**LAB 4: FIREWALL**

## Setting up SEED labs

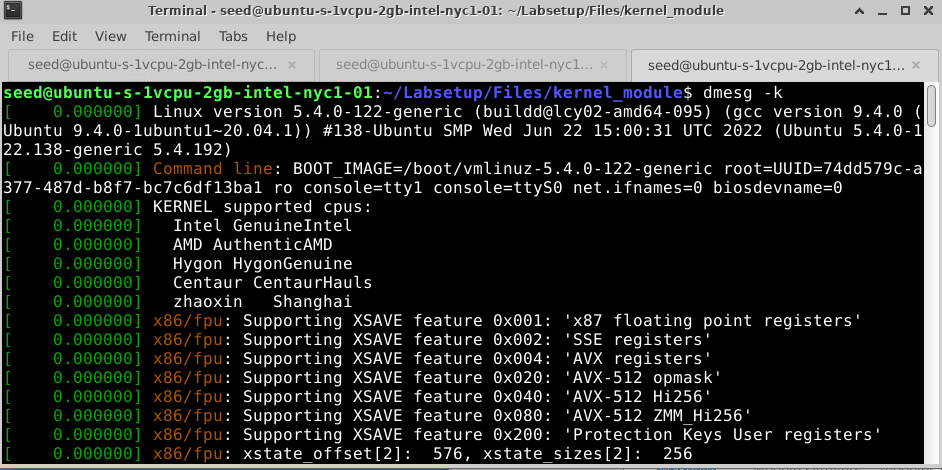
The following is the environment setup for the lab:

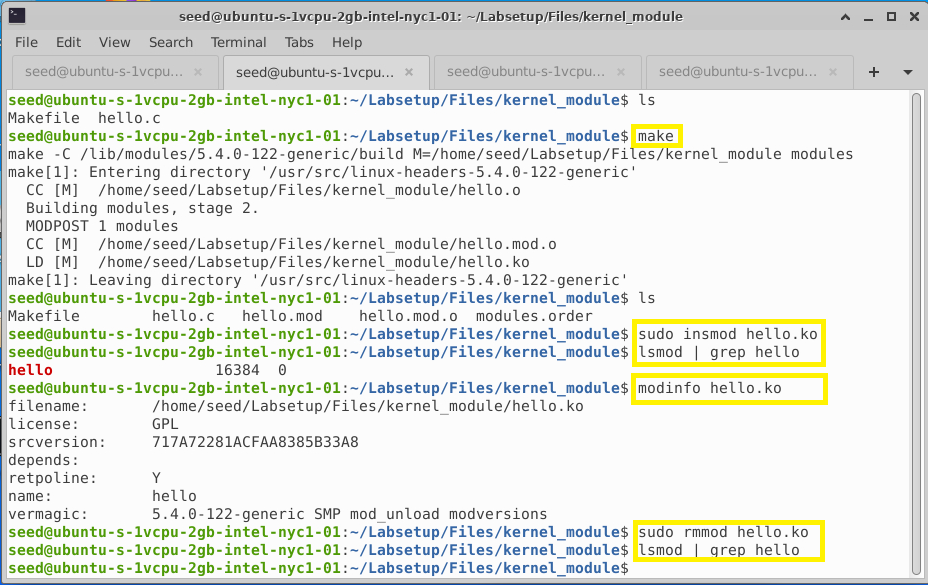
|  |  |
| --- | --- |
| **SYSTEM** | **IP ADDRESS** |
| Host 1 | 192.168.60.5 |
| Host 2 | 192.168.60.6 |
| Host 3 | 192.168.60.7 |
| Host A | 10.9.0.5 |
| Router | 10.9.0.11 |

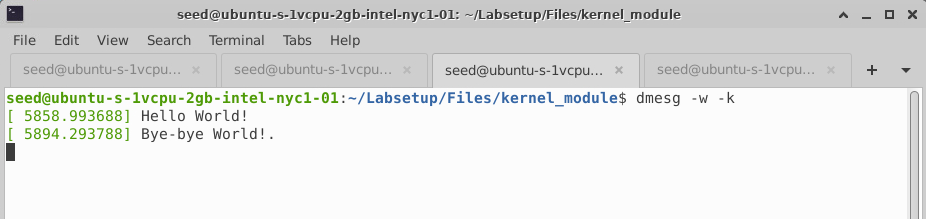


## TASK 1: Implementing a Simple Firewall

### Task 1.A (Implement a Simple Kernel Module)

The dmesg command displays kernel-related messages retrieved from the kernel ring buffer

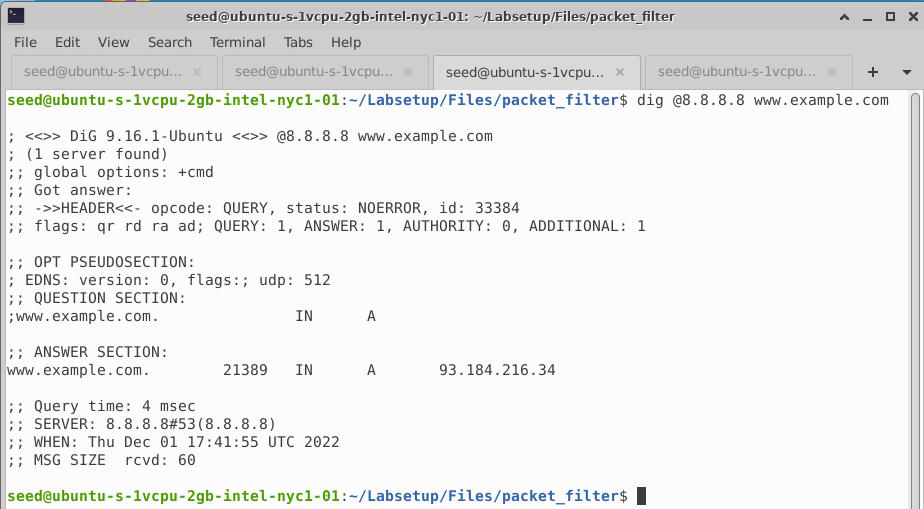
Executed all the commands to load, list, remove and view details of the Linux Kernel Module.  




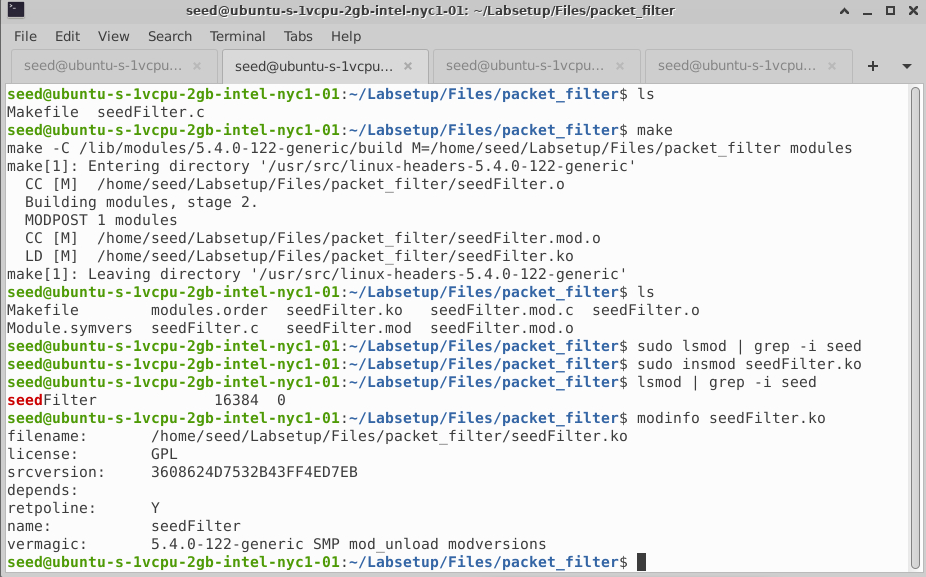
**EXPLANATION:**

It prints out "Hello World!" when the module is loaded using command “sudo insmod hello.ko” and when the module is removed from the kernel using command “sudo rmmod hello”, it prints out "Bye-bye World!" from cleanup() function.

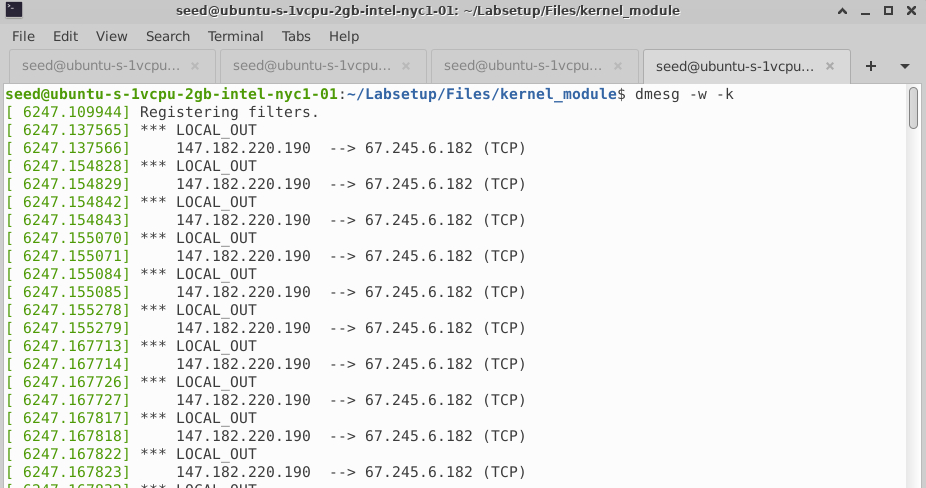
### Task 1.B (Implement a Simple Firewall Using **Netfilter**)

We used the dig command to generate UDP packets to 8.8.8.8, which is Google’s DNS server and received following response on successful connection.

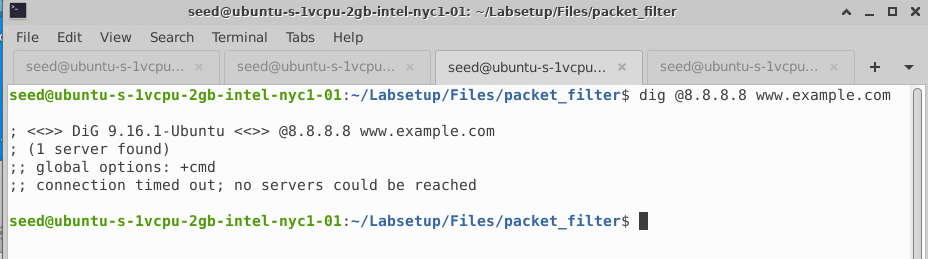
1. We compiled the sample code using ‘make’ command with the provided Makefile.

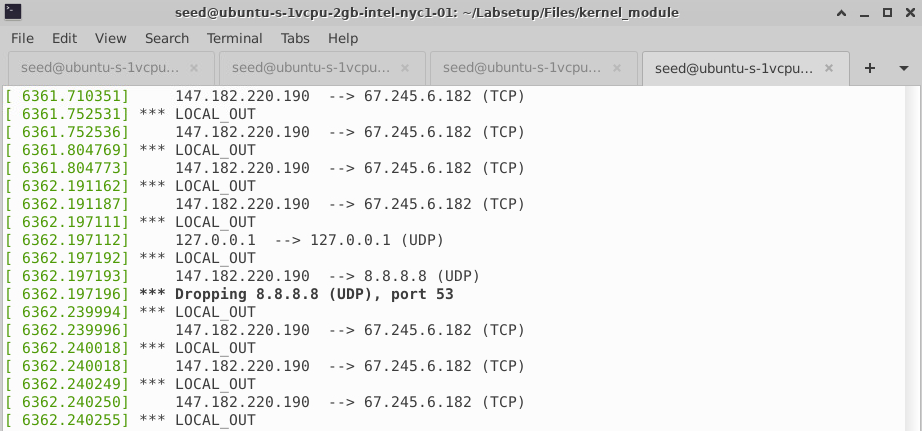


On executing “suso insmod seedFilter.ko” we get following output for dmesg command



As the UDP command is blocked in seedFilter.c, dig command to 8.8.8.8 will be blocked and dropped as seen below.





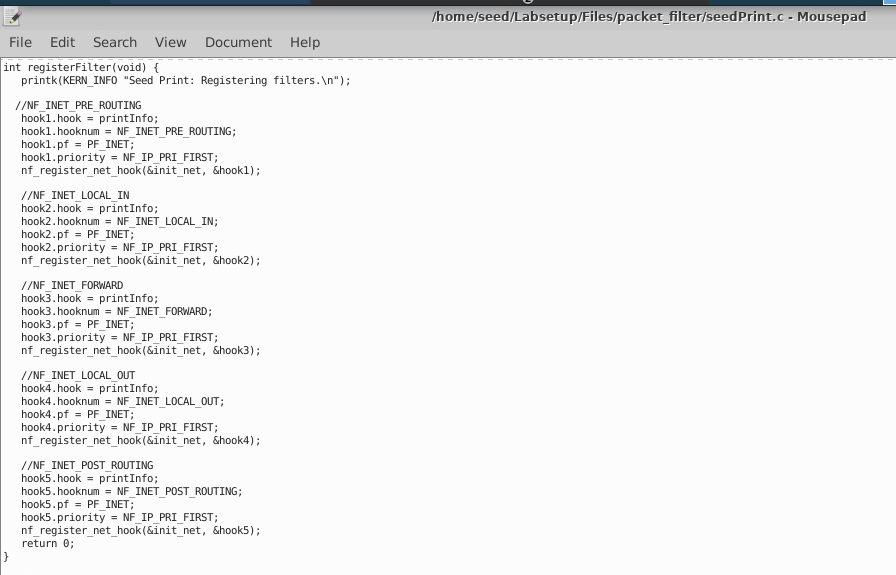
On “sudo rmmod seedFilter” command, we get the last kernel message from printInfo()

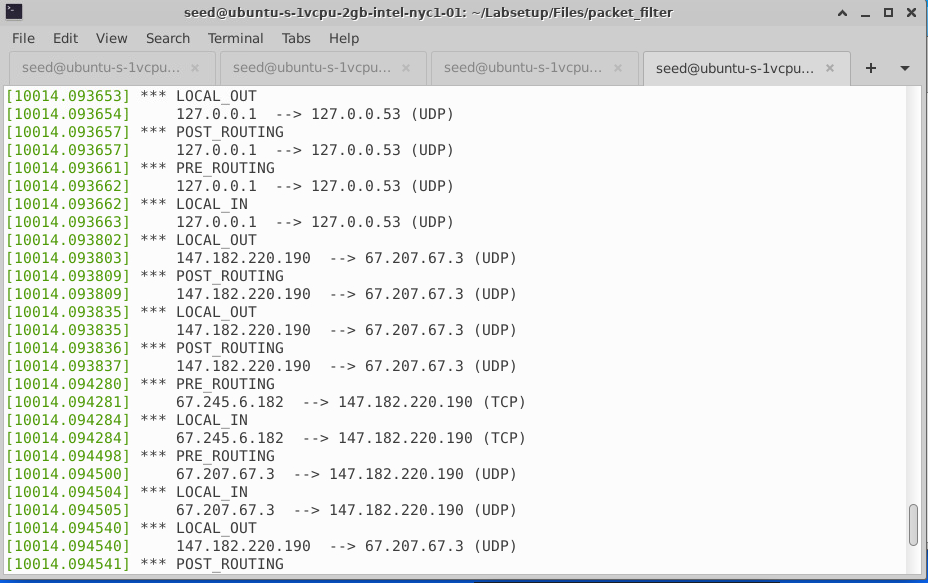


1. New module “seedPrint.c” has been created to understand on what condition will each of the hook function be invoked as seen below:

In this code snippet, when a packet gets to the LOCAL\_IN, PRE\_ROUTING and other hooks, the function printInfo() will be invoked. In printInfo() function,hook number from the

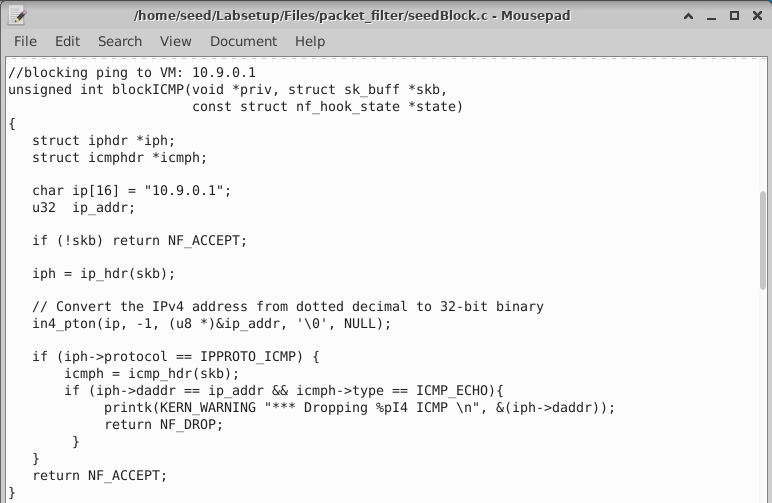
state argument is retrieved and ip\_hdr() function get the pointer for the IP header, %pI4 format string specifier is used to print out the source and destination IP addresses.

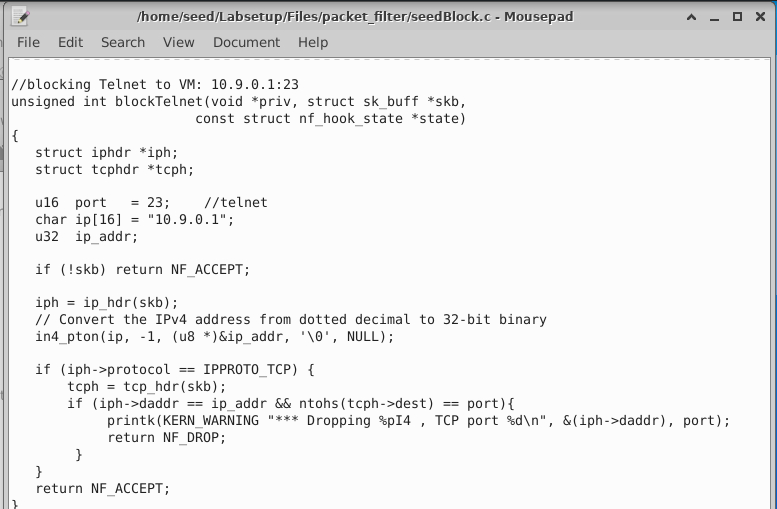


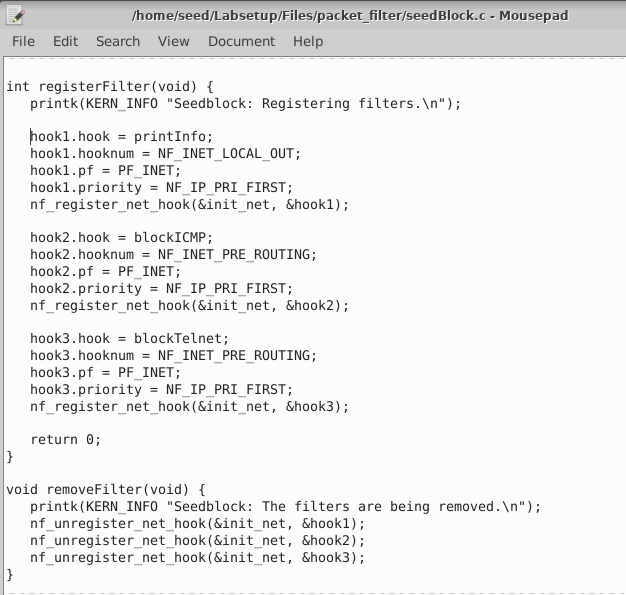


3) Implemented two different hook functions **blockICMP()** and **blockTelnet()**, registered them to the same netfilter hook.

In both functions, IP address is converted to the dotted decimal format to be compared with the binary number stored inside packets. If they match the address and protocol, the NF DROP will be returned to netfilter, which will drop the packet. Otherwise, the NF ACCEPT will be returned, and netfilter will let the packet continue its journey.



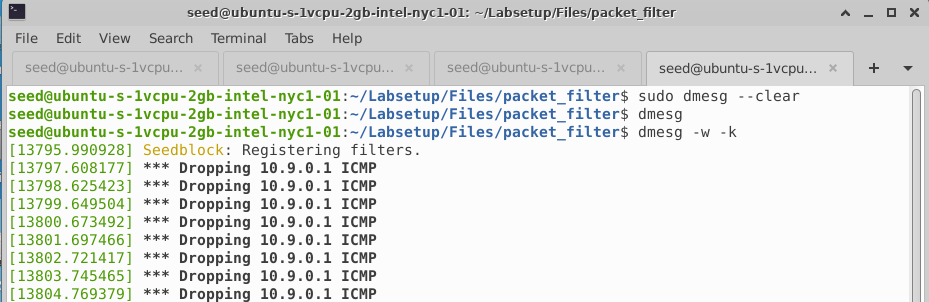


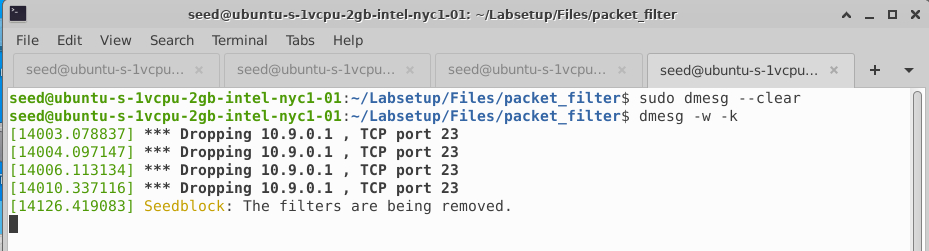




Started the docker containers and ran the following commands in Host A (10.9.0.5)







**EXPLANATION:**

We understood how to implement simple firewall that requires our code to be loaded into the kernel using the hooks provided by netfilter to do packet filtering.

Kernel modules have a priority number to help determine the order in which they will be called when the hook is triggered. This provides the means for multiple modules (or multiple instances of the same module) to be connected to each of the hooks with deterministic ordering.

Netfilter supports 5 hooks inside the kernel which can be used to load the kernel module and implement packet filtering firewall. The 5 hooks are NF\_IP\_PRE\_ROUTING, NF\_IP\_POST\_ROUTING, NF\_IP\_FORWARD, NF\_IP\_LOCAL\_IN, NF\_IP\_LOCAL\_OUT. All these hooks are tested using “seedPrint.c”

For Ingress Filtering we use NF\_IP\_PRE\_ROUTING and for Egress Filtering - NF\_IP\_POST\_ROUTING

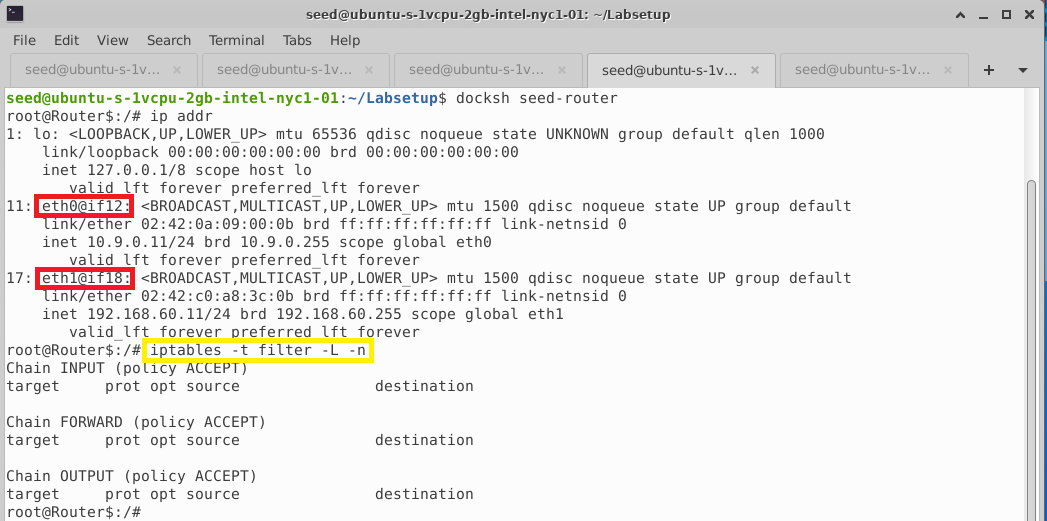
In “seedBlock.c”, we prevented ICMP and TCP (23) packets to 10.9.0.1.

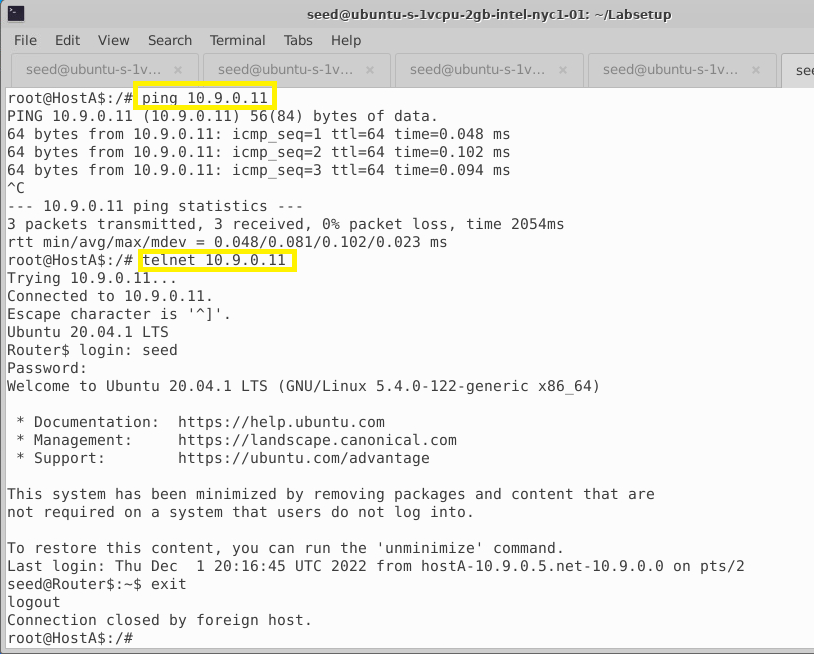
We used the command “dmesg -k -w” to wait for kernel messages and use “sudo dmesg –clear” to clear all the exisiting messages in the kernel.

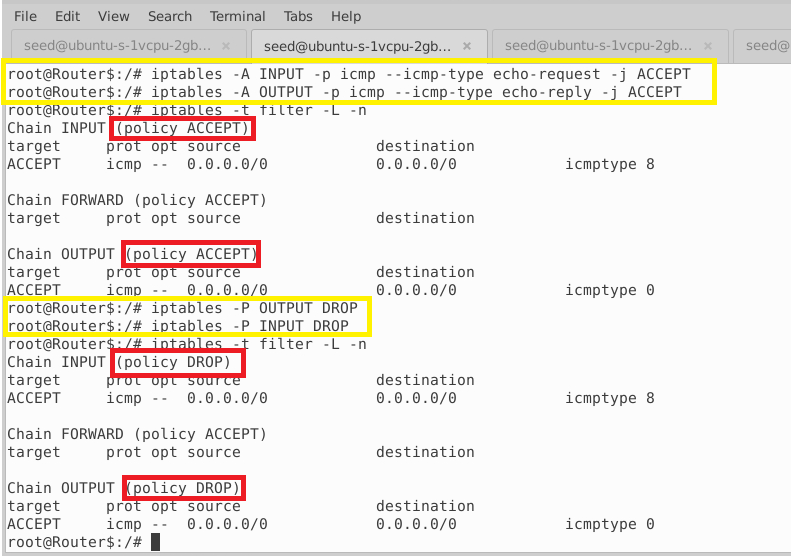
## Task 2: Experimenting with Stateless Firewall

### Task 2.A (Protecting the Router):

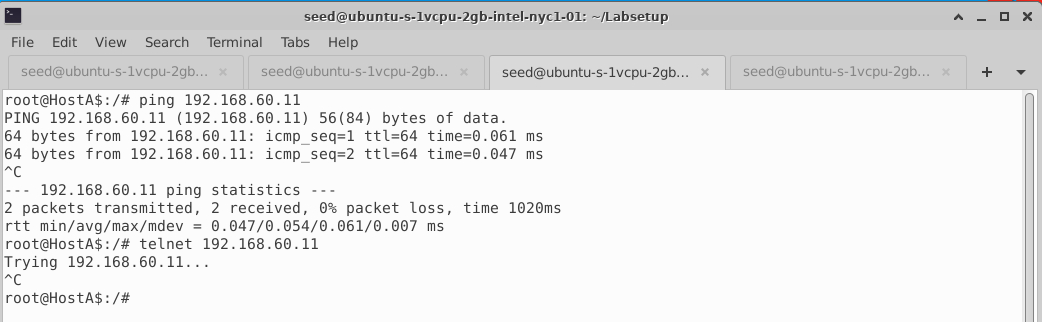
We see that seed-router has 2 ports – **eth0** and **eth1** using **ip addr** command



Before applying firewall rules, we are able to ping and connect to seed-router (10.9.0.11) from HostA (10.9.0.5)



After applying the rules:

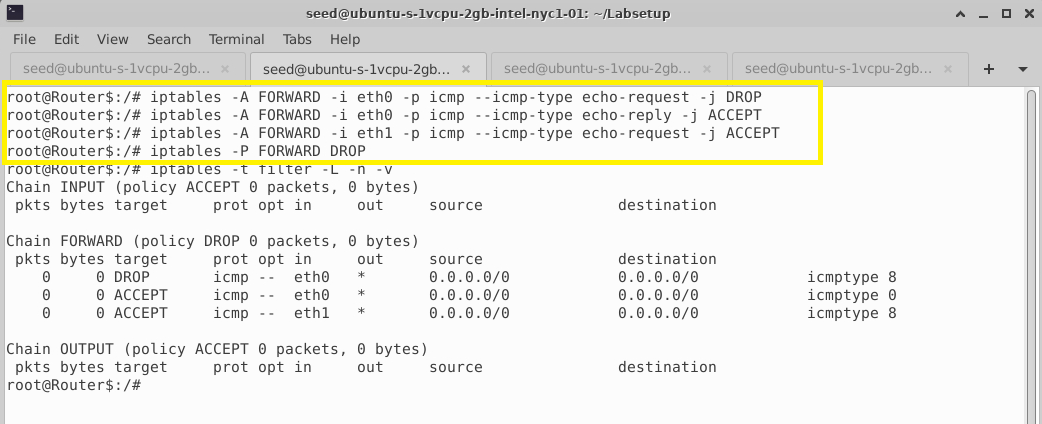


**EXPLANATION:**

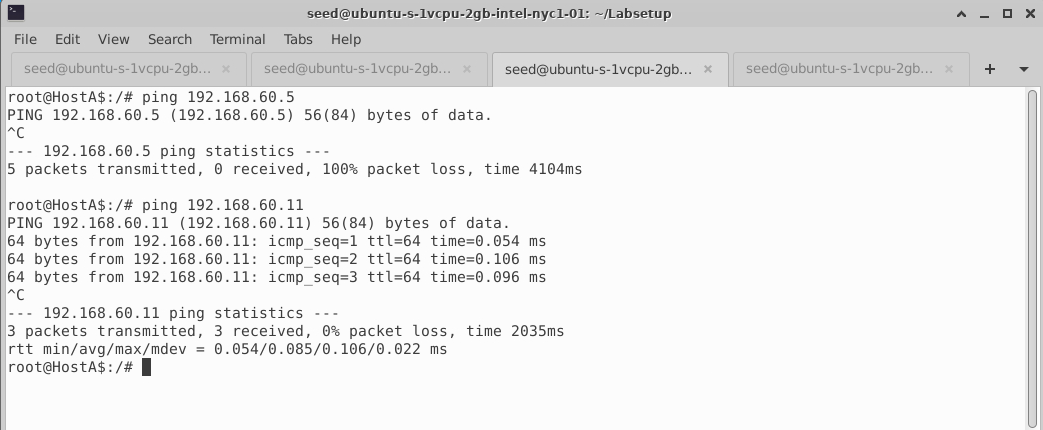
We executed the iptables command as given and observed that -P option will change the default policy of the Chain. We infer that ICMP type 8 is for REQUEST and ICMP type0 is for REPLY. After applying the rules, all packets except the ICMP packets from Host A to router are dropped.

### Task 2.B (Protecting the Internal Network):

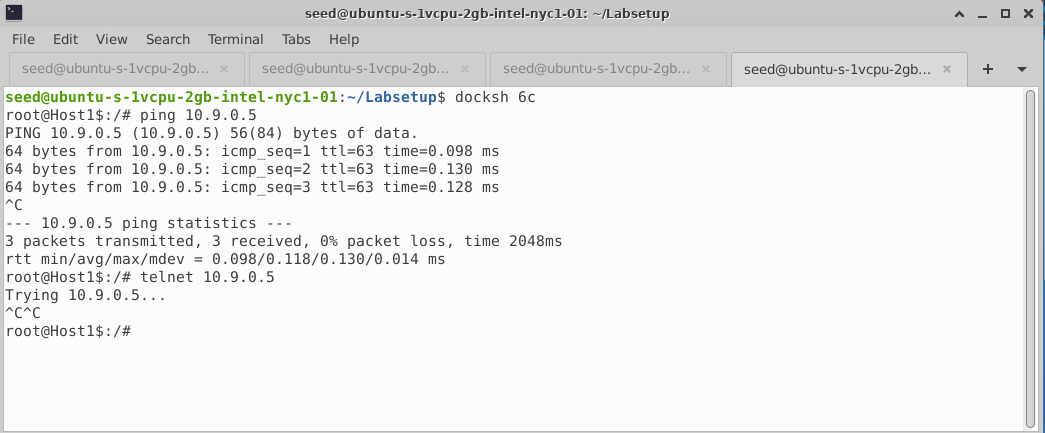
We setup the following firewall rules with iptables commands in the router to protect the internal network and achieved the requirements:



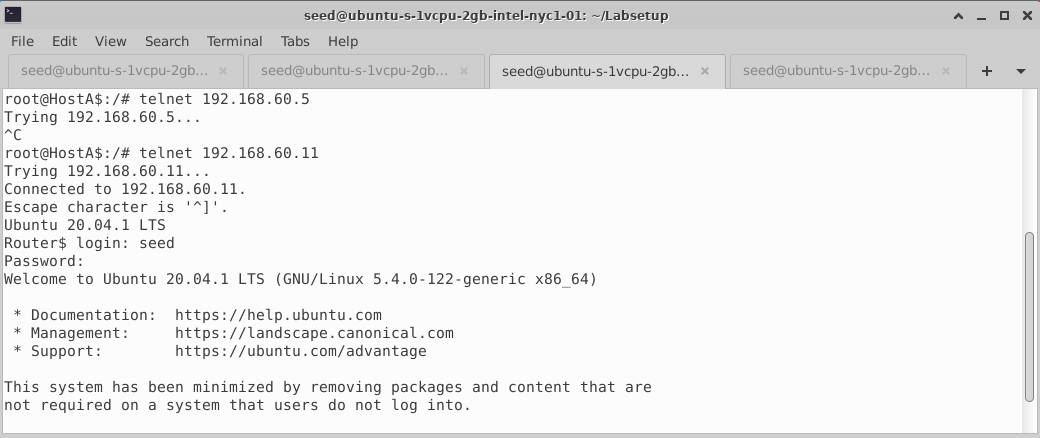
After this, outside hosts (HostA) cannot ping internal hosts (Host1) but can ping the router.



Internal hosts (Host1) can only ping outside hosts (HostA) but cannot send any other packets.



External hosts cannot send any type of packet to internal hosts

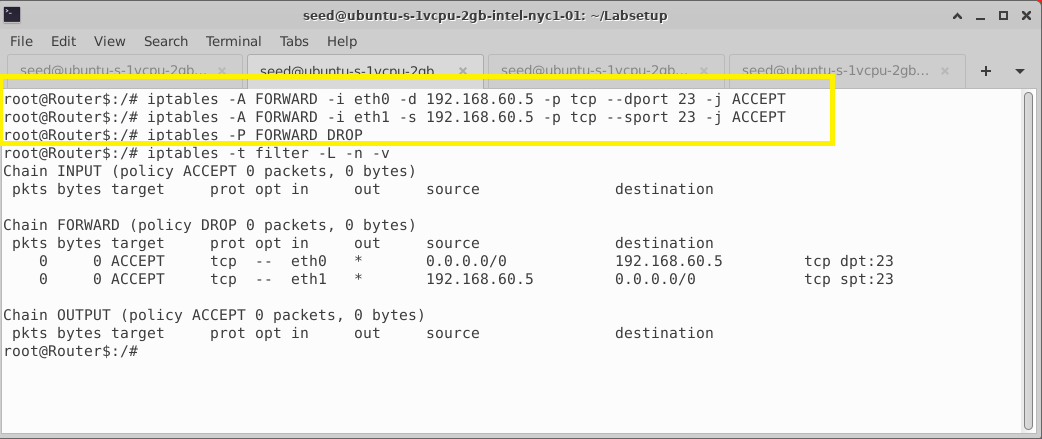


**EXPLANATION:**

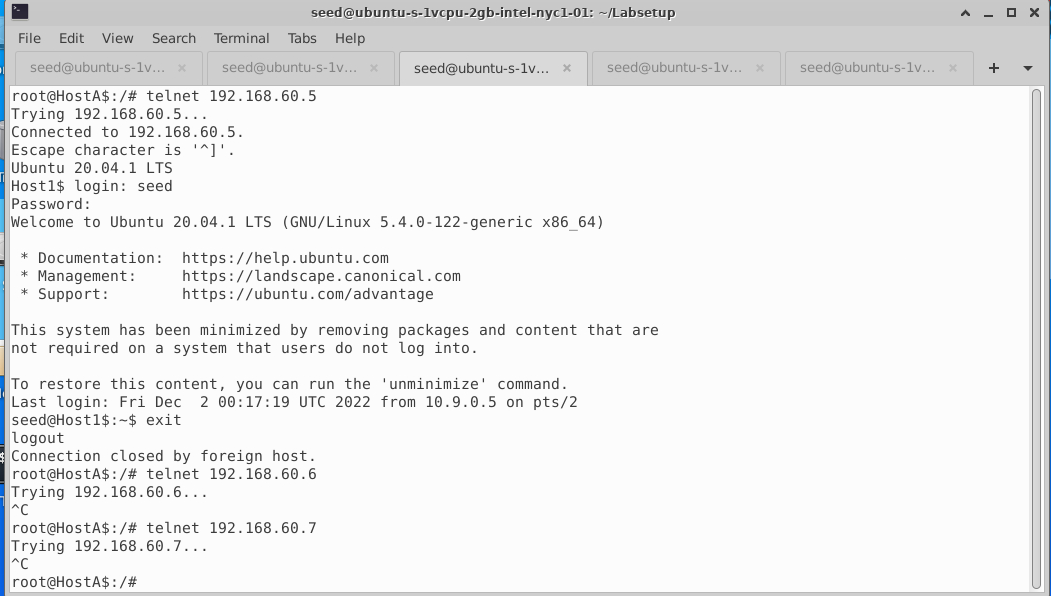
We create firewall rules on the router to protect the internal network from external server (HostA) from ICMP traffic by using the "-p icmp" options to specify the match options related to the ICMP protocol.

### Task 2.C (Protecting Internal Servers):

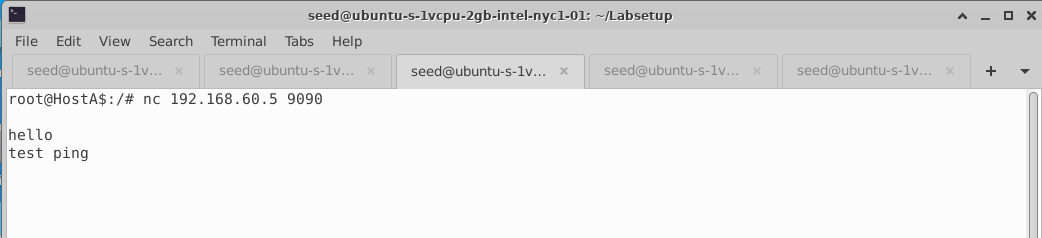
We setup the following firewall rules with iptables commands in the router to protect the internal network and achieved the requirements:

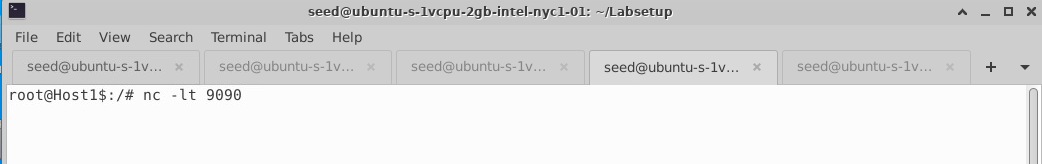


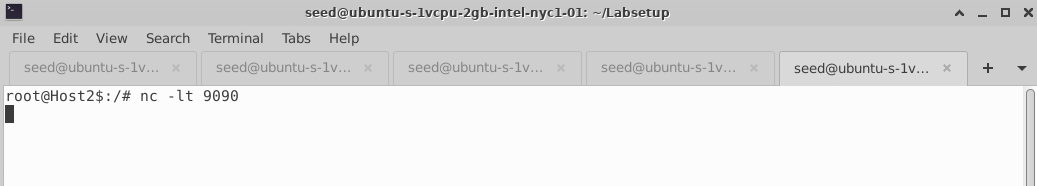
All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.



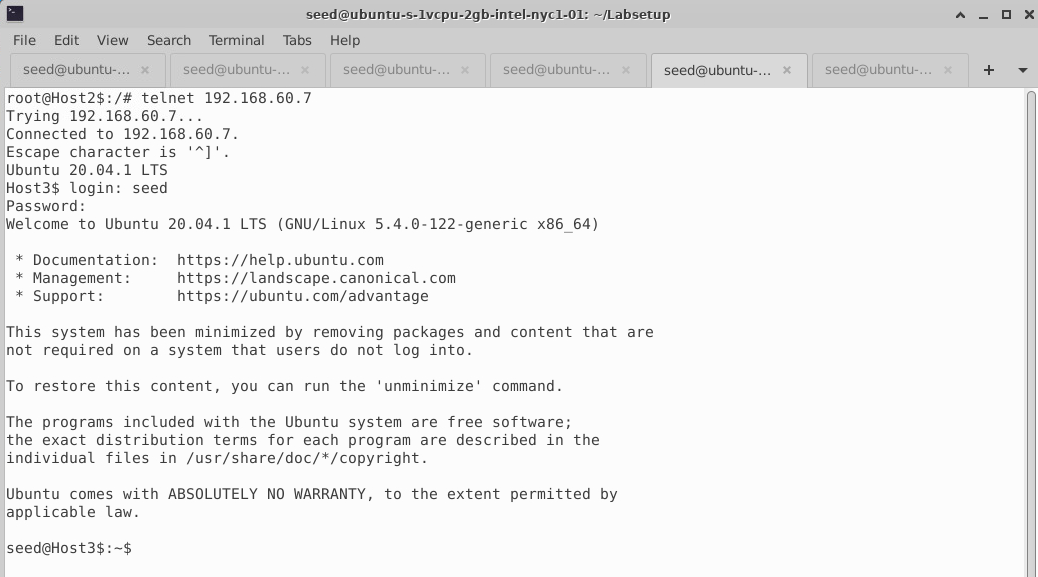
Outside host A cannot access other internal servers and other ports in 192.168.60.5 (Host1) which is tested with netcat on port 9090



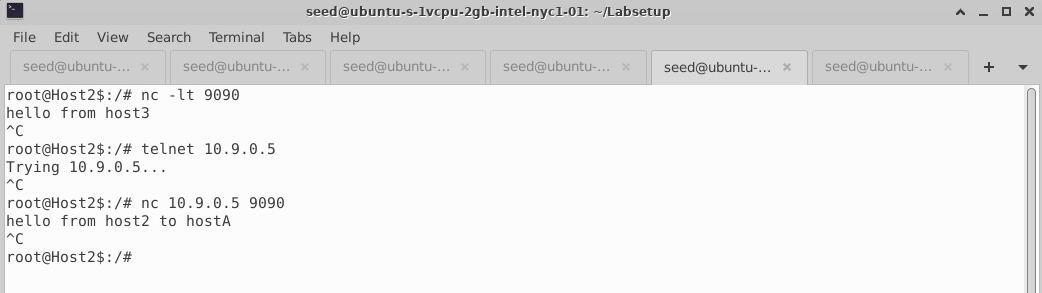


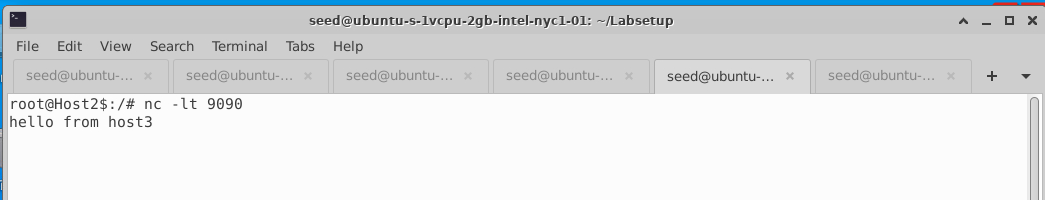


Internal server (Host2- 192.168.60.6) can access the another server (Host3 -192.168.60.7).

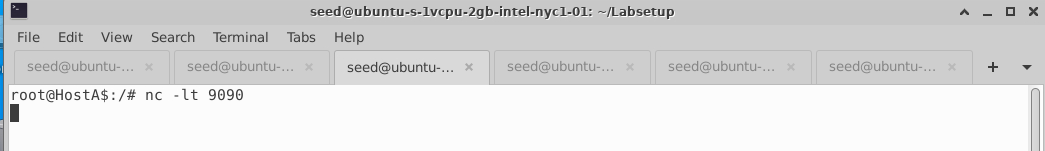


Internal hosts can access all other internal servers (example Host3 to Host2 – 192.168.60.7) in any port.





Internal servers (Host3- 192.168.60.7) cannot access the external server (HostA -10.9.0.5).



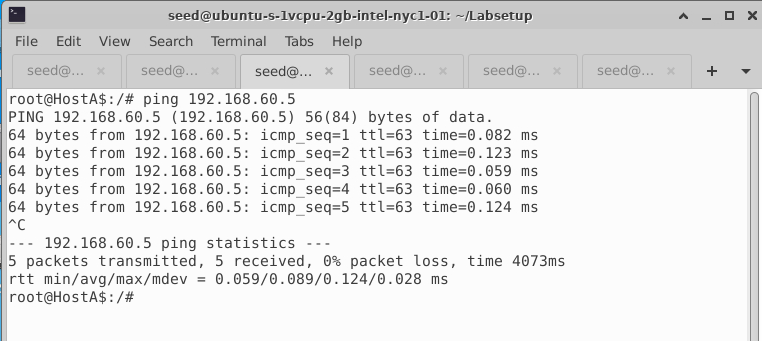
**EXPLANATION:**

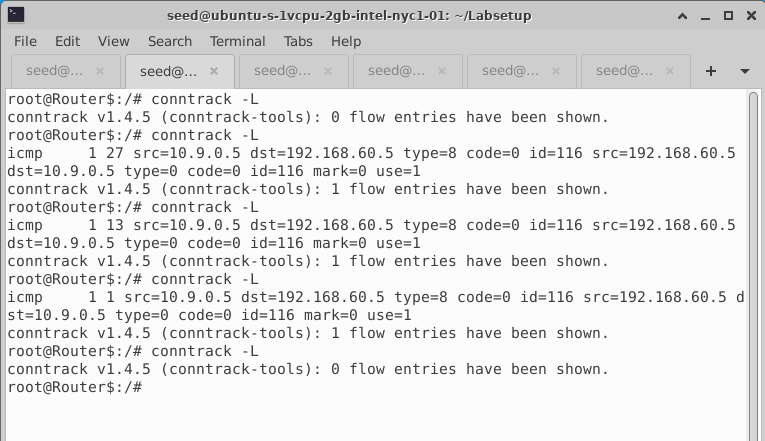
Similar to previous sub task we achieve internal TCP server protection with the above rules on the router. We mention both set of rules in FORWARD chain to compare the packet state against the ruleset in the filter table until the first match, or until the default policy of the chain is executed.

## Task 3: Connection Tracking and Stateful Firewall

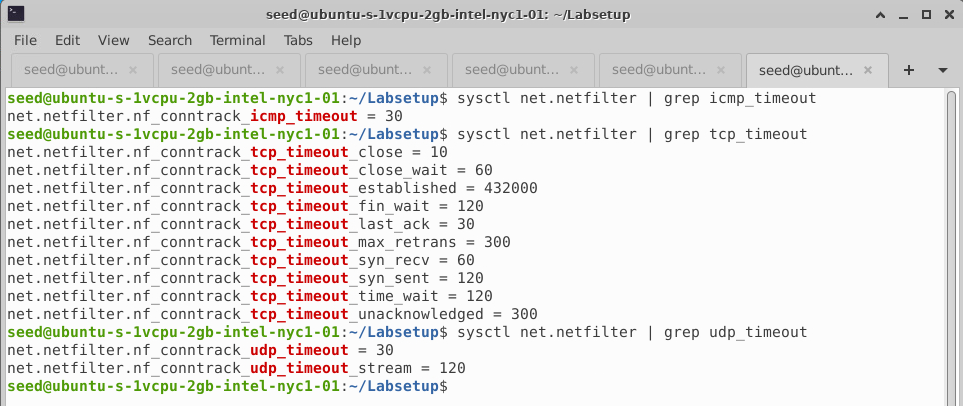
### Task 3.A (Experiment with the Connection Tracking):

**ICMP experiment:** We pinged 192.168.60.5 (Host1) from 10.9.0.5 (HostA) and checked the connection tracking information. ICMP connection state information is maintained in memory tables for 30 seconds.

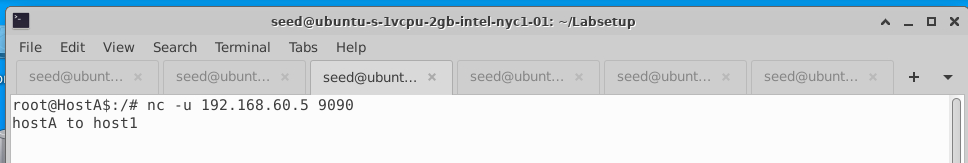


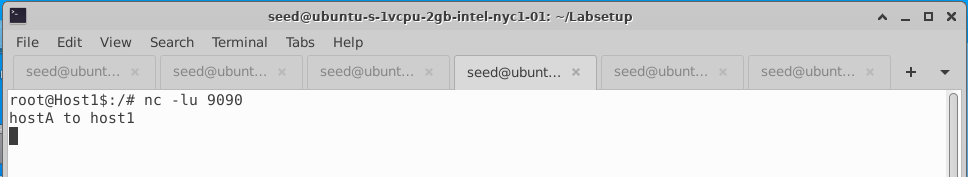


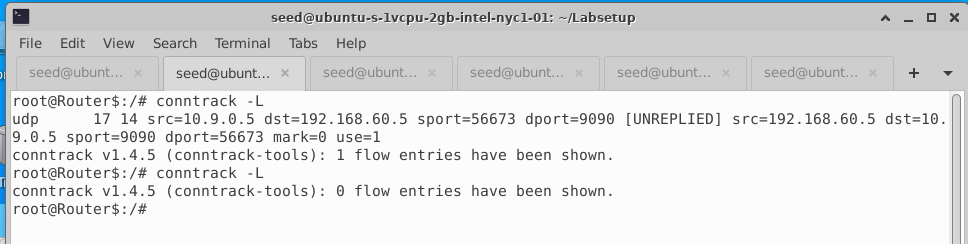
Timeouts for each protocol are listed below using the following command. Timeouts are set in /usr/src/linux/net/ipv4/netfilter/ip\_conntrack\_proto\_tcp.c at compile time.



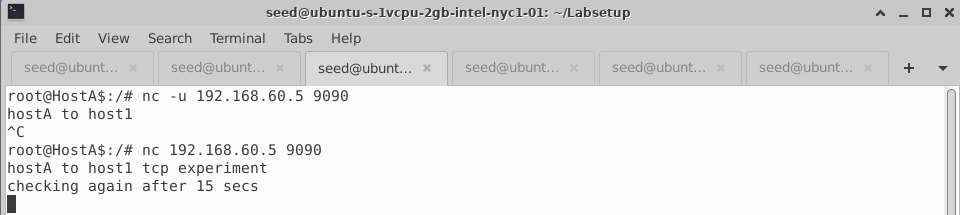
**UDP experiment:** Ran the following command for UDP packet and checked the connection tracking information on the router which is stored in memory for upto 30 seconds.

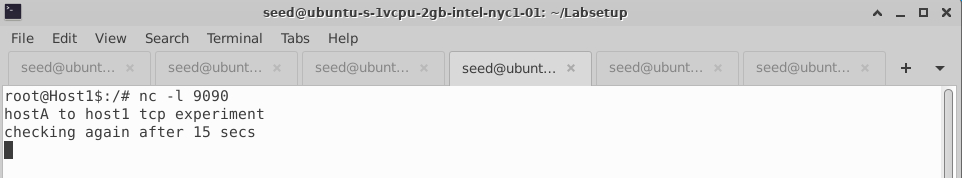


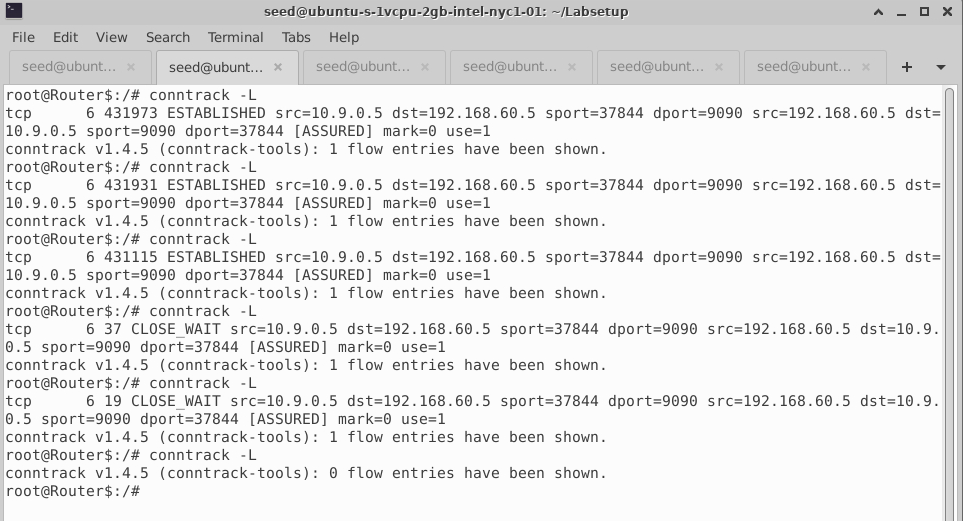




**TCP experiment:** Ran the following command for UDP packet and checked the connection tracking information on the router. It is observed that connection state is in memory until connection is closed and waits for upto 60 seconds once connection is closed.

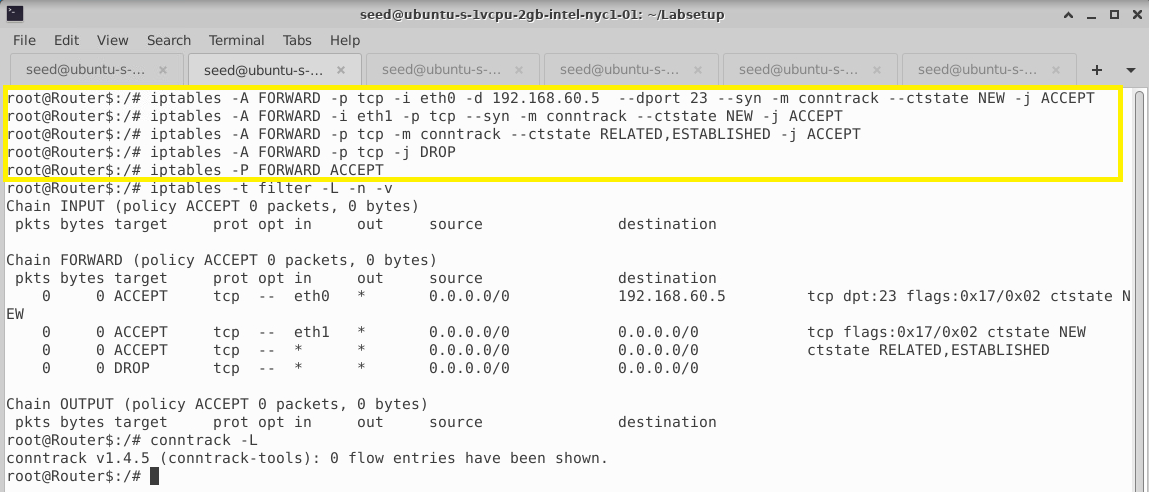




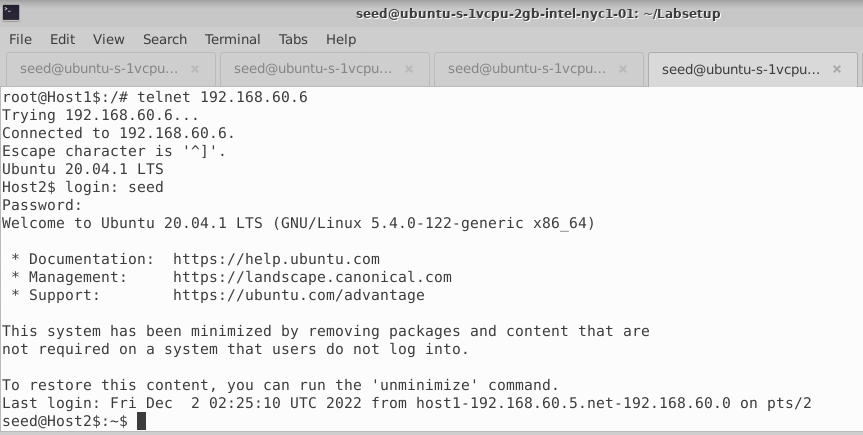


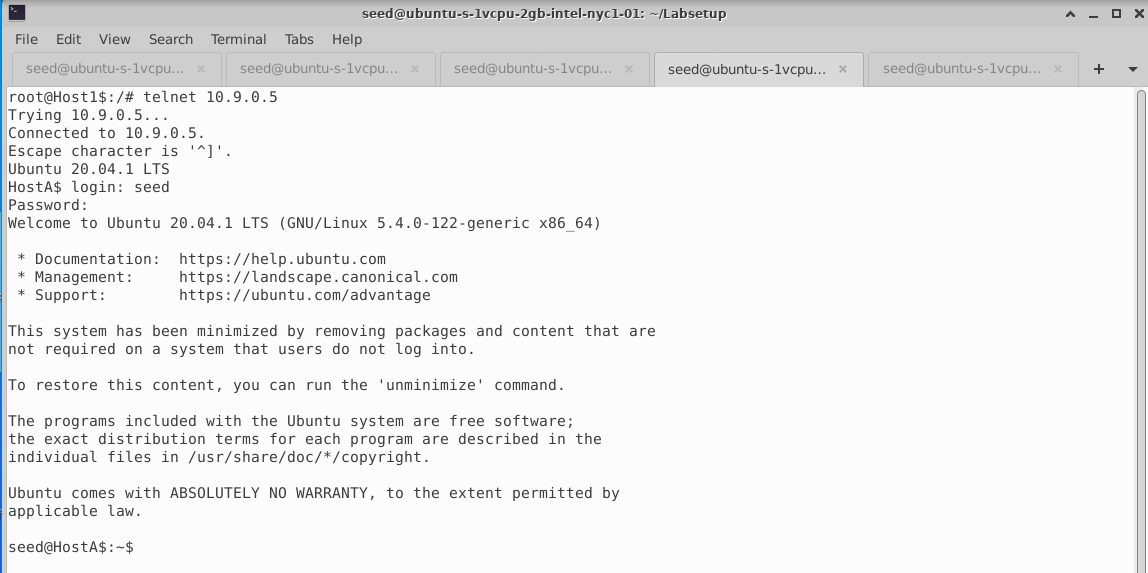
### Task 3.B (Setting Up a Stateful Firewall):

Added similar rules as in Task (2C) but with” conntrack” to keep track of the packets transferred.

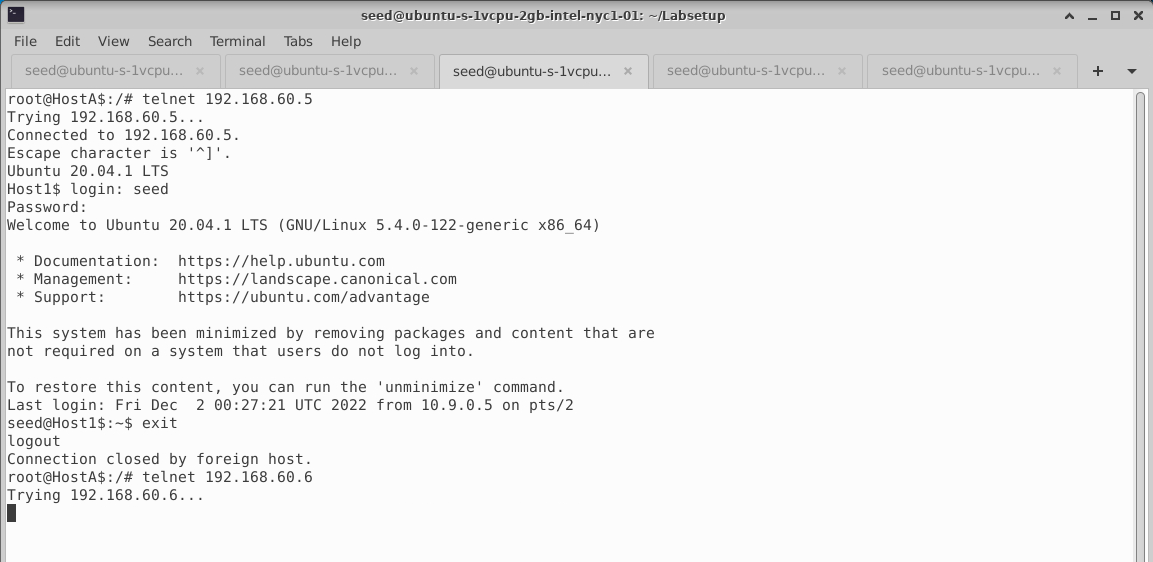


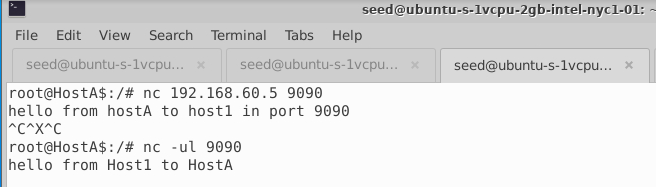
Internal hosts can access any internal and external servers (HostA -10.9.0.5) which was not allowed in Task (2C)

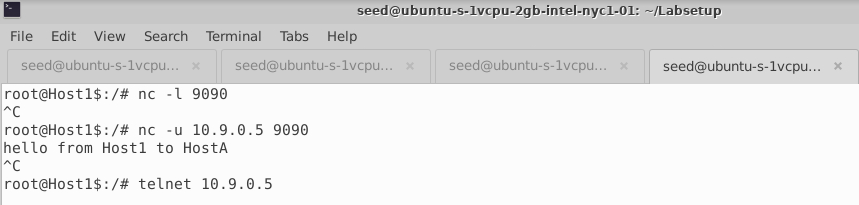




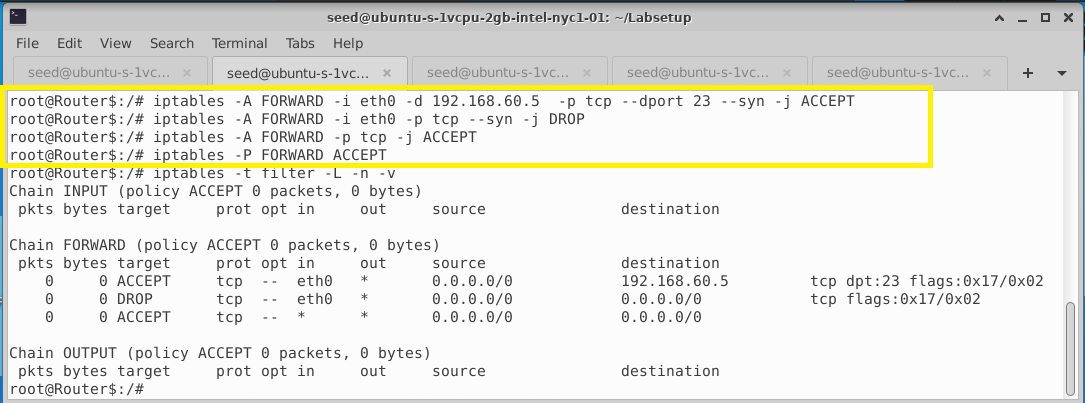
All the internal hosts run a telnet server (listening to port 23). Outside hosts (HostA) can only access the telnet server on 192.168.60.5, not the other internal hosts.







We implemented same rules without conntrack to check on how the packet transmission behaves.



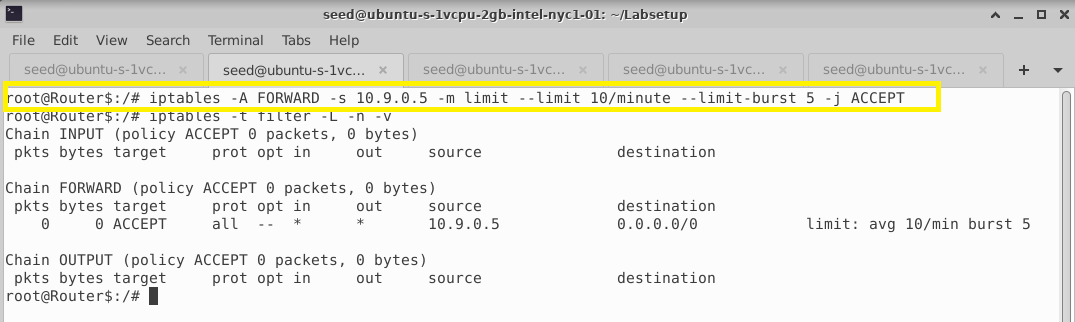
**EXPLANATION:**

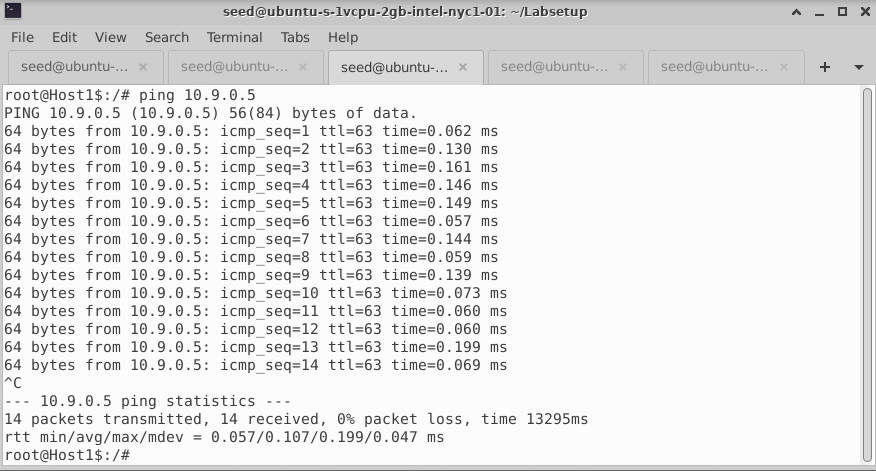
In this task, we understood how stateful firewall works with the help of “conntrack” mechanism inside the kernel which keeps track of the network connections and patterns of behavior. If a data packet has some suspicious activity this stateful firewall can recognize it and drops it unlike stateless firewall which doesn’t store any history of previous connections and data packets. In stateful firewall, we have the advantage of filtering packets based on the state of the packet.

We are able to set rules based on the state of the packet (ESTABLISHED, NEW, RELATED) in stateful firewall. As the packet states cannot be tracked in stateless firewall, stateful firewalls prove to be more secure than stateless firewalls.

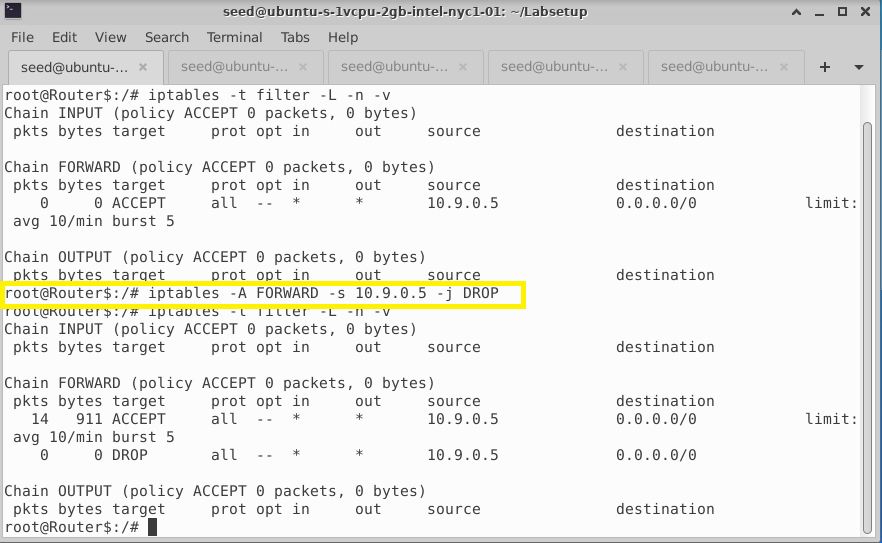
## Task 4: Limiting Network Traffic

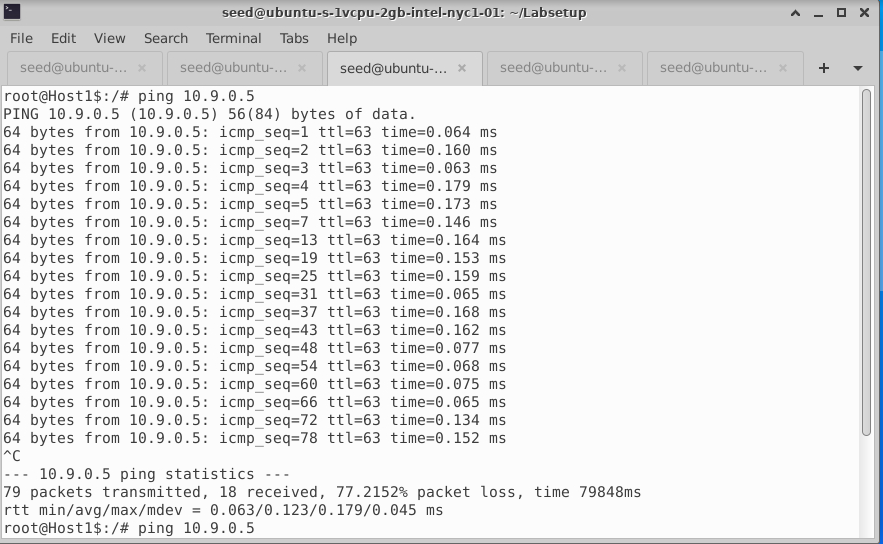
We implement the given rule in the router with limit option. But it does not have any effect on the limiting the traffic on the router and no packets are dropped.





After implementing the second command, we observe that part of the packets from external hosts are dropped due to limitation and there is no restriction on internal packets.





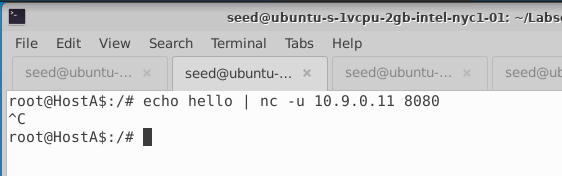
**EXPLANATION:**

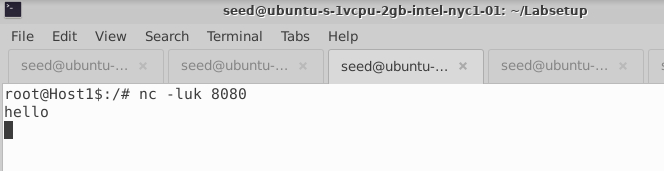
Since the default policy of the FORWARD chain is ACCEPT, all the packets are transmitted even though limit is set in first rule. Once the second rule is added to the table, the average number of packets and limit on burst number works as expected.

## Task 5: Load Balancing

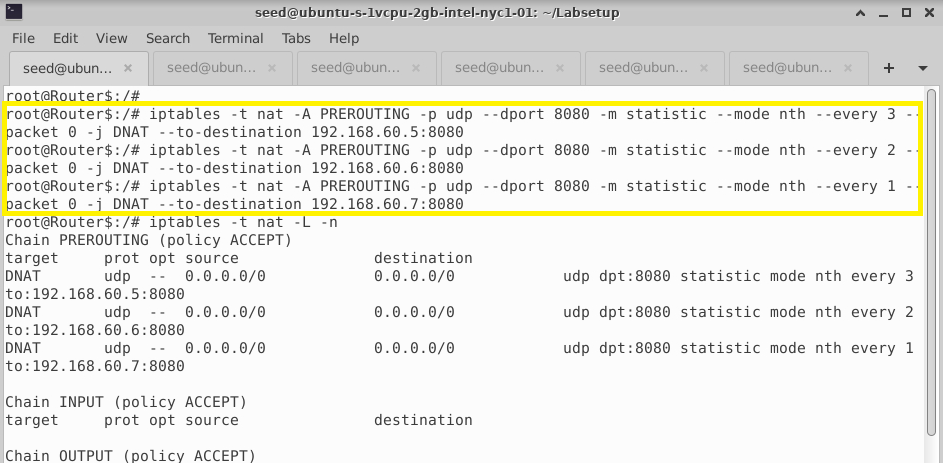
First, we execute the command to redirect every 3rd packet to 192.168.60.5(Host1) and test it.

## 



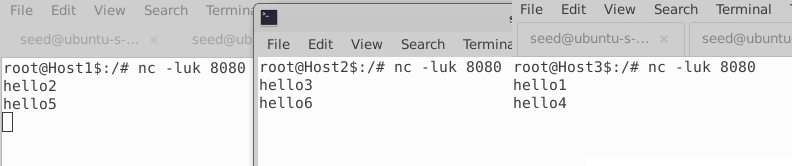


To distribute those connections between Server1 (Host1), Server2 (Host2) and Server3 (Host3) using round-robin, we define the following rules. This algorithm takes two different parameters: every (n) and packet(p). The rule will be evaluated every n packet starting at the packet p.

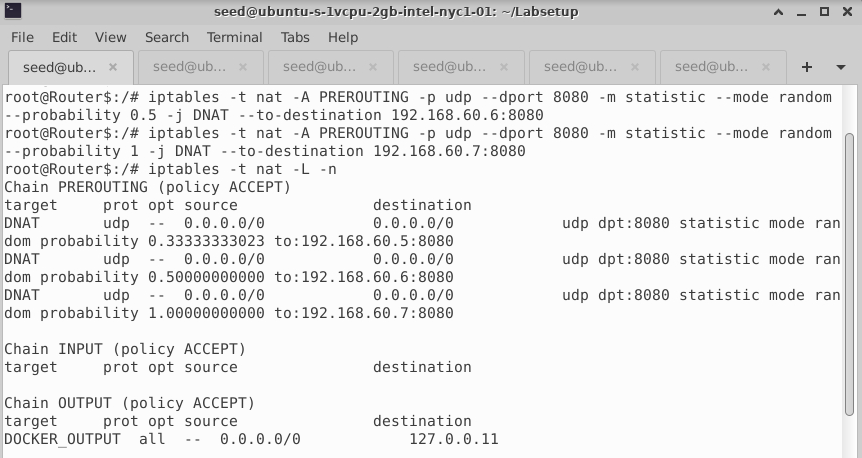


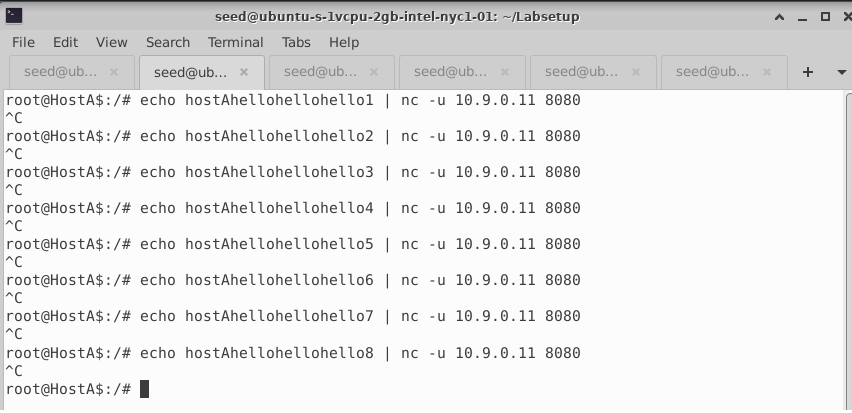
We test whether three internal hosts get the equal number of packets from external server (Host A) as below.

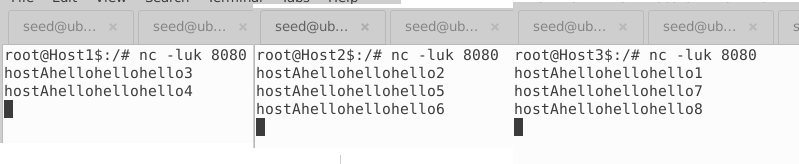




To load balance traffic on 3 different servers using probability, we define the following rules.







Note 3 different probabilities are defined and not 0.33 everywhere. The reason is that the rules are executed sequentially. With a probability of 0.33, the first rule will be executed 33% of the time and skipped 66% of the time. With a probability of 0.5, the second rule will be executed 50% of the time and skipped 50% of the time. However, since this rule is placed after the first one, it will only be executed 66% of the time. Hence this rule will be applied to only (50%\*66%=33%) of requests. Since only 33% of the traffic reaches the last rule, it must always be applied.

**EXPLANATION:**

While performing both load balancing, we need to wait for UDP timeout (30 seconds) for the packet to be redirected to each server equally.

In nth mode, load balancer directs requests in a rotating fashion, with the first server in the group fielding a request and then moving to the bottom, where it awaits its turn to be called upon again. This technique ensures each server handles about the same number of connections.

In random mode, it can be seen from the above that the probability of hitting the three rules is the same but depends on the weights. In addition, if we want to modify the hit rate of the three rules, we can adjust it through the --probability parameter.

In iptables, the rules are matched in top to bottom order. Therefore, when designing iptables rules, the rules must be sorted strictly. Therefore, the order of the above three rules cannot be changed, otherwise the equalization cannot be achieved.

## Task 6: Write-Up:

Firewalls are an important tool that can be configured to protect your servers and infrastructure. Main functionalities include filtering data, redirecting traffic, and protecting against network attacks. Firewalls are important because they are a single point where you can impose security and auditing. They may be installed at an organization's network perimeter to guard against external threats, or within the network to create segmentation and guard against insider threats too.

There are different types of firewalls depending on where they are used:

1. Application Layer Firewall
2. Network Layer or Packet filtering firewall

In this lab, we worked on Packet filtering that will only check for the IP addresses and not verify the data of the packet. We only set rules on source, destination IP and type of packets.

Packet filter types:

* Stateless firewall - examines each packet independently and does not know whether any given packet is part of an existing stream of traffic. It can be vulnerable to IP spoofing attacks.
* Stateful firewall - maintains a table that keeps track of all open connections for which we used 'conntrack' above.

There are various firewalls on Linux such as iptables, nftables, UFW, etc. All of these firewall tools are user-space utility programs that rely on Netfilter. Netfilter is the Linux kernel subsystem that allows various networking-related operations to be implemented and aids in developing firewall using the Linux Kernel Module(LKM). These modules can be inserted into the kernel and removed on demand.

Iptables builds upon netfilter to provide a powerful firewall, which can be configured by adding rules. There are different structures provided by iptables for packet filtering:

* Tables - allows to process packets in specific ways. We mainly used FILTER and NAT table.
* Chains - allow you to filter packets at various points. PREROUTING, INPUT, OUTPUT, FORWARD, POSTROUTING.
* Targets – decide fate of the packet whether to DROP, ACCEPT, REJECT.

The netfilter kernel hooks are close enough to the networking stack to provide powerful control over packets as they are processed by the system. The iptables firewall leverages these capabilities to provide a flexible, extensible method of communicating policy requirements to the kernel. By learning about how these pieces fit together, we utilize them better to control and secure server environments.