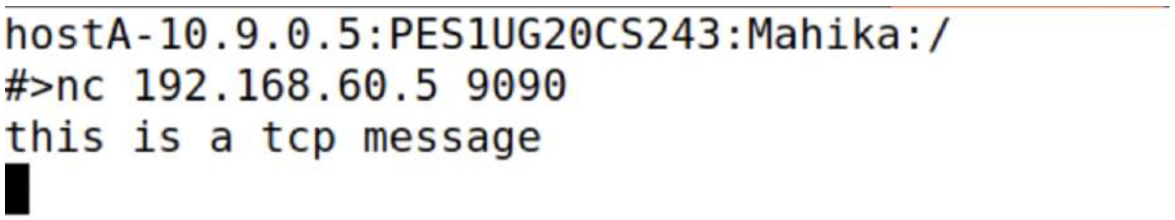


```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>nc -l 9090
this is a tcp message
```

On host A - 10.9.0.5

Command:

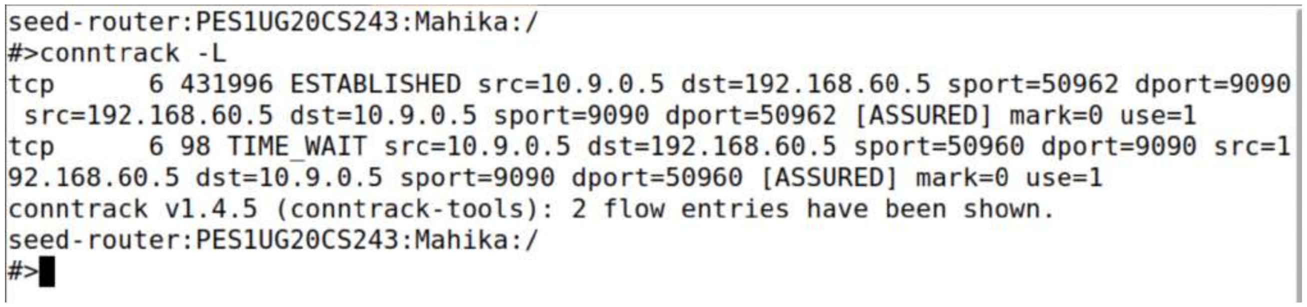
# nc 192.168.60.5 9090



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>nc 192.168.60.5 9090
this is a tcp message
```

On seed router

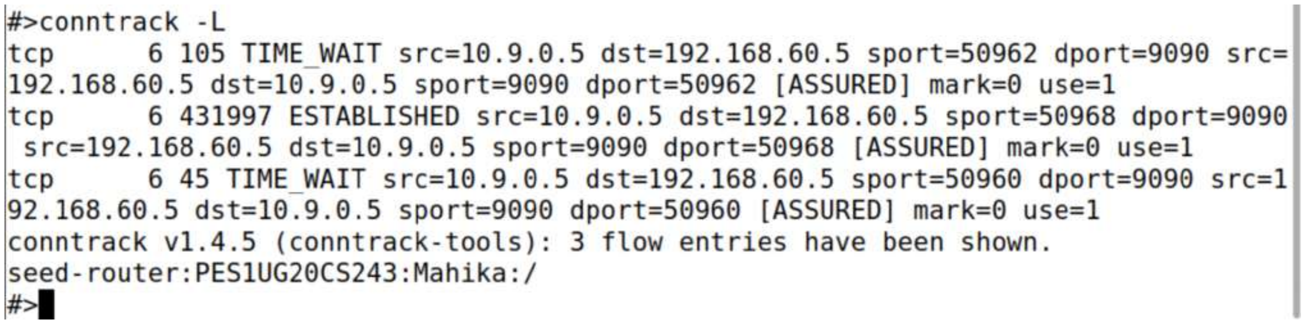
# conntrack -L



```
seed-router:PES1UG20CS243:Mahika:/
#>conntrack -L
tcp      6 431996 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=50962 dport=9090
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50962 [ASSURED] mark=0 use=1
tcp      6 98 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50960 dport=9090 src=1
92.168.60.5 dst=10.9.0.5 sport=9090 dport=50960 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>
```

Execute the above command continuously and see if you spot the change in the timer.

The tcp connection is established one more time:



```
#>conntrack -L
tcp      6 105 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50962 dport=9090 src=
192.168.60.5 dst=10.9.0.5 sport=9090 dport=50962 [ASSURED] mark=0 use=1
tcp      6 431997 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=50968 dport=9090
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50968 [ASSURED] mark=0 use=1
tcp      6 45 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50960 dport=9090 src=1
92.168.60.5 dst=10.9.0.5 sport=9090 dport=50960 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 3 flow entries have been shown.
seed-router:PES1UG20CS243:Mahika:/
#>
```

We can see that the time to wait has increased.

Task 3.B: Setting Up a Stateful Firewall

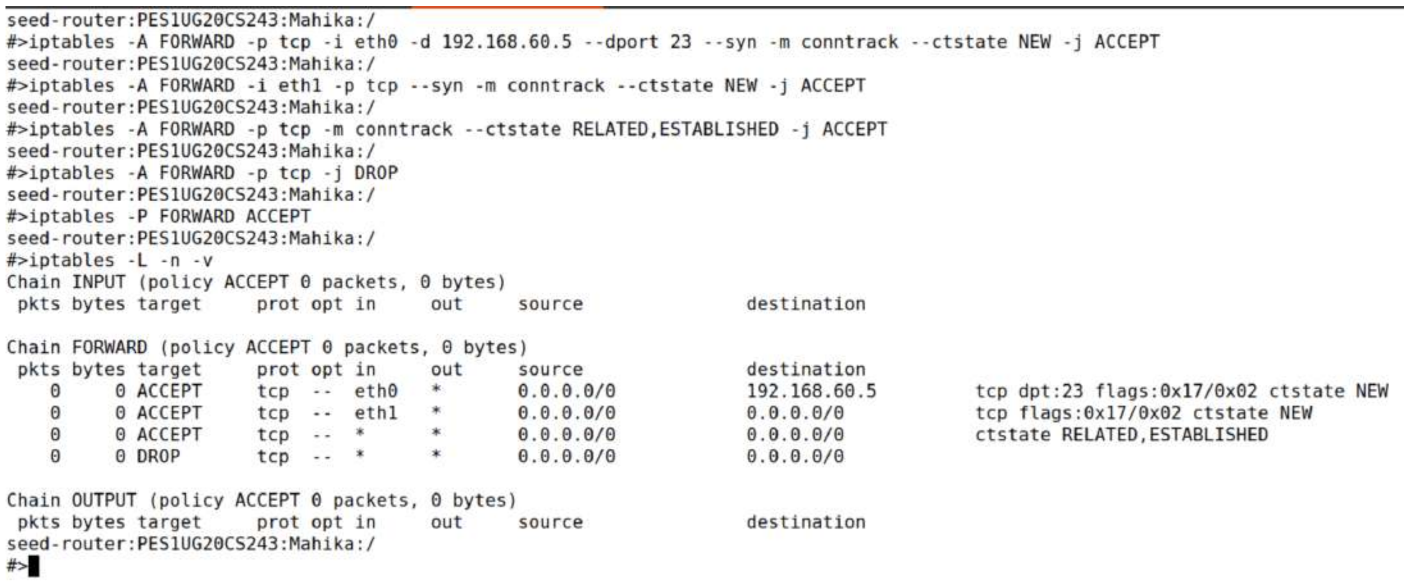
Now we are ready to set up firewall rules based on connections. In the following example, the "-m conntrack" option indicates that we are using the conntrack module, which is a very important module for iptables; it tracks connections, and iptables replies on the tracking information to build stateful firewalls. The --ctstate ESTABLISHED,RELATED indicates whether a packet belongs to an ESTABLISHED or RELATED connection. The rule allows TCP packets belonging to an existing connection to pass through

For this task we have to rewrite the firewall rules in Task 2.C, but this time, we will add a rule allowing internal hosts to visit any external server (this was not allowed in Task 2.C).

On seed-router execute

Commands:

```
        # iptables -A FORWARD -p tcp -i eth0 -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 RELATED,ESTABLISHED
-j ACCEPT
        # iptables -A FORWARD -p tcp -j DROP
        # iptables -P FORWARD ACCEPT
        # iptables -L -n -v
```



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -p tcp -j DROP
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P FORWARD ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>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
    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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
seed-router:PES1UG20CS243:Mahika:/
#>
```

The first command sets the rule to only accept tcp telnet packets with SYN bit set, with destination address as 192.168.60.5, where the state of the connection is NEW, at eth0 interface

The second command sets the rule to only accept tcp packets with SYN bit set, where the state of the connection is either RELATED or ESTABLISHED

The third command is to set the FORWARD policy as DROP for other tcp packets

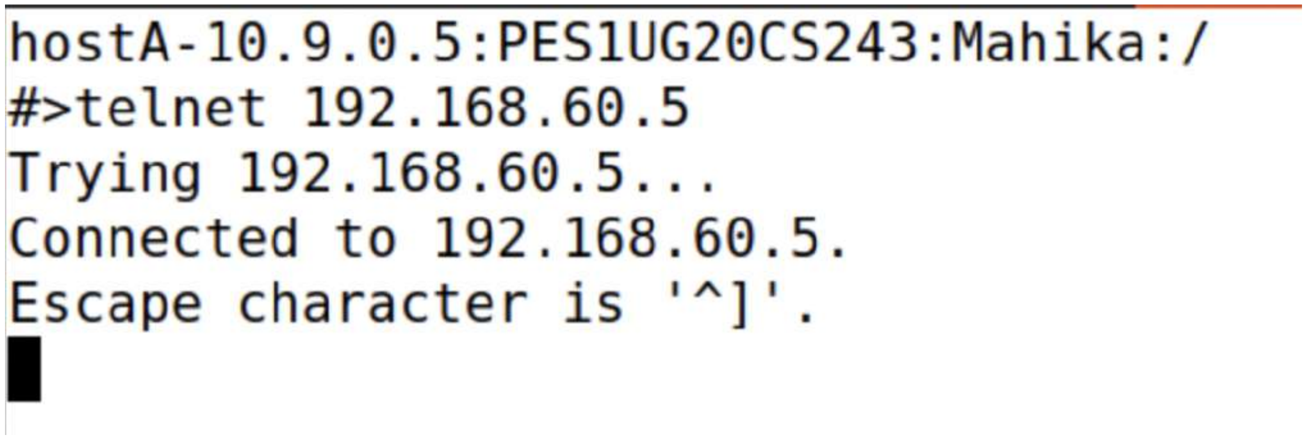
The fourth command sets the default FORWARD policy as ACCEPT

Now we shall see if these **restrictions have been enforced** in the network. 1. All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.

On host A - 10.9.0.5

Command:

# telnet 192.168.60.5



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
█
```

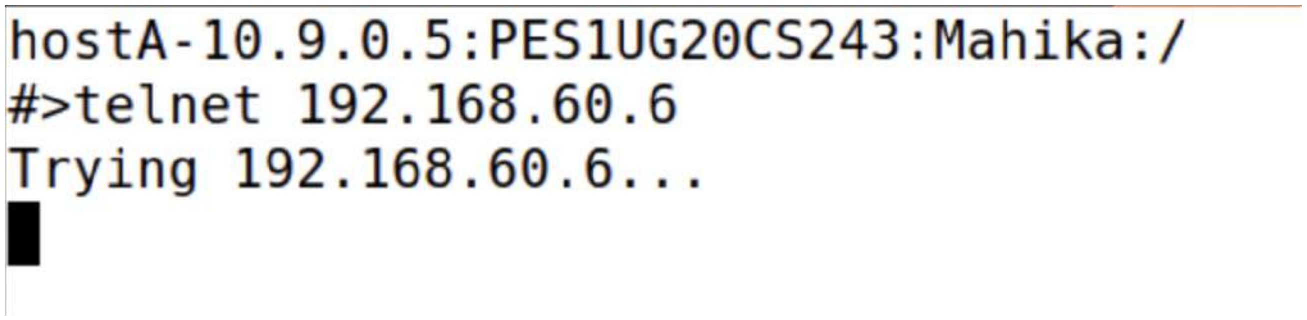
Here we can see that the telnet connection is established as all the rules are satisfied, i.e the destination is 192.168.60.5, and telnet connection is made at port 23, eth0

2. Outside hosts cannot access other internal servers.

On host A - 10.9.0.5

Command:

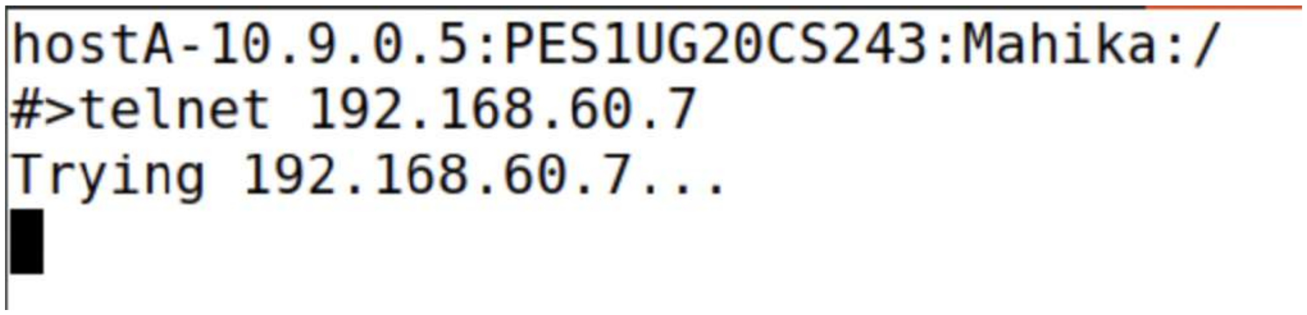
# telnet 192.168.60.6



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.6
Trying 192.168.60.6...
█
```

We can see that the telnet connection can not be established as outside hosts cannot access internal servers and the destination is not 192.168.60.5. The default for tcp packets is DROP.

# telnet 192.168.60.7



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.7
Trying 192.168.60.7...
█
```

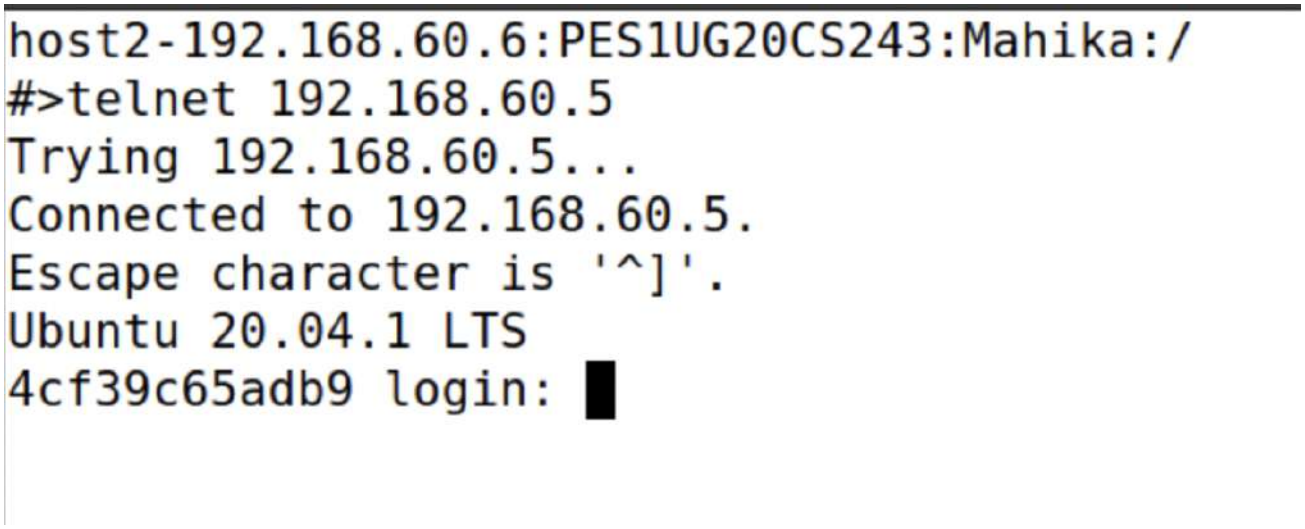
Similarly this telnet connection cannot be established.

3. Internal hosts can access all the internal servers.

On host2 - 192.168.60.6

Command:

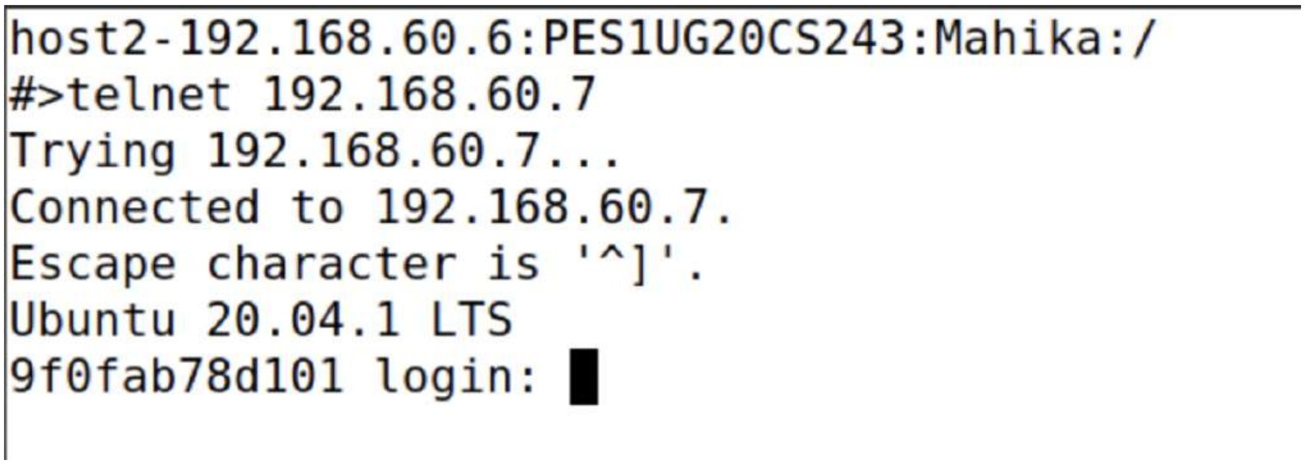
# telnet 192.168.60.5



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4cf39c65adb9 login: █
```

We can see that the telnet connection is established as internal hosts can access internal servers, and the Default setting for forward is ACCEPT.

# telnet 192.168.60.7



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
9f0fab78d101 login: █
```

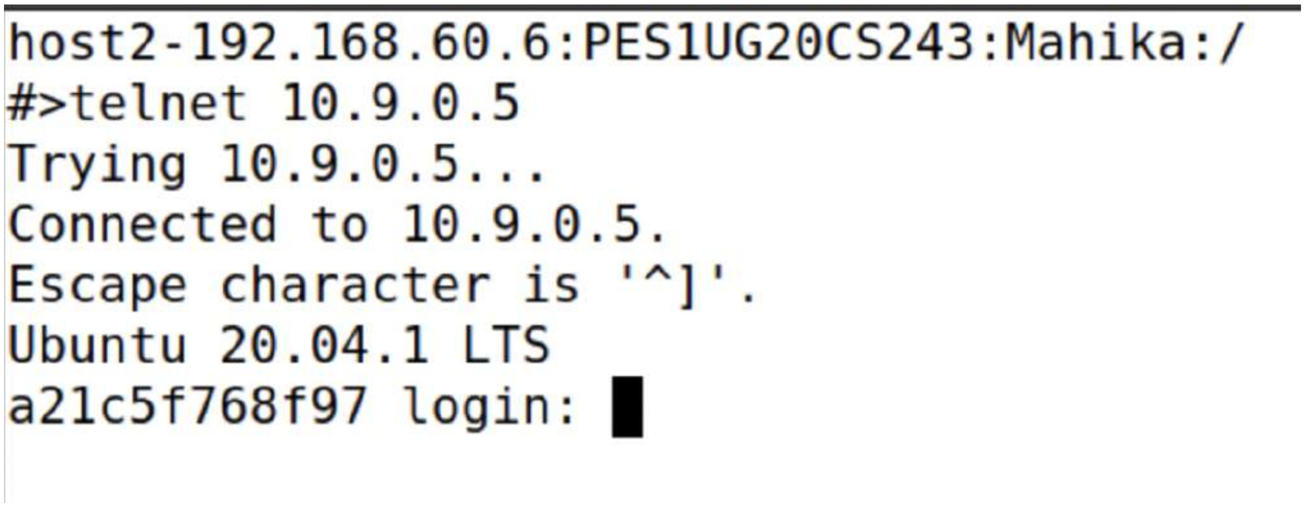
Similarly telnet connection is established over here.

4. Internal hosts can access external servers.

On host2 - 192.168.60.6

Command:

# telnet 10.9.0.5



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a21c5f768f97 login: 
```

Here internal hosts can access external servers as the rule for new connections on eth1 is accepted.

Cleanup

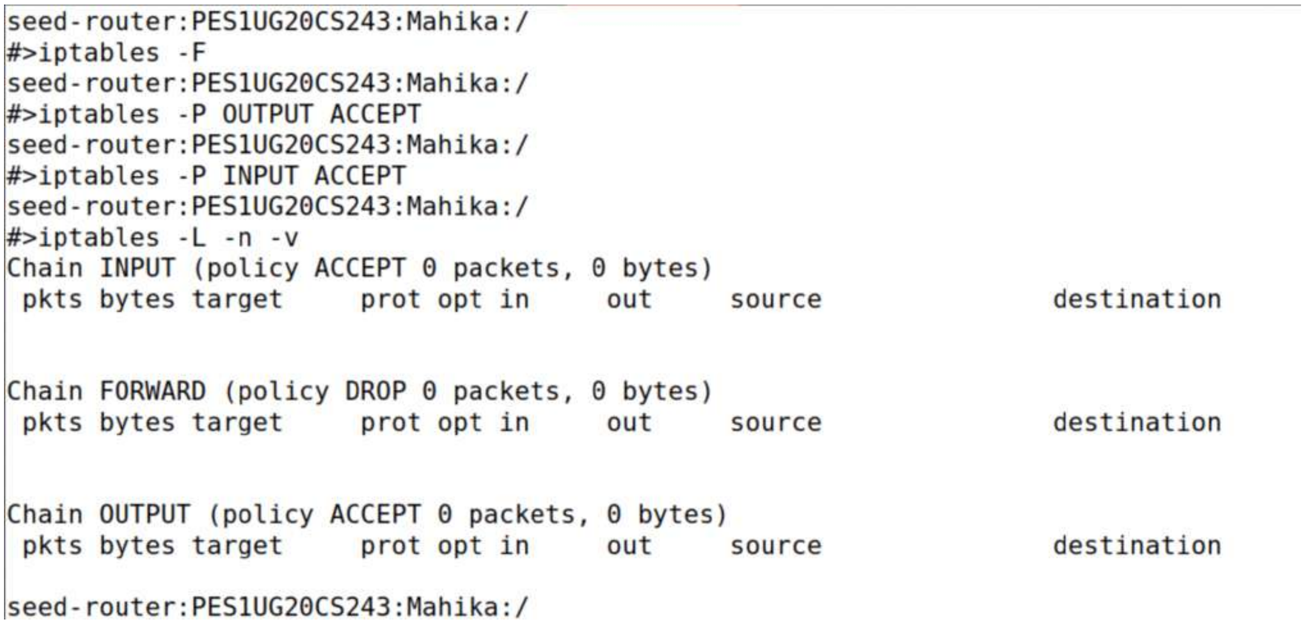
Before moving on to the next task, please restore the filter table to its original state by running the **following commands:**

On seed-router

# iptables -F

# iptables -P OUTPUT ACCEPT

# iptables -P INPUT ACCEPT



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -F
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P OUTPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P INPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>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


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

seed-router:PES1UG20CS243:Mahika:/
```

Task 4: Limiting Network Traffic

In addition to blocking packets, we can also limit the number of packets that can pass through the firewall. This can be done using the limit module of iptables. In this task, we will use this module to limit how many packets from 10.9.0.5 are allowed to get into the internal network. You can use "iptables -m limit -h" to see the manual

Please run the following commands on the router, and then ping 192.168.60.5 from 10.9.0.5. Describe your observation. Please conduct the experiment with and without the second rule, and then explain whether the second rule is needed or not, and why.

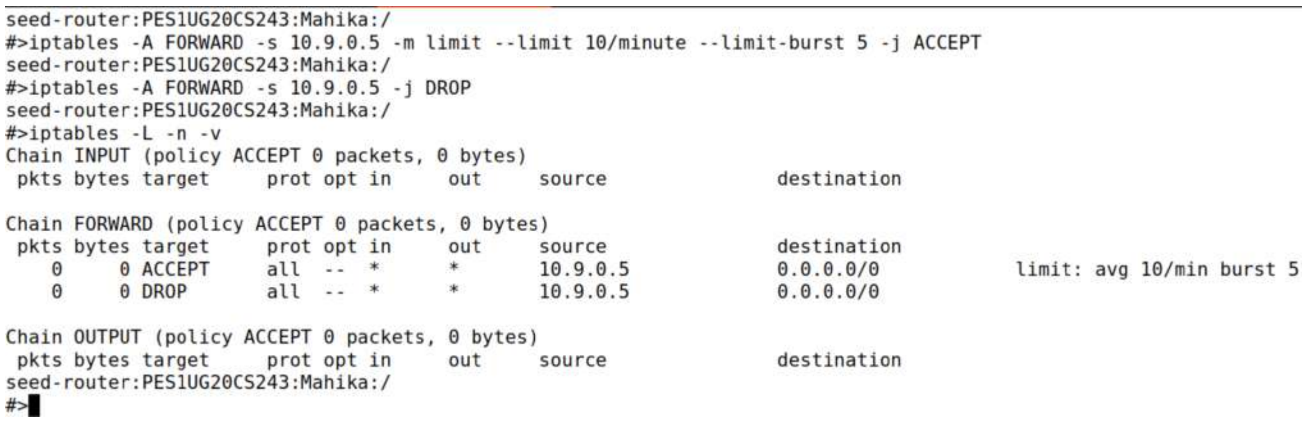
On seed router execute -

Command:

# 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

# iptables -L -n -v



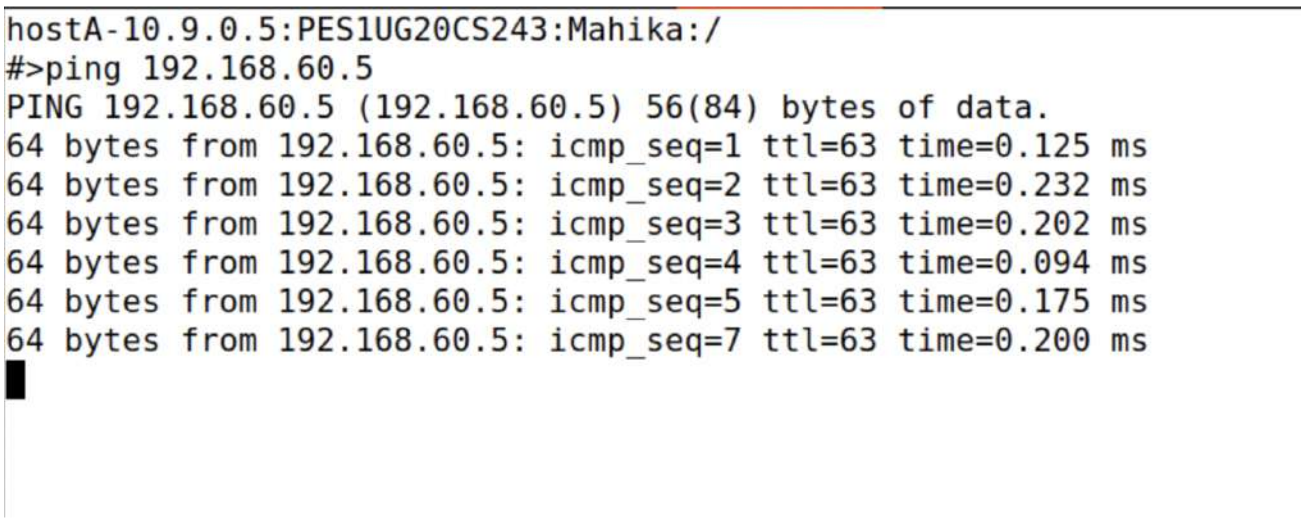
```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -s 10.9.0.5 -j DROP
seed-router:PES1UG20CS243:Mahika:/
#>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 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
    0     0 ACCEPT     all  --  *      *       10.9.0.5             0.0.0.0/0            limit: avg 10/min burst 5
    0     0 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
seed-router:PES1UG20CS243:Mahika:/
#>
```

The first rule sets that when source is 10.9.0.5, the limit is 5 packets allowed with the rate of 10 packets per minute. The secod rule drops all packets coming from source 10.9.0.5.

On host A - 10.9.0.5

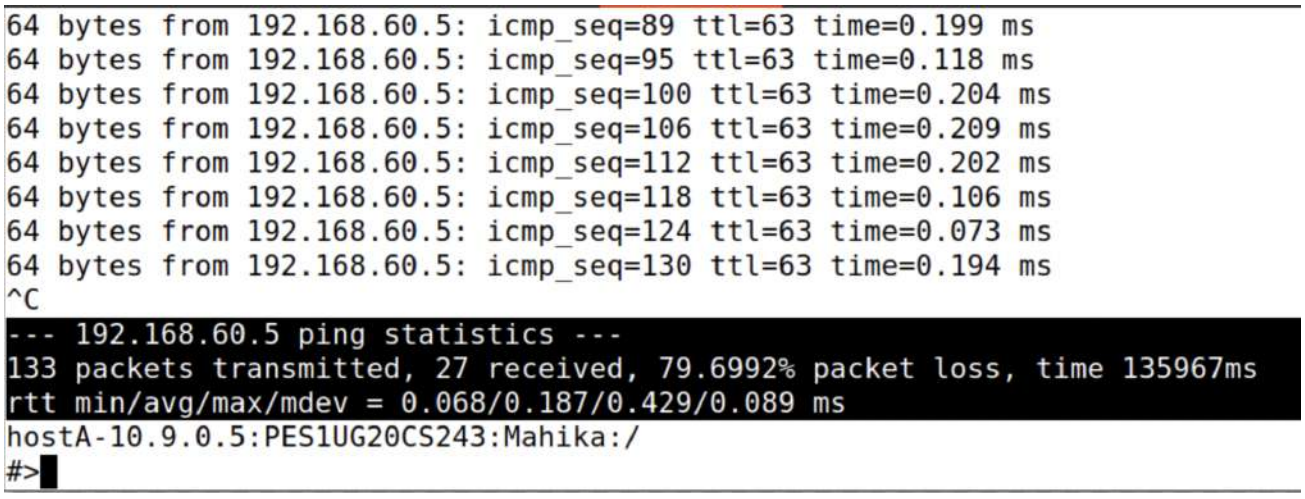
Command:

# ping 192.168.60.5



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>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=0.125 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.232 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.202 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.094 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.175 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.200 ms
```

Here we see that The time between each request increases and the ping stops eventually.



```
64 bytes from 192.168.60.5: icmp_seq=89 ttl=63 time=0.199 ms
64 bytes from 192.168.60.5: icmp_seq=95 ttl=63 time=0.118 ms
64 bytes from 192.168.60.5: icmp_seq=100 ttl=63 time=0.204 ms
64 bytes from 192.168.60.5: icmp_seq=106 ttl=63 time=0.209 ms
64 bytes from 192.168.60.5: icmp_seq=112 ttl=63 time=0.202 ms
64 bytes from 192.168.60.5: icmp_seq=118 ttl=63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=124 ttl=63 time=0.073 ms
64 bytes from 192.168.60.5: icmp_seq=130 ttl=63 time=0.194 ms
^C
--- 192.168.60.5 ping statistics ---
133 packets transmitted, 27 received, 79.6992% packet loss, time 135967ms
rtt min/avg/max/mdev = 0.068/0.187/0.429/0.089 ms
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

The packets were dropped after 5 packets were received.

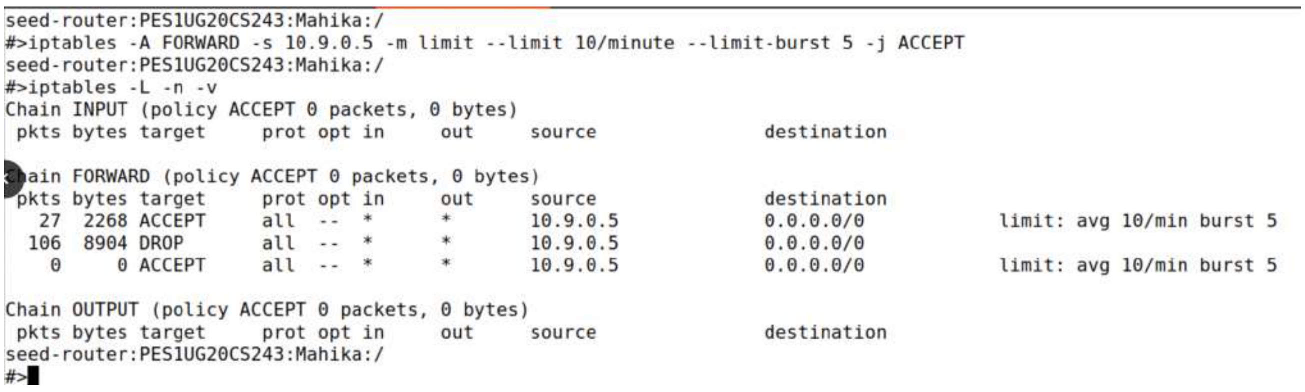
On seed router execute -

Command:

# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5

-j ACCEPT

# iptables -L -n -v

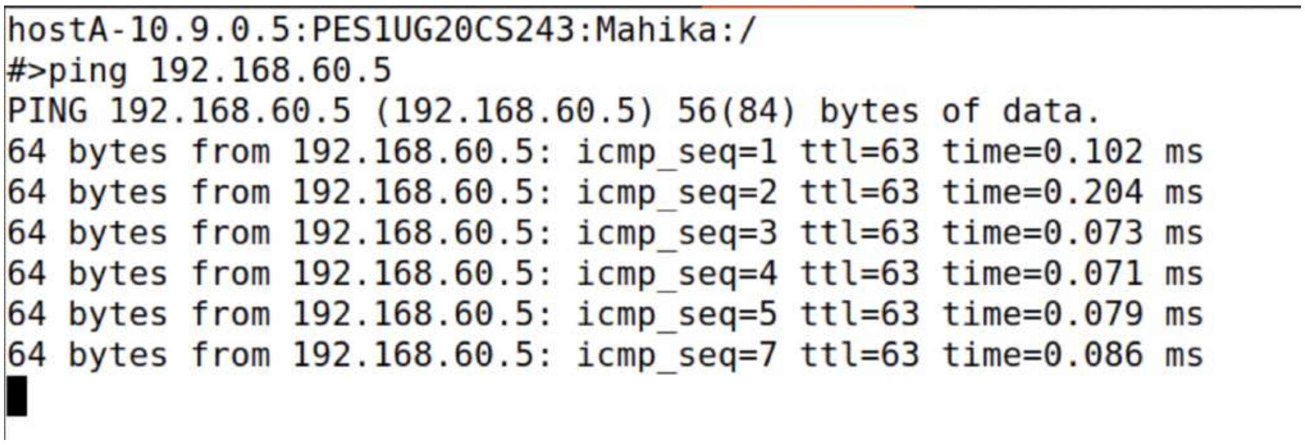


```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   27  2268 ACCEPT     all  --  *      *       10.9.0.5             0.0.0.0/0            limit: avg 10/min burst 5
  106  8904 DROP       all  --  *      *       10.9.0.5             0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       10.9.0.5             0.0.0.0/0            limit: avg 10/min burst 5

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
seed-router:PES1UG20CS243:Mahika:/
#>
```

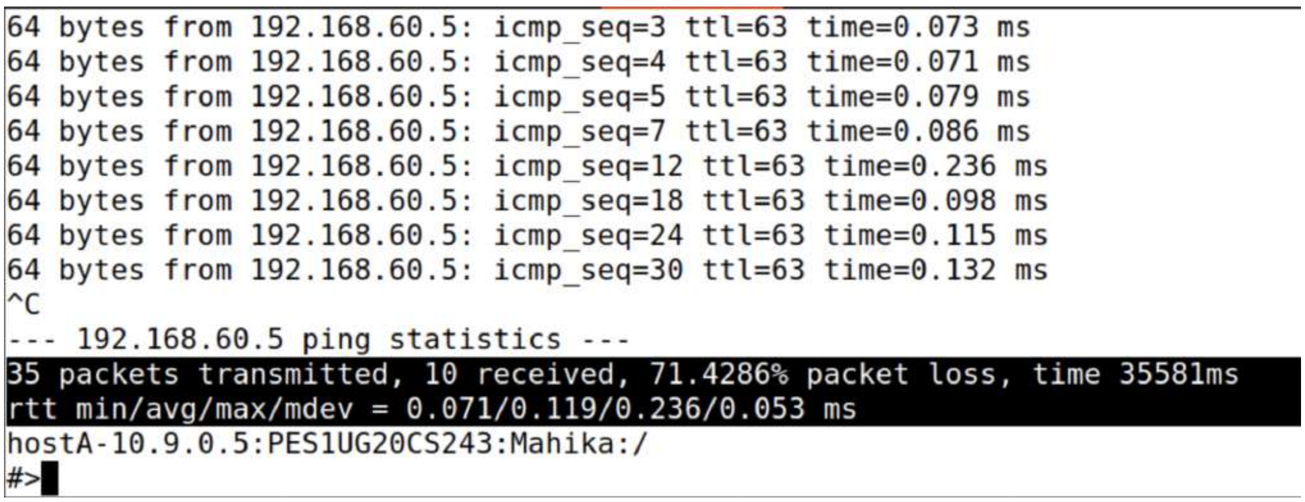
On host A - 10.9.0.5

Command:

# ping 192.168.60.5



```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>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=0.102 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.204 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.073 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.079 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.086 ms
```



```
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.073 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.079 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.086 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.236 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.098 ms
64 bytes from 192.168.60.5: icmp_seq=24 ttl=63 time=0.115 ms
64 bytes from 192.168.60.5: icmp_seq=30 ttl=63 time=0.132 ms
^C
--- 192.168.60.5 ping statistics ---
35 packets transmitted, 10 received, 71.4286% packet loss, time 35581ms
rtt min/avg/max/mdev = 0.071/0.119/0.236/0.053 ms
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

Packets were dropped after 5, and hence we can see that the second rule was not necessary, as we get the same result in both cases.

Task 5: Load Balancing

The iptables are very powerful. In addition to firewalls, it has many other applications. We will not be able to cover all its applications in this lab, but we will be experimenting with one of the applications, load balancing. In this task, we will use it to balance three UDP servers running in the internal network. Let's first start the server on each of the hosts: 192.168.60.5, 192.168.60.6, and 192.168.60.7 (the -k option indicates that the server can receive UDP datagrams from multiple hosts):

```
# nc -luk 8080
```

We can use the statistic module to achieve load balancing.
There are two modes: random and nth. We will conduct experiments using both of them.

Using the nth mode (round-robin) - On the router container, we set the following rule, which applies to all the UDP packets going to port 8080. The nth mode of the statistic module is used; it implements a round-robin load balancing policy:

In this subtask we shall implement policies to equally divide the incoming packets between the three interval servers.

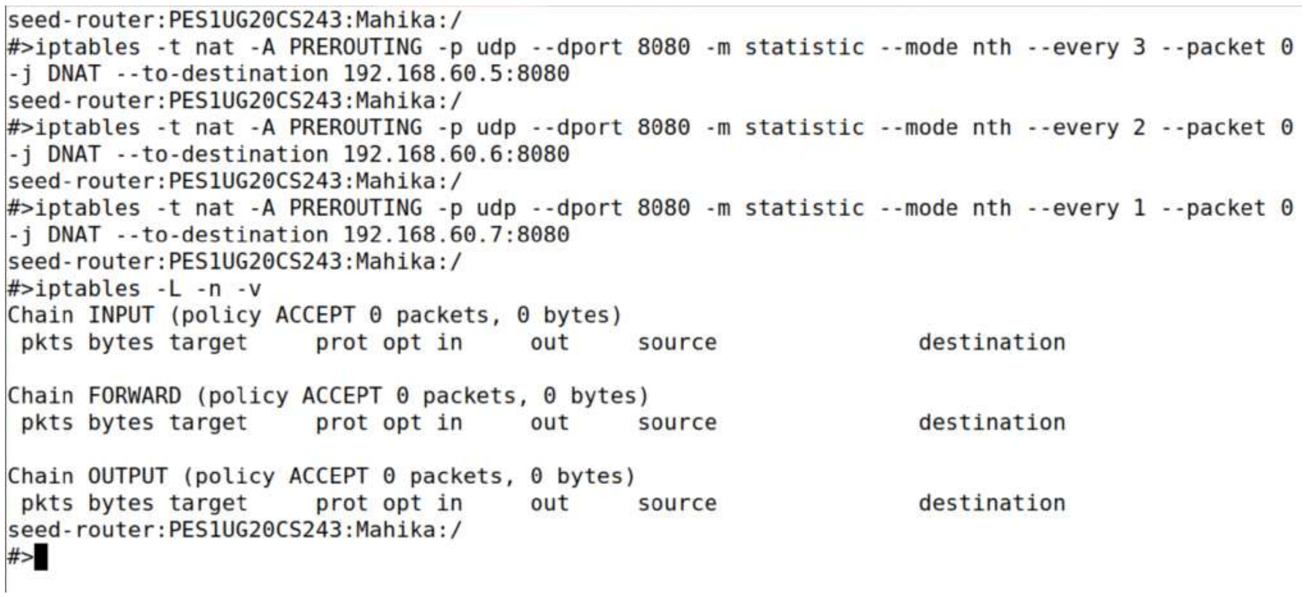
On the seed-router container:

Command:

```
# 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

# iptables -L -n -v



```
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
#>
```

These rules state that in the nat table, which shows all new connections, all new connections will be pre-routed to the destination servers 192.168.60.5, 192.168.60.6, and 192.168.60.7 in a round robin manner.

On host1 - 192.168.60.5, host2 - 192.168.60.6 and host3 - 192.168.60.7 start the server

Command:

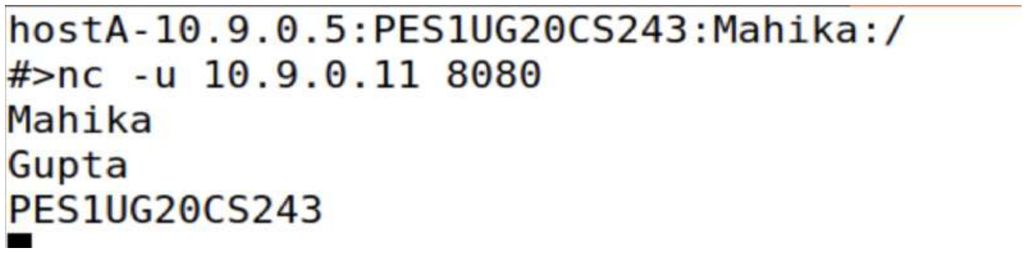
# nc -luk 8080

On host A - 10.9.0.5

Command:

# nc -u 10.9.0.11 8080

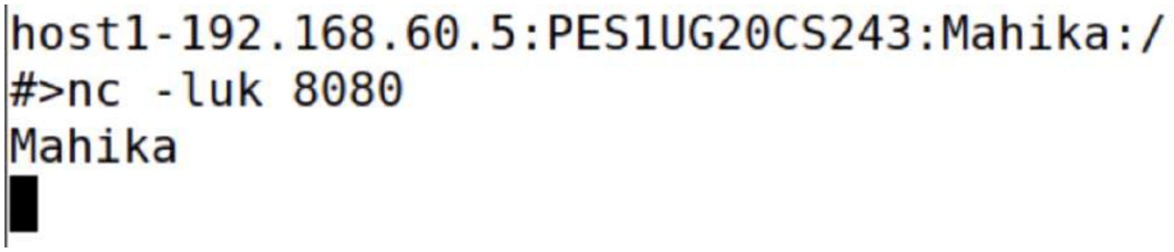
< enter 3 words, wait 30 seconds before entering the next word>



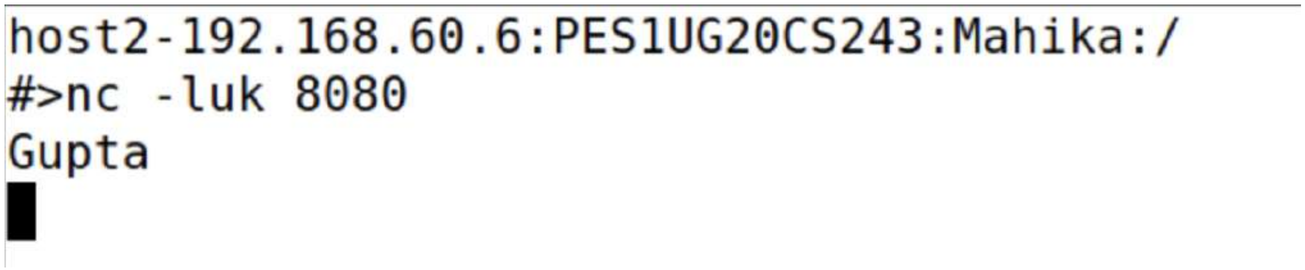
```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>nc -u 10.9.0.11 8080
Mahika
Gupta
PES1UG20CS243
```

‘Hello 1’ appears in the host 1 terminal, ‘Hello 2’ appears in the host 2 terminal etc. If you do not have a waiting period all the three statements will be considered as a single packet, so please wait for sometime before entering some text.

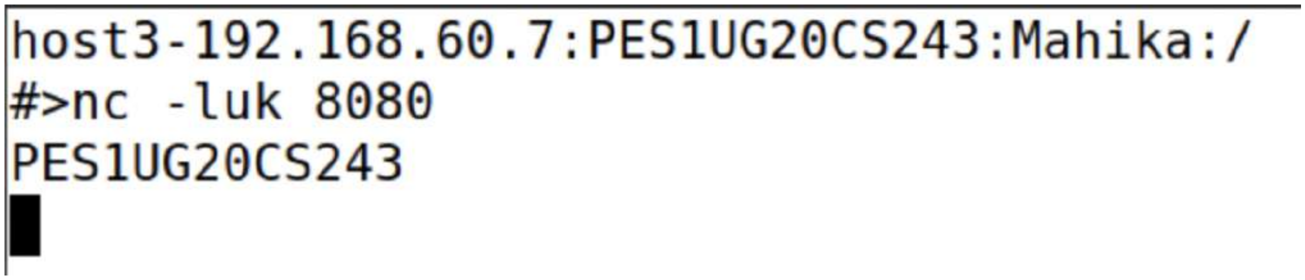
Students should enter their First Name, Last Name and SRN.



```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>nc -luk 8080
Mahika
```

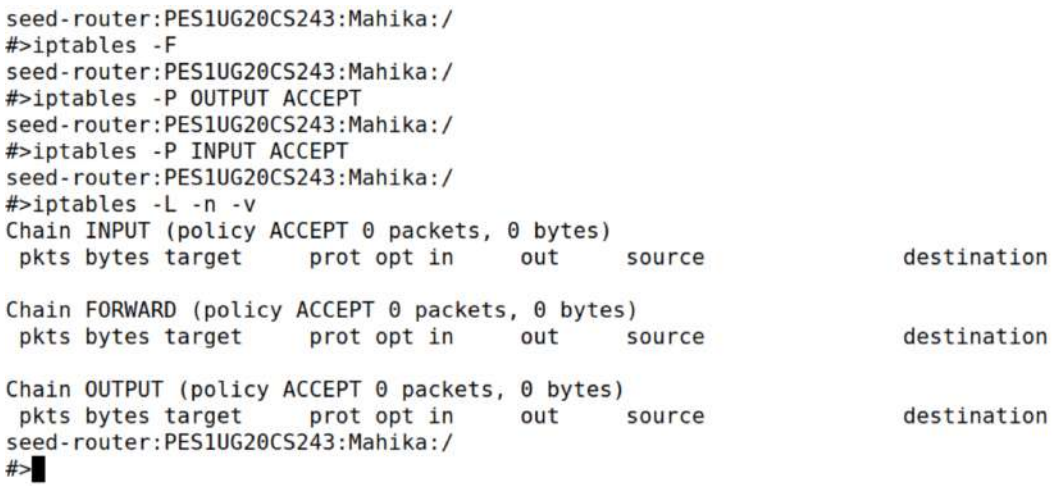


```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>nc -luk 8080
Gupta
```



```
host3-192.168.60.7:PES1UG20CS243:Mahika:/
#>nc -luk 8080
PES1UG20CS243
```

As we can see, the three words show up in each server in the respective order in which they were typed, which displays the round robin implementation.



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -F
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P OUTPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>iptables -P INPUT ACCEPT
seed-router:PES1UG20CS243:Mahika:/
#>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
seed-router:PES1UG20CS243:Mahika:/
#>
```

Using the random mode - Let's use a different mode to achieve load balancing. The following rule will select a matching packet with the probability P. You need to replace P with a probability number.

On the seed-router container:

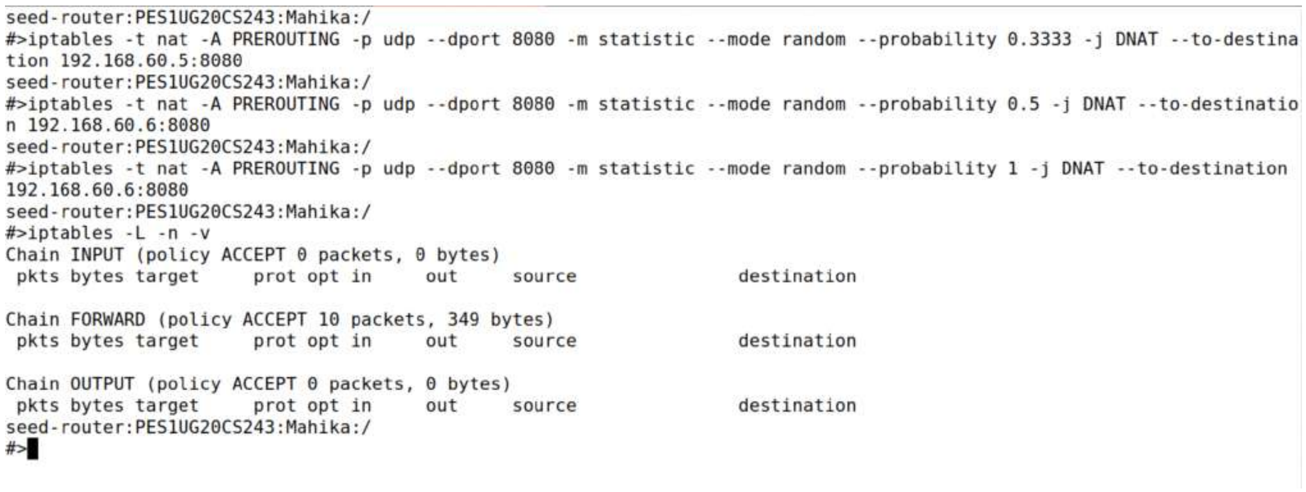
Command:

# 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 -m statistic --mode random --probability 1 -j DNAT --to-destination 192.168.60.6:8080

# iptables -L -n -v



```
seed-router:PES1UG20CS243:Mahika:/
#>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.3333 -j DNAT --to-destina
tion 192.168.60.5:8080
seed-router:PES1UG20CS243:Mahika:/
#>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-destinatio
n 192.168.60.6:8080
seed-router:PES1UG20CS243:Mahika:/
#>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1 -j DNAT --to-destination
192.168.60.6:8080
seed-router:PES1UG20CS243:Mahika:/
#>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 10 packets, 349 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
seed-router:PES1UG20CS243:Mahika:/
#>
```

According to these rules, all the udp messages should be displayed randomly at each host.

On host1 - 192.168.60.5, host2 - 192.168.60.6, host3 - 192.168.60.7 start the server

Command:

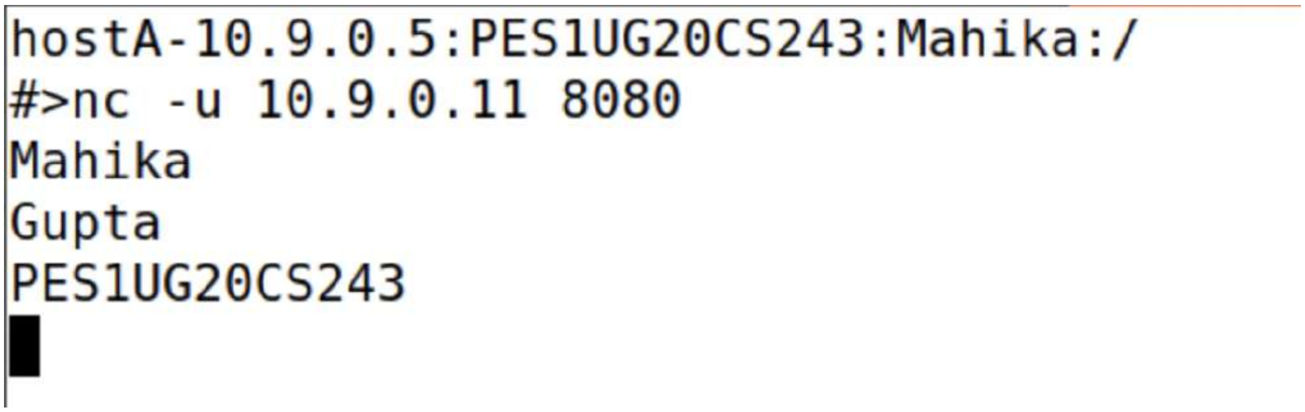
```
# nc -luk 8080
```

On host A - 10.9.0.5

Command:

```
# nc -u 10.9.0.11 8080
```

< enter 3 words, wait 30 seconds before entering the next word>



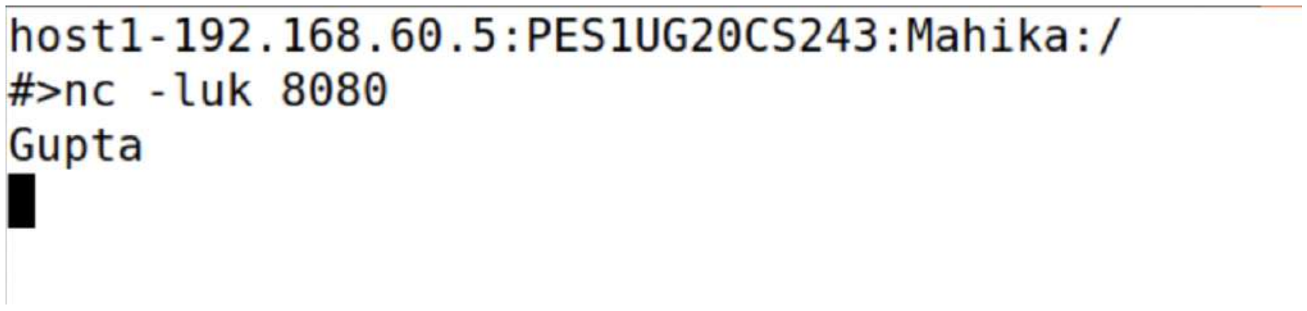
```
hostA-10.9.0.5:PES1UG20CS243:Mahika:/
#>nc -u 10.9.0.11 8080
Mahika
Gupta
PES1UG20CS243
```

For example :

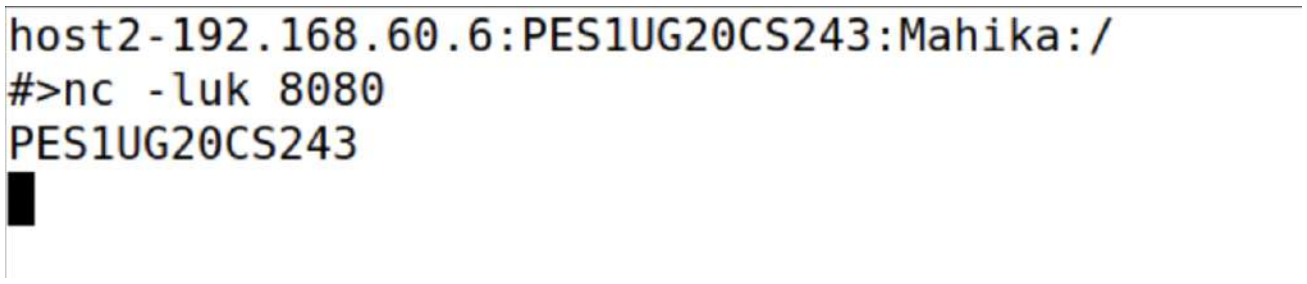
```
# nc -u 10.9.0.11 8080
Hello 1
Wait 30 seconds
Hello 2
Wait 30 seconds
Hello 3
```

This time each server **<u>may not get the same amount of traffic</u>**.

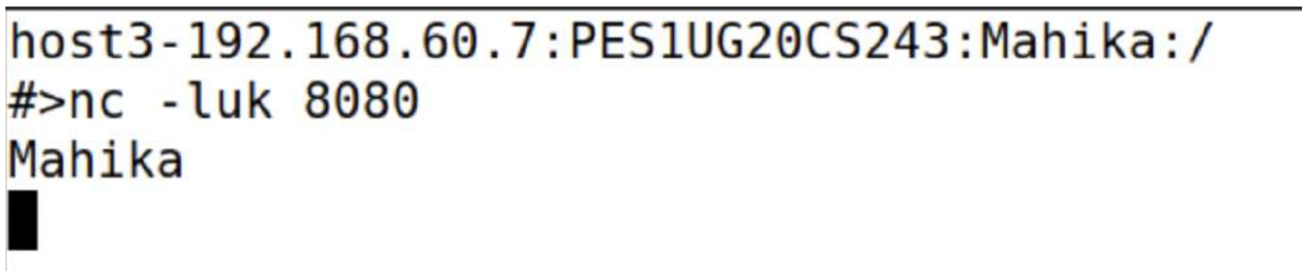
Please provide some explanation for the rules with the appropriate screenshots.



```
host1-192.168.60.5:PES1UG20CS243:Mahika:/
#>nc -luk 8080
Gupta
█
```



```
host2-192.168.60.6:PES1UG20CS243:Mahika:/
#>nc -luk 8080
PES1UG20CS243
█
```



```
host3-192.168.60.7:PES1UG20CS243:Mahika:/
#>nc -luk 8080
Mahika
█
```

As we can see the three messages are displayed in random host terminals, in no particular order.