**Crypto and Network Security**

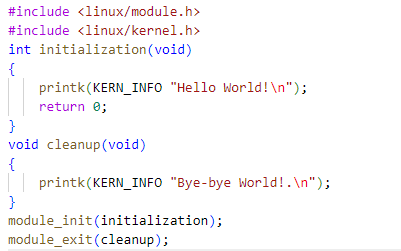
**Lab – 5**

**Name: GAURAV SETTY**

**Email: settgm01@pfw.edu**

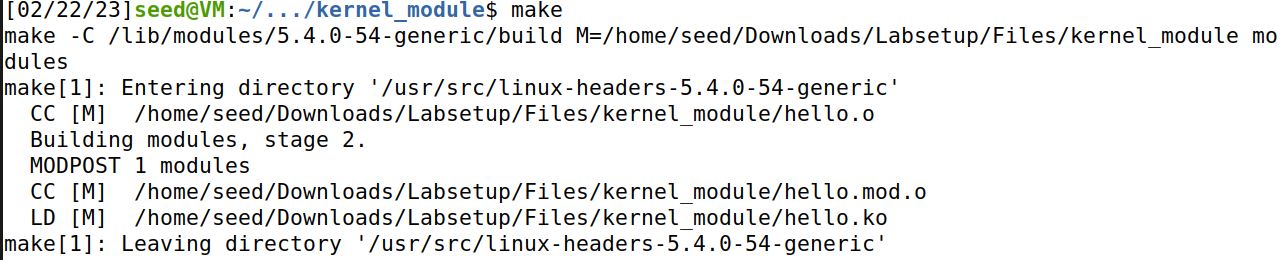
**Task 1.A Implementing a simple kernel module (LKM)**

Hello.c

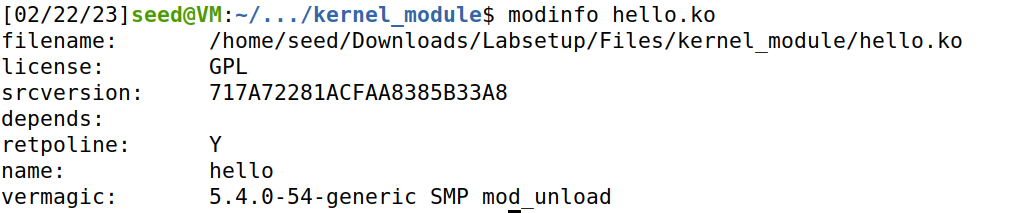


During kernel initialization, the above code emits Hello World and Bye-Bye World. whenever the module is taken out. With the make command, let's compile the module. After that, we'll install the kernel mod and then uninstall it.

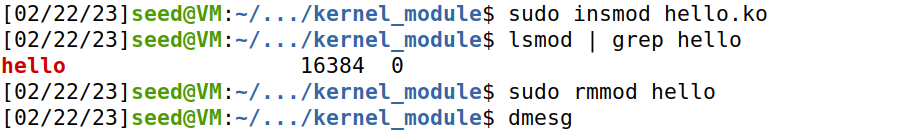
Then, to observe its installation and deletion, we will examine the kernel logs.



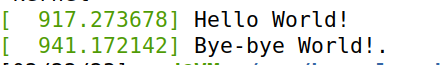
To view the LKM information, run modinfo on the hello.ko file.



Let’s run the following command :



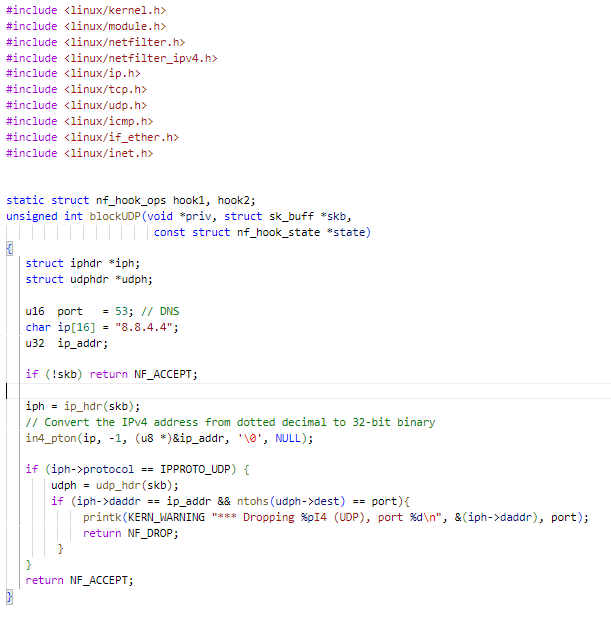
The messages Hello World and Bye-bye World are displayed here:

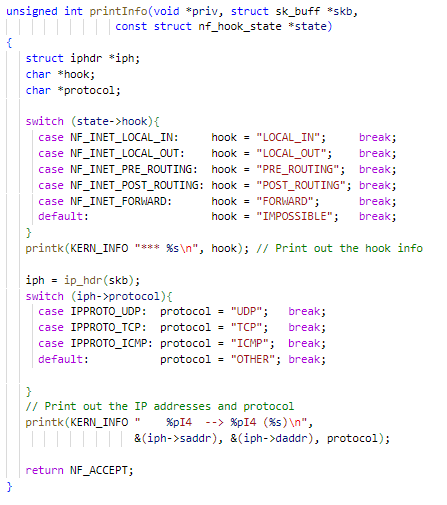


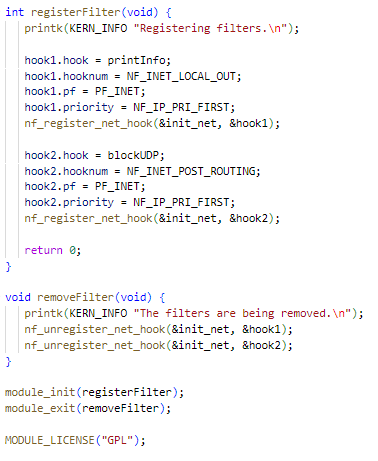
**Task 1.B Implement a simple firewall using netfilter**

**Part 1:**

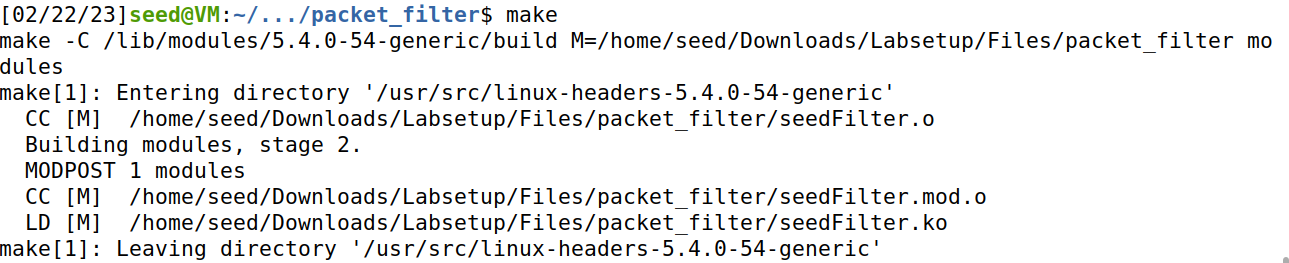
Code:



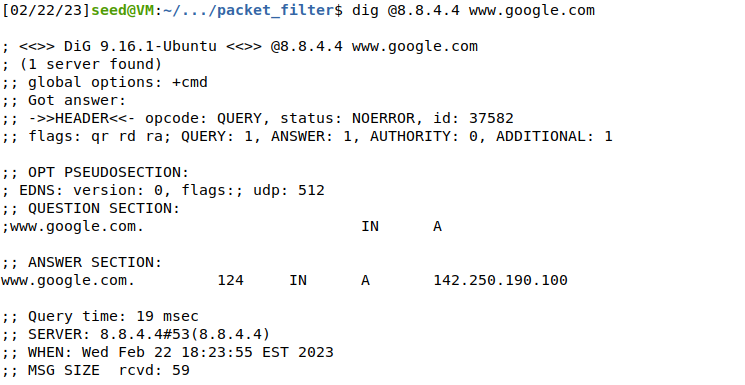




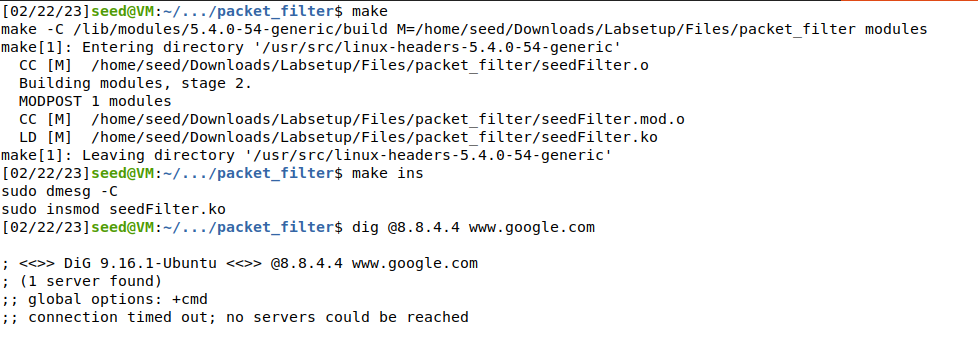
Run Make file:



Dig into [www.google.com](http://www.google.com):



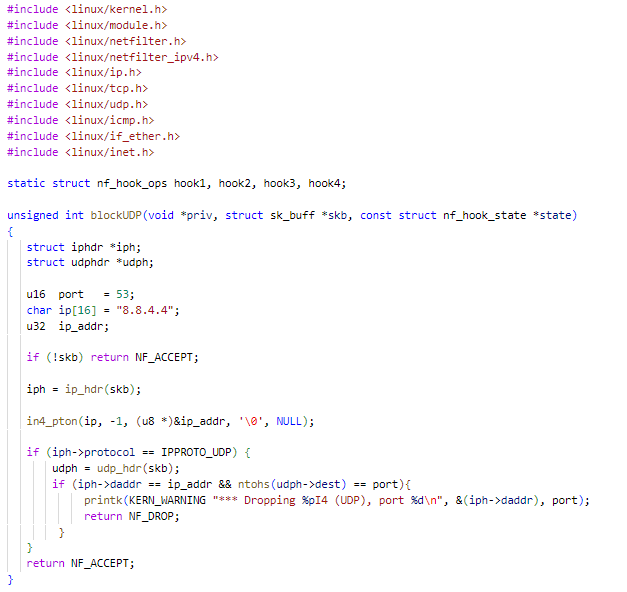
We can see that it operates without issue. Let's proceed to adding the module to the kernel now:

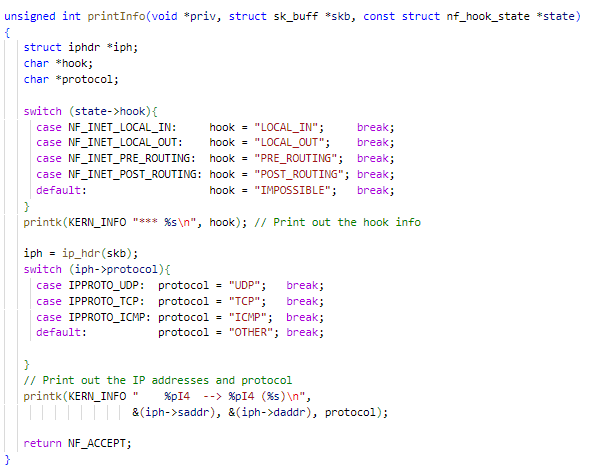


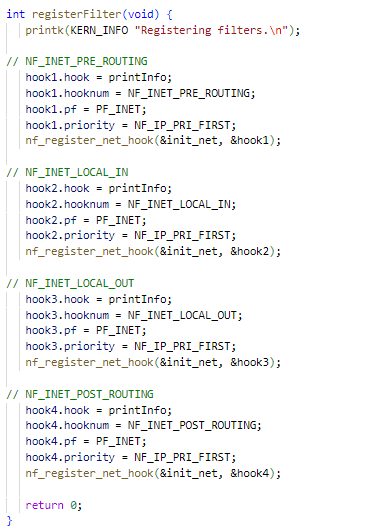
We can see the Request was now blocked. So, our firewall is in place.

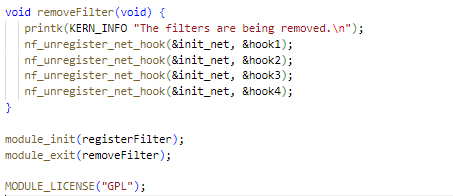
**Part 2:**

Code:

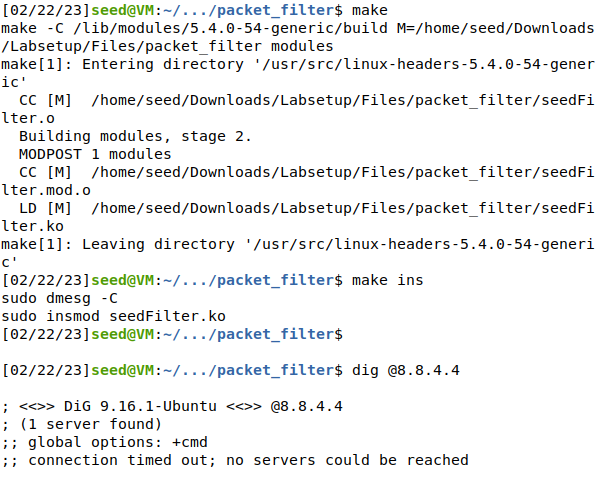




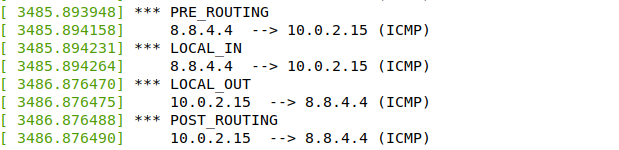




Let's now execute the Makefile, add the module to the kernel, and do a test.

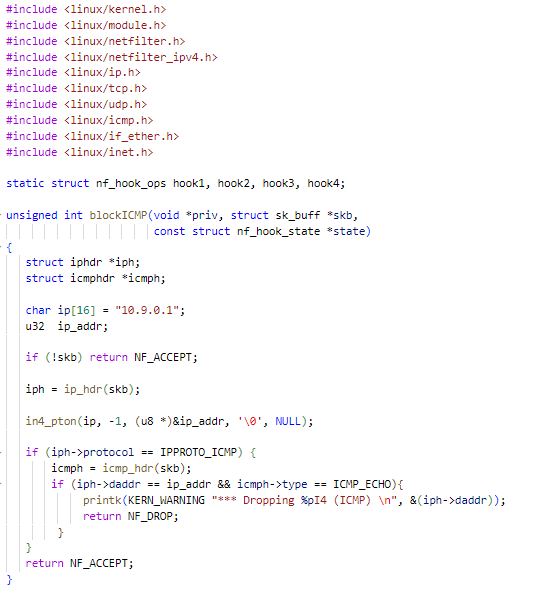


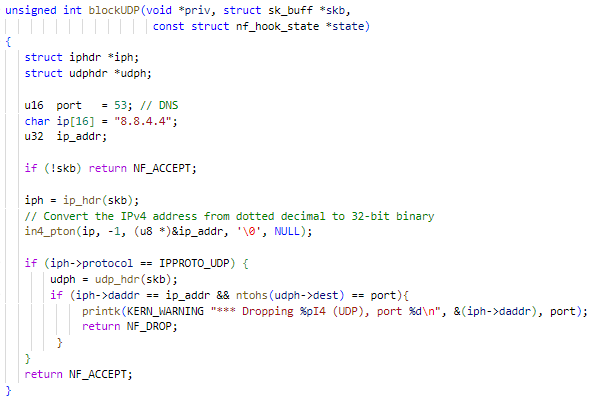
Here we can see the request was blocked by all Netfilter hooks.

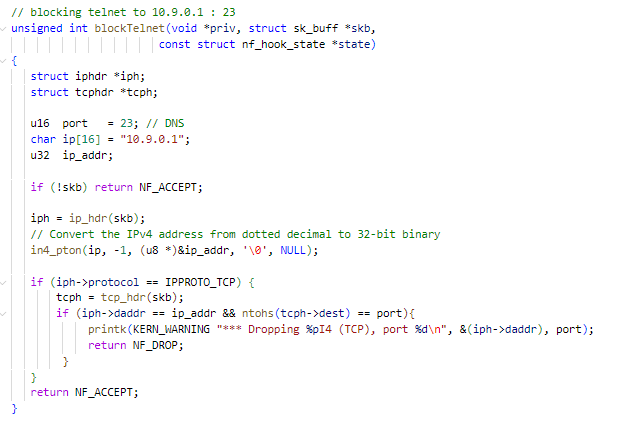


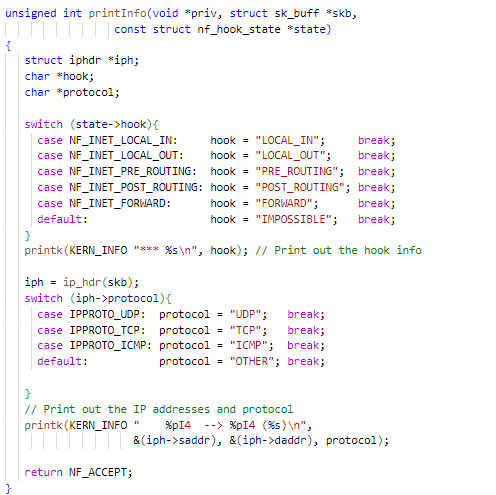
**Part 3:**

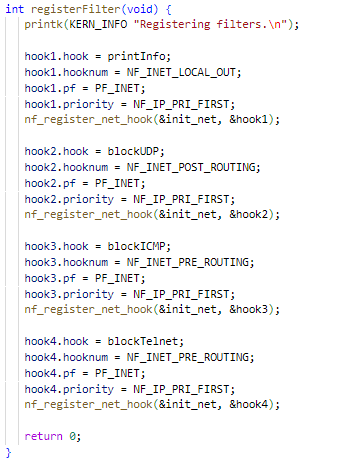
Code:

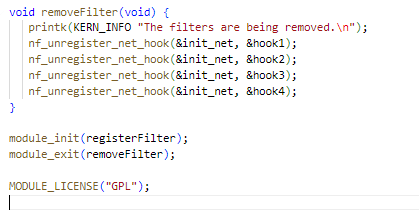




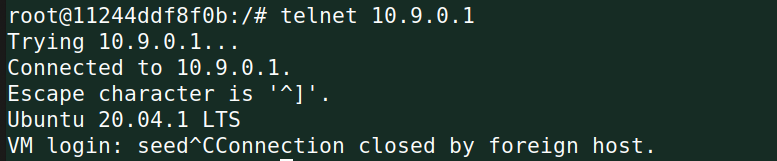






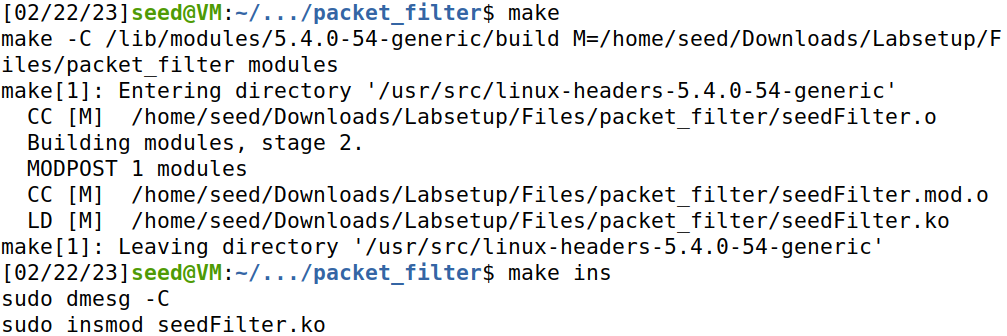


Lets try to telnet the VM before running the code:

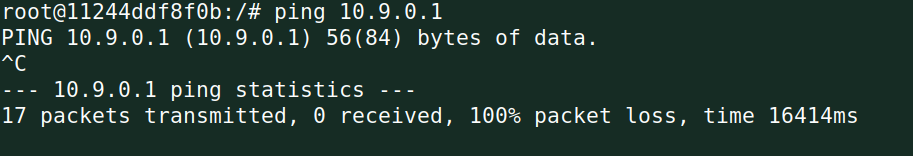


We can see that connection is formed.

Now, let’s install the module to the kernel and test:



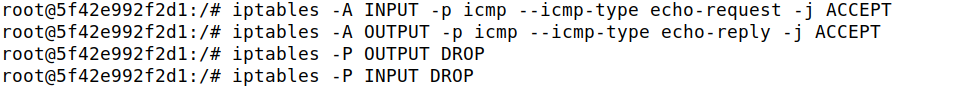
Now let’s try to ping VM again:



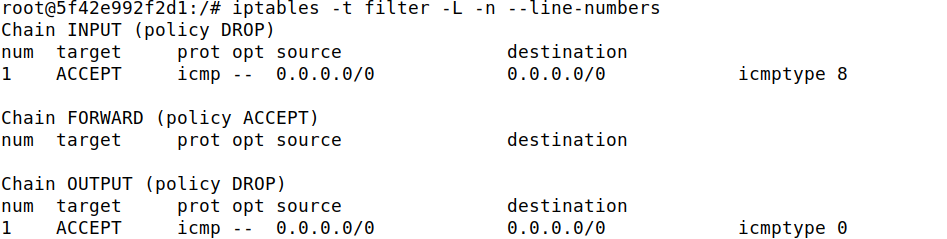
We can see that it failed.

**Task 2.A Protecting the router**

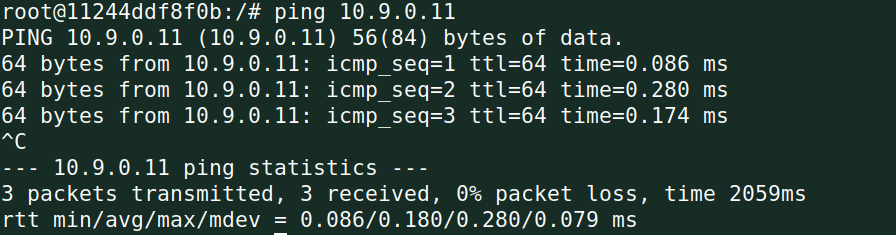
Running the iptables rules in the router:

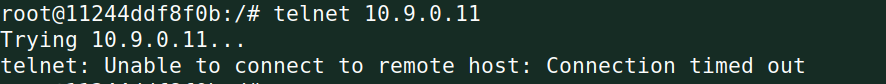


Filter table:



Now, we’ll try to ping and telnet:



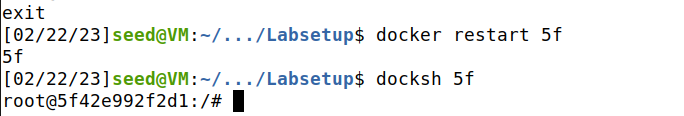


We can see that ping went through but not telnet.

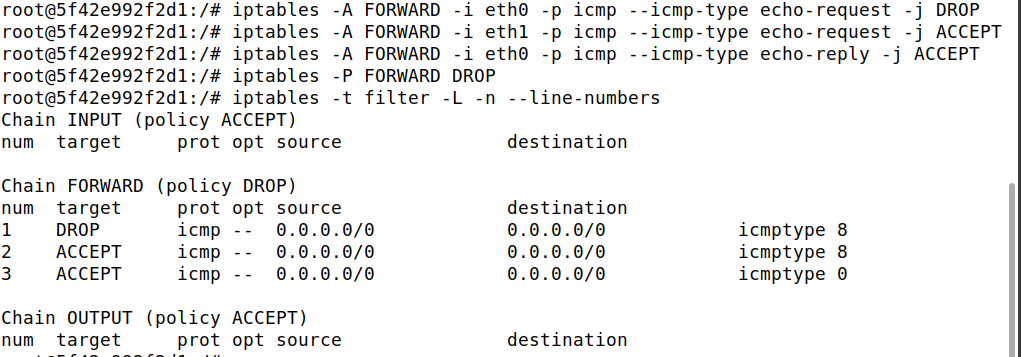
Because the rules given allow ICMP echo requests and echo answers across the firewall, which are frequently used for ping, we can see that ping was successful but not telnet in this case. As a result, ping will not be blocked by this firewall. All all incoming and outgoing traffic will be blocked by these rules, with the exception of ICMP echo requests and echo replies.

**Task 2.B Protecting the internal network**

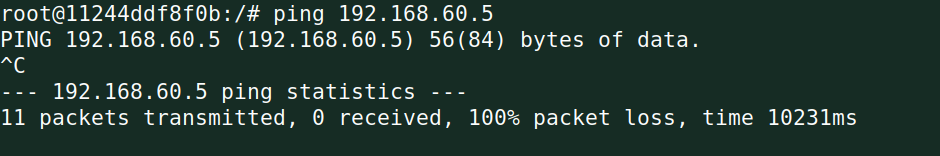
Reverting the changes done before:

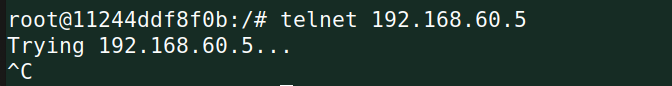


Running the iptables rules in the router:



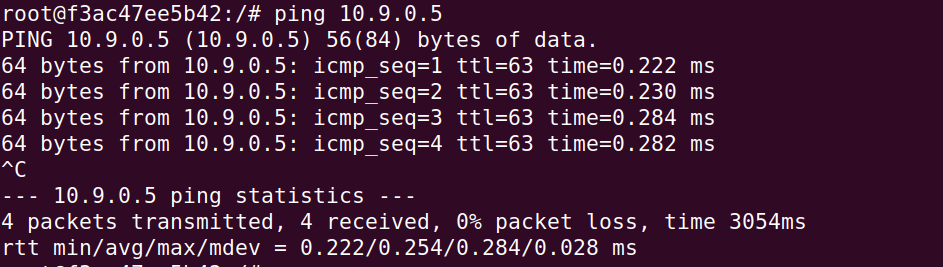
Using an outside host to ping the network. Let's ping and telnet Host A's IP address, 192.168.60.5





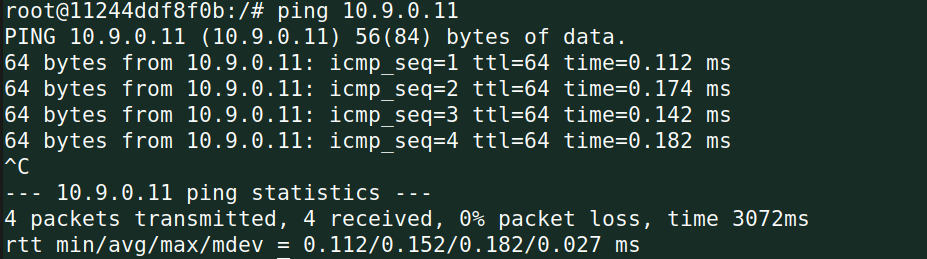
The Echo was a failure, as is evident. This is because external hosts are unable to ping internal hosts because we have restricted ICMP echo requests and replies. Moreover, Telnet fails, resulting in the blocking of all packets between internal and external hosts. Moreover, pinging an internal network from a remote computer is prohibited.

Now let’s ping External Host from internal hosts :



It works!

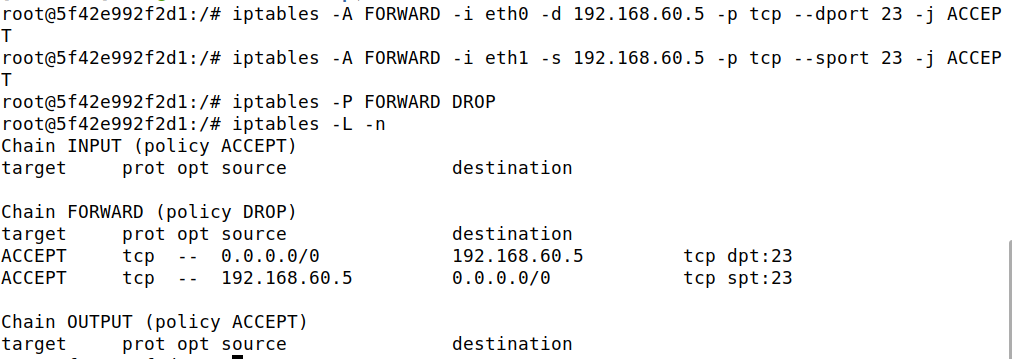
Now let’s try pinging the host from an Outside Host :



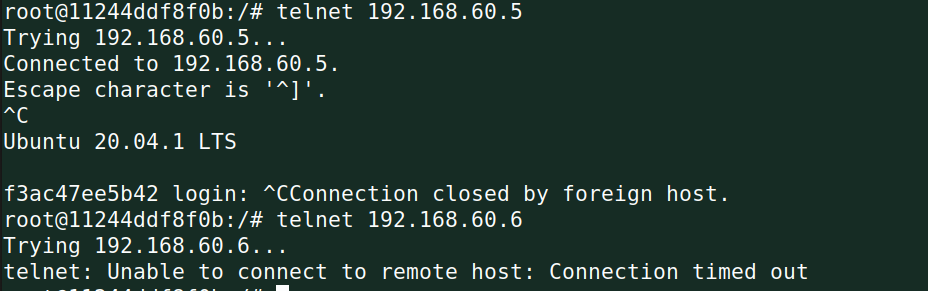
Because we didn't modify the router to stop receiving packets, it worked.

**Task 2.C Protecting Internal Servers**

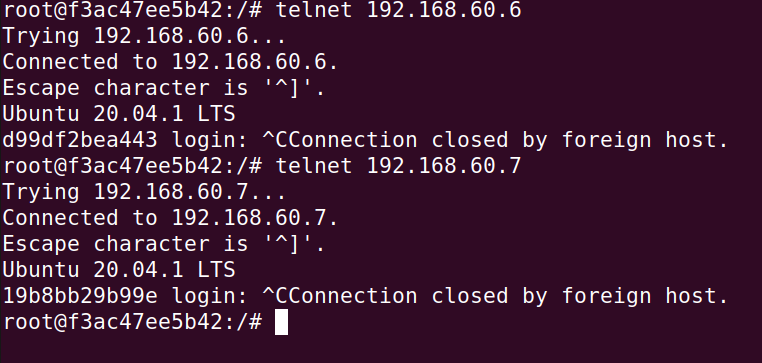
Running the iptables rules in the router:



Once we've given permission, let's try to telnet 192.168.60.5 from a new outside host (HOST A), it’ll work. But when we try to connect 192.168.60.6, it will not work.

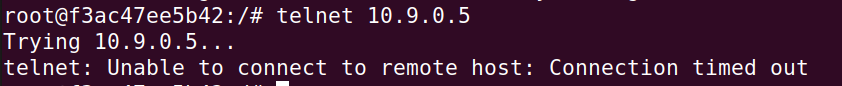


This means Outside servers can access telnet server only on 192.168.60.5 and no other internal hosts.

Now, let’s try to access Internal Server from an internal server(192.168.60.5):   


It works!

Now we try to connect Internal Server to External server:



It failed.

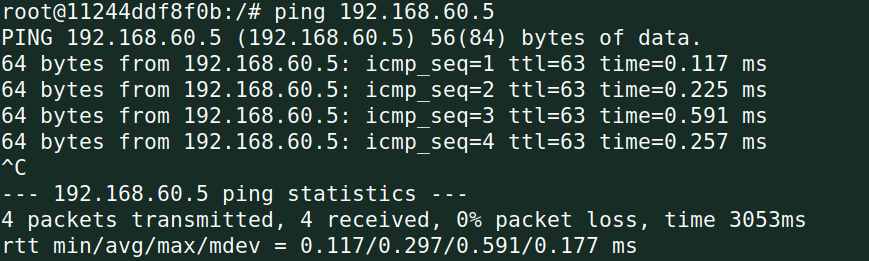
We can infer from the above findings that the internal servers are secure.

**Task 3: Connection Tracking and Stateful Firewall**

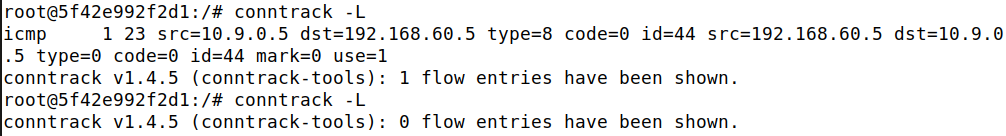
**Task3A: Experiment with connection tracking**

1. ICMP experiment

Sending out a ping from Host A to Host 1.



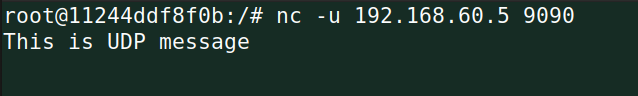
Now after exiting the ping we look at conntrack utility

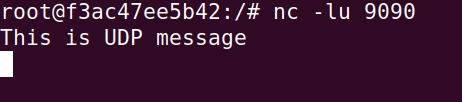


The connection will be stored for 23 seconds more. So, the ICMP connection is stored for **30 seconds**.

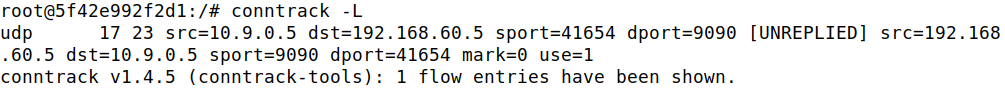
1. UDP experiment

A UDP server is started and a message is sent to it





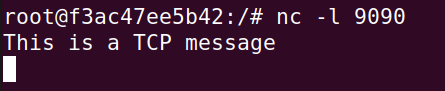
Now let us look at the conntrack command:

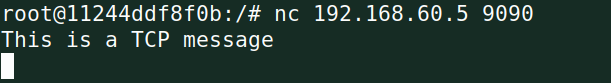


Here also we can see that the connection will be stored for 23 seconds more. A UDP connection is also stored for 30 seconds.

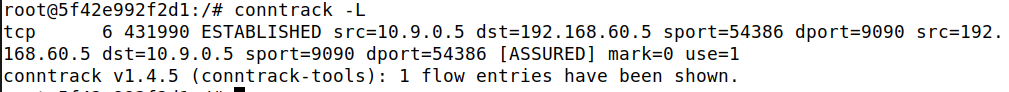
1. TCP experiment

A TCP server is started and a message is sent to it





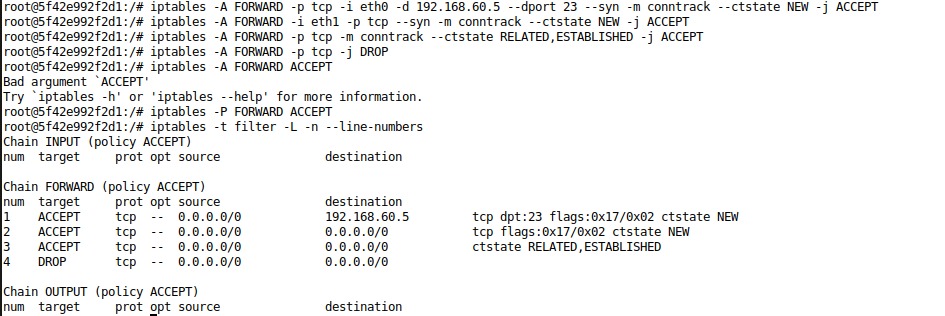
Now we can look at the conntrack command



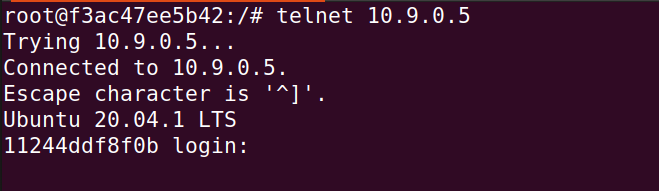
The established connection will be stored for a very long time (4331990 seconds). However, after the connection is terminated the timer reduces to 120 seconds. Hence, a TCP connection is tracked for 2 minutes.

**Task 3B: Setting up a stateful firewall**

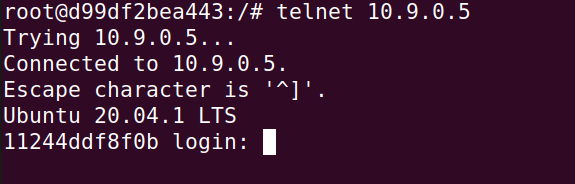
iptables commands:



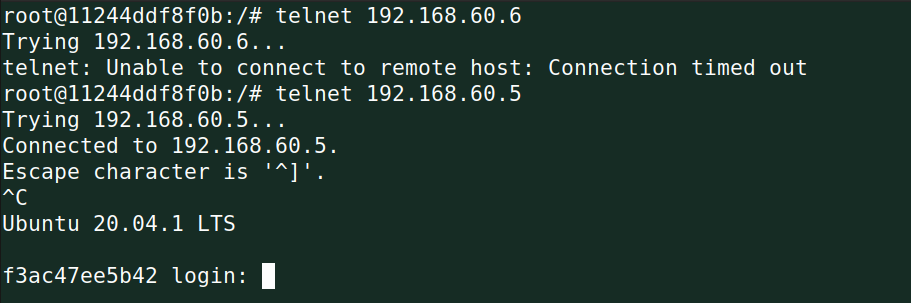
Telnet from host 1 to host A is successful (Internal to external)



Telnet from host 2 to host A is successful (internal to external)



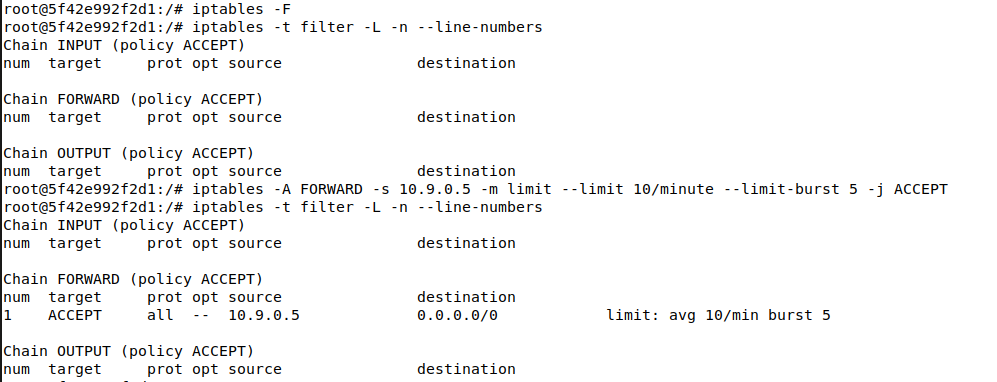
Telnet from host A to host 2 fails but from host A to host 1 is successful



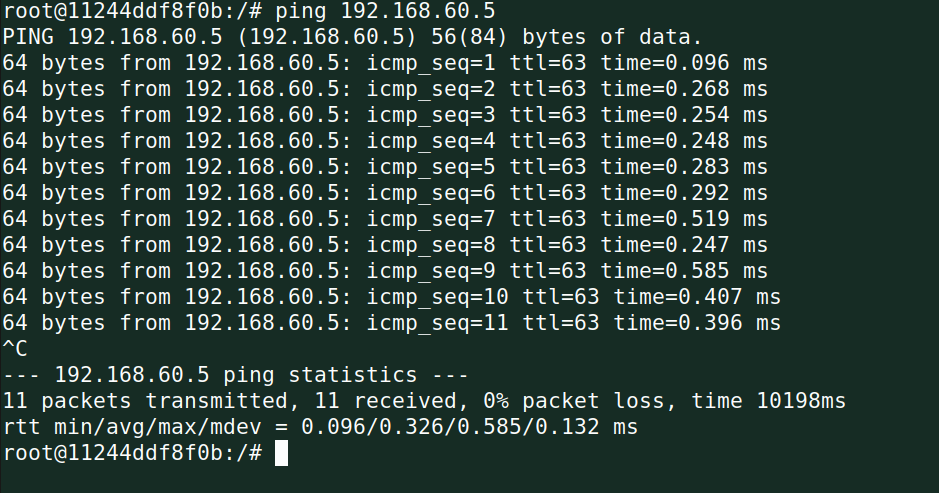
Without conntrack these rules can be implemented by blocking the SYN packet from external hosts so that external hosts cannot complete a connection. Conntrack uses resources to track connections, so it is more resource intensive. It is easier to write understandable rules about related connections using it.

**Task 4: Limiting Network Traffic**

1. Without 2nd rule

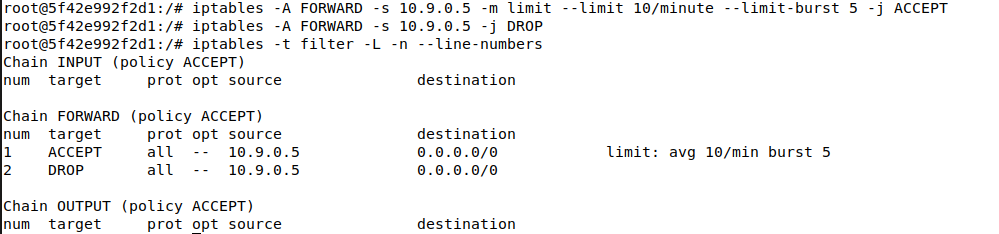


Sending ping from Host A to Host 1

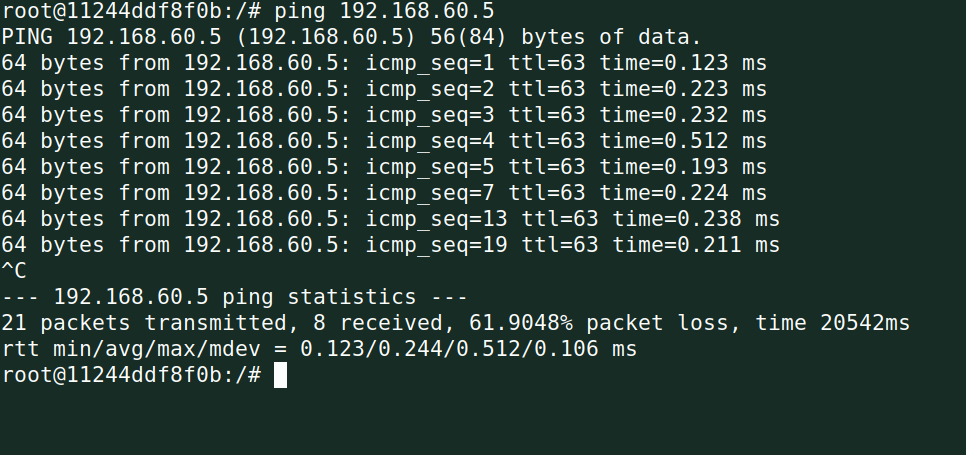


The rule applies first to allow the burst of 5 packets and limiting the rest of the packets based on 10 per minute. However, since the default policy of FORWARD chain is to allow all packets through, the packets which are rejected by the first rule are also forwarded to Host 1. Therefore, we cannot see any difference in the ping replies.

Now we will apply the second rule



Sending ping from Host A to Host 1

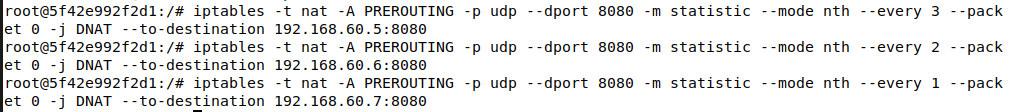


Now that the policy of FORWARD chain has been set to DROP, we see that after the first 5 packets burst limit, 10 packets are being allowed every minute. We can see this by looking at the sequence numbers after 5 (7,13,19 ...). Therefore, the network traffic has been successfully limited

**Task 5: Load Balancing**

1. Using Round Robin

Commands for load balancing:



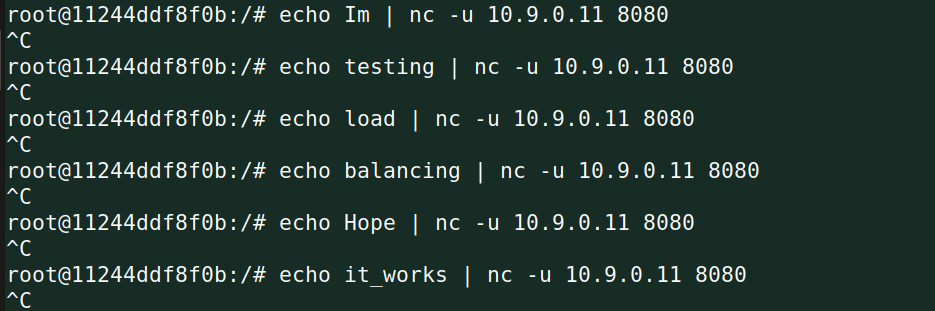
Packet 0 of every 3 will be sent to 192.168.60.5

Packet 0 of remaining 2 will be sent to 192.168.60.6

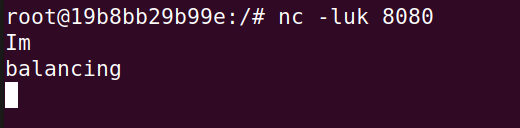
Packet 0 of all other packets will be sent to 192.168.60.7

This is how round robin load balancing has been achieved using statistic module.

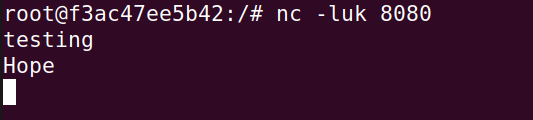
Let’s test the load balancer by sending a few packets to the external interface of the router (10.9.0.11)



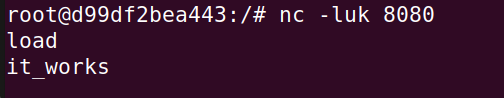
Server 192.168.60.5



Server 192.168.60.6



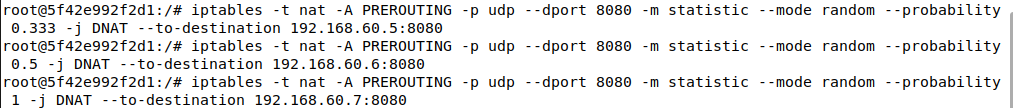
Server 192.168.60.7



The packets have been uniformly distributed amongst the 3 servers using nth mode (round robin) strategy.

1. Using random mode

Commands for load balancing:



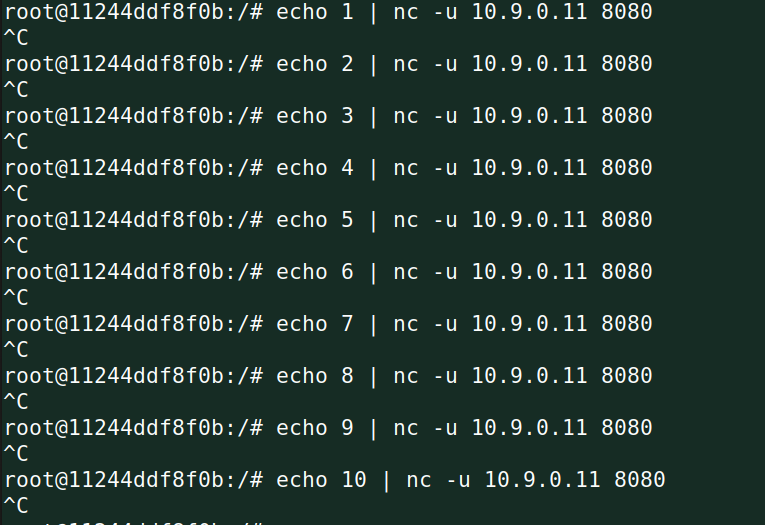
33% chance that the first packet is sent to 192.168.60.5

If not sent to 192.168.60.5, there is a 50% chance that it would be sent to 192.168.60.6 (Overall probability = 0.67\*0.5 = 33%).

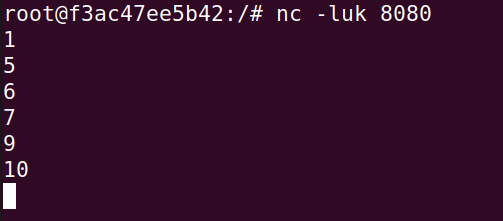
If not sent to 192.168.60.6 also, there is a 100% chance that it would be sent to 192.168.60.7 (Overall probability = 100 – 33 –33 = 33 %)

This load balancer will achieve close to equal load balancing over a large number of packets because it is based on chance instead of order.

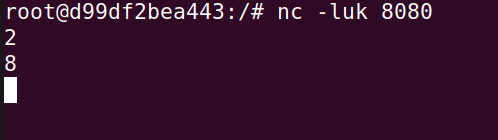
Let’s send out packets to test this load balancer



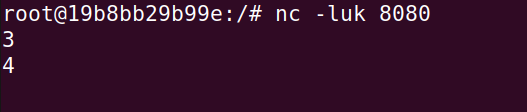
Server 192.168.60.5



Server 192.168.60.6



Server 192.168.60.7



As we can see in the above screenshots, the packets are not distributed uniformly. This is because it is based on probability. Over a larger number of packets, we will achieve close to equal distribution.