PES UNIVERSITY

Department of Computer Science & Engineering

Computer Network Security

UE23CS343AB6

Assignment 7 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

Computer Network Security

UE23CS343AB6

Explanation of the config files as part of this lab setup:

This lab uses a small multi-container environment (router container + helper code) implemented with Docker images, source code and host-mounted volumes. The top-level tree is:

```
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %ls
 docker-compose.yml      router
 Files                   volumes
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %tree
 .
 ├── docker-compose.yml
 ├── Files
 │   ├── kernel_module
 │   │   ├── hello.c
 │   │   └── Makefile
 │   └── packet_filter
 │       ├── Makefile
 │       └── seedFilter.c
 ├── router
 │   └── Dockerfile
 └── volumes
     ├── kernel_module
     │   ├── hello.c
     │   └── Makefile
     └── packet_filter
         ├── Makefile
         ├── seedBlock.c
         ├── seedFilter.c
         └── seedPrint.c

 8 directories, 12 files
 pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

Below is the role and purpose of each file and its configurations:

docker-compose.yml

Defines the multi-container composition for the lab. It describes services to bring up (for example, a router service), how volumes are mounted from ./volumes into containers, network configuration, and any runtime privileges (e.g., privileged: true or cap_add: NET_ADMIN) required to load kernel modules or use raw sockets. This is the entrypoint for launching the lab environment with a single docker-compose up command.

Files/

A repository copy of the source used in the lab. These are kept as reference or for building locally before being mounted into containers.

- Files/kernel_module/hello.c
  A simple example kernel module source (commonly a “hello world” kernel module). Used to demonstrate how to compile, load, and unload a kernel module inside a privileged container or on the host during the lab.
- Files/kernel_module/Makefile
  Makefile to compile the kernel module (hello.c) against the appropriate kernel headers. Shows compile targets such as make, make clean, and possibly an install/insmod helper.
- Files/packet_filter/seedFilter.c
  A user-space packet filtering program (single source) that demonstrates packet capture/filtering logic or integration points with the kernel module. The Makefile compiles this into a runnable binary used to observe/filter traffic inside the router container.

router/

Contains the Dockerfile used to build the router container image that runs the lab’s networking components (packet filter, kernel module build tools, or test scripts).

- router/Dockerfile
  The image definition used for the router service. Typical responsibilities:
  - Starts from a base image (e.g., ubuntu), installs required build tools (build-essential, linux-headers), and network utilities (ip, tcpdump) so you can compile and load kernel modules and run packet filtering binaries inside the container.
  - Copies or expects mounted volumes (from volumes/) so you can build from the host-provided source.

    - May set an entrypoint or default command to keep the container running (e.g., a small shell wrapper or supervisor) so you can docker exec in to build/load modules or run tests.

volumes/

Host-side directories that are mounted into containers by docker-compose.yml. These contain the actual code the containers run or build. Using volumes makes iteration easy: edit files on your host and the container sees the changes immediately.

- volumes/kernel_module/hello.c
  Same kernel module source as in Files/ but placed here to be mounted into the router container. You typically docker exec into the container, cd /mnt/volumes/kernel_module, then make and insmod ./hello.ko (requires privileged container and matching kernel headers).
- volumes/kernel_module/Makefile
  The build instructions for the kernel module used inside the container. Keeps compilation consistent when building inside the lab container.
- volumes/packet_filter/Makefile
  Makefile to build the packet filter and other small user-space utilities. When mounted into the router container you run make to produce the binaries.
- volumes/packet_filter/seedBlock.c
  A source file that likely implements blocking logic (e.g., dropping certain packets). Used as one behavior of the packet-filtering toolset.
- volumes/packet_filter/seedFilter.c
  A packet capture / filter implementation — probably the main filtering binary that inspects packets and optionally hands decisions to seedBlock or logs decisions.
- volumes/packet_filter/seedPrint.c
  A helper program that prints/logs packets or events (useful to observe what the filter sees). Helpful for debugging and lab demonstrations (shows matched packets, counters, etc).

Change shell name:

PS1="terminal1:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="terminal2:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="seed-router:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="host1-192.168.60.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="host2-192.168.60.6:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="host3-192.168.60.7:PES1UG23CS433:PranavHemanth:\w\n\$>"

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: 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.

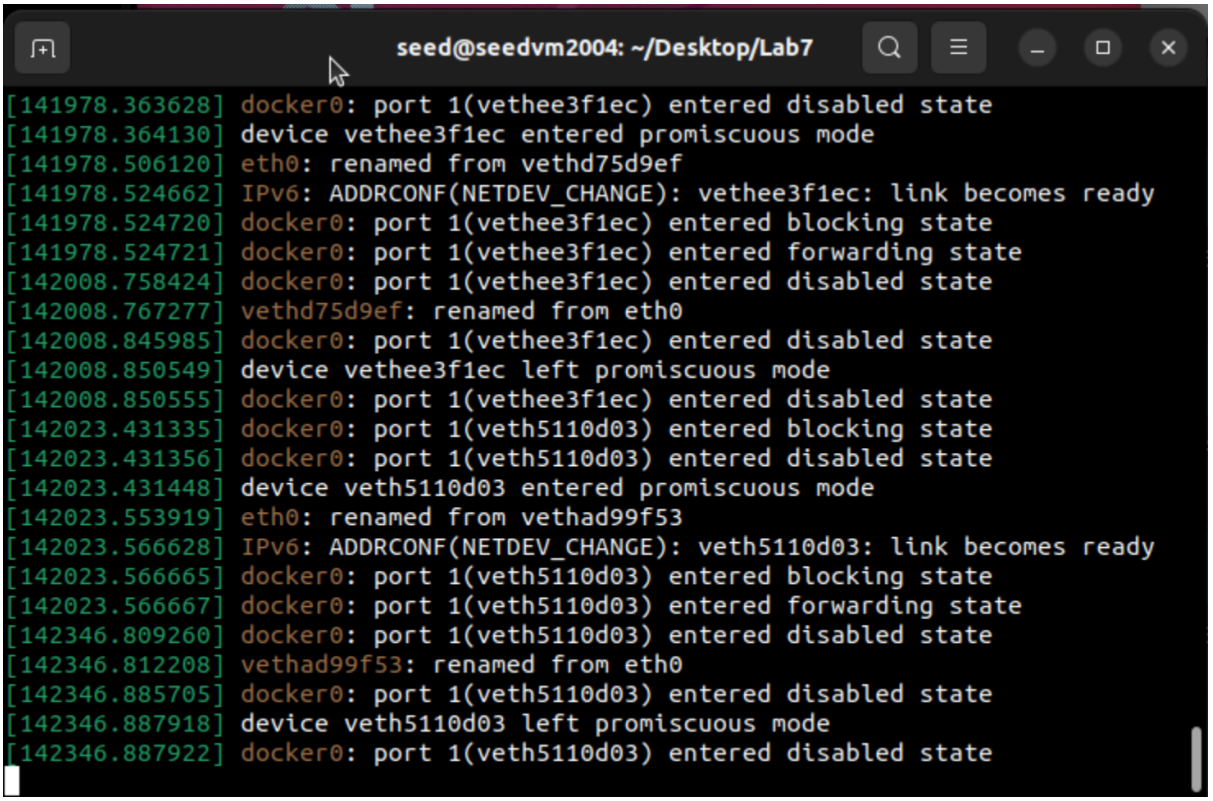
Task 1.A: Implement a Simple Kernel Module

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.

Step1: On the VM- Open two Terminal Tabs, one to load the module and the other to view the messages.

terminal 1:

1) Initial Kernel Buffer:
   Command: sudo dmesg -k -w



terminal 2:

2) Commands:

$ make

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$make
make -C /lib/modules/5.15.0-153-generic/build M=/home/seed/Desktop/Lab7/volumes/
kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-153-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
  You are using:           gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-153-generic'
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$
```

$ sudo insmod hello.ko (inserting a module)

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$ls
hello.c   hello.mod    hello.mod.o  Makefile       Module.symvers
hello.ko  hello.mod.c  hello.o      modules.order
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$sudo insmod hello.ko
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$
```

$ lsmod | grep hello (list modules)

```
$lsmod | grep hello
hello                  16384  0
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$
```

$ sudo rmmod hello

```
$sudo rmmod hello
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/kernel_module
$
```

terminal 1:

```
[141978.524721] docker0: port 1(vethee3f1ec) entered forwarding state
[142008.758424] docker0: port 1(vethee3f1ec) entered disabled state
[142008.767277] vethd75d9ef: renamed from eth0
[142008.845985] docker0: port 1(vethee3f1ec) entered disabled state
[142008.850549] device vethee3f1ec left promiscuous mode
[142008.850555] docker0: port 1(vethee3f1ec) entered disabled state
[142023.431335] docker0: port 1(veth5110d03) entered blocking state
[142023.431356] docker0: port 1(veth5110d03) entered disabled state
[142023.431448] device veth5110d03 entered promiscuous mode
[142023.553919] eth0: renamed from vethad99f53
[142023.566628] IPv6: ADDRCONF(NETDEV_CHANGE): veth5110d03: link becomes ready
[142023.566665] docker0: port 1(veth5110d03) entered blocking state
[142023.566667] docker0: port 1(veth5110d03) entered forwarding state
[142346.809260] docker0: port 1(veth5110d03) entered disabled state
[142346.812208] vethad99f53: renamed from eth0
[142346.885705] docker0: port 1(veth5110d03) entered disabled state
[142346.887918] device veth5110d03 left promiscuous mode
[142346.887922] docker0: port 1(veth5110d03) entered disabled state
[143930.554411] hello: loading out-of-tree module taints kernel.
[143930.568461] hello: module verification failed: signature and/or required key
 missing - tainting kernel
[143930.601743] Hello World!
[143990.698431] Bye-bye World!.
```

Here when the module is inserted and removed, we monitor the kernel messages with sudo dmesg -k -w and observe the corresponding output.

Explanation:
In this task, we developed and tested a simple Loadable Kernel Module (LKM) to show how modules can be dynamically inserted into or removed from the Linux kernel without requiring a system reboot. The hello.c program outputs “Hello World!” when the module is loaded and “Bye-bye World!” when it is unloaded.

These messages are printed to the kernel log, which can be monitored using the dmesg command. To see this behavior, two terminals are used: one to continuously monitor kernel messages with sudo dmesg -k -w, and another to compile and manage the module using commands such as make, sudo insmod hello.ko (to insert the module), lsmod | grep hello (to verify it is loaded), and sudo rmmod hello (to remove it).

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.

Task:

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.

We can see that seedBlock is uncommented instead of seedFilter in the Makefile. For the task we need seedFilter, hence we edit the Makefile

```
Makefile  seedBlock.c  seedFilter.c  seedPrint.c
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$cat Makefile
#obj-m += seedFilter.o
#obj-m += seedPrint.o
obj-m += seedBlock.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

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

ins:
        sudo dmesg -C
        sudo insmod seedFilter.ko

rm:
        sudo rmmod seedFilter

terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$nano Makefile
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$gedit Makefile
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
```

Step1: 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

3) Command: dig @8.8.8.8 www.example.com

VM:

```
terminal1:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7
$dig @8.8.8.8 www.example.com

; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37827
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        238     IN      CNAME   www.example.com-v4.edgesuite.net.
www.example.com-v4.edgesuite.net. 14567 IN CNAME a1422.dscr.akamai.net.
a1422.dscr.akamai.net.  20      IN      A       23.44.10.218
a1422.dscr.akamai.net.  20      IN      A       23.44.10.227

;; Query time: 43 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Fri Oct 17 03:09:48 UTC 2025
;; MSG SIZE  rcvd: 154

terminal1:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7
$
```

This output shows a normal and successful dig command. A DNS query was sent to Google's server (8.8.8.8), which correctly responded with the IP addresses for www.example.com. This confirms that the network connection is functioning properly before the firewall is activated.

Step2: Open a new terminal to view kernel messages and execute the following -

4) Command: sudo dmesg -k -w

```
eration="profile_replace" profile="unconfined" name="snap.firefox.hook.install"
pid=119610 comm="apparmor_parser"
[149013.238130] audit: type=1400 audit(1760597885.711:116): apparmor="STATUS" op
eration="profile_replace" profile="unconfined" name="snap.firefox.hook.post-refr
esh" pid=119611 comm="apparmor_parser"
[150739.749866] e1000e 0000:02:00.0 ens160: NIC Link is Down
[151134.006497] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[157052.313815] e1000e 0000:02:00.0 ens160: NIC Link is Down
[157065.247208] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[168698.988008] e1000e 0000:02:00.0 ens160: NIC Link is Down
[168819.320644] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[168993.353389] e1000e 0000:02:00.0 ens160: NIC Link is Down
[169013.821960] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[169023.603086] e1000e 0000:02:00.0 ens160: NIC Link is Down
[169150.657967] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[169905.517159] e1000e 0000:02:00.0 ens160: NIC Link is Down
[169920.039333] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
```

5) Uncomment - ‘obj-m += seedFilter.o’ and comment the other two.

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$cat Makefile
obj-m += seedFilter.o
#obj-m += seedPrint.o
#obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

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

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter

terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

6) Command:
   $ make

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$make
make -C /lib/modules/5.15.0-153-generic/build M=/home/seed/Desktop/Lab7/volumes/
packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-153-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
  You are using:           gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
  CC [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedFilter.o
  MODPOST /home/seed/Desktop/Lab7/volumes/packet_filter/Module.symvers
  CC [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedFilter.ko
  BTF [M] /home/seed/Desktop/Lab7/volumes/packet_filter/seedFilter.ko
Skipping BTF generation for /home/seed/Desktop/Lab7/volumes/packet_filter/seedFi
lter.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-153-generic'
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

$ sudo insmod seedFilter.ko

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$ls
Makefile        seedBlock.c    seedFilter.mod    seedFilter.o
modules.order   seedFilter.c   seedFilter.mod.c  seedPrint.c
Module.symvers  seedFilter.ko  seedFilter.mod.o
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$sudo insmod seedFilter.ko
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
```

$ lsmod | grep seedFilter

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$lsmod | grep seedFilter
seedFilter             16384  0
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

terminal1:

```
[168993.353389] e1000e 0000:02:00.0 ens160: NIC Link is Down
[169013.821960] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[169023.603086] e1000e 0000:02:00.0 ens160: NIC Link is Down
[169150.657967] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[169905.517159] e1000e 0000:02:00.0 ens160: NIC Link is Down
[169920.039333] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[170524.828575] Registering filters.
[170548.037363] *** LOCAL_OUT
[170548.037560]     172.16.73.128  --> 34.107.243.93 (TCP)
[170548.044444] *** LOCAL_OUT
[170548.044464]     172.16.73.128  --> 34.107.243.93 (TCP)
[170616.312830] *** LOCAL_OUT
[170616.312847]     172.16.73.128  --> 8.8.8.8 (UDP)
[170616.312925] *** Dropping 8.8.8.8 (UDP), port 53
[170621.313636] *** LOCAL_OUT
[170621.313644]     172.16.73.128  --> 8.8.8.8 (UDP)
[170621.313716] *** Dropping 8.8.8.8 (UDP), port 53
[170626.321812] *** LOCAL_OUT
[170626.321865]     172.16.73.128  --> 8.8.8.8 (UDP)
[170626.322055] *** Dropping 8.8.8.8 (UDP), port 53
```

This dmesg log shows why the dig command timed out. The kernel is printing messages from your module, showing it's dropping the outgoing UDP packets destined for 8.8.8.8 on port 53 (the DNS port). The filter is correctly identifying and blocking the target traffic.

terminal2:

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$dig @8.8.8.8 www.example.com
;; communications error to 8.8.8.8#53: timed out
;; communications error to 8.8.8.8#53: timed out
;; communications error to 8.8.8.8#53: timed out

; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; no servers could be reached
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

Here, the seedFilter.ko module was successfully loaded using insmod and verified with lsmod. When the same digcommand is executed afterward, it results in a “timed out” error, indicating that the firewall is now active and functioning as intended.

7) Remove the module by executing
   Command: $ sudo rmmod seedFilter

terminal1:

```
[171271.276823] *** LOCAL_OUT
[171271.276830]     172.16.73.128  --> 151.101.1.91 (TCP)
[171271.292805] *** LOCAL_OUT
[171271.292812]     172.16.73.128  --> 151.101.1.91 (TCP)
[171271.298690] *** LOCAL_OUT
[171271.298694]     172.16.73.128  --> 34.107.243.93 (TCP)
[171323.282093] *** LOCAL_OUT
[171323.282184]     172.16.73.128  --> 34.107.243.93 (TCP)
[171323.290864] *** LOCAL_OUT
[171323.290872]     172.16.73.128  --> 34.107.243.93 (TCP)
[171323.291070] *** LOCAL_OUT
[171323.291073]     172.16.73.128  --> 34.107.243.93 (TCP)
[171323.295803] *** LOCAL_OUT
[171323.295807]     172.16.73.128  --> 34.107.243.93 (TCP)
[171324.288976] *** LOCAL_OUT
[171324.288989]     172.16.73.128  --> 151.101.1.91 (TCP)
[171324.290589] *** LOCAL_OUT
[171324.290598]     172.16.73.128  --> 151.101.1.91 (TCP)
[171324.290784] *** LOCAL_OUT
[171324.290788]     172.16.73.128  --> 151.101.1.91 (TCP)
[171324.294645] *** LOCAL_OUT
[171324.294650]     172.16.73.128  --> 151.101.1.91 (TCP)
[171415.656021] The filters are being removed.
```

terminal2:

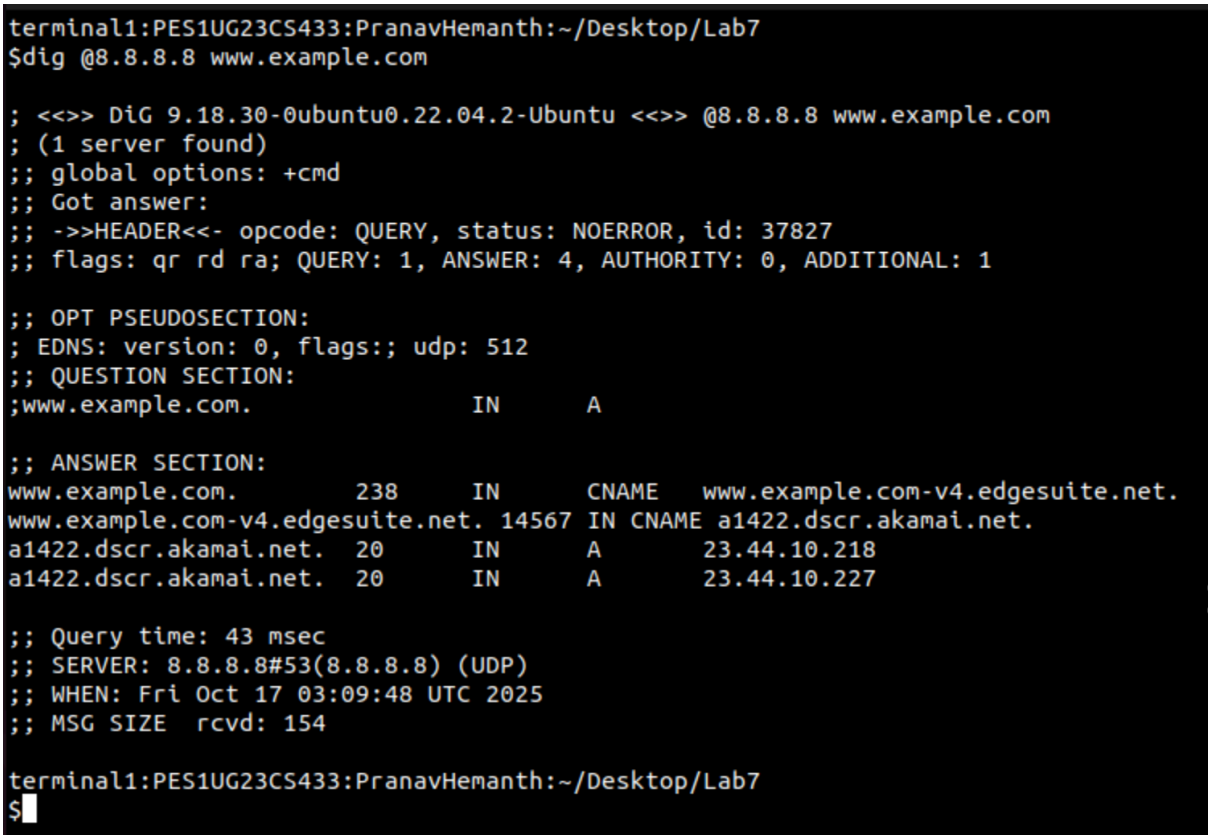
```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$sudo rmmod seedFilter
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$lsmod | grep seedFilter
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

8) To clear the kernel messages execute -
   Command: $ dmesg -C

```
terminal1:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7
$sudo dmesg -C
terminal1:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7
$sudo dmesg -k -w
```

Step3: Lets retry running dig

9) Command: dig @8.8.8.8 www.example.com



Running the dig command works normally, as expected

Explanation:

In this task, we implemented a simple firewall using the Linux Netfilter framework by loading a custom kernel module (LKM). The module, *seedFilter.ko*, filters outgoing UDP packets destined for Google's DNS server (8.8.8.8) on port 53. Before enabling the firewall, a DNS lookup using the dig command works normally, confirming internet connectivity. After editing the Makefile to enable the firewall module and successfully compiling and inserting it into the kernel, the firewall begins intercepting and dropping the targeted packets. This behavior is confirmed through kernel logs in dmesg, where the dropped packets are reported, and through the dig command timing out, showing that DNS requests are blocked as intended. Finally, removing the module restores normal functionality, allowing DNS packets to pass again.

Task:

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.
    a. NF_INET_PRE_ROUTING
    b. NF_INET_LOCAL_IN
    c. NF_INET_FORW ARD
    d. NF_INET_LOCAL_OUT
    e. NF_INET_POST_ROUTING

Step1: Change the code for this task

10) Uncomment - 'obj-m += seedPrint.o' and comment the other two.

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$gedit Makefile
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$cat Makefile
#obj-m += seedFilter.o
obj-m += seedPrint.o
#obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

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

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter

terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

Step2: Open a new terminal to view kernel messages and execute the following -

11) Command: sudo dmesg -k -w

```
terminal1:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7
$sudo dmesg -k -w
[171660.399587] e1000e 0000:02:00.0 ens160: NIC Link is Down
[171674.242002] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[172161.913594] device br-b277b226ad85 entered promiscuous mode
[172192.580423] device br-b277b226ad85 left promiscuous mode
[172200.360087] device br-b277b226ad85 entered promiscuous mode
[172281.318177] device br-b277b226ad85 left promiscuous mode
[172878.860747] device br-b277b226ad85 entered promiscuous mode
```

(Here i opened wireshark just to monitor hence the promiscuous mode)

12)Command:

$ make

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$make
make -C /lib/modules/5.15.0-153-generic/build M=/home/seed/Desktop/Lab7/volumes/
packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-153-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
  You are using:           gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
  CC [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedPrint.o
  MODPOST /home/seed/Desktop/Lab7/volumes/packet_filter/Module.symvers
  CC [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedPrint.mod.o
  LD [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedPrint.ko
  BTF [M] /home/seed/Desktop/Lab7/volumes/packet_filter/seedPrint.ko
Skipping BTF generation for /home/seed/Desktop/Lab7/volumes/packet_filter/seedPr
int.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-153-generic'
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

$ sudo insmod seedPrint.ko

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$ls
Makefile        seedFilter.c      seedFilter.mod.o  seedPrint.mod
modules.order   seedFilter.ko     seedFilter.o      seedPrint.mod.c
Module.symvers  seedFilter.mod    seedPrint.c       seedPrint.mod.o
seedBlock.c     seedFilter.mod.c  seedPrint.ko      seedPrint.o
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$sudo insmod seedPrint.ko
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
```

$ lsmod | grep seedPrint

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$lsmod | grep seedPrint
seedPrint              16384  0
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

terminal1:

```
terminal1:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7
$sudo dmesg -k -w
[171660.399587] e1000e 0000:02:00.0 ens160: NIC Link is Down
[171674.242002] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[172161.913594] device br-b277b226ad85 entered promiscuous mode
[172192.580423] device br-b277b226ad85 left promiscuous mode
[172200.360087] device br-b277b226ad85 entered promiscuous mode
[172281.318177] device br-b277b226ad85 left promiscuous mode
[172878.860747] device br-b277b226ad85 entered promiscuous mode
[173192.383821] Registering filters.
```

13)Now run the below command

Command:

dig @8.8.8.8 www.example.com

terminal2:

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$dig @8.8.8.8 www.example.com

; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44346
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        27      IN      CNAME   www.example.com-v4.edgesuite.net.
www.example.com-v4.edgesuite.net. 6220 IN CNAME a1422.dscr.akamai.net.
a1422.dscr.akamai.net.  20      IN      A       23.44.10.227
a1422.dscr.akamai.net.  20      IN      A       23.44.10.218

;; Query time: 48 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Fri Oct 17 04:08:43 UTC 2025
;; MSG SIZE  rcvd: 154

terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

This time, running the dig command works normally, as expected, since this module is designed to print output rather than block execution.

terminal1:

```
[173313.089056]     127.0.0.1  --> 224.0.0.251 (UDP)
[173313.089116] *** PRE_ROUTING
[173313.089117]     127.0.0.1  --> 224.0.0.251 (UDP)
[173313.089123] *** LOCAL_IN
[173313.089125]     127.0.0.1  --> 224.0.0.251 (UDP)
[173313.089801] *** LOCAL_OUT
[173313.089804]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089828] *** POST_ROUTING
[173313.089829]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089870] *** PRE_ROUTING
[173313.089871]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089875] *** LOCAL_IN
[173313.089875]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089889] *** POST_ROUTING
[173313.089891]     10.9.0.1  --> 224.0.0.251 (UDP)
[173342.467119] *** LOCAL_OUT
[173342.467136]     172.16.73.128  --> 8.8.8.8 (UDP)
[173342.467266] *** POST_ROUTING
[173342.467266]     172.16.73.128  --> 8.8.8.8 (UDP)
[173342.515902] *** PRE_ROUTING
[173342.515906]     8.8.8.8  --> 172.16.73.128 (UDP)
[173342.515936] *** LOCAL_IN
[173342.515937]     8.8.8.8  --> 172.16.73.128 (UDP)
```

14)Remove the module by executing

Command: $ sudo rmmod seedPrint

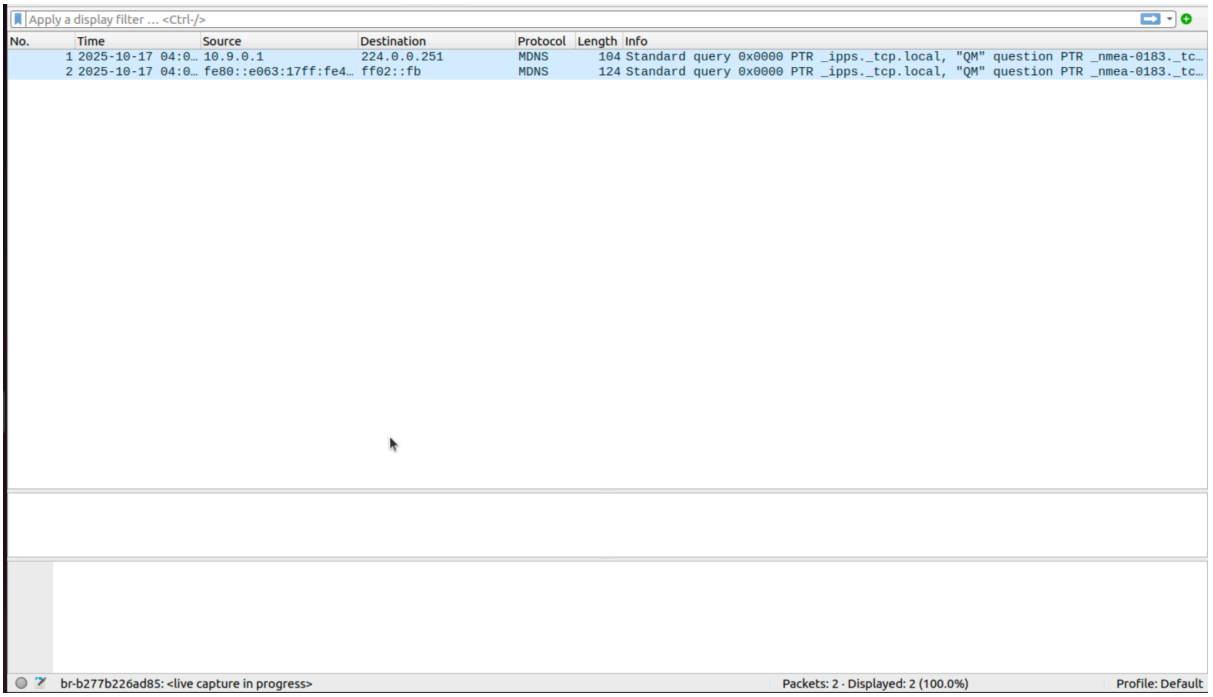
terminal1:

```
[173313.089116] *** PRE_ROUTING
[173313.089117]     127.0.0.1  --> 224.0.0.251 (UDP)
[173313.089123] *** LOCAL_IN
[173313.089125]     127.0.0.1  --> 224.0.0.251 (UDP)
[173313.089801] *** LOCAL_OUT
[173313.089804]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089828] *** POST_ROUTING
[173313.089829]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089870] *** PRE_ROUTING
[173313.089871]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089875] *** LOCAL_IN
[173313.089875]     10.9.0.1  --> 224.0.0.251 (UDP)
[173313.089889] *** POST_ROUTING
[173313.089891]     10.9.0.1  --> 224.0.0.251 (UDP)
[173342.467119] *** LOCAL_OUT
[173342.467136]     172.16.73.128  --> 8.8.8.8 (UDP)
[173342.467266] *** POST_ROUTING
[173342.467266]     172.16.73.128  --> 8.8.8.8 (UDP)
[173342.515902] *** PRE_ROUTING
[173342.515906]     8.8.8.8  --> 172.16.73.128 (UDP)
[173342.515936] *** LOCAL_IN
[173342.515937]     8.8.8.8  --> 172.16.73.128 (UDP)
[173445.593726] The filters are being removed.
```

terminal2:

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$sudo rmmod seedPrint
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$lsmod | grep seedPrint
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

Wireshark: (no useful packet capture)



Explanation:
In this task, we used the Netfilter framework to observe how packets move through different stages of the Linux networking stack. The seedPrint.ko kernel module was modified to attach the printInfo function to all five Netfilter hook points:

- NF_INET_PRE_ROUTING: Packets arriving from the network before routing decisions
- NF_INET_LOCAL_IN: Packets destined for the local machine
- NF_INET_FORWARD: Packets being routed through the machine to another destination
- NF_INET_LOCAL_OUT: Outgoing packets generated by the local system
- NF_INET_POST_ROUTING: Packets leaving the system after routing is completed

After loading the module, the dig command was run again. Since this module does not filter or drop traffic, DNS requests proceeded normally. However, kernel messages showed logging from each hook, demonstrating exactly where packets are intercepted during transmission and reception. Removing the module stopped the logging and returned the system to normal.

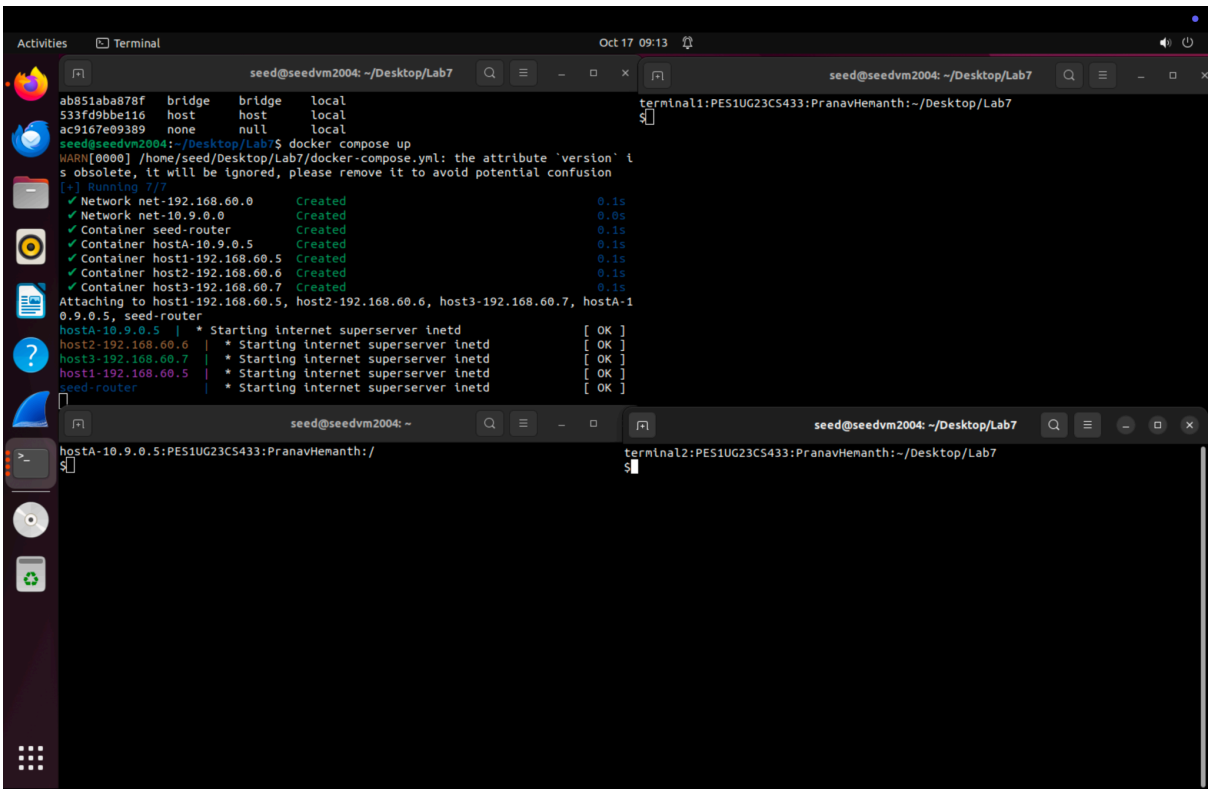
This experiment helps visualize the Linux packet processing path and understand when each Netfilter hook gets invoked based on the packet’s direction and destination.

Task:

3. Implement two more hooks to achieve the following:
    a. preventing other computers to ping the VM, and
    b. 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

   Three window setup:



Step1: Open a new terminal to view kernel messages and execute the following -

15) Command: sudo dmesg -k -w

16)

```
[180952.882402] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[181008.583444] e1000e 0000:02:00.0 ens160: NIC Link is Down
[181264.641181] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[182336.518425] e1000e 0000:02:00.0 ens160: NIC Link is Down
[182361.477063] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[182628.943116] e1000e 0000:02:00.0 ens160: NIC Link is Down
[182636.052154] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[182730.232006] e1000e 0000:02:00.0 ens160: NIC Link is Down
[182743.840793] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183016.578784] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183064.312569] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183104.645321] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183117.128770] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183122.241320] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183127.369041] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
```

Step2: Change the code for this task

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

```
#obj-m += seedFilter.o
#obj-m += seedPrint.o
obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

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

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter

```

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$gedit Makefile
^C
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$cat Makefile
#obj-m += seedFilter.o
#obj-m += seedPrint.o
obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

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

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter

terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

18)Command:

$ make

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$make
make -C /lib/modules/5.15.0-153-generic/build M=/home/seed/Desktop/Lab7/volumes/
packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-153-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
  You are using:           gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
  CC [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedBlock.o
  MODPOST /home/seed/Desktop/Lab7/volumes/packet_filter/Module.symvers
  CC [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedBlock.mod.o
  LD [M]  /home/seed/Desktop/Lab7/volumes/packet_filter/seedBlock.ko
  BTF [M] /home/seed/Desktop/Lab7/volumes/packet_filter/seedBlock.ko
Skipping BTF generation for /home/seed/Desktop/Lab7/volumes/packet_filter/seedBl
ock.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-153-generic'
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

$ sudo insmod seedPrint.ko

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$ls
Makefile        seedBlock.mod    seedFilter.ko     seedPrint.c      seedPrint.o
modules.order   seedBlock.mod.c  seedFilter.mod    seedPrint.ko
Module.symvers  seedBlock.mod.o  seedFilter.mod.c  seedPrint.mod
seedBlock.c     seedBlock.o      seedFilter.mod.o  seedPrint.mod.c
seedBlock.ko    seedFilter.c     seedFilter.o      seedPrint.mod.o
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$sudo insmod seedBlock.ko
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

$ lsmod | grep seedPrint

```
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$lsmod | grep seedBlock
seedBlock              16384  0
terminal2:PES1UG23CS433:PranavHemanth:~/Desktop/Lab7/volumes/packet_filter
$
```

terminal1:

```
[182336.518425] e1000e 0000:02:00.0 ens160: NIC Link is Down
[182361.477063] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[182628.943116] e1000e 0000:02:00.0 ens160: NIC Link is Down
[182636.052154] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[182730.232006] e1000e 0000:02:00.0 ens160: NIC Link is Down
[182743.840793] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183016.578784] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183064.312569] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183104.645321] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183117.128770] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183122.241320] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183127.369041] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183406.197561] Registering filters.
[183429.679860] *** LOCAL_OUT
[183429.680071]     172.16.73.128  --> 34.107.243.93 (TCP)
[183429.690897] *** LOCAL_OUT
[183429.690904]     172.16.73.128  --> 34.107.243.93 (TCP)
```

19)Now run the below command on Host A - 10.9.0.5

Command:

$ ping 10.9.0.1

hostA-10.9.0.5:

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8176ms

hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

terminal1:

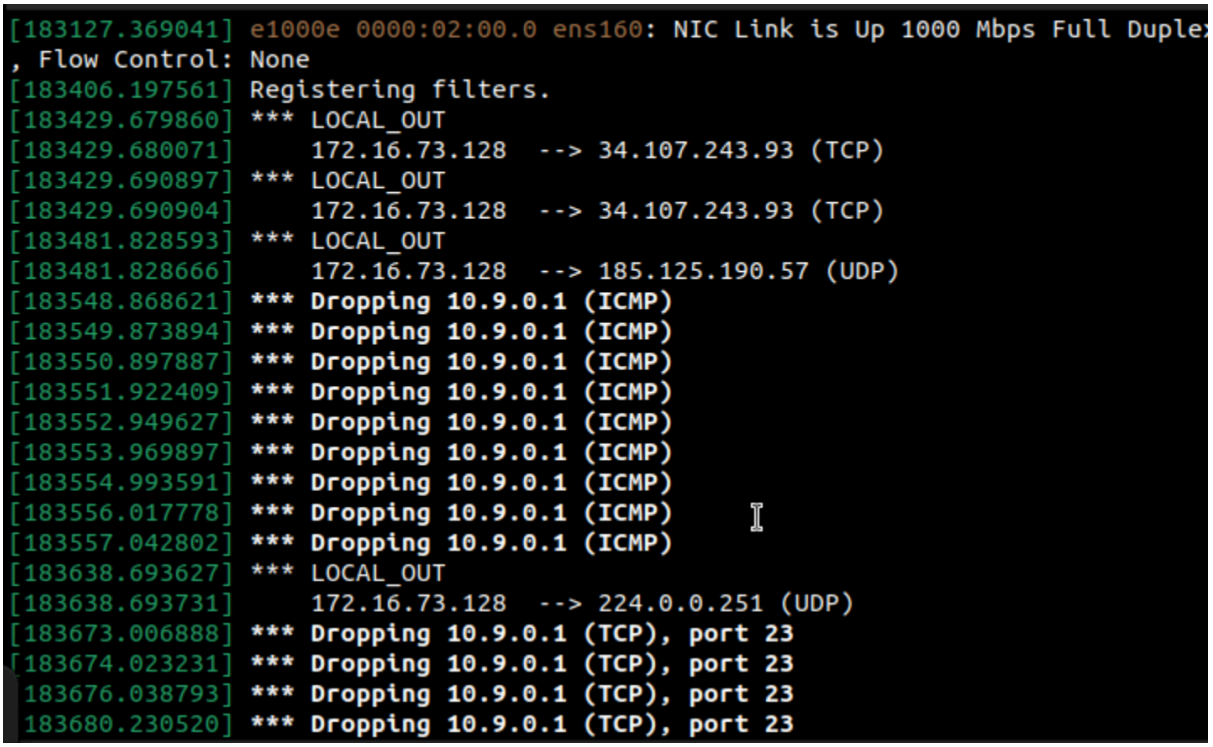
```
, Flow Control: None
[183104.645321] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183117.128770] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183122.241320] e1000e 0000:02:00.0 ens160: NIC Link is Down
[183127.369041] e1000e 0000:02:00.0 ens160: NIC Link is Up 1000 Mbps Full Duplex
, Flow Control: None
[183406.197561] Registering filters.
[183429.679860] *** LOCAL_OUT
[183429.680071]     172.16.73.128  --> 34.107.243.93 (TCP)
[183429.690897] *** LOCAL_OUT
[183429.690904]     172.16.73.128  --> 34.107.243.93 (TCP)
[183481.828593] *** LOCAL_OUT
[183481.828666]     172.16.73.128  --> 185.125.190.57 (UDP)
[183548.868621] *** Dropping 10.9.0.1 (ICMP)
[183549.873894] *** Dropping 10.9.0.1 (ICMP)
[183550.897887] *** Dropping 10.9.0.1 (ICMP)
[183551.922409] *** Dropping 10.9.0.1 (ICMP)
[183552.949627] *** Dropping 10.9.0.1 (ICMP)
[183553.969897] *** Dropping 10.9.0.1 (ICMP)
[183554.993591] *** Dropping 10.9.0.1 (ICMP)
[183556.017778] *** Dropping 10.9.0.1 (ICMP)
 183557.042802] *** Dropping 10.9.0.1 (ICMP)
]
```

$ telnet 10.9.0.1

hostA-10.9.0.5:

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet 10.9.0.1
Trying 10.9.0.1...
^C
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

terminal1:



Explanation:

In this task, we extended the Netfilter-based firewall to block two specific types of incoming traffic: ICMP echo requests (ping) and TCP connections to port 23 (telnet). We created two separate filtering functions but registered both to the same Netfilter hook, since both packet types are inbound and should be dropped before reaching the system. After compiling and loading the seedBlock.ko module, the VM became protected from external ping and telnet requests.

Testing from another host (10.9.0.5) confirmed the behavior. Ping attempts to the VM's IP address (10.9.0.1) resulted in 100 percent packet loss, and telnet attempts hung without establishing a connection. Meanwhile, kernel log messages showed that the firewall was detecting and dropping ICMP packets and TCP packets destined for port 23. This validates that the firewall rules are correctly identifying and blocking the targeted traffic.

The experiment demonstrates how custom Netfilter hooks can enforce basic firewall policies and prevent unauthorized access to critical services on the VM.

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.

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.

Task 2.A: Protecting the Router

Step1: Ensure all containers are running

```
seed@seedvm2004:~$ docker ps
CONTAINER ID   IMAGE                                   COMMAND
CREATED       STATUS          PORTS     NAMES
dccca3a58e38   seed-router-image                       "bash -c ' ip route …"
6 days ago    Up 4 minutes              seed-router
344d644c5360   handsonsecurity/seed-ubuntu:large-arm   "bash -c ' ip route …"
6 days ago    Up 4 minutes              host3-192.168.60.7
44e8f7e115a1   handsonsecurity/seed-ubuntu:large-arm   "bash -c ' ip route …"
6 days ago    Up 4 minutes              host1-192.168.60.5
64850cb17f44   handsonsecurity/seed-ubuntu:large-arm   "bash -c ' ip route …"
6 days ago    Up 4 minutes              hostA-10.9.0.5
a57272dc8a30   handsonsecurity/seed-ubuntu:large-arm   "bash -c ' ip route …"
6 days ago    Up 4 minutes              host2-192.168.60.6
seed@seedvm2004:~$
```

Step2: In order to view the current policies run the below command (on seed router)

20)Command: iptables -t filter -L -n

```
seed-router:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$
```

Above are initial configs

Step3: Now run the below on seed router and try to access from hostA

21) 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:

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 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:PES1UG23CS433:PranavHemanth:/
```

Step4: Now try to access (ping and telnet) the router from Host A - 10.9.0.5

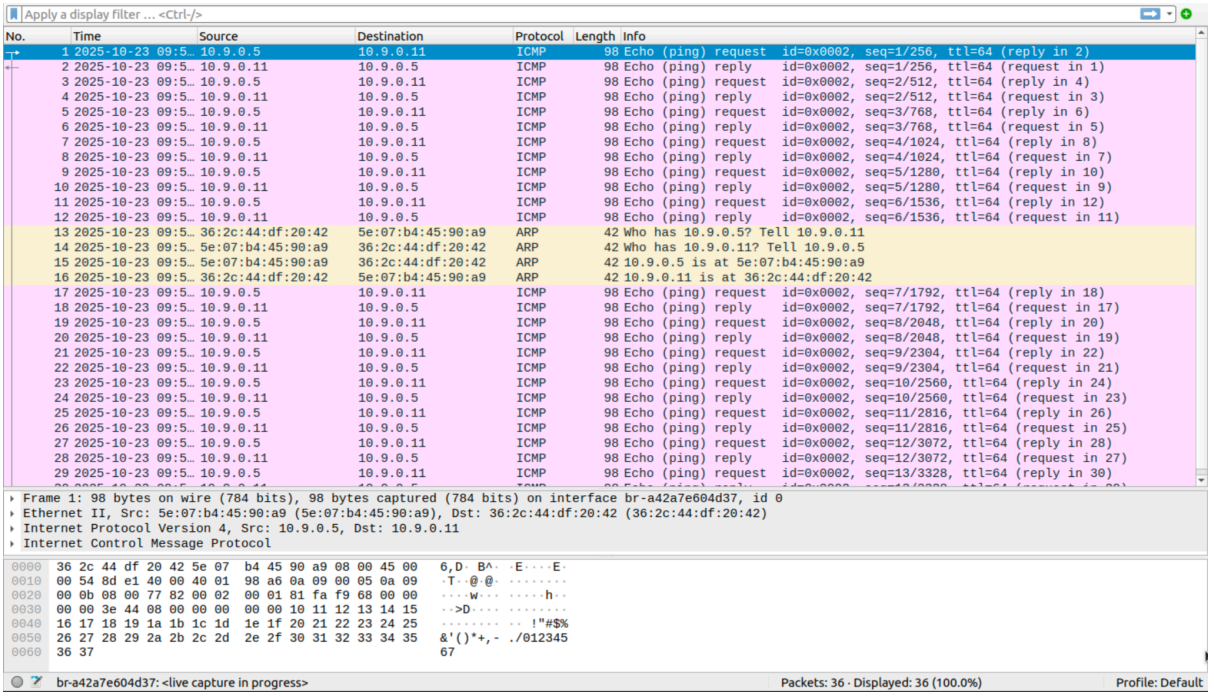
22) Command:

ping seed-router

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$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=0.40
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.16
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.14
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.14
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.13
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=6 ttl=64 time=0.11
 ms
^C
--- seed-router ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5102ms
rtt min/avg/max/mdev = 0.115/0.184/0.401/0.098 ms
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

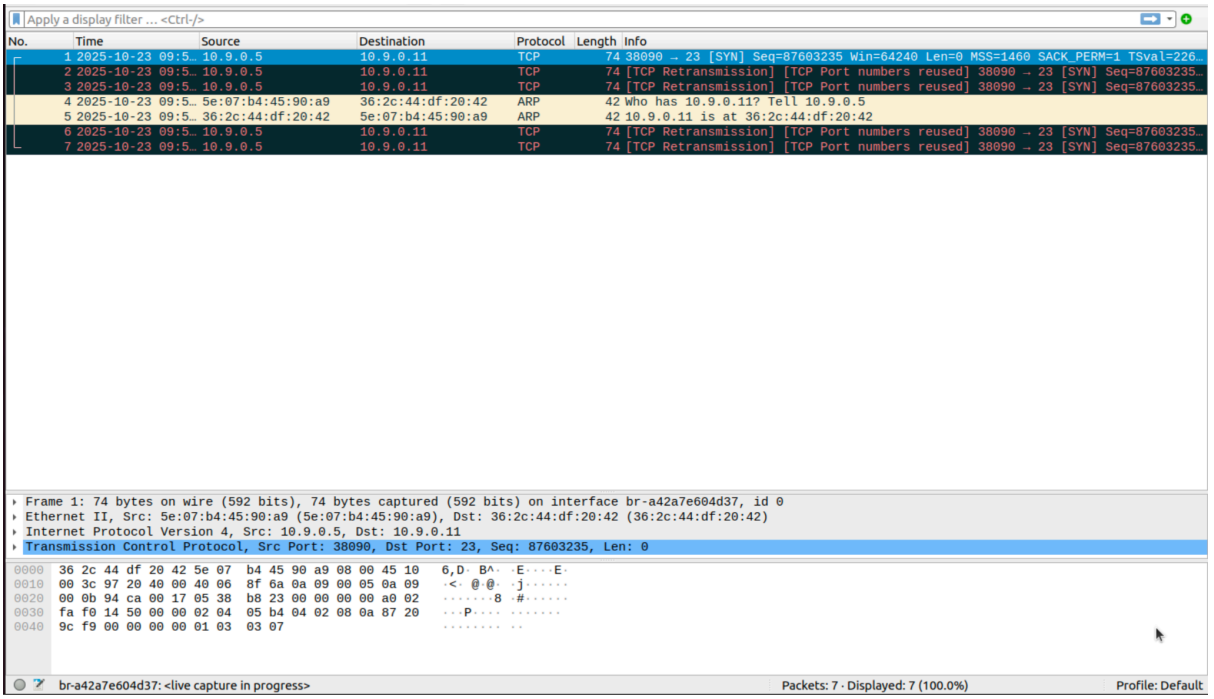
Wireshark:



telnet seed-router

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet seed-router
Trying 10.9.0.11...
^C
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

Wireshark:



Questions :

(1) Can you ping the router?

Answer: Yes. The ping succeeds with 0 percent packet loss, because ICMP echo-request and echo-reply packets were explicitly allowed in the firewall rules.

(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).

Answer: No. The telnet connection fails and eventually times out, because TCP packets (port 23) are not allowed by any rule and are therefore dropped by the default INPUT and OUTPUT DROP policies.

Step5: Cleanup

23)Command:

iptables -F

iptables -P OUTPUT ACCEPT

iptables -P INPUT ACCEPT

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$
```

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> (Here - $ docker restart dccca3a58e38)

Explanation:
In this task, we explored Linux's built-in firewall, iptables, to protect the router by enforcing stateless filtering rules. Initially, all firewall chains (INPUT, OUTPUT, FORWARD) had default ACCEPT policies, which allowed unrestricted traffic. We then modified the router's firewall behavior to adopt a default-deny stance by setting both INPUT and OUTPUT chains to DROP, while adding specific exceptions to allow ICMP ping traffic:

- Allow incoming ICMP echo-requests
- Allow outgoing ICMP echo-replies

After applying these rules, we tested connectivity from Host A. Ping requests to the seed-router were successful because the inserted ACCEPT rules specifically permitted ICMP request and reply packets. However, telnet attempts failed because TCP packets do not match any ACCEPT rule and were instead dropped by the default DROP policies. Packet captures confirmed that telnet traffic was silently discarded, while ping traffic continued to pass through correctly.

This experiment demonstrates how stateless firewall rules operate, how rule order and default policies influence traffic decisions, and how selective protocol-based restrictions can secure a networked device against unauthorized access.

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 FORW ARD chain for this purpose.

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.

Step1: Execute the following iptables commands on the seed-router container-24)Command:

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:

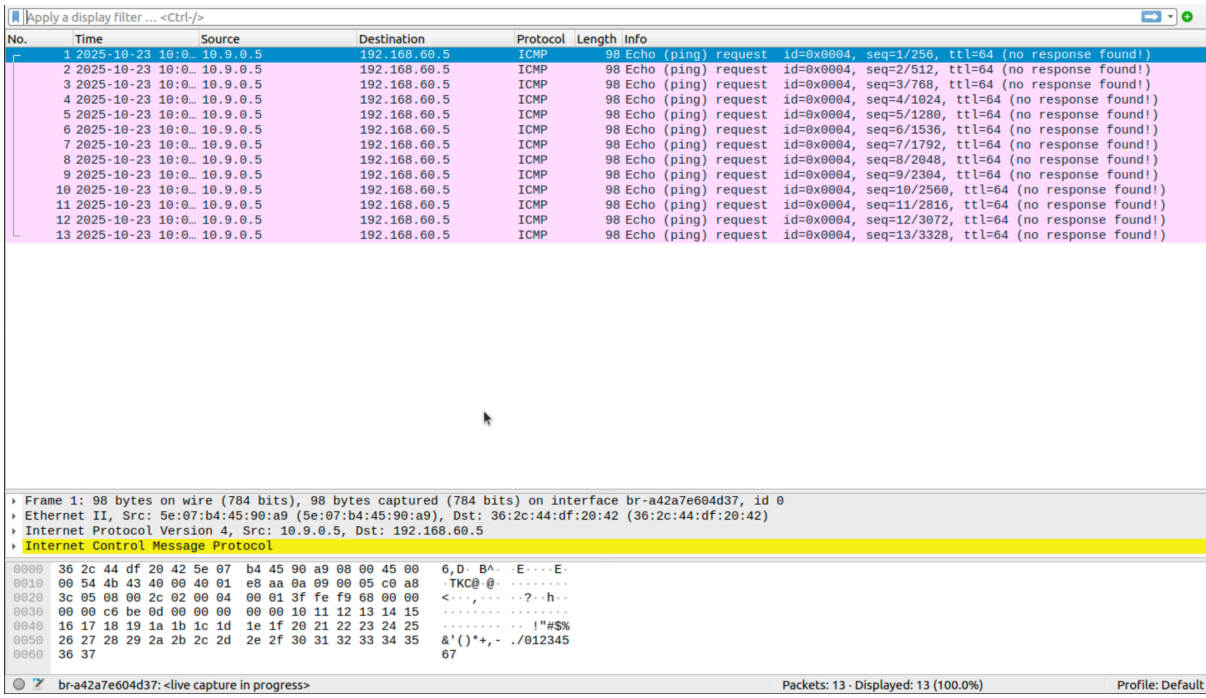
```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P FORWARD DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$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 DROP       icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0           icmptype 8
    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:PES1UG23CS433:PranavHemanth:/
$
```

Step2: Now we shall see if these restrictions have been enforced in the network.

1. Outside hosts cannot ping internal hosts.
    a. Command: ping 192.168.60.5 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
13 packets transmitted, 0 received, 100% packet loss, time 12265ms

hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

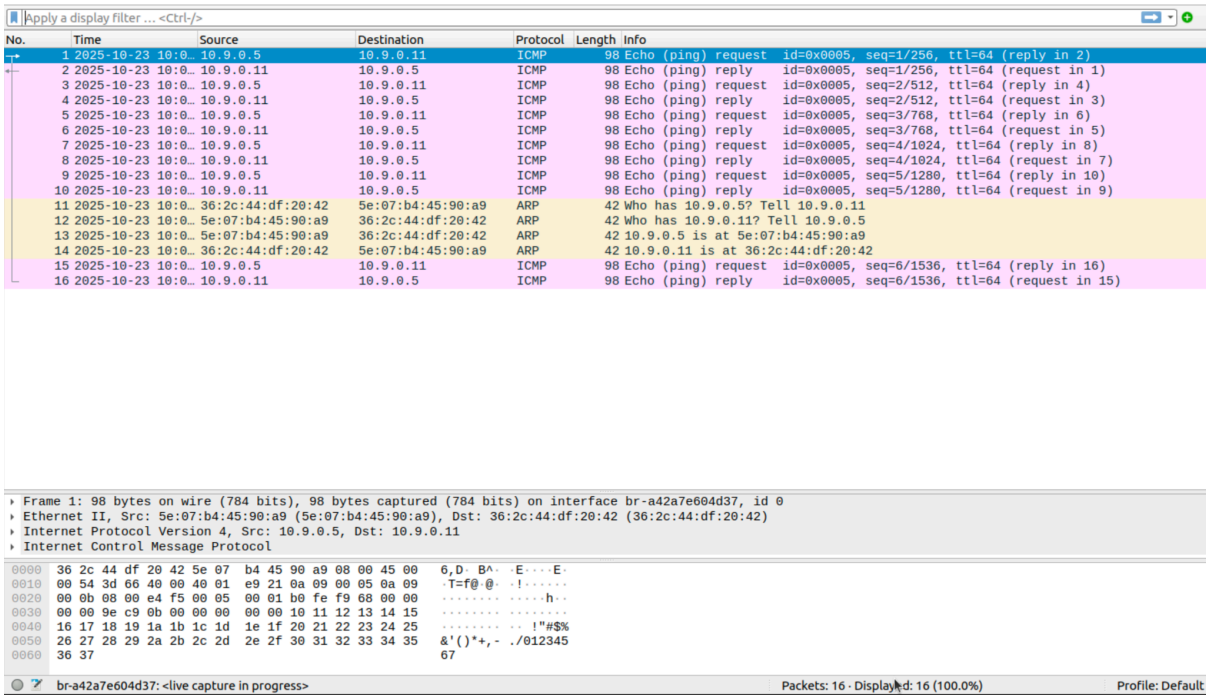
Wireshark:



2. Outside hosts can ping the router.
    a. Command: ping seed-router (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$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=0.43
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.15
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.14
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.12
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.14
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=6 ttl=64 time=0.09
 ms
^C
--- seed-router ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5090ms
rtt min/avg/max/mdev = 0.097/0.182/0.430/0.112 ms
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

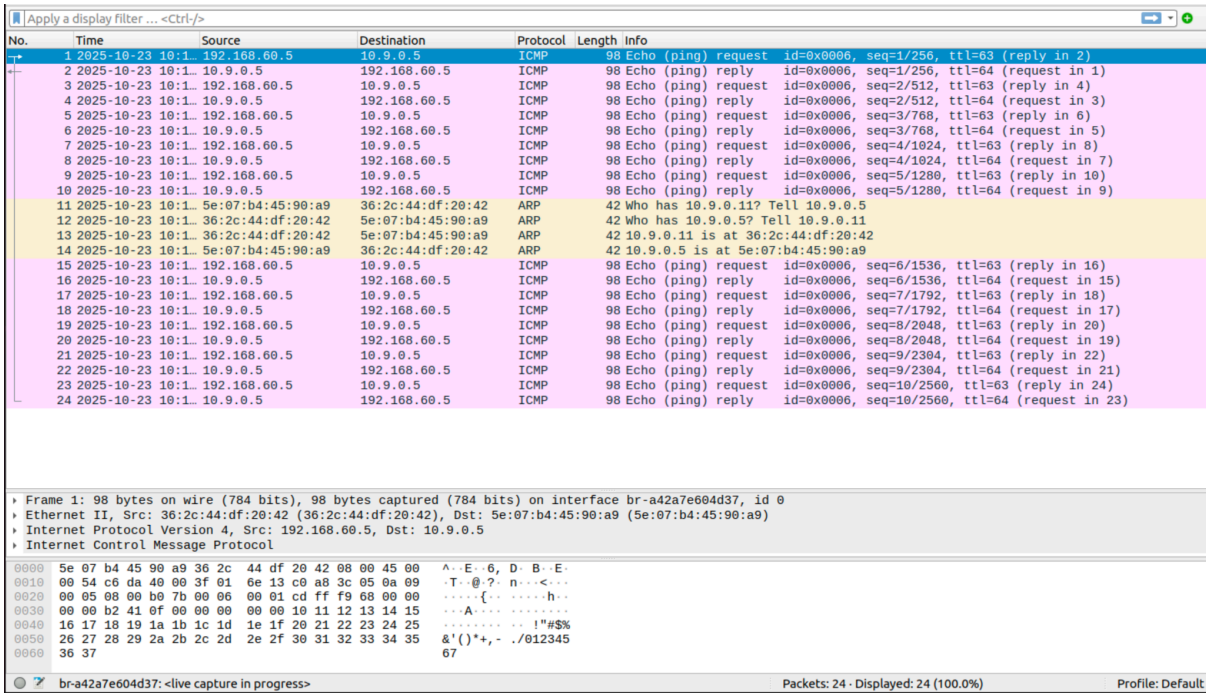
Wireshark:



3. Internal hosts can ping Outside Hosts.
    a. Command: ping 10.9.0.5 (On host1-192.168.60.5)

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth/
$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=1.27 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.211 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.163 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.176 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.229 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.140 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.116 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.166 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=63 time=0.178 ms
64 bytes from 10.9.0.5: icmp_seq=10 ttl=63 time=0.163 ms
^C
--- 10.9.0.5 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9175ms
rtt min/avg/max/mdev = 0.116/0.281/1.269/0.330 ms
host1-192.168.60.5:PES1UG23CS433:PranavHemanth/
$
```

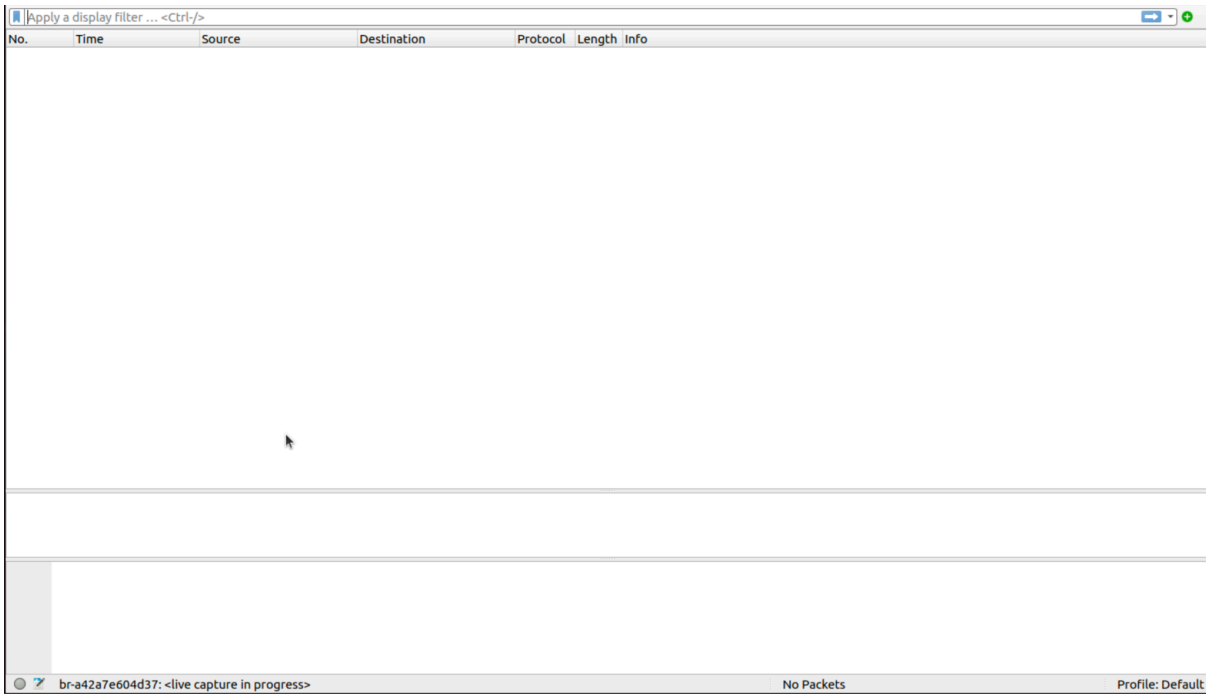
Wireshark:



4. All other packets between the internal and external networks should be blocked.
    a. Command: telnet 10.9.0.5 (On host1-192.168.60.5)

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth/
$telnet 10.9.0.5
Trying 10.9.0.5...
^C
host1-192.168.60.5:PES1UG23CS433:PranavHemanth/
$
```

Wireshark:



(No packets show as they all are dropped)

Explanation:
In this task, we configured iptables on the seed-router to protect the internal network (192.168.60.0/24) while still allowing necessary connectivity through the FORWARD chain. We enforced a default-deny policy, only allowing specific ICMP traffic to be forwarded between the internal and external networks.
Three key rules were added:

- Block ICMP echo-requests coming from the external interface (eth0) to prevent outside hosts from pinging internal machines.
- Allow ICMP echo-requests originating from the internal network (eth1) so internal hosts can ping external systems.
- Allow ICMP echo-replies returning from external hosts so internal ping responses can be received.

With these rules active, the FORWARD chain policy was set to DROP, ensuring all other forwarded traffic is blocked.

Testing verified the expected behavior:

- Outside hosts could ping the router itself, since that uses the INPUT chain.
- Outside hosts could not ping internal hosts (blocked by our DROP rule).
- Internal hosts could ping external hosts because both request and reply ICMP packets were allowed.
- Other forwarded traffic, such as telnet, was blocked by the default DROP policy.

This experiment demonstrates how iptables can selectively restrict forwarding traffic to protect an internal network while still maintaining limited and controlled communication with external hosts.

Step3: Cleanup

25)Command:

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

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

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> (Here - $ docker restart dccca3a58e38)

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.

Step1: Execute the following iptables commands on the seed-router container-
26)Command:

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:

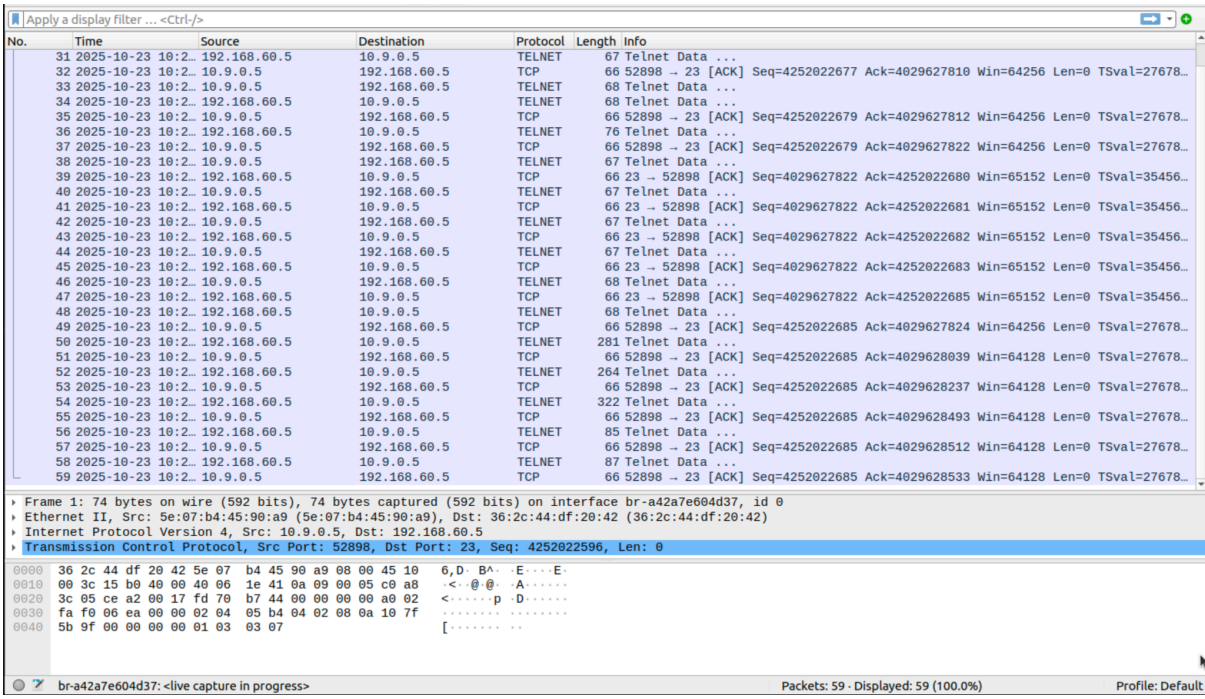
```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P FORWARD DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$
```

Step2: Now we shall see if these restrictions have been enforced in the network.

5. 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.
    a. Command: telnet 192.168.60.5 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
44e8f7e115a1 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-157-generic aarch64)

 * 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@44e8f7e115a1:~$
```

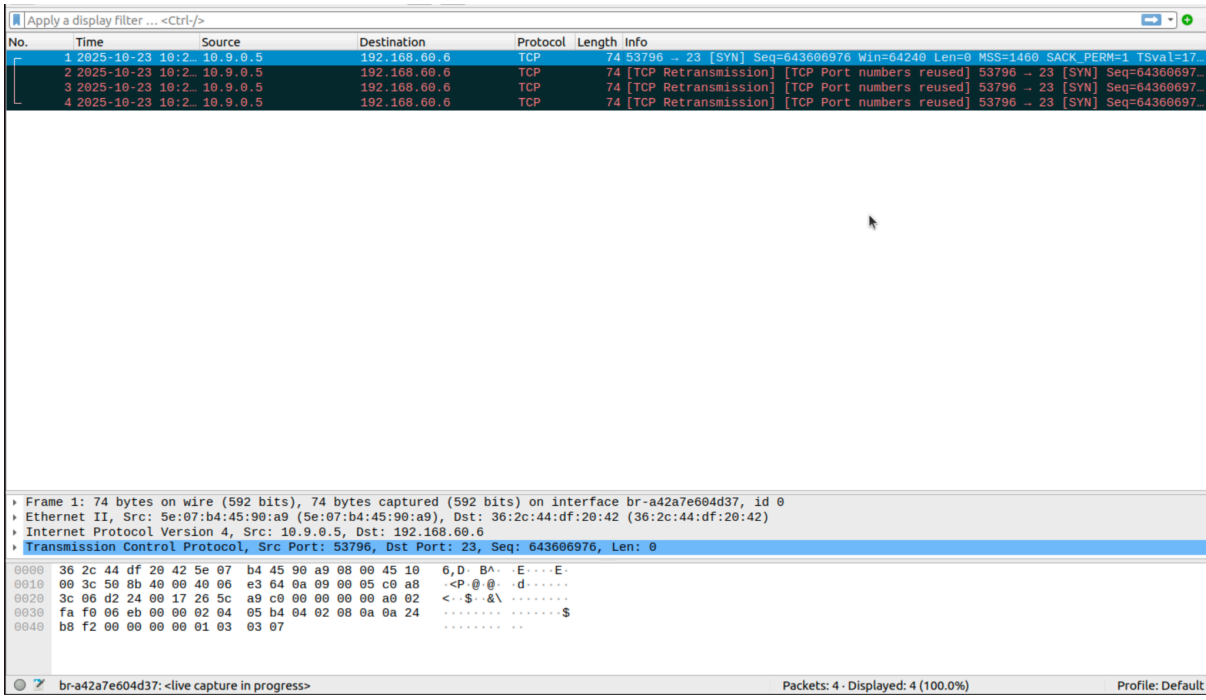
Wireshark:



6. Outside hosts cannot access other internal servers.
    a. Command: telnet 192.168.60.6 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.6
Trying 192.168.60.6...
^C
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

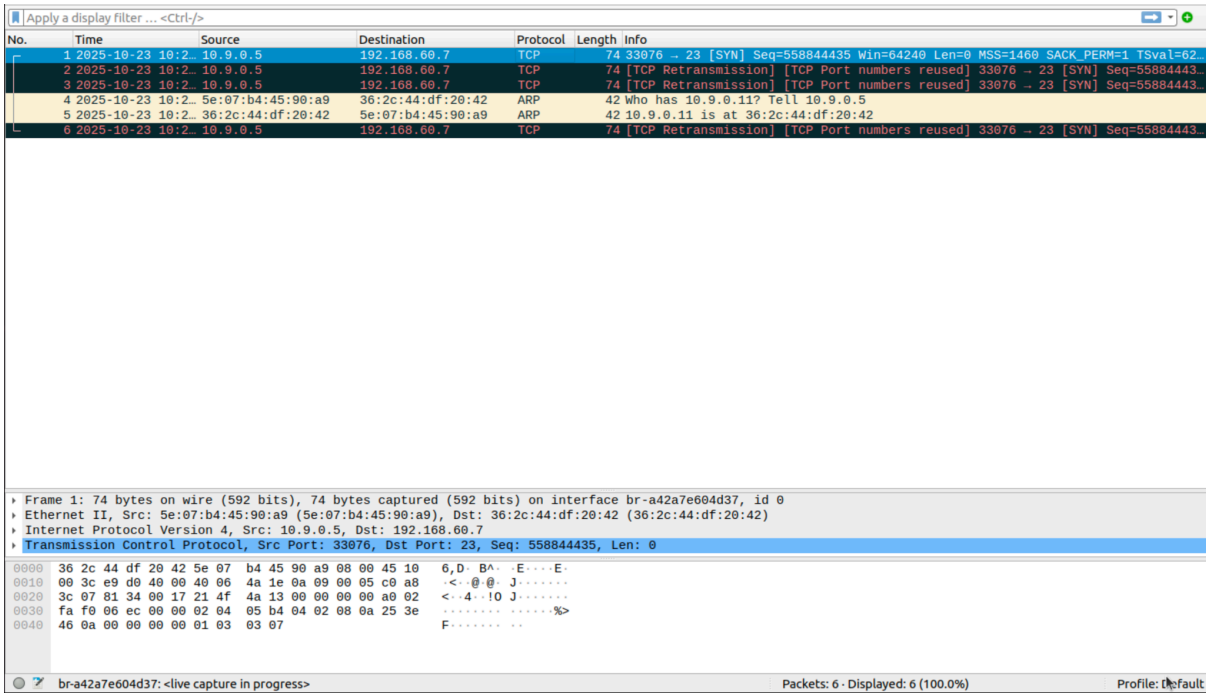
Wireshark:



b. Command: telnet 192.168.60.7 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.7
Trying 192.168.60.7...
^C
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

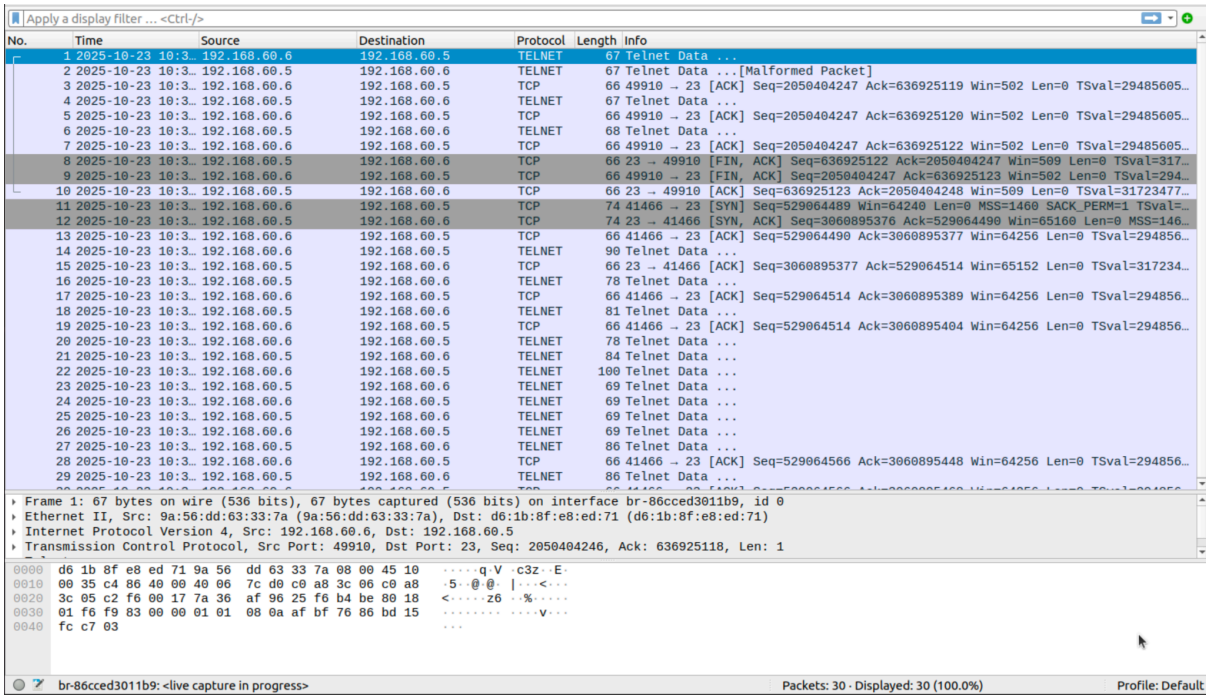
Wireshark:



7. Internal hosts can access all the internal servers.
    a. Command: telnet 192.168.60.5 (On host2 - 192.168.60.6)

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
44e8f7e115a1 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-157-generic aarch64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Thu Oct 23 10:25:21 UTC 2025 from 10.9.0.5 on pts/2
seed@44e8f7e115a1:~$
```

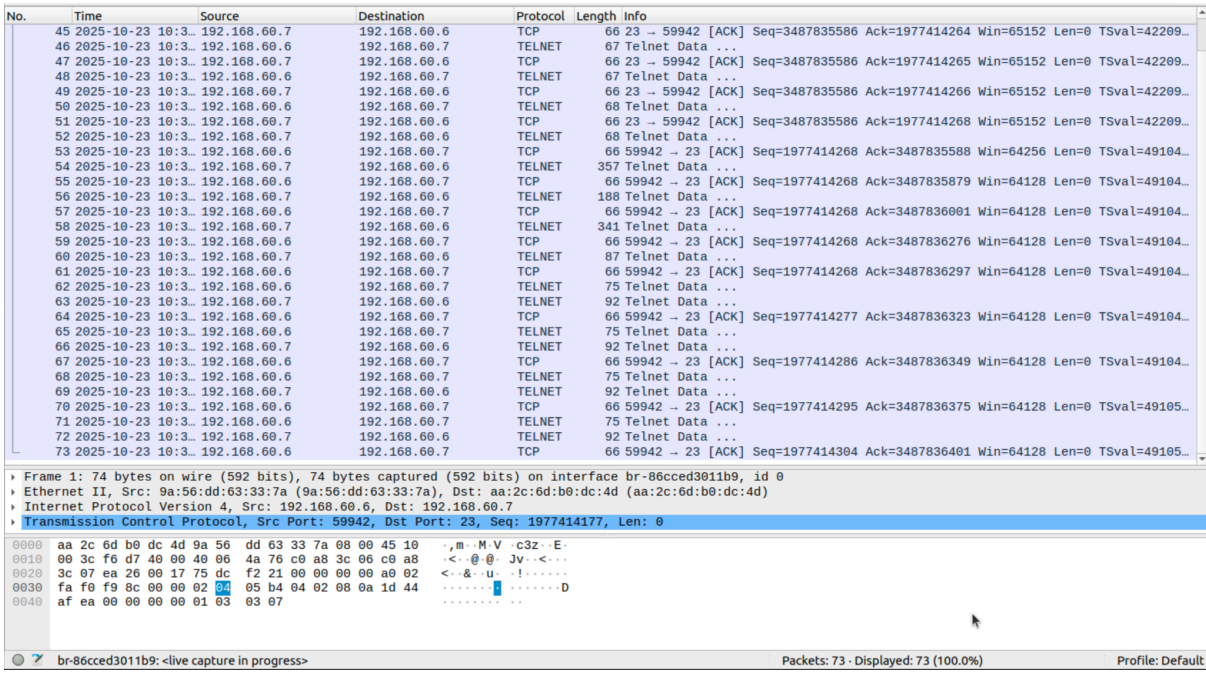
Wireshark:



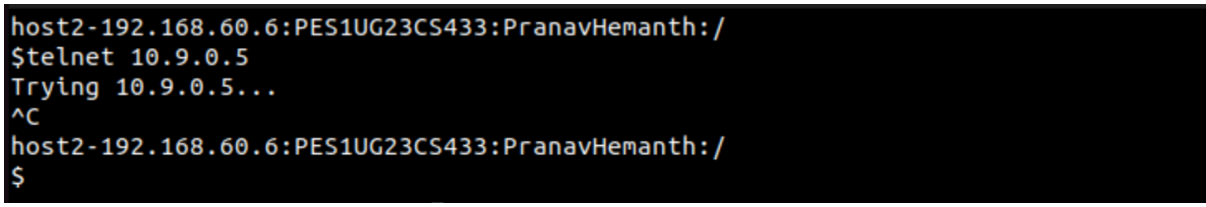
b. Command: telnet 192.168.60.7 (On host2 - 192.168.60.6)

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
344d644c5360 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-157-generic aarch64)

 * 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@344d644c5360:~$
```

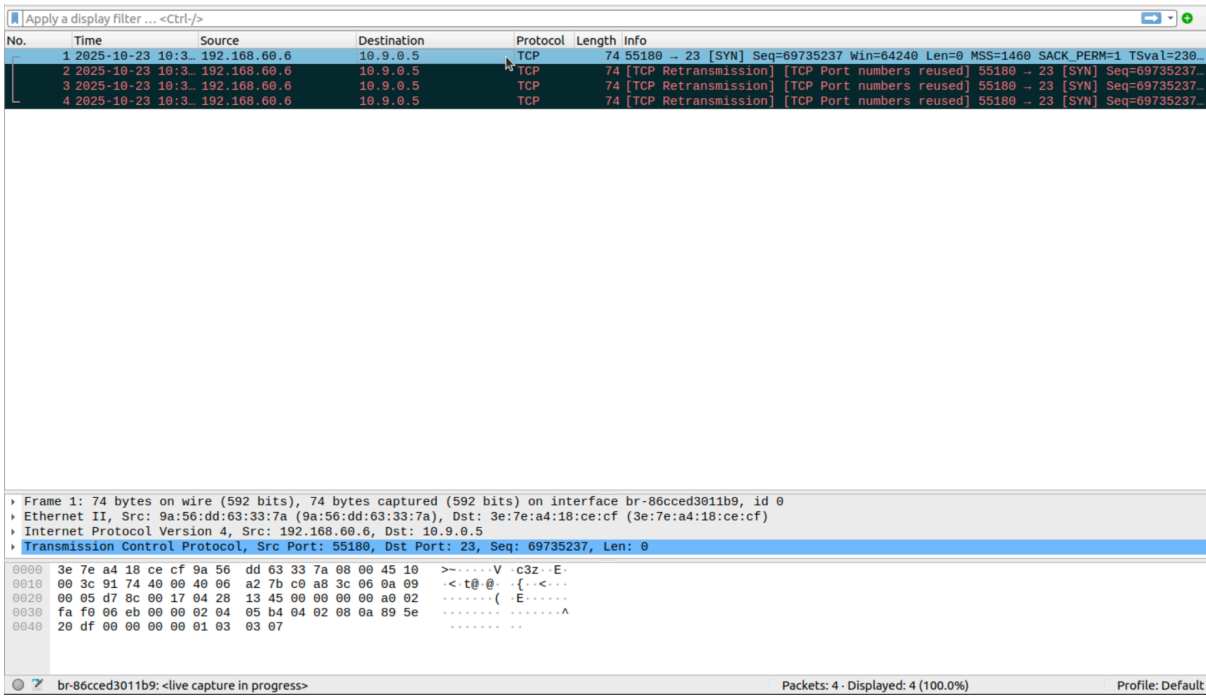
Wireshark:



8. Internal hosts cannot access external servers.
   a. Command: telnet 10.9.0.5 (On host2 - 192.168.60.6)



Wireshark:



Explanation:
In this task, we configured the router's firewall using iptables to control access to internal TCP servers. The internal network (192.168.60.0/24) has multiple hosts running telnet servers, but only the server at 192.168.60.5 should be accessible from the external network. We added two specific ACCEPT rules to the FORWARD chain that allow only telnet traffic to and from 192.168.60.5, then set the default FORWARD policy to DROP so that all other forwarded traffic is blocked.

Testing confirmed the correct behavior:

- External hosts could successfully telnet into 192.168.60.5.
- External hosts could not access telnet servers on 192.168.60.6 or 192.168.60.7.
- Internal hosts could telnet to any internal server. These packets do not traverse the router and therefore bypass the FORWARD chain entirely.
- Internal hosts were blocked from telnetting to external systems, since those packets are forwarded through the router and hit the default DROP policy.

This demonstrates precise segmentation and protection of internal services. Access is granted only where explicitly permitted, while all other internal-external communications are restricted to prevent unauthorized entry or data exposure.

Step3: Cleanup

27) Command:
    iptables -F
    iptables -P OUTPUT ACCEPT
    iptables -P INPUT ACCEPT

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

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> (Here - $ docker restart dccca3a58e38)

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.

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

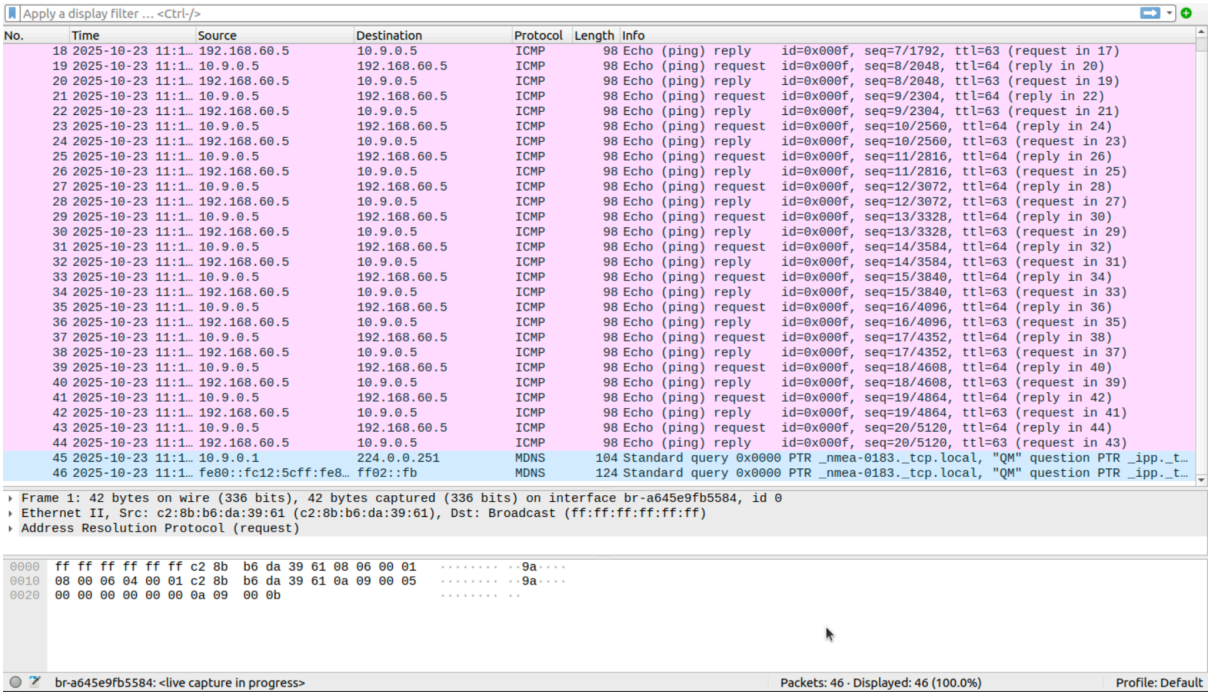
The goal of the task is to use a series of experiments to help students understand the connection concept in this tracking mechanism, especially for the ICMP and UDP protocols, because unlike TCP , they do not have connections.

Step1: ICMP Experiment

28)Command: ping 192.168.60.5 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$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=0.831 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.160 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.153 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.143 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.233 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.138 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.139 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.150 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.154 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.154 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.151 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.144 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.155 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.166 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.081 ms
^C
--- 192.168.60.5 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14325ms
rtt min/avg/max/mdev = 0.081/0.196/0.831/0.171 ms
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

Wireshark:



29)Command: conntrack -L(On seed-router)

```
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 27 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 24 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 21 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 20 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 20 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
icmp     1 17 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=16 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=16 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

Questions:

How long can the ICMP connection state be kept?

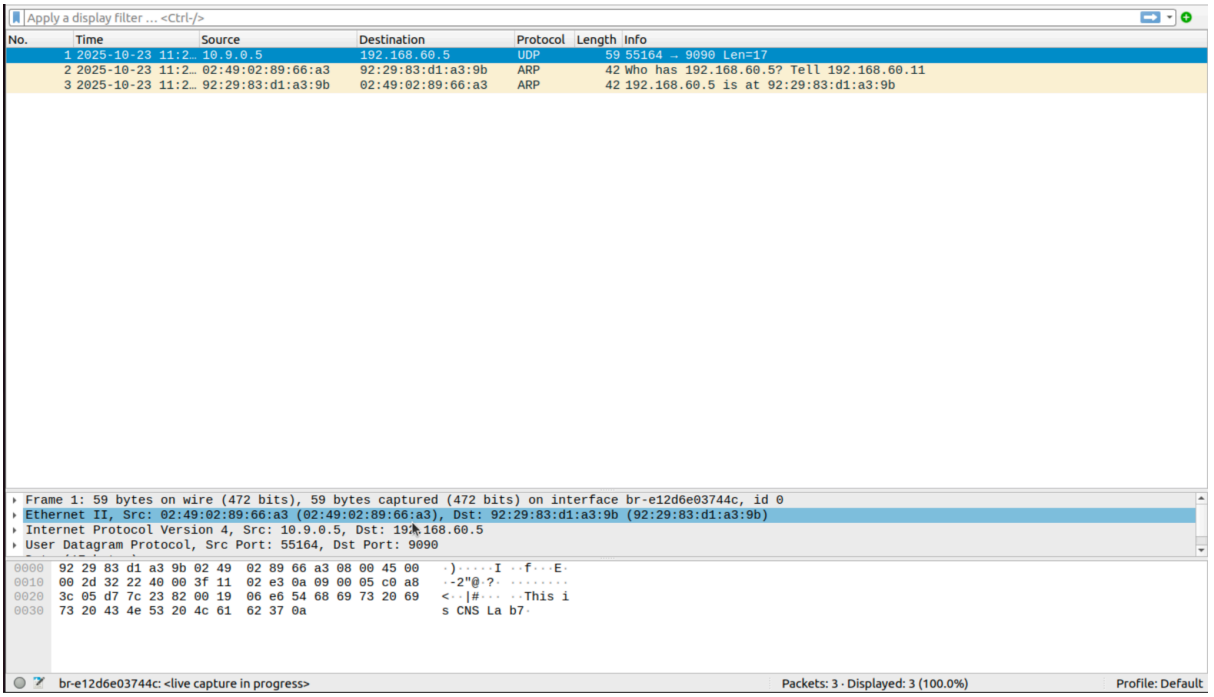
Answer: ICMP entries remain in the conntrack table for **around 30 seconds** after the ping stops. The timeout counts down until it reaches zero, then the flow is removed.

Step2: UDP Experiment

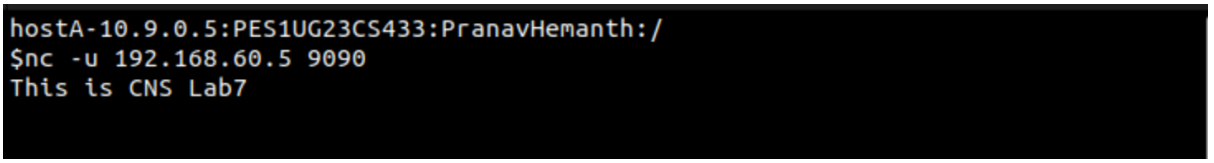
30)Command: nc -lu 9090 (On host1 - 192.168.60.5)

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$nc -lu 9090
This is CNS Lab7
```

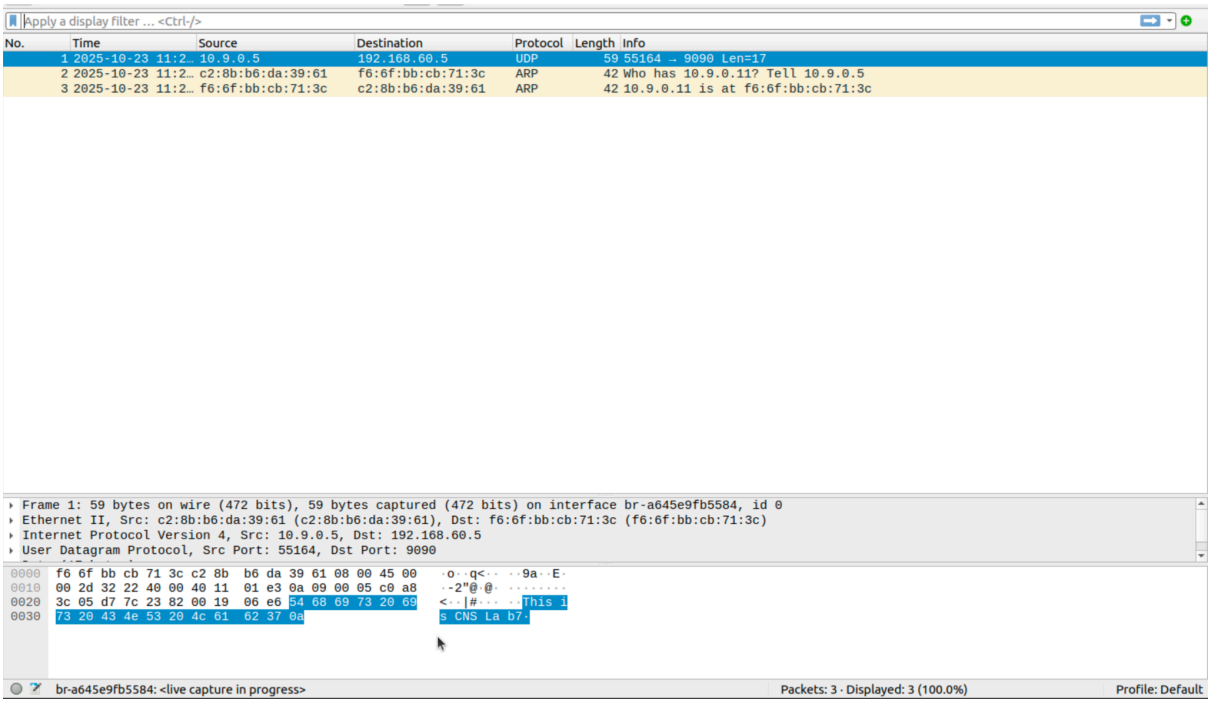
Wireshark:



31)Command: nc -u 192.168.60.5 9090 (On Host A - 10.9.0.5)



Wireshark:



32)Command: conntrack -L(On seed-router)

```
$conntrack -L
udp      17 15 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 15 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 14 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 14 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 13 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 7 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 6 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 6 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 3 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 2 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
udp      17 0 src=10.9.0.5 dst=192.168.60.5 sport=50139 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50139 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

Questions:

How long can the UDP connection state be kept?

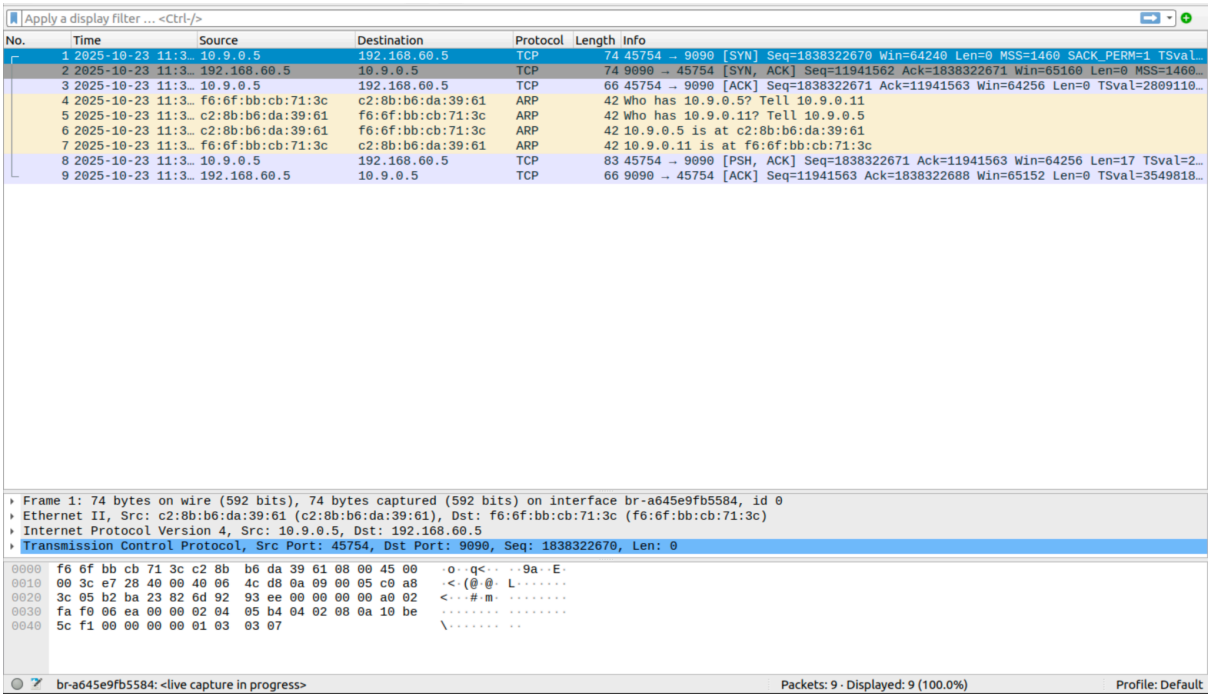
Answer: UDP states are kept for a shorter period, roughly 10–30 seconds, and entries are marked UNREPLIED when there is no return packet. They disappear quickly due to the lack of session control or teardown.

Step3: TCP Experiment

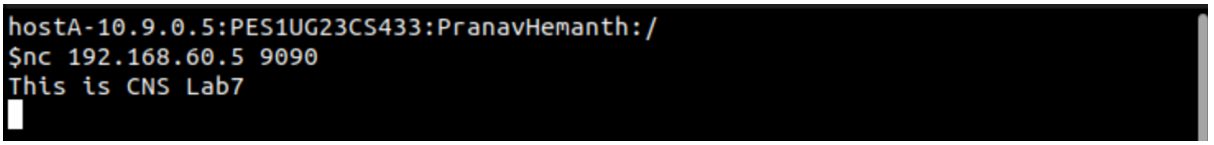
33) Command: nc -l 9090 (On host1 - 192.168.60.5)

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$nc -l 9090
This is CNS Lab7
```

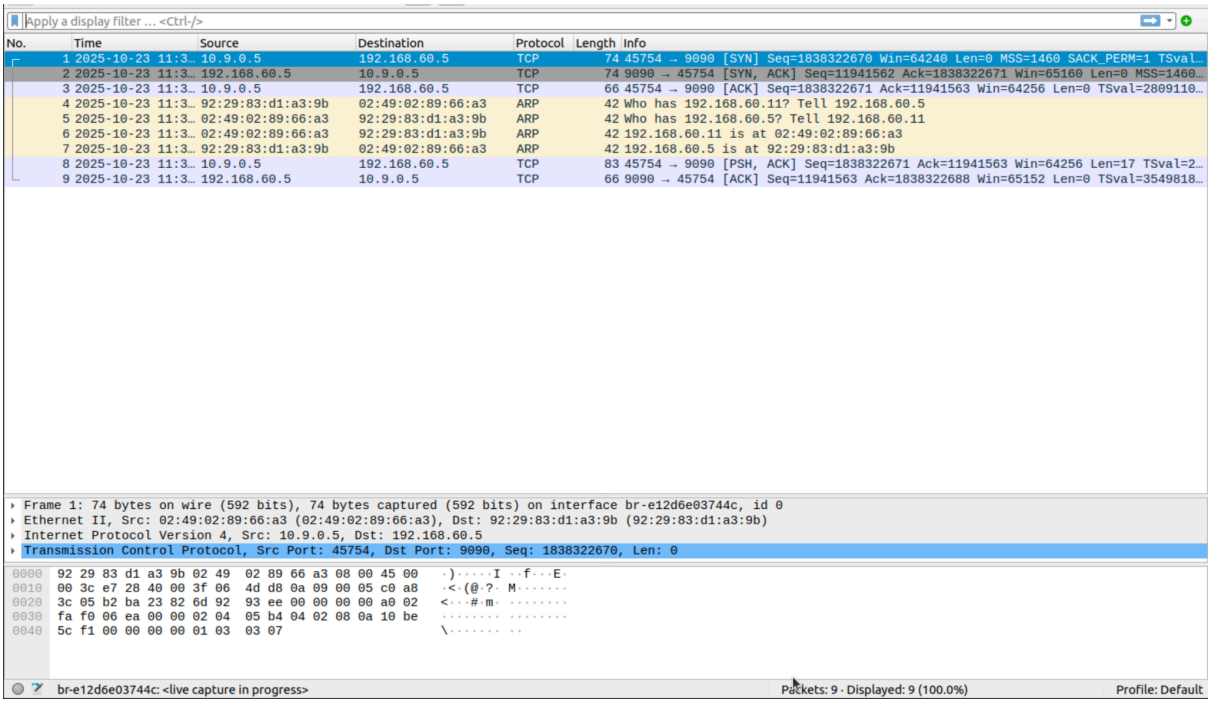
Wireshark:



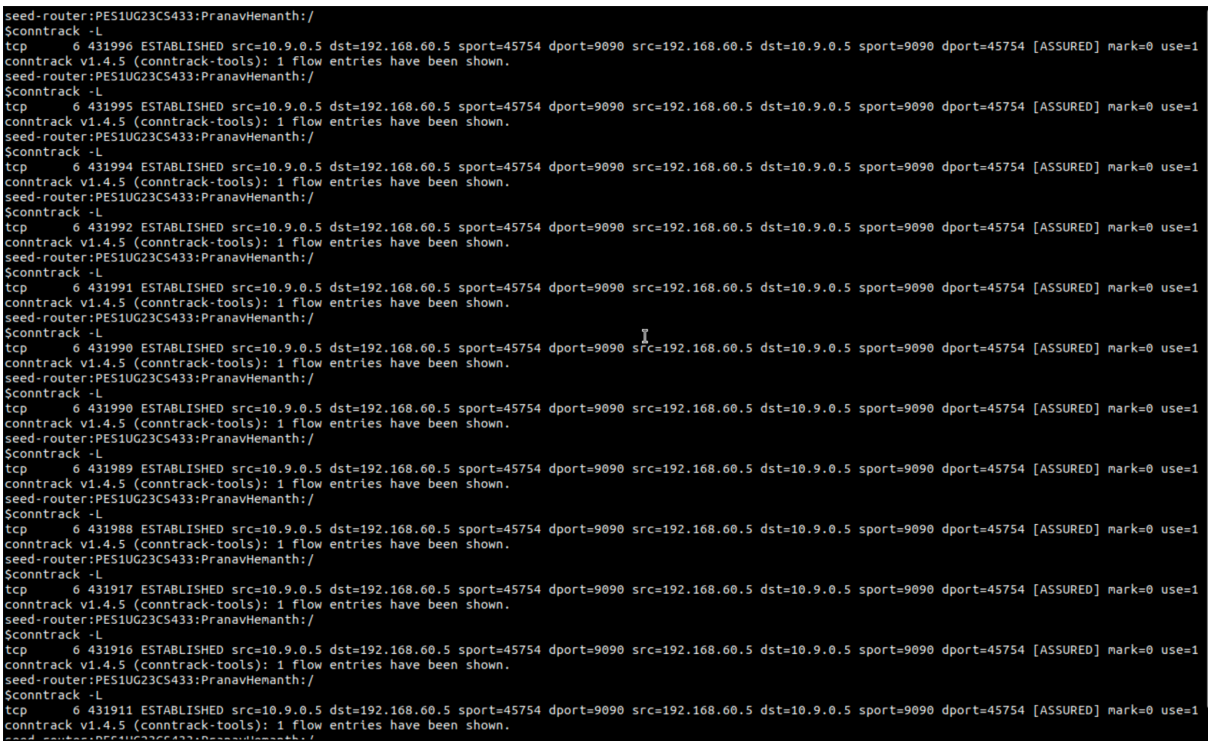
34)Command: nc 192.168.60.5 9090 (On Host A - 10.9.0.5)



Wireshark:



35)Command: conntrack -L(On seed-router)



Questions:

1. **How long can the TCP connection state be kept?**
   Answer: TCP ESTABLISHED flows show a large timeout value of hundreds of thousands of seconds, typically about 5 days of idle timeout, because TCP has reliable session state and teardown behavior.

2. Close the TCP connection and execute conntrack -L on the router container. Do you spot any difference?
   Answer: Yes. Once the connection is closed:
   - The state changes from ESTABLISHED to TIME_WAIT or CLOSE,
   - Then the entry remains only briefly before being removed from the table.
     This demonstrates proper tracking of TCP session lifecycle.

Explanation:

This task introduced connection tracking, which is essential for building stateful firewalls. Unlike stateless filtering, which treats every packet independently, conntrack keeps track of the state of communication between hosts. Even for protocols like ICMP and UDP that do not have a formal “connection,” conntrack still groups packets into flows and assigns temporary state.

Using the command conntrack -L on the seed-router, we inspected how the kernel tracks ICMP, UDP, and TCP traffic as we generated it from different hosts.

We observed:

- ICMP (ping) creates temporary tracked entries for echo-request and echo-reply packets, which expire shortly after traffic stops.
- UDP packets are also tracked briefly even if there is no reply. These entries disappear quickly since UDP lacks connection teardown.
- TCP has a well-defined state machine, so conntrack recognizes SYN, handshake completion, data transfer, and teardown. It keeps ESTABLISHED connections for a long timeout.

This helped show why stateful firewalls can allow return traffic without requiring explicit reverse-direction rules. The firewall can track and match packets as part of an already-approved flow, simplifying rule creation and improving security.

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 ruleallowing internal hosts to visit any external server (this was not allowed in Task 2.C).

Step1: Execute the following iptables commands on the seed-router container-36)Command:

```
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:

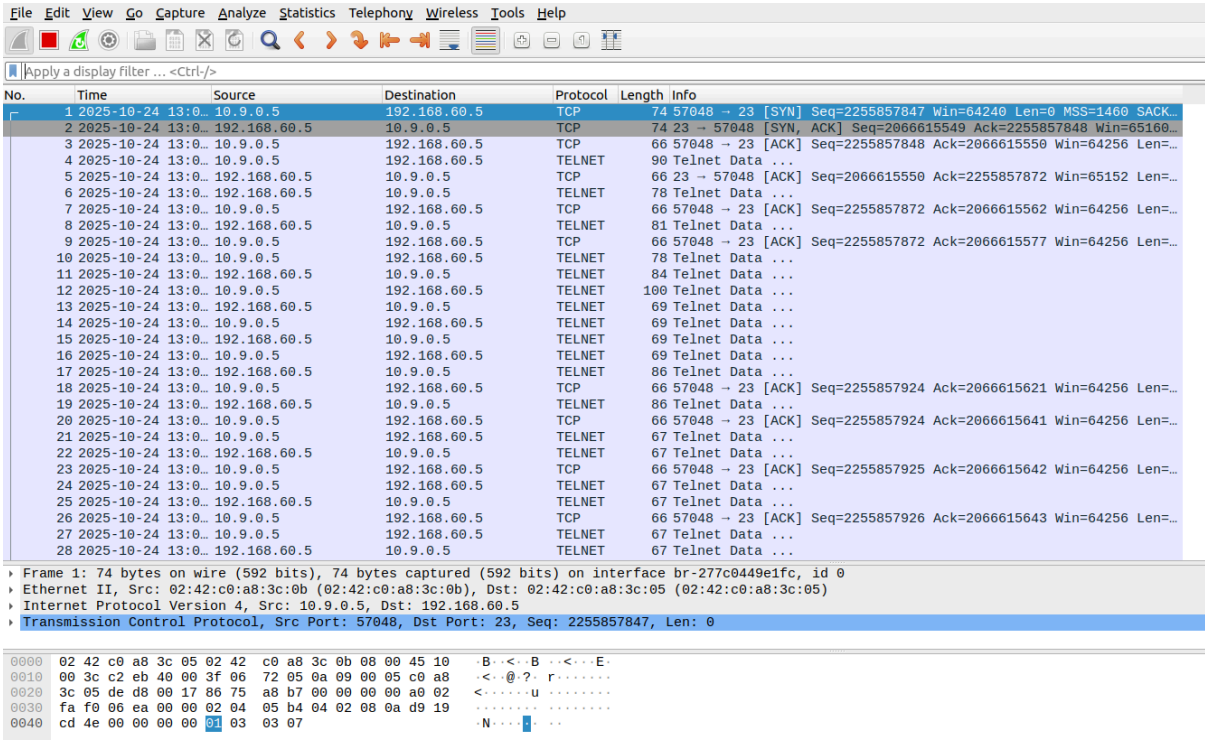
```
seed-router:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -p tcp -j DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$>iptables -P FORWARD ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$>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:PES1UG23CS433:PranavHemanth:/
$>
```

Step2: Now we shall see if these restrictions have been enforced in the network.

9. 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.
    a. Command: telnet 192.168.60.5 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c04c5edada49 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-139-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@c04c5edada49:~$
```

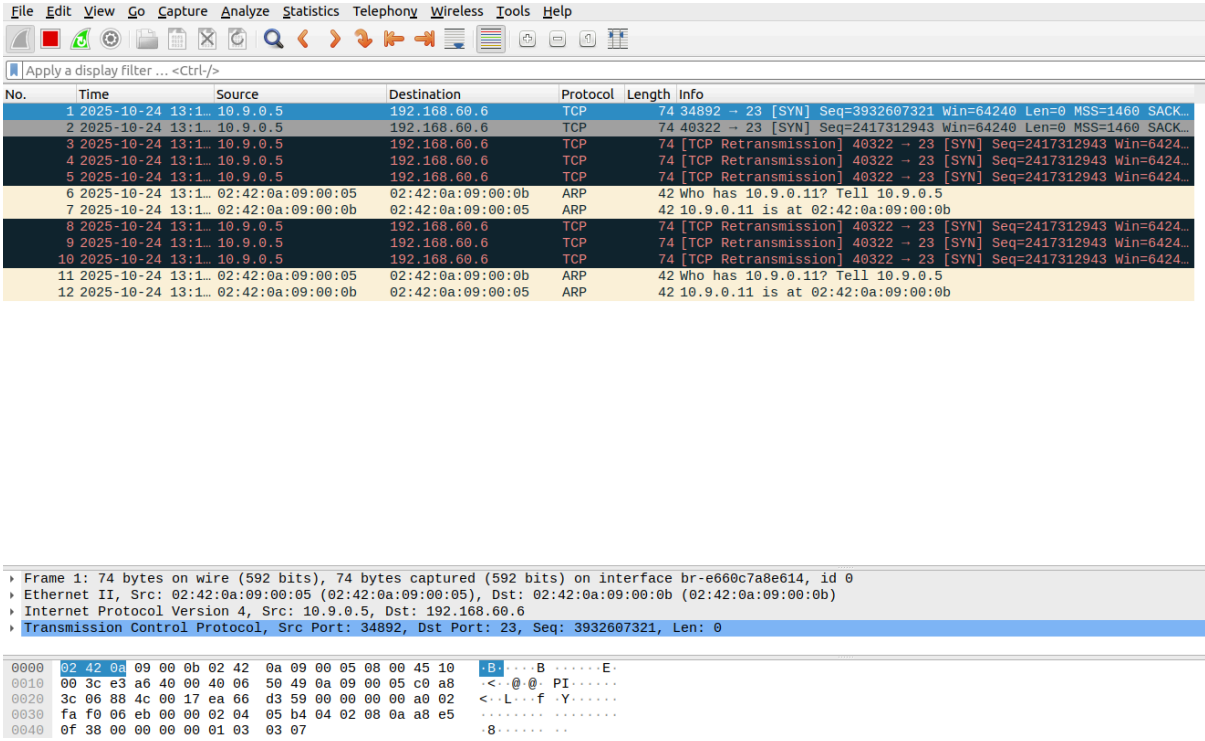
Wireshark:



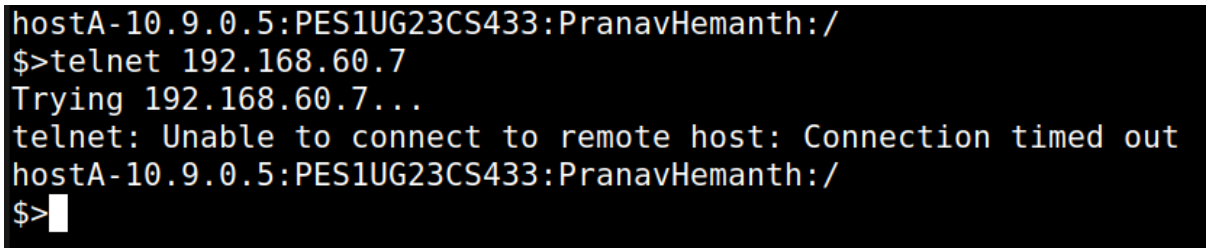
10. Outside hosts cannot access other internal servers.
    a. Command: telnet 192.168.60.6 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>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:PES1UG23CS433:PranavHemanth:/
$>
```

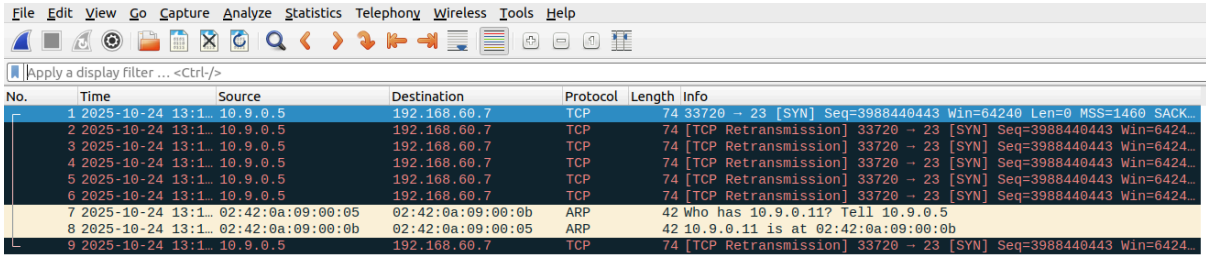
Wireshark:

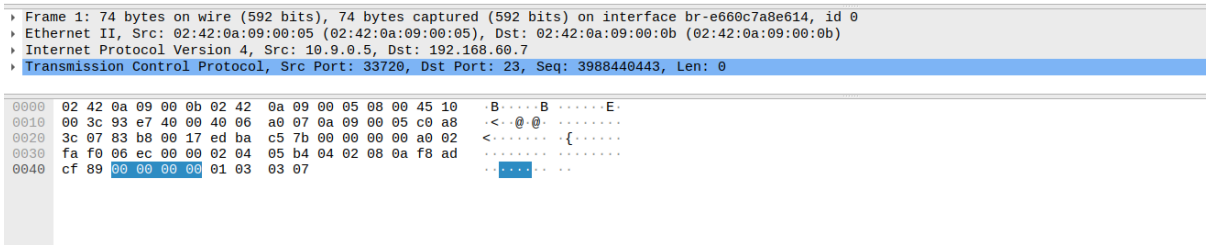


b. Command: telnet 192.168.60.7 (On Host A - 10.9.0.5)



Wireshark:

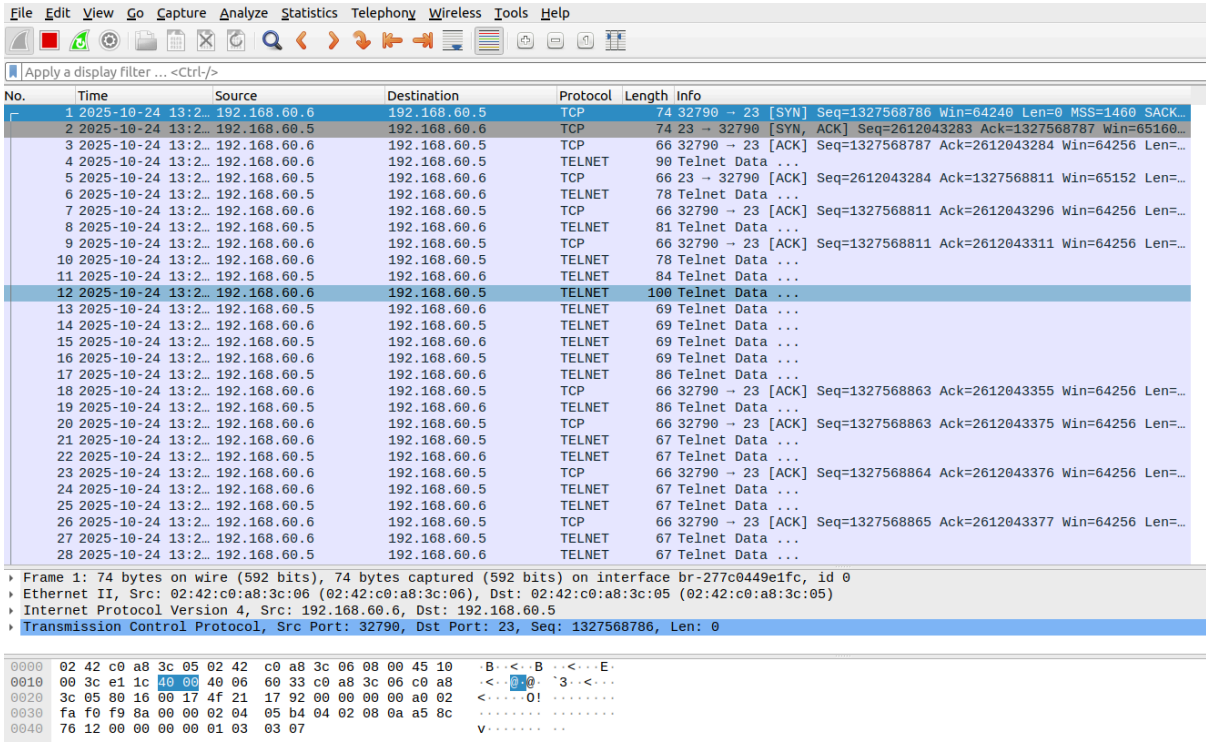




11. Internal hosts can access all the internal servers.
    a. Command: telnet 192.168.60.5 (On host2 - 192.168.60.6)

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c04c5edada49 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-139-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Fri Oct 24 17:20:17 UTC 2025 from www.SeedLabSQLInjection.com on pts/2
seed@c04c5edada49:~$
```

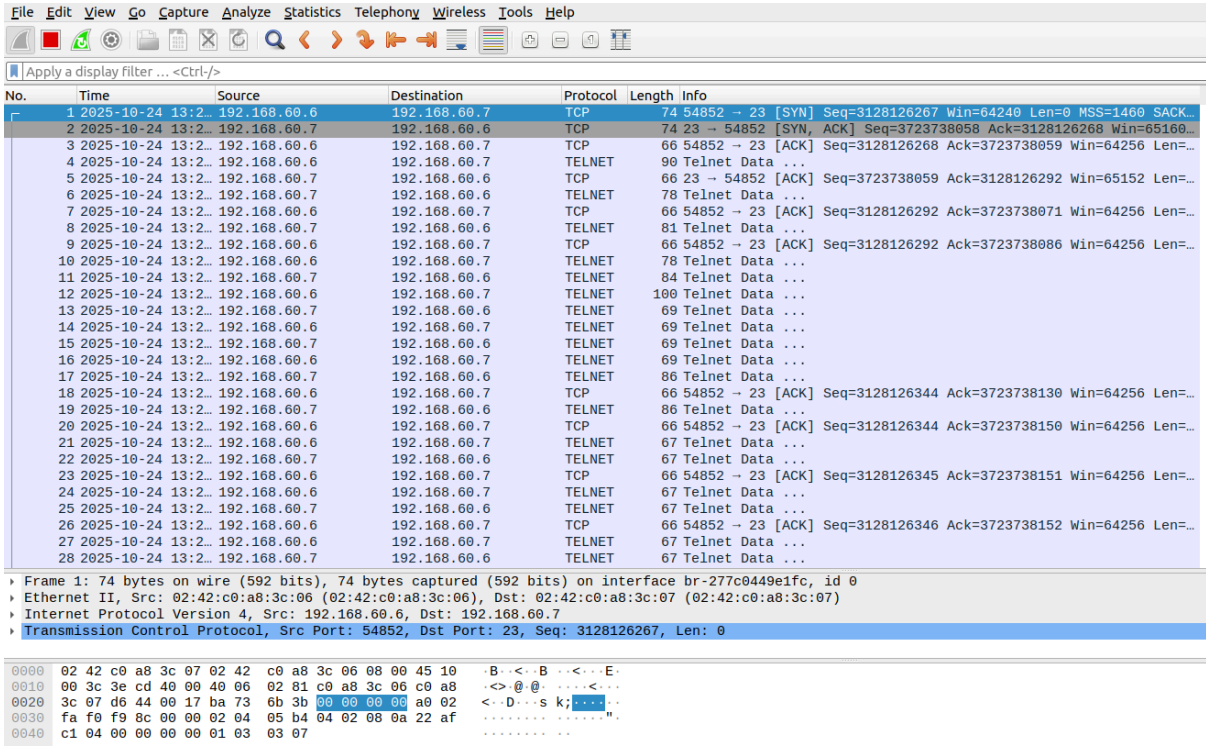
Wireshark:



b. Command: telnet 192.168.60.7 (On host2 - 192.168.60.6)

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$>telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
d6239420659d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-139-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@d6239420659d:~$
```

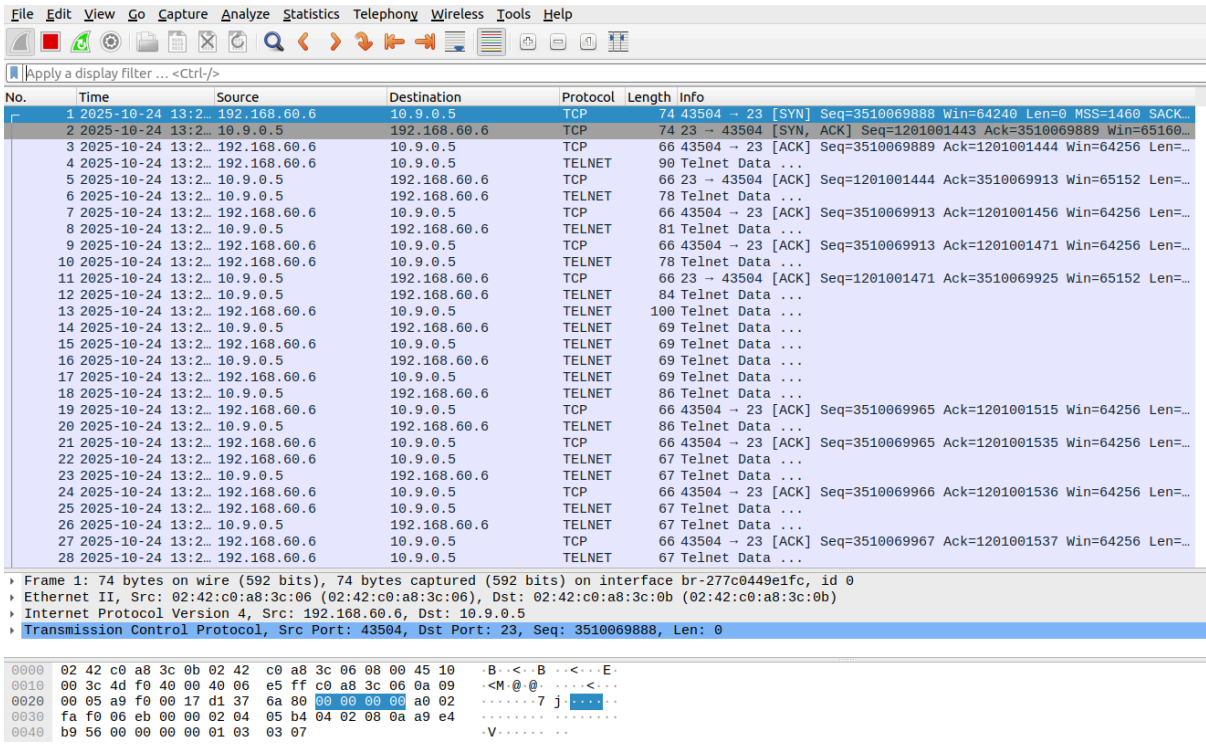
Wireshark:



12. Internal hosts can access external servers.
    a. Command: telnet 10.9.0.5 (On host2 - 192.168.60.6)

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
972e43bfc613 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-139-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@972e43bfc613:~$
```

Wireshark:



Explanation:
In this task, we switched from a stateless firewall to a stateful one using the kernel's connection tracking module (conntrack). Stateful firewalls understand the context of network traffic. Instead of inspecting each packet independently, they track active connections and determine whether a packet belongs to a valid, ongoing session.

We rewrote the firewall rules so that:

1. External hosts are only allowed to start Telnet connections to one internal server: 192.168.60.5:23
2. Internal hosts can initiate any external TCP connection
3. Return traffic for accepted connections is automatically allowed due to ESTABLISHED,RELATED handling
4. All other TCP traffic is dropped

Once the rules were applied, we tested different connection scenarios:

- From Host A (external), Telnet to 192.168.60.5 succeeded because it matched the NEW allowed rule.
- External Telnet to 192.168.60.6 or .7 failed since no rule permits NEW connections to them.
- Communication between internal hosts always worked because it never touches the router's FORWARD chain.
- Internal hosts could now access external servers, unlike in Task 2.C, because NEW outbound connections are explicitly allowed.

Overall, connection tracking made our firewall smarter, simpler, and more secure by allowing replies for approved connections while blocking all unauthorized attempts.

The firewall rules for the stateful setup explained individually are as follows:

1. Allow NEW TCP SYN from the Internet to 192.168.60.5:23
    - This rule ensures that only the intended internal server is reachable from outside, restricting external access to other internal hosts.
2. Allow NEW TCP SYN from the internal LAN to any destination
    - This rule enables internal hosts to initiate connections to external servers, supporting normal outbound traffic.
3. Allow RELATED or ESTABLISHED connections
    - This rule permits reply traffic for approved connections. It forms the core of the stateful firewall, automatically allowing responses without additional rules, simplifying management and improving security.
4. Drop all remaining TCP traffic
    - This rule blocks unauthorized connection attempts and acts as a security cleanup measure, ensuring that any traffic not explicitly allowed is discarded.

Step3: Cleanup

37) Command:

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

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

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> (Here - $ docker restart dccca3a58e38)

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.

Conduct the experiment with and without the second rule, and then explain whether the second rule is needed or not, and why.

Step1: On seed router execute (With 2nd rule)

38)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:

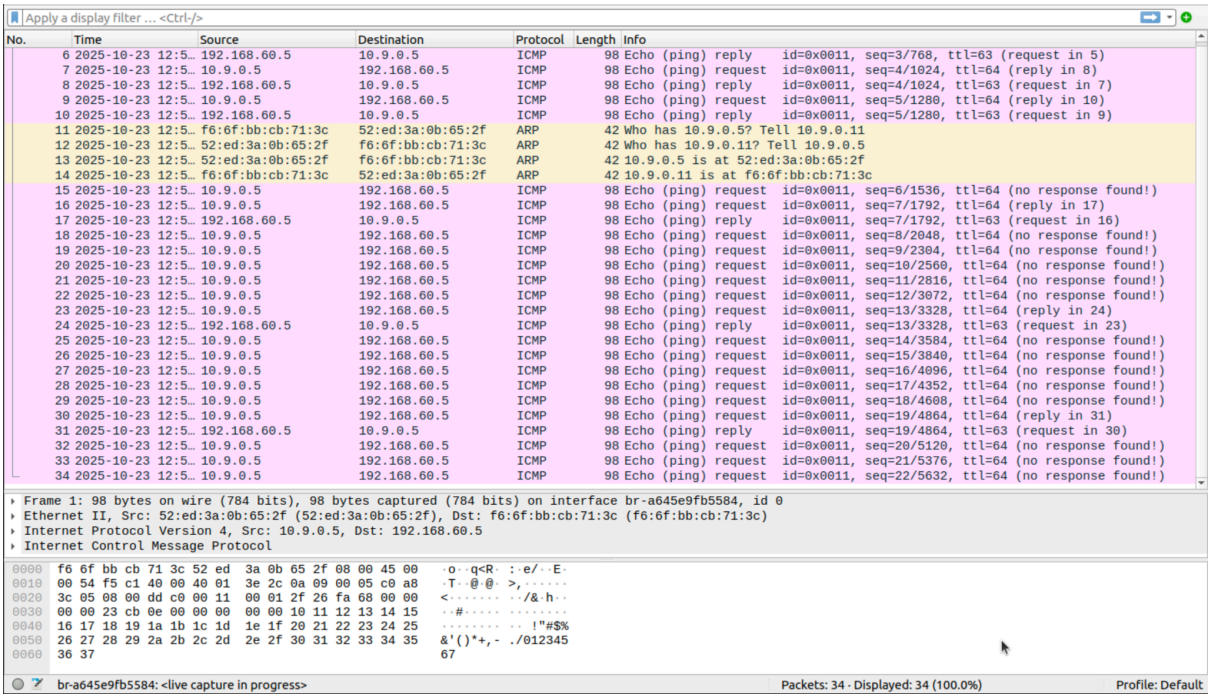
```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -s 10.9.0.5 -j DROP
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -L -n -v
Chain INPUT (policy ACCEPT 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:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5            0.0.0.0/0           limit: avg 10/min burst 5
DROP       all  --  10.9.0.5            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

Step2: Now we shall see if these restrictions have been enforced in the network.

39)Command: ping 192.168.60.5 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$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=2.25 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.263 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.224 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.192 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.228 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.141 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.231 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.186 ms
^C
--- 192.168.60.5 ping statistics ---
22 packets transmitted, 8 received, 63.6364% packet loss, time 21489ms
rtt min/avg/max/mdev = 0.141/0.464/2.250/0.675 ms
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

Wireshark:



Step3: Cleanup

40)Command:

iptables -F

iptables -P OUTPUT ACCEPT

iptables -P INPUT ACCEPT

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

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> (Here - $ docker restart dccca3a58e38)

Explanation: Experiment 1:- With the Second DROP Rule

In this experiment, the firewall on the router was configured with two FORWARD rules for traffic from 10.9.0.5. The first rule uses the limit module to accept up to 10 packets per minute with a burst of 5, and the second rule explicitly drops any remaining packets from the same source that exceed this limit.

Testing this setup by pinging 192.168.60.5 from Host A shows that the connection experiences significant packet loss (about 70%), which demonstrates that the firewall is correctly throttling traffic. The initial burst of packets is allowed through, followed by a slow trickle at the defined rate, while excess packets are discarded by the DROP rule. Without this second rule, the packets exceeding the limit would fall through to the chain's default policy and be accepted, rendering the rate-limiting ineffective.

Purpose of the rules:

1. iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT — defines the allowance, accepting packets that are within the specified rate.
2. iptables -A FORWARD -s 10.9.0.5 -j DROP — enforces the limit by dropping packets that exceed the allowance.

Step4: On seed router execute (Without 2nd rule)

41) 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:

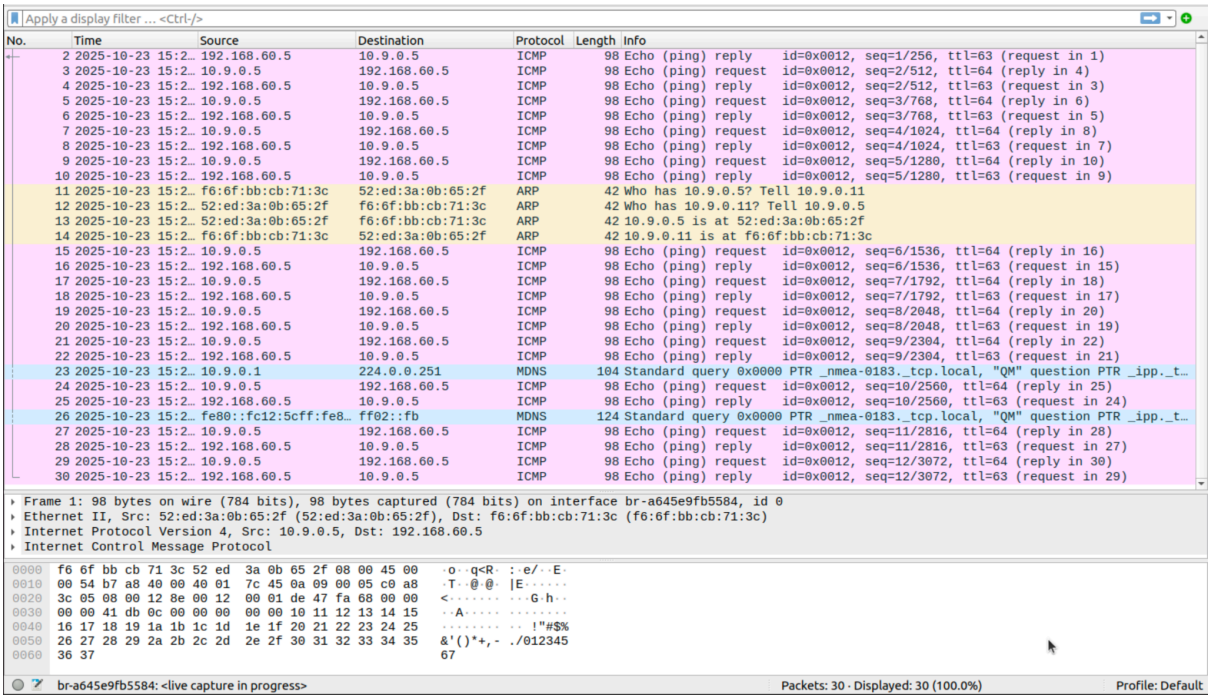
```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j A
CCEPT
$iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


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

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

seed-router:PES1UG23CS433:PranavHemanth:/
$
```

Step5: Now we shall see if these restrictions have been enforced in the network.

42) Command: ping 192.168.60.5 (On Host A - 10.9.0.5)

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$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=0.557 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.165 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.226 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.184 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.209 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.196 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.183 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.178 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.183 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.111 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.166 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.179 ms
^C
--- 192.168.60.5 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11252ms
rtt min/avg/max/mdev = 0.111/0.211/0.557/0.107 ms
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

43) Wireshark:



Step6: Cleanup

44)Command:

iptables -F

iptables -P OUTPUT ACCEPT

iptables -P INPUT ACCEPT

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

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> (Here - $ docker restart dccca3a58e38)

Explanation: Experiment 2:- Without the Second DROP Rule

Here, only the first rule with the limit module was added, and the chain's default policy remained ACCEPT. When pinging 192.168.60.5 from Host A, the connection is fully successful with 0% packet loss, showing that the rate limit is ineffective. The excess packets that would have been blocked by the second rule are instead allowed to pass through.

Purpose of the single rule:

- iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT — allows packets within the specified rate, but does not automatically block packets exceeding the limit. Without an explicit DROP rule, any packet beyond the limit is simply allowed by the chain's default ACCEPT policy, defeating the purpose of the rate limit.

Overall Conclusion:

The second DROP rule is essential for enforcing the rate limit. It ensures that any packets exceeding the allowed threshold are explicitly discarded, otherwise the limit module alone cannot restrict traffic effectively.

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.

Step1: We shall implement policies to equally divide the incoming packets between the three interval servers. On the seed-router container:

45) 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:

```
seed-router:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$
```

Step2: Start servers on host1 - 192.168.60.5, host2 - 192.168.60.6 and host3 - 192.168.60.7

46) Command: nc -luk 8080

host1 - 192.168.60.5:

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
```

host2 - 192.168.60.6:

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
```

host3 - 192.168.60.7:

```
host3-192.168.60.7:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
```

47)Command: Now on hostA - 10.9.0.5
nc -u 10.9.0.11 8080
< enter 3 words, wait 30 seconds before entering the next word>

hostA - 10.9.0.5:

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>nc -u 10.9.0.11 8080
Pranav
Hemanth
PES1UG23CS433
```

host1 - 192.168.60.5:

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$>nc -luk 8080
Pranav
```

host2 - 192.168.60.6:

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$>nc -luk 8080
Hemanth
```

host3 - 192.168.60.7:

```
host3-192.168.60.7:PES1UG23CS433:PranavHemanth:/
$>nc -luk 8080
PES1UG23CS433
```

Step3: Cleanup

48)Command:
iptables -F
iptables -P OUTPUT ACCEPT
iptables -P INPUT ACCEPT

```
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -F
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P OUTPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -P INPUT ACCEPT
seed-router:PES1UG23CS433:PranavHemanth:/
$iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
seed-router:PES1UG23CS433:PranavHemanth:/
$
```

Explanation: Experiment 1: Round-Robin Mode (Nth Packet)

In this experiment, the router is configured to distribute incoming UDP traffic on port 8080 evenly across three internal servers (192.168.60.5, 192.168.60.6, 192.168.60.7) using the statistic module in nth mode. Three DNAT rules are added to the PREROUTING chain of the nat table. The first rule redirects every third packet to Host1, the second redirects every second packet to Host2, and the final rule acts as a catch-all for the remaining packets, sending them to Host3.

Testing this setup from HostA, each UDP packet sent to the router is correctly distributed: the first packet reaches Host1, the second reaches Host2, and the third reaches Host3. This confirms that the router successfully implements round-robin load balancing, ensuring that the packets are evenly split across all servers in a predictable sequence.

Purpose of the rules:

- --every 3 --to-destination 192.168.60.5:8080 — sends every third packet to Host1.
- --every 2 --to-destination 192.168.60.6:8080 — sends every second packet to Host2.
- --every 1 --to-destination 192.168.60.7:8080 — catch-all rule for remaining packets to Host3.

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> (Here - $ docker restart dccca3a58e38)

Step4: Using the random mode. The following rule will select a matching packet with the probability P . You need to replace P with a probability number.

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:

```
seed-router:PES1UG23CS433:PranavHemanth:/
$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
seed-router:PES1UG23CS433:PranavHemanth:/
$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
seed-router:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$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:PES1UG23CS433:PranavHemanth:/
$
```

Step5: Start servers on host1 - 192.168.60.5, host2 - 192.168.60.6 and host3 - 192.168.60.7

49)Command: nc -luk 8080

host1 - 192.168.60.5:

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
```

host2 - 192.168.60.6:

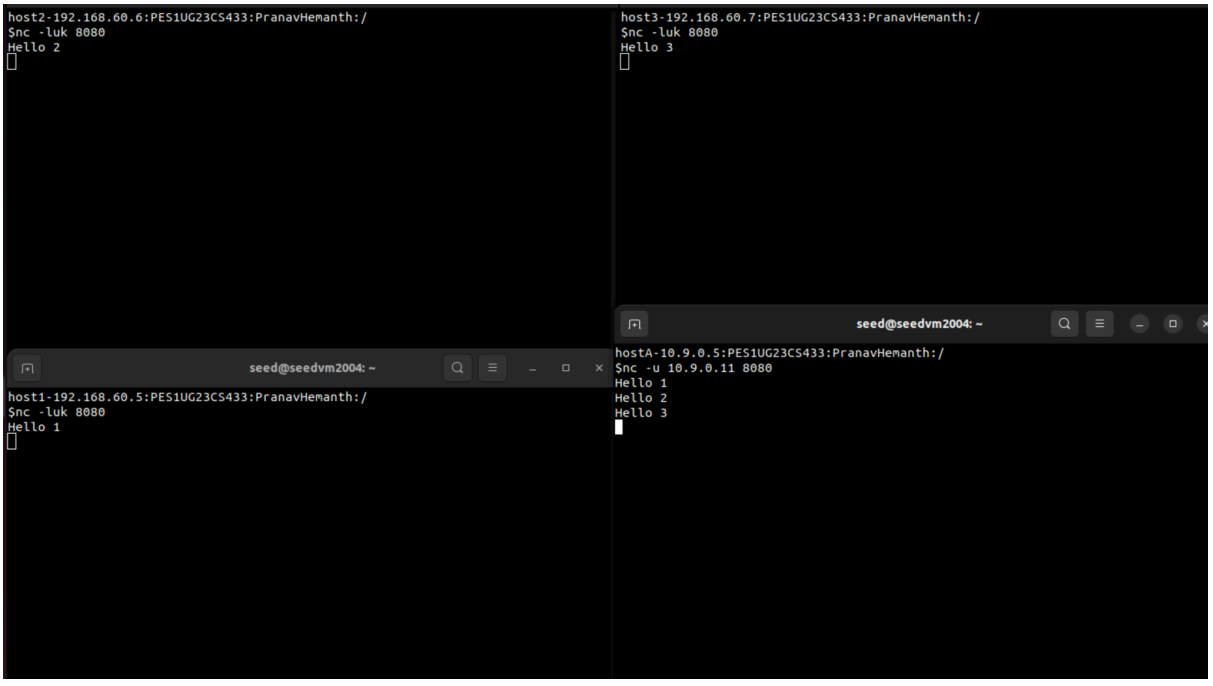
```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
```

host3 - 192.168.60.7:

```
host3-192.168.60.7:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
```

50) Command: Now on hostA - 10.9.0.5
nc -u 10.9.0.11 8080
< enter 3 words, wait 30 seconds before entering the next word>

All windows:



hostA - 10.9.0.5:

```
hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$nc -u 10.9.0.11 8080
Hello 1
Hello 2
Hello 3
```

host1 - 192.168.60.5:

```
host1-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
Hello 1
```

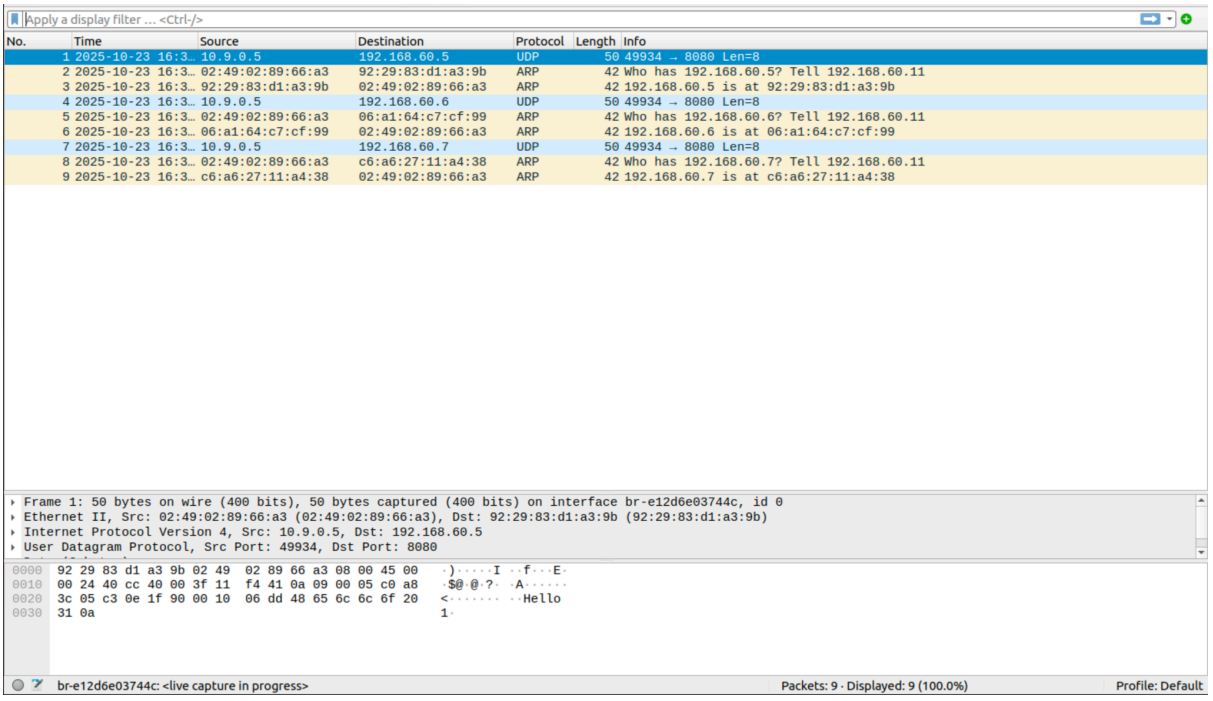
host2 - 192.168.60.6:

```
host2-192.168.60.6:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
Hello 2
```

host3 - 192.168.60.7:

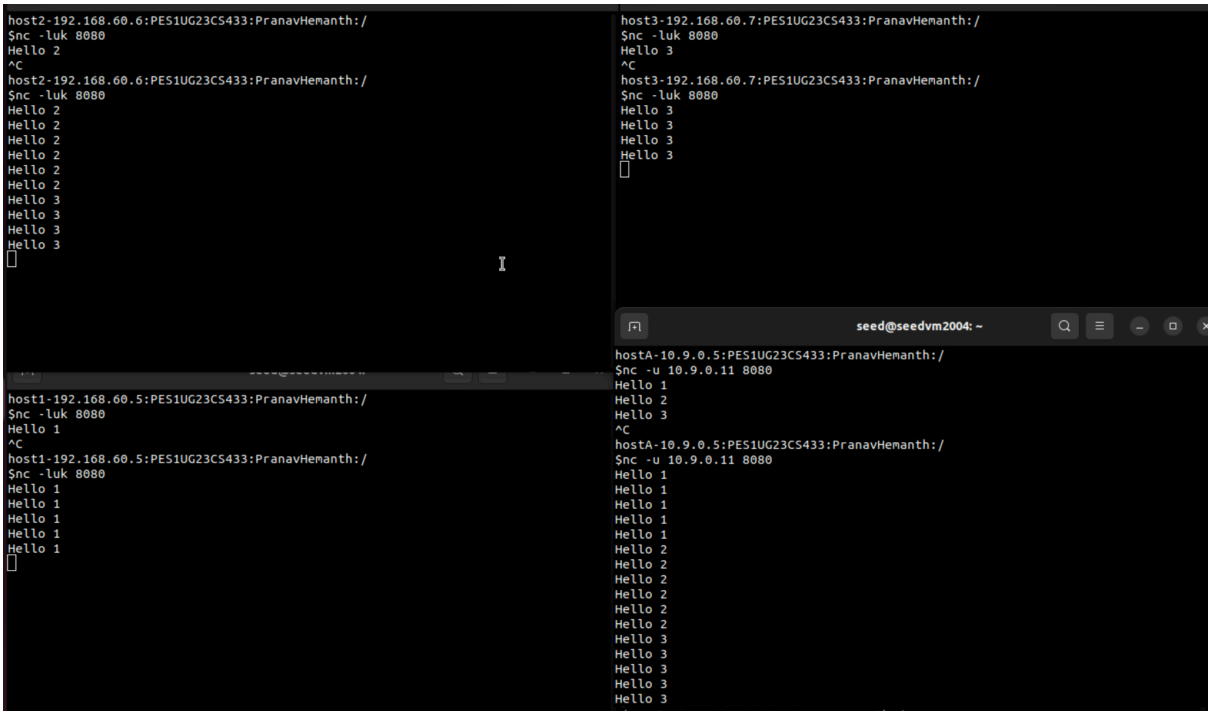
```
host3-192.168.60.7:PES1UG23CS433:PranavHemanth:/
$nc -luk 8080
Hello 3
```

Wireshark:



The above is not a good illustration of the random mode. Hence the below experiment to try packet probability:

All windows:



We are not able to view easily the difference since the packets hit probability incidentally in a way consistent to the non random mode

Explanation: Experiment 2: Random Mode (Probability)

In this experiment, the router is configured to distribute UDP traffic using the random mode of the statistic module. Three DNAT rules are added to the PREROUTING chain with cascading probabilities: 0.3333 for Host1, 0.5 for Host2, and 1 for Host3. Each incoming packet is evaluated probabilistically: it may hit one of the first two rules with the defined chance or be caught by the final catch-all rule.

Testing this setup from HostA, the three packets sent to the router are distributed across the three servers according to their probabilities. The first packet hits the 33.33% chance and is sent to Host1, the second misses the first but hits the 50% chance of the second rule and is sent to Host2, and the third misses both probabilistic rules and is sent to Host3. This demonstrates that random-mode load balancing achieves an approximate even distribution of traffic in a non-deterministic manner.

Purpose of the rules:

- --probability 0.3333 --to-destination 192.168.60.5:8080 — sends packets to Host1 with ~1/3 probability.
- --probability 0.5 --to-destination 192.168.60.6:8080 — sends remaining packets to Host2 with ~1/3 probability.
- --probability 1 --to-destination 192.168.60.7:8080 — catch-all sends all leftover packets to Host3.

Overall Conclusion:

Round-robin mode provides predictable, evenly spaced packet distribution, while random mode provides probabilistic load balancing, which approximates equal distribution but with some variability. Both methods allow iptables to function as a simple UDP load balancer, redirecting incoming traffic across multiple internal servers.