# Firewall Assignment

Assignment 8

Network Security (CS6903)

Devang Dubey (*cs21mtech14013*)

Kamal Shrestha (*cs21mtech16001*)

Nilesh Shivanand Kale (*cs21mtech11022*)

Pradhumn Kanase (*cs21mtech11018*)

May 2, 2022

# Table of contents

# Abstract

We have created a firewall system. A firewall monitors the incoming and outgoing traffic in the network  A firewall is a network security device that monitors incoming and outgoing network traffic and permits or blocks data packets based on a set of security rules.

First, we try to implement a simple firewall system with two network interface cards connecting to the external network (Internet) and the internal network which is supposed to be secured. In this system we hardcore a simple rules set. Then we improve the firewall to make it more advanced by extending the supported rule set up to layer 4 (including MAC, IPv4 IPv6, ICMP for IPv4/v6, TCP/UDP ) and not hard-coding the ruleset. Then we analyzed the performance of the implemented firewall, calculating the packet per second the implementation can handle for different scenarios. We have then tried to improve the performance of the firewall. In the final step, we have shown how to detect attacks such as DoS.

# Setup

Three VM has been set up in which VM1 is the internal network, VM2 is the firewall and VM3 is the external network that is connected to the internet. We have done the setup on 2 laptops.

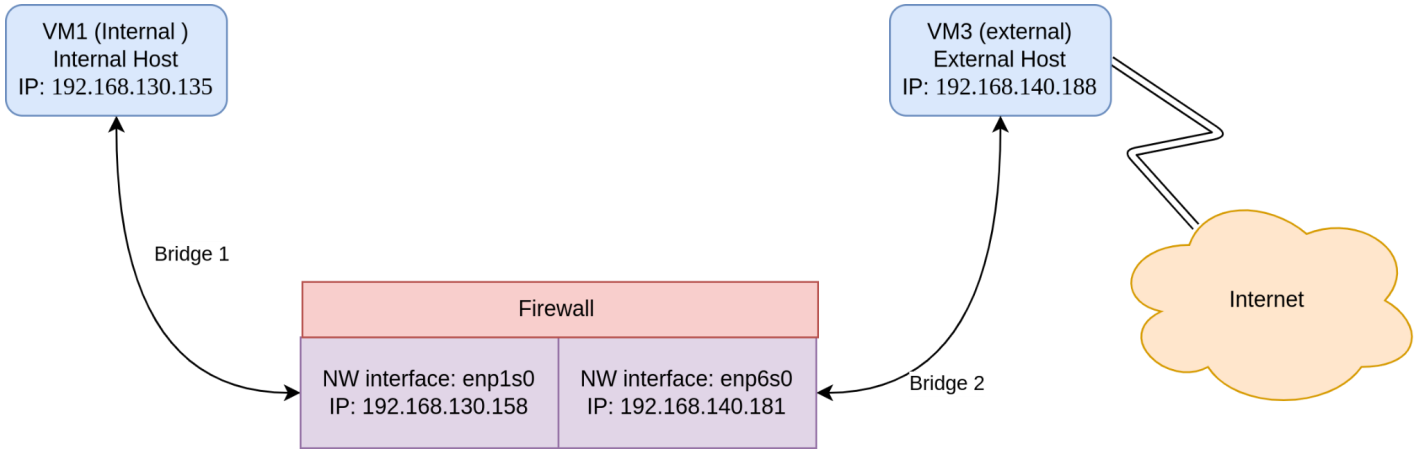



*Fig: Setup on laptop 1*

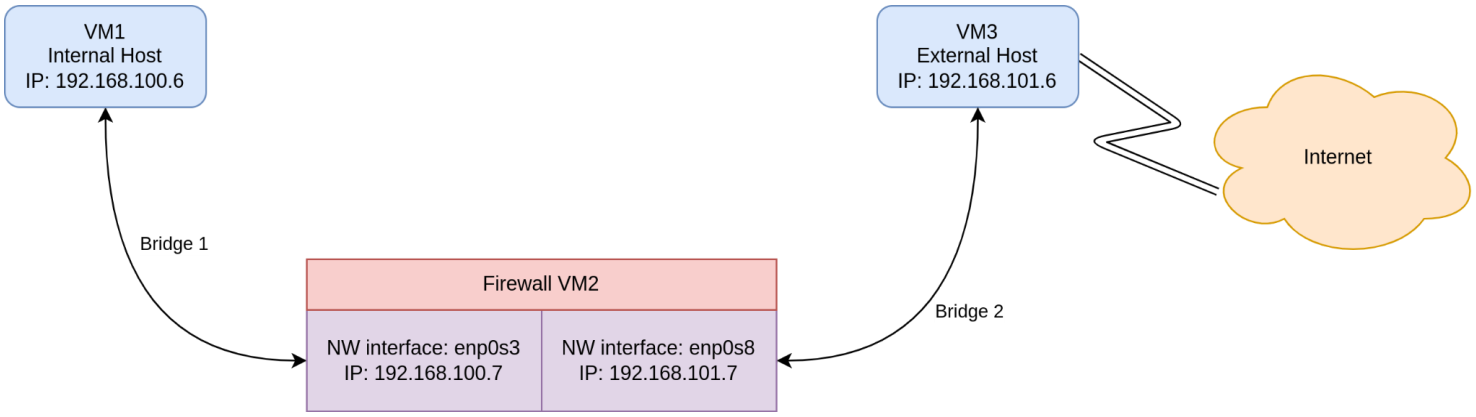



*Fig: Setup on laptop 2*

# Configuration

The following are the Ip addresses and the route table configurations of the VM's

## Laptop 1

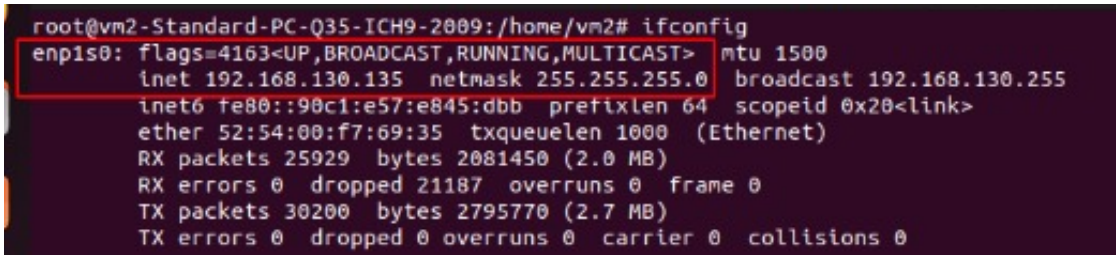



*Fig: VM1 config*

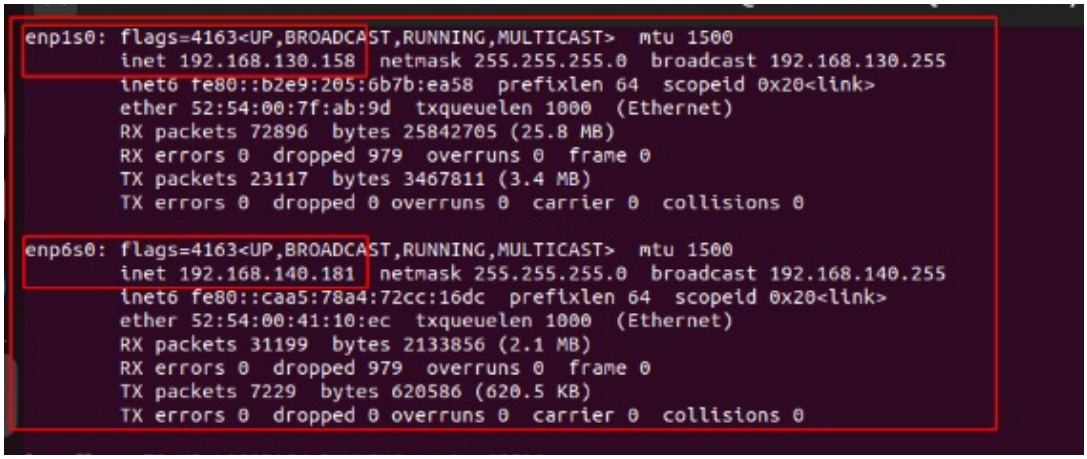



*Fig: VM2 config (Firewall)*

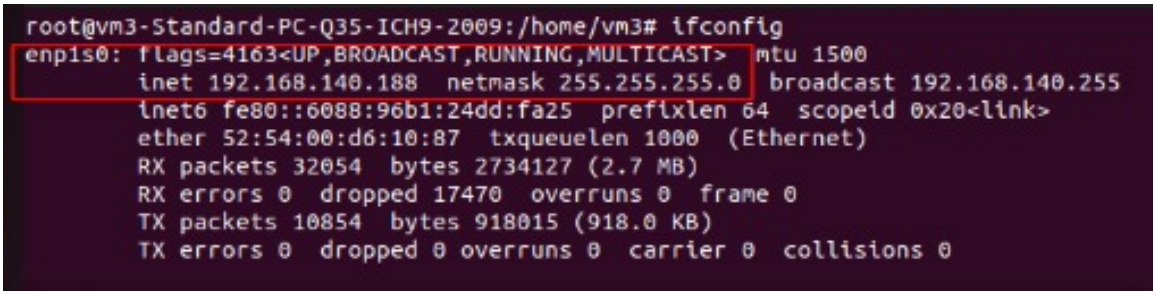



*Fig: VM3 config*

```
root@vm2-Standard-PC-Q35-ICH9-2009:/home/vm2# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.130.158 0.0.0.0         UG    0      0        0 enp1s0
0.0.0.0         192.168.130.1   0.0.0.0         UG    20100  0        0 enp1s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp1s0
192.168.130.0   0.0.0.0         255.255.255.0   U     100    0        0 enp1s0
root@vm2-Standard-PC-Q35-ICH9-2009:/home/vm2#
```

*Fig: VM1 route tables*

```
root@vm1-Standard-PC-Q35-ICH9-2009:/home/vm1# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.130.1   0.0.0.0         UG    100    0        0 enp1s0
0.0.0.0         192.168.140.1   0.0.0.0         UG    101    0        0 enp6s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp1s0
192.168.130.0   0.0.0.0         255.255.255.0   U     100    0        0 enp1s0
192.168.140.0   0.0.0.0         255.255.255.0   U     101    0        0 enp6s0
root@vm1-Standard-PC-Q35-ICH9-2009:/home/vm1# echo 0 > /proc/sys/net/ipv4/ip_forward
```

*Fig: VM2 route table (Firewall)*

```
root@vm3-Standard-PC-Q35-ICH9-2009:/home/vm3# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.140.1   0.0.0.0         UG    100    0        0 enp1s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp1s0
192.168.130.0   192.168.140.181 255.255.255.0   UG    0      0        0 enp1s0
root@vm3-Standard-PC-Q35-ICH9-2009:/home/vm3# iptables -t nat -A POSTROUTING -o enp1s0 -j MASQUERADE
```

*Fig: VM3 route table*

## Laptop 2

```
vm1@vm1-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.6  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::5366:8c4a:246:14f  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:ab:ba:9d  txqueuelen 1000  (Ethernet)
        RX packets 26660  bytes 38182592 (38.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9251  bytes 784607 (784.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

*Fig: VM1 config*

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.7  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::166e:594d:2ec2:a8c0  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:7f:36:15  txqueuelen 1000  (Ethernet)
        RX packets 9383  bytes 841165 (841.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27083  bytes 61086567 (61.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.101.7  netmask 255.255.255.0  broadcast 192.168.101.255
        inet6 fe80::9e23:8be6:bcda:59ac  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:30:76:f2  txqueuelen 1000  (Ethernet)
        RX packets 124781  bytes 179598422 (179.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 29928  bytes 3266059 (3.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

*Fig: VM2 config (Firewall)*

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.101.6  netmask 255.255.255.0  broadcast 192.168.101.255
        inet6 fe80::92a4:fcad:a734:4c43  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:72:b6:ab  txqueuelen 1000  (Ethernet)
        RX packets 155826  bytes 183891554 (183.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 154738  bytes 282492175 (282.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

*Fig: VM 3 config (external)*

## Testing

To check if the VM's are working properly we are going to ping Google from VM1

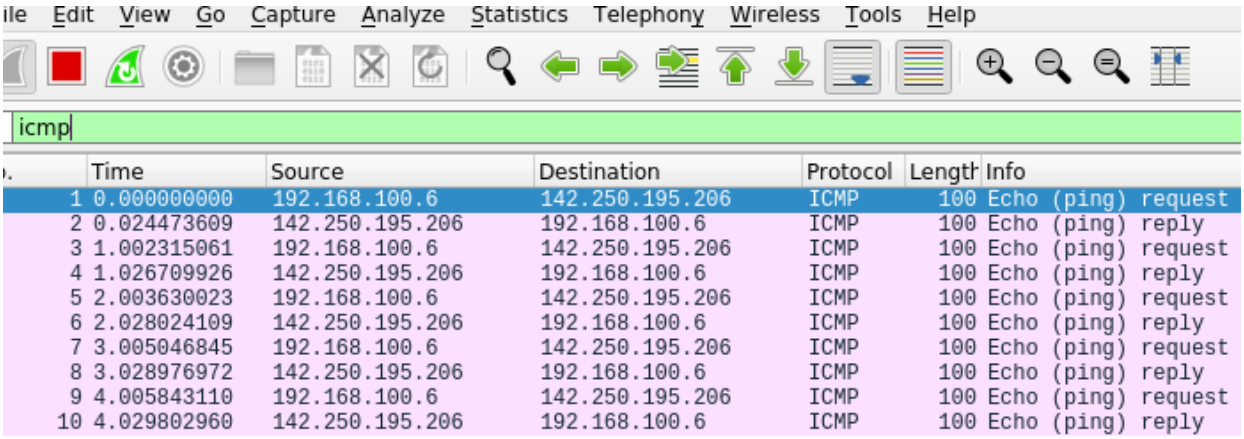
```
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.195.206
```

*Fig: Google ip add lookup*

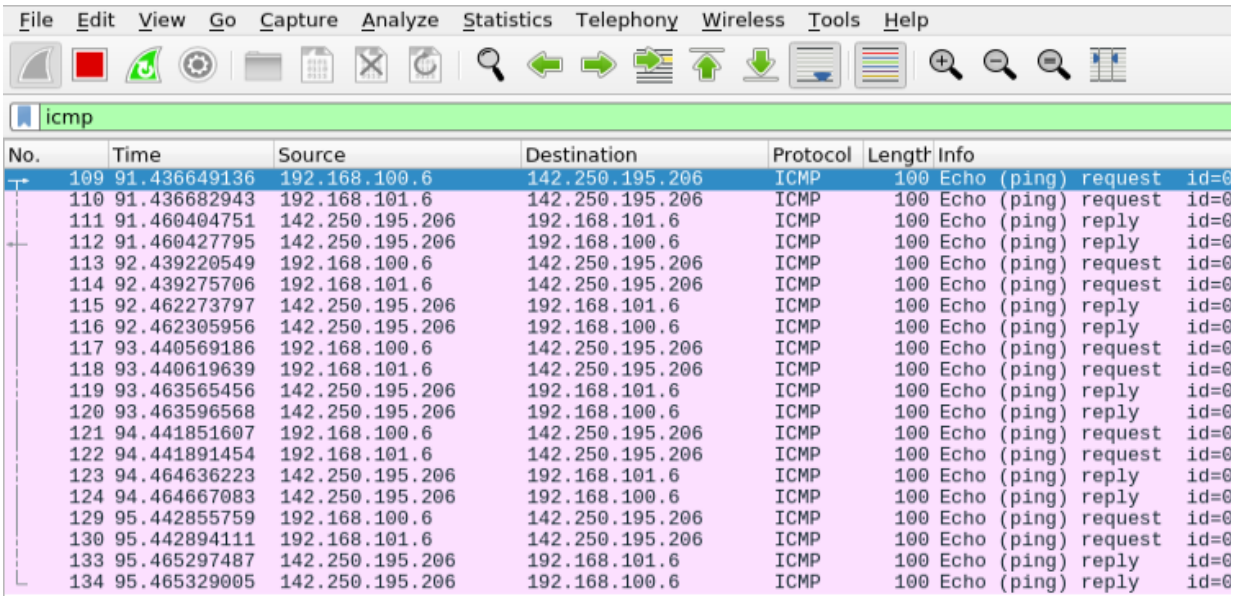
```
vm1@vm1-VirtualBox:~$ ping 142.250.195.206
PING 142.250.195.206 (142.250.195.206) 56(84) bytes of data.
64 bytes from 142.250.195.206: icmp_seq=1 ttl=53 time=24.5 ms
64 bytes from 142.250.195.206: icmp_seq=2 ttl=53 time=24.4 ms
64 bytes from 142.250.195.206: icmp_seq=3 ttl=53 time=24.4 ms
64 bytes from 142.250.195.206: icmp_seq=4 ttl=53 time=24.0 ms
64 bytes from 142.250.195.206: icmp_seq=5 ttl=53 time=24.0 ms
^C
--- 142.250.195.206 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 24.030/24.280/24.486/0.205 ms
```

*Fig: ping to google using VM1 reply incoming*




| | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 2 | 0.024473609 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 3 | 1.002315061 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 4 | 1.026709926 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 5 | 2.003630023 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 6 | 2.028024109 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 7 | 3.005046845 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 8 | 3.028976972 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 9 | 4.005843110 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 10 | 4.029802960 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |

*Fig: Wireshark capture for above at VM1*




| No. | Time | Source | Destination | Protocol | Length | Info | |
|---|---|---|---|---|---|---|---|
| 109 | 91.436649136 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 110 | 91.436682943 | 192.168.101.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 111 | 91.460404751 | 142.250.195.206 | 192.168.101.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 112 | 91.460427795 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 113 | 92.439220549 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 114 | 92.439275706 | 192.168.101.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 115 | 92.462273797 | 142.250.195.206 | 192.168.101.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 116 | 92.462305956 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 117 | 93.440569186 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 118 | 93.440619639 | 192.168.101.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 119 | 93.463565456 | 142.250.195.206 | 192.168.101.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 120 | 93.463596568 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 121 | 94.441851607 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 122 | 94.441891454 | 192.168.101.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 123 | 94.464636223 | 142.250.195.206 | 192.168.101.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 124 | 94.464667083 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 129 | 95.442855759 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 130 | 95.442894111 | 192.168.101.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request | id=0 |
| 133 | 95.465297487 | 142.250.195.206 | 192.168.101.6 | ICMP | 100 | Echo (ping) reply | id=0 |
| 134 | 95.465329005 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply | id=0 |

*Fig: Wireshark capture for above at firewall*

# Task 1

## Creating a simple Firewall using Socket Programming

In this part, we have implemented a simple firewall. This firewall works at layer 3 i.e IP layer. In this, there is a precoded list of IP addresses that can be blocked. By **default, it allows all the packets**. It checks if the IP address is in the blocked list, if it is there it makes allow = False.

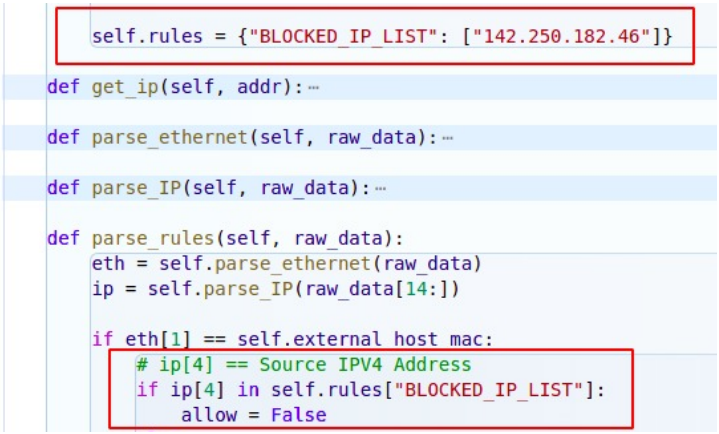



*Fig: Hardcoded IP addresses*

Command for the simple firewall.

**Command:**

```
python3 firewall.py -s
```

As we can see **142.250.195.206** is the blocked ip address. We have run a ping command from the internal host to **142.250.195.206** i.e to the Google server. The request of the ping request is passed by the firewall but the reply from the Google server is not allowed by the firewall. This happens because the Google IP address is on the blocking list.

```
vm1@vm1-VirtualBox:~$ ping 142.250.195.206
PING 142.250.195.206 (142.250.195.206) 56(84) bytes of data.
^C
--- 142.250.195.206 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8194ms
```

*Fig: Tried ping command again from internal host VM1*

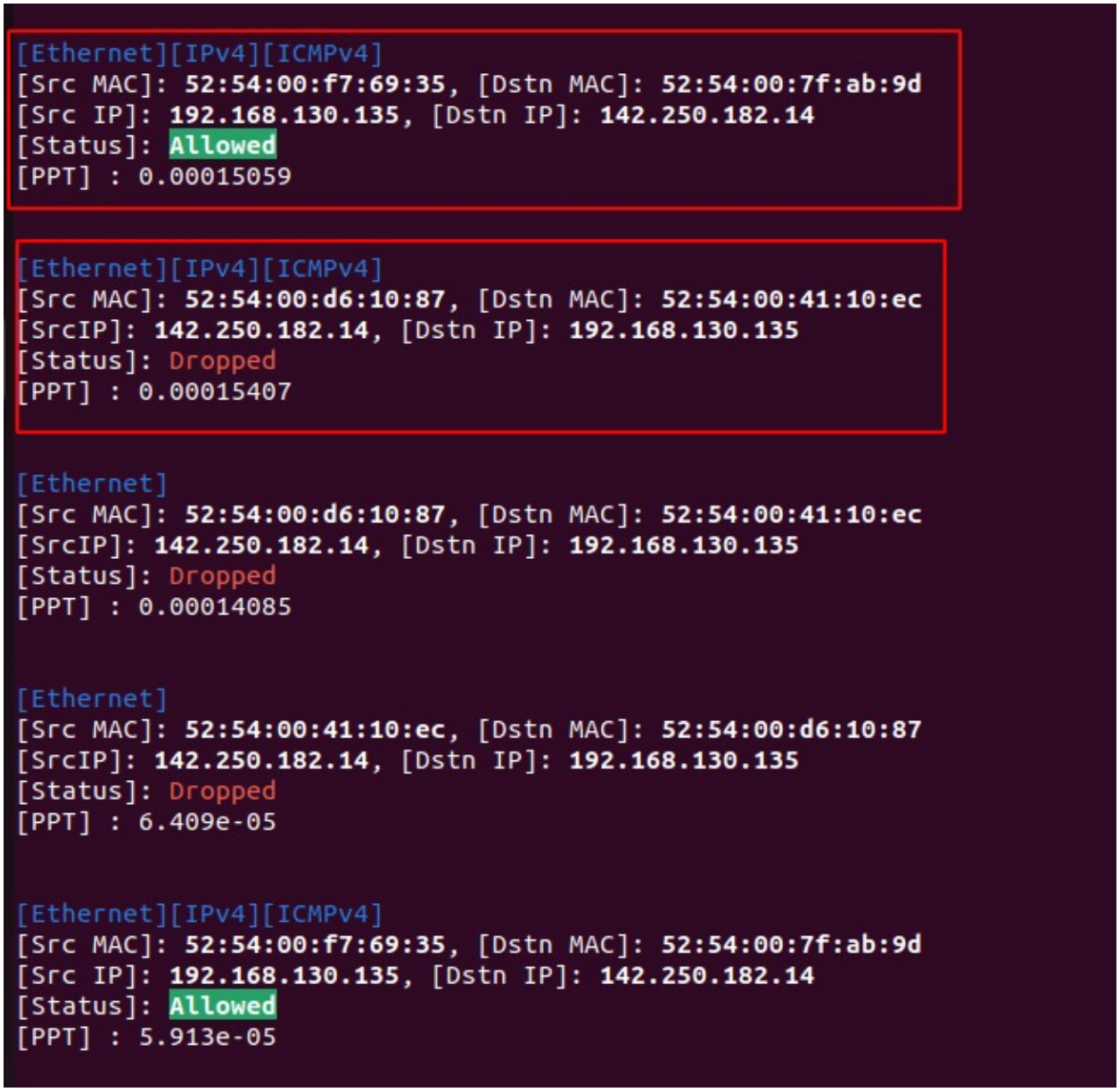
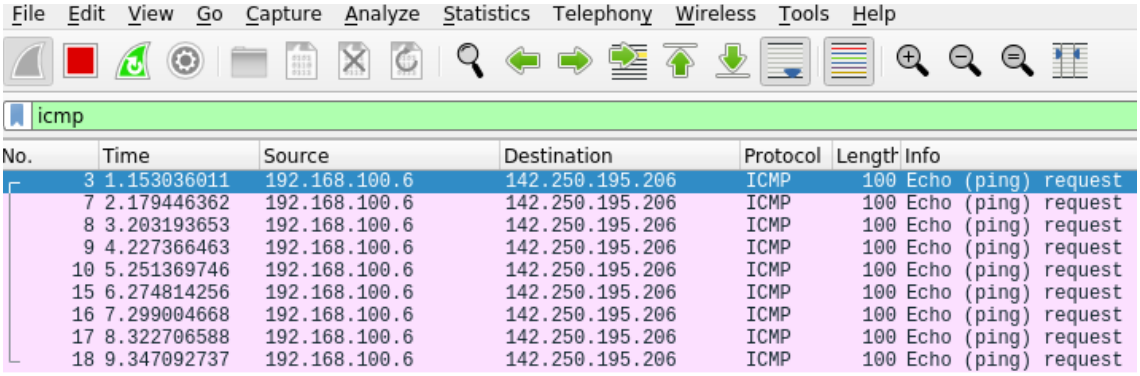



*Fig: Packets allowed and discarded by the blocking process (ScreenShot from Laptop 1)*

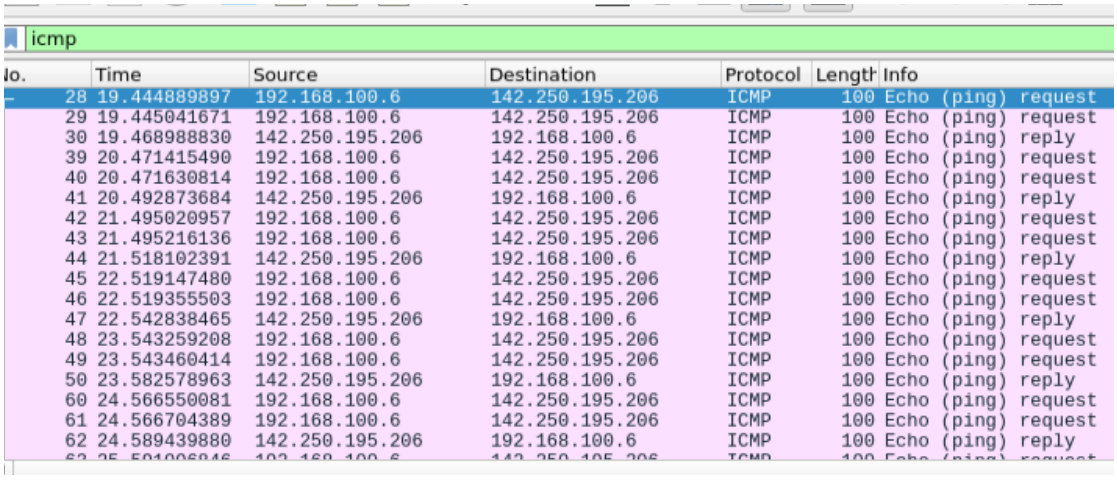
**PCAP analysis:**

The PCAP on VM1 shows that there is no reply packet from the Google server (142.250.195.206) since the IP has been blocked by the firewall in the source IP field.




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 3 | 1.153036011 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 7 | 2.179446362 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 8 | 3.203193653 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 9 | 4.227366463 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 10 | 5.251369746 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 15 | 6.274814256 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 16 | 7.299004668 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 17 | 8.322706588 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 18 | 9.347092737 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |

*Fig: Wireshrk capture at VM1 shows that reply is blocked by firewall*




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 28 | 19.444889897 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 29 | 19.445041671 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 30 | 19.468988830 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 39 | 20.471415490 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 40 | 20.471630814 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 41 | 20.492873684 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 42 | 21.495020957 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 43 | 21.495216136 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 44 | 21.518102391 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 45 | 22.519147480 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 46 | 22.519355503 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 47 | 22.542838465 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 48 | 23.543259208 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 49 | 23.543460414 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 50 | 23.582578963 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |
| 60 | 24.566550081 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 61 | 24.566704389 | 192.168.100.6 | 142.250.195.206 | ICMP | 100 | Echo (ping) request |
| 62 | 24.589439880 | 142.250.195.206 | 192.168.100.6 | ICMP | 100 | Echo (ping) reply |

*Fig: Reply received at firewall but is not forwarded, it is blocked.*

# Task 2

## Extending the ruleset and its operation on the Firewall

In this task, we need to add additional rules to improve the functionality of the firewall. We are filtering at layer 2 [Ether], layer 3 [IP], and layer 4 [TCP, UDP]. We are doing dynamic rule management by addition, deletion, and updating the rules in the firewall. This is done by storing the rules of the firewall in a json file. The json file is edited in order to update the rules.

**Command:**

```
python3 firewall.py
```

## How our Firewall Works

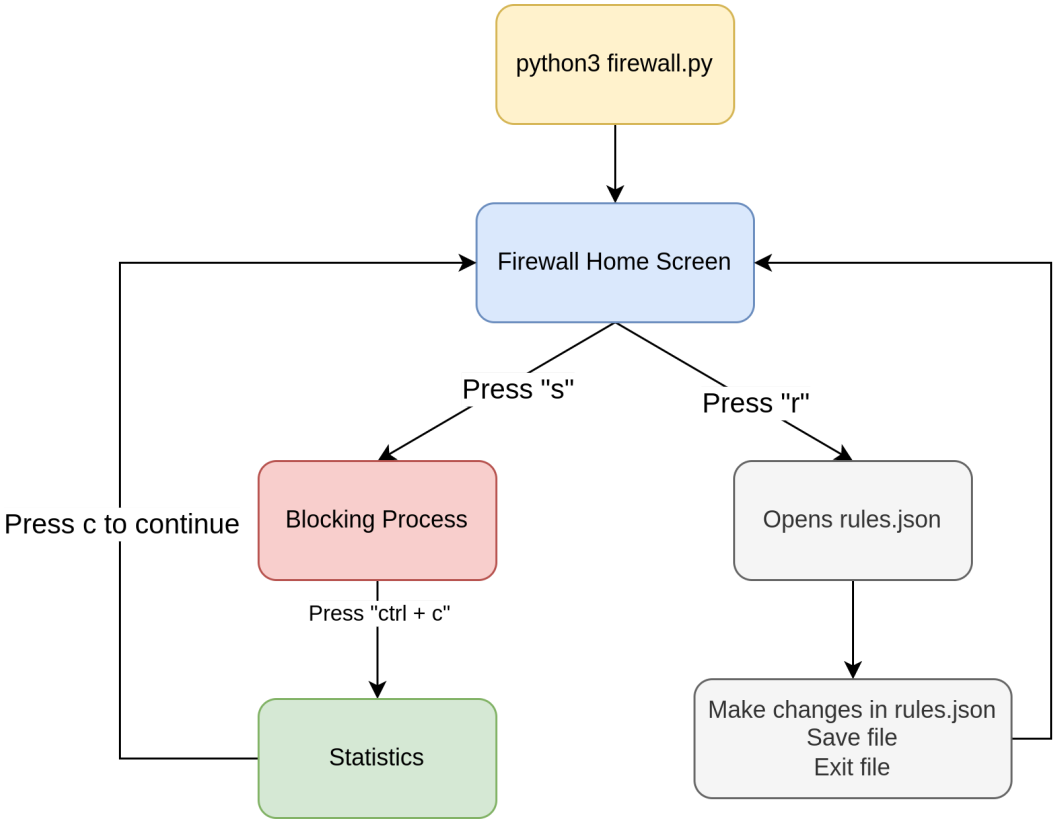



*Fig: Firewall CFG*

***Firewall Home Screen :*** Home Screen

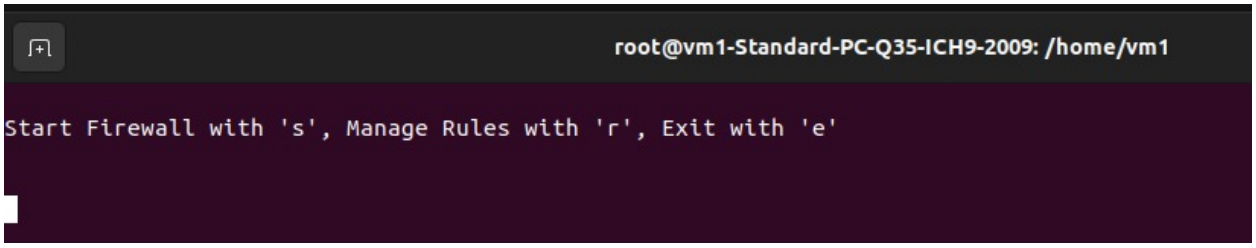



*Fig: Home screen*

***Blocking Process :***

Blocking process looks at every single packet and decides whether it can be allowed or it needs to be blocked based on the rules mentioned in the rules.json. In our Firewall, by **default**, it **blocks all the packets**. If we want to allow any type of packet we need to mention it in rules.json

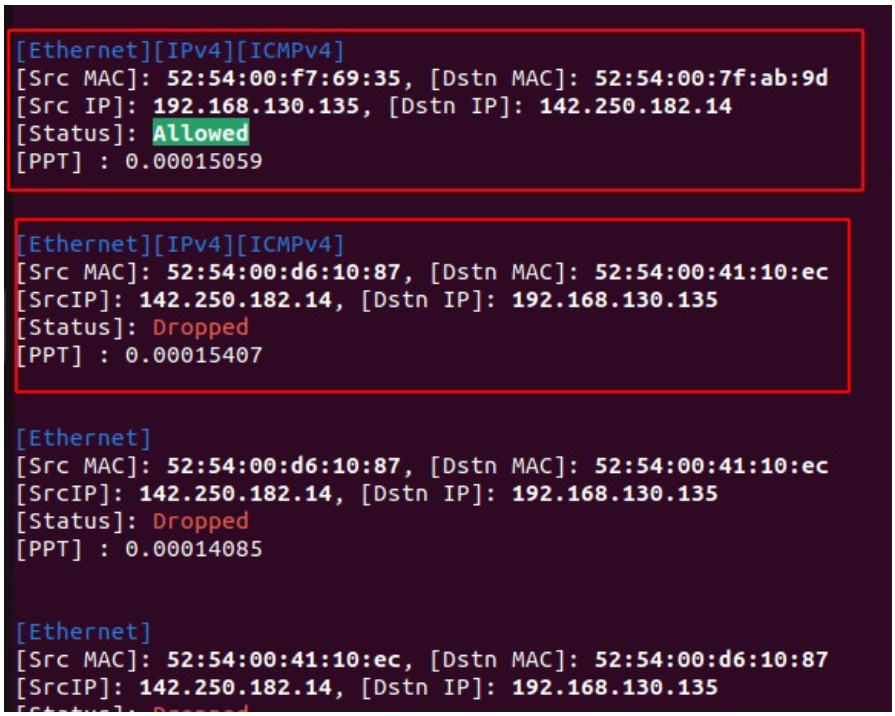



*Fig: blocking process running*

***Statistics :***

When the Firewall run is complete the firewall program returns the Statistics page which returns the number of allowed and dropped packets. It also returns the average time taken to process each packet.

```
Firewall Capture Statistics

No of packets allowed : 78

No of packets dropped : 132

Mean Packet Processing Time : 0.0001248

No of rules in system : 3
Press 'c' to continue...
```

*Fig: Statistics page*

***rules.json:***

rules.json file is the file from which we read the rules which are used in te blocking process. We can also add, delete or update rules in this.


*Fig: rules.json file*

When *"s"* is pressed the blocking process is started. When *"r"* is pressed then the rules.json file is opened in which we can add, remove and update the rules. After every run of the firewall the program prints the statistics of the code.

# Types of Rules

## L2: Mac Layer Filtering Rule

Filtering packets based on Mac address

```
"L2": [
    {
        "rule_id": 168,
        "dstn_mac": "52:54:00:f7:69:35",
        "rule": "Allow"
    }
],
```

*Fig: Allowed packet rule*

## IP Packet Filtering Rule

Filtering packets based on IP rule

```
"L3V4": [
    {
        "rule_id": 57,
        "ipv4protocol": 3,
        "rule": "Allow"
    }
],
```

*Fig: Allowed packet rule*

## UDP Packet filtering Rule

Filtering packets based on IP rule

```
"L4UDP": [
    {
        "rule_id": 505,
        "udpsrc_port": 403,
        "udpdest_port": 9876,
        "rule": "Allow"
    }
]
```

*Fig: Allowed packet rule*

## TCP Packet filtering Rule

Filtering packets based on IP rule


*Fig: Allowed packet rule*

# Task 3

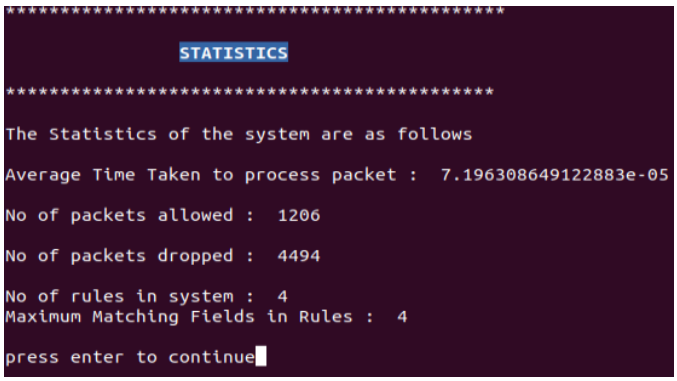
## Performance examination and improvement

In this part, we have tried to analyze the system. We have used the metric processing time per packet for analyzing the performance of the system. We have increased the number of rules and tried to observe how this affects the performance of the system.

We are generating traffic on VM3 using *"generate_traffic.py"* which uses the *"nping"* command. This file generates mixed traffic (i.e TCP/UDP/IP/IP) continuously and sends it to VM1.

***Command:*** *python3 traffic_generator.py g*
This command will generate a total 1000 packets and send to the destination IP which is hardcoded in the file itself .

At the same time, we also wrote code to create rules using random parameters.
***Command*** : *python3 rules_generator.py rm No_Of_Rules_In_Each_Category*.
No_Of_Rules_In_Each_Category is an integer value stating how many rules will be created for each traffic category like TCP, UDP etc.



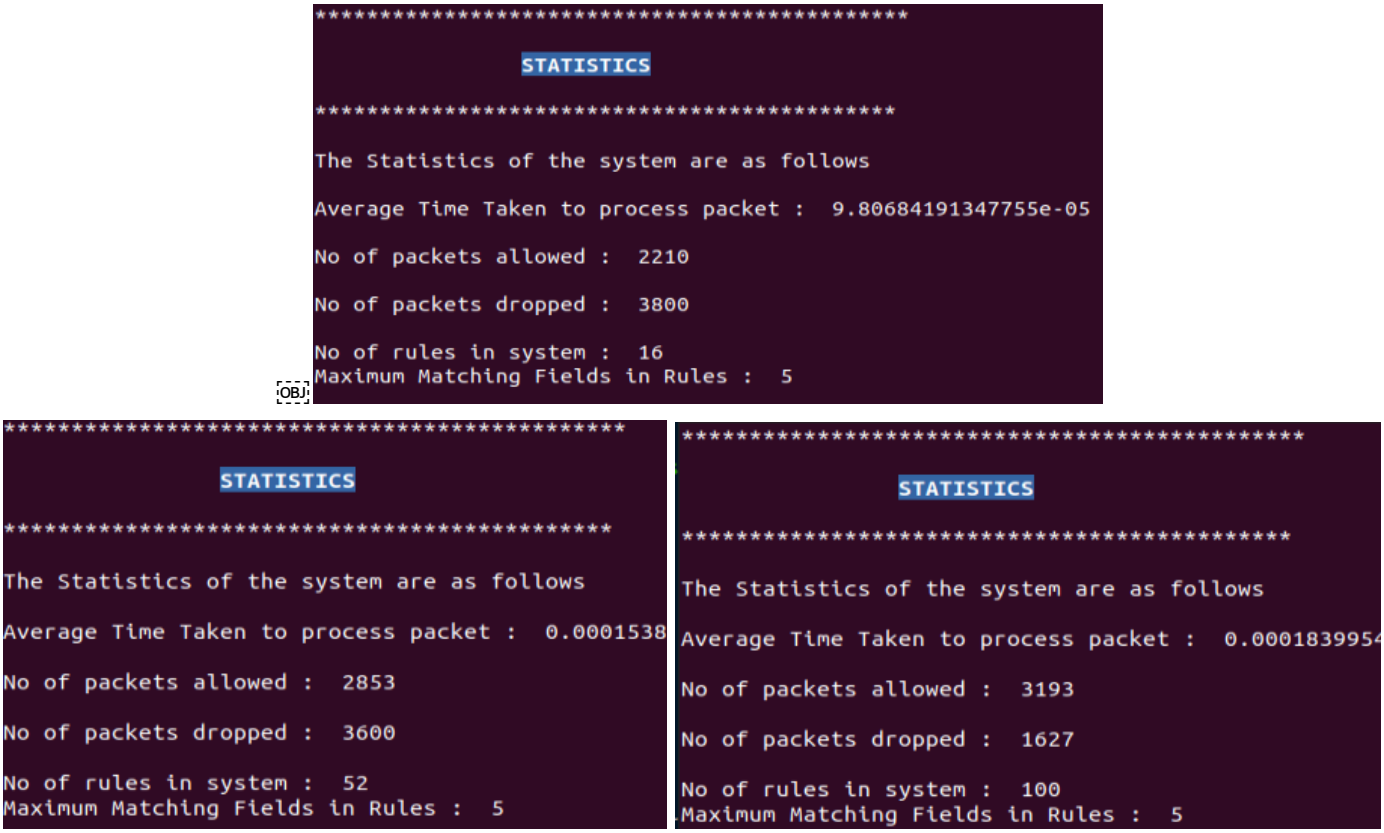




*Fig: Statistics*

| Statistics | |
|---|---|
| **No of Rules** | **Processing time per packet** |
| 4 | 0.00007196 |
| 8 | 0.00008349 |
| 16 | 0.00009806 |
| 50 | 0.0001538 |
| 100 | 0.0001839 |

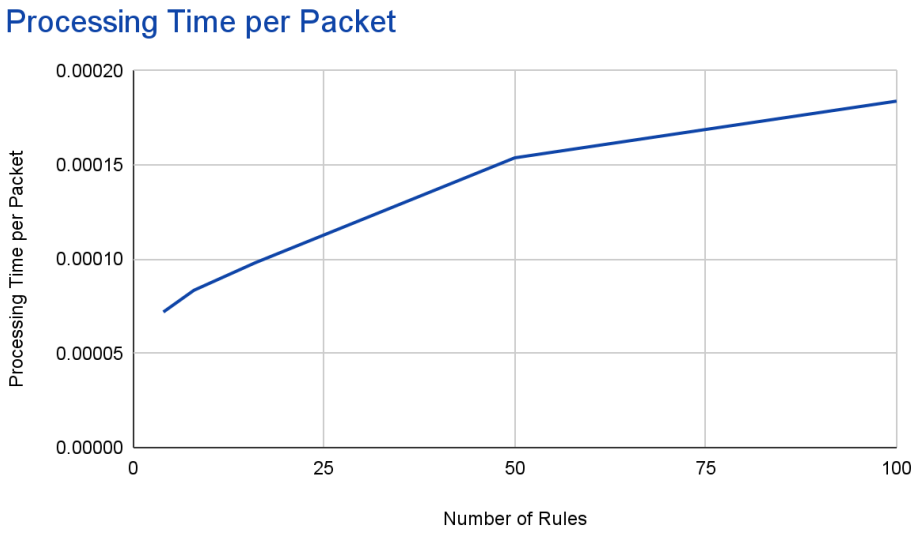



*Fig: Analysis of statistics*

We can see clearly as the number of Rules is increased the the time taken for processing the packet also increases.

# Task 4 - Part B:

Detecting attacks in the network using Firewall

## DoS Attack

A Denial-of-Service (DoS) attack is one that attempts to bring a machine or network to a halt, rendering it unreachable to its intended users. DoS attacks work by inundating the target with traffic or delivering it information that causes it to crash. The DoS attack deprives genuine users, such as employees, of the service or resource they expected in both cases.

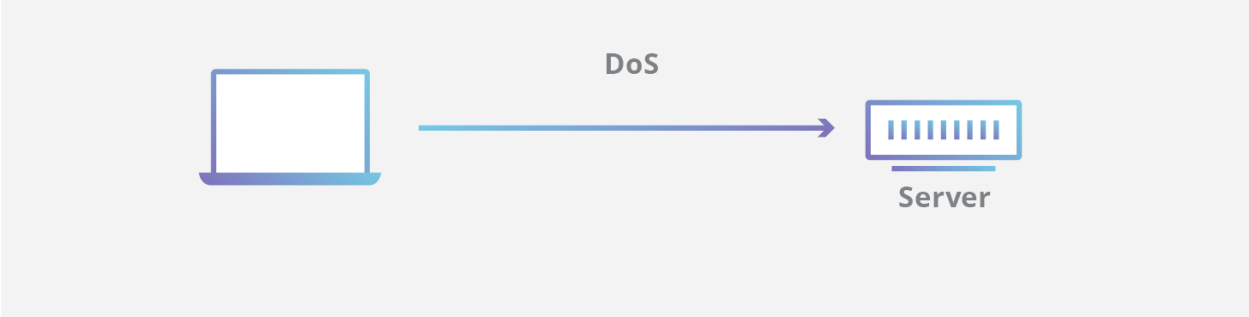



Fig: DoS Attack (ref: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/)

## Prevention using Firewall

In this task we're going to detect the Dos attack that we are performing on VM1. In order to prevent the DoS attack we have tried to put a cap on the number of packets incoming from one IP. If a lot of IP packets are incoming from one IP then the firewall system detects that there is some DoS attack and then blocks that particular IP. Here is our implementation of the DoS attack:

```
root@vm2-Standard-PC-Q35-ICH9-2009:/home/vm2# ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.130.135  netmask 255.255.255.0  broadcast 192.168.130.255
        inet6 fe80::90c1:e57:e845:dbb  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:f7:69:35  txqueuelen 1000  (Ethernet)
        RX packets 21288  bytes 1784205 (1.7 MB)
        RX errors 0  dropped 17582  overruns 0  frame 0
        TX packets 28359  bytes 2675492 (2.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 822  bytes 76500 (76.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 822  bytes 76500 (76.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@vm2-Standard-PC-Q35-ICH9-2009:/home/vm2# nping -c 200 --delay 20ms --tcp -p 9876 -S
192.168.130.135 192.168.140.188
```

Fig: Internal VM1

In this the Internal machine i.e. VM1 (192.168.130.135) is generating ping messages and is continuously sending it to the external machine i.e. VM3 (192.168.140.188). This ping is sending 200 packets and they are sent periodically at a delay of 20ms from each other.

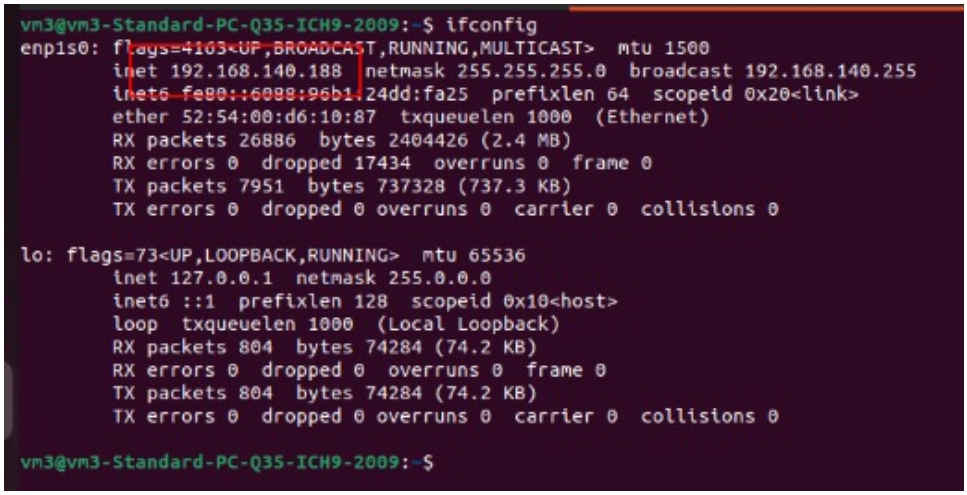



*Fig: IP address of external machine VM3*


*Fig: Command for DoS attack prevention in VM2 (Firewall)*

What this command does is it enables the DoS prevention mechanism and if the number of packets from an IP address is more than 100 then the Firewall blocks the IP address.


*Fig: Rules description*

As we can see in the Fig above the rule with rule_id: 100, tells to allow the packets with which have destination address as **192.168.140.188**

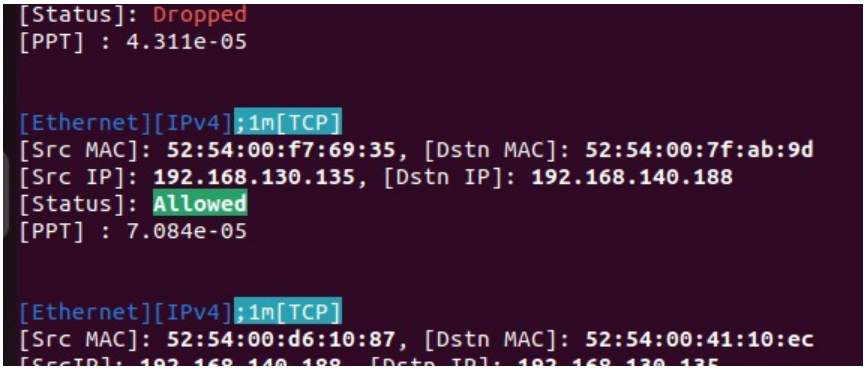



*Fig: Firewall ON, DoS is not detected yet*

Since the number of Packets is not more than 100 DoS is not detected yet.

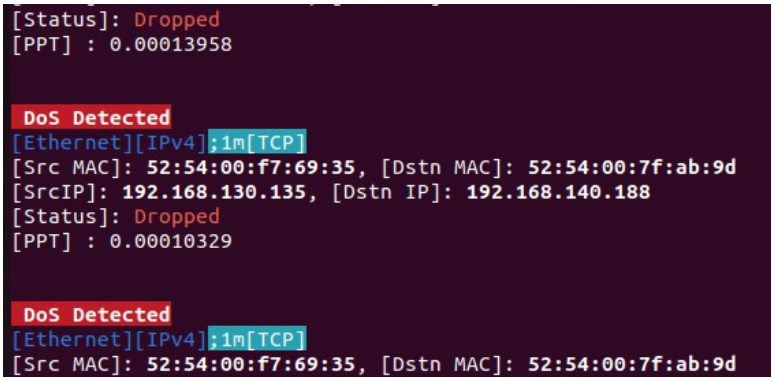



*Fig: Firewall ON, DoS is Detected*

In the above Figure the number of packets has crossed 100 mark hence the Firewall has started to block the packets.

In the figure below, we can see that once the number of packets reaches 100 then the IP address of VM1 **(192.168.130.135)** is blocked and no more packets are incoming from it. Thus the number of allowed packets plateaus after a point.

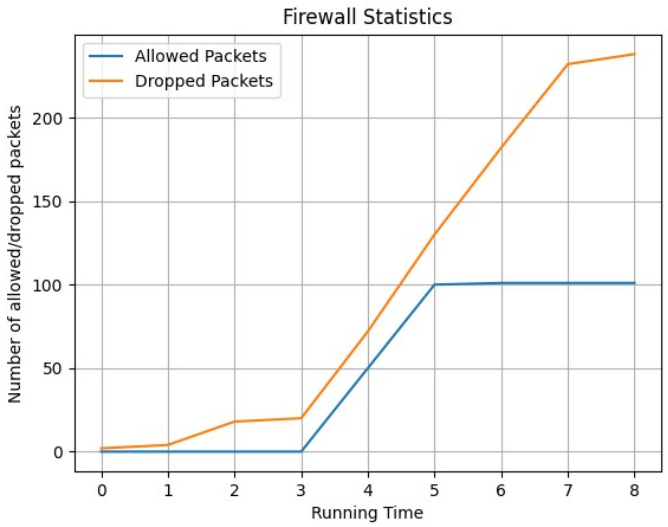



*Fig: Cumulative graph of allowed packets and Dropped packets*

This figure clearly demonstrates how the packets gets dropped when a firewall detects DDoS.