

The following is the Network Setup in this lab for Task 1 through 4 and 8 (Same NAT network):

Name	Role	IP Address	MAC Address
SEEDUbuntu	Machine A	10.0.2.7	08:00:27:b7:ba:af
SEEDUbuntu1	Machine B	10.0.2.8	08:00:27:cd:2d:fd

Task 1: Using Firewall

In this task, we set up some firewall policies, and observe the behavior of the system before and after the policies become effective. We run the firewall on Machine A. Before we set up the firewall, we verify that Machine A and B can telnet to each other as follows:

From Machine A to Machine B – Successful Telnet connection:

```
[04/01/20]seed@VM:~$ telnet 10.0.2.8
Trying 10.0.2.8...
Connected to 10.0.2.8.
Escape character is '^'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Feb 26 19:44:54 EST 2020 from 10.0.2.7 on pts/0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

[04/01/20]seed@VM:~$ exit
logout
Connection closed by foreign host.
[04/01/20]seed@VM:~$
```

From Machine B to Machine A – Successful Telnet connection:

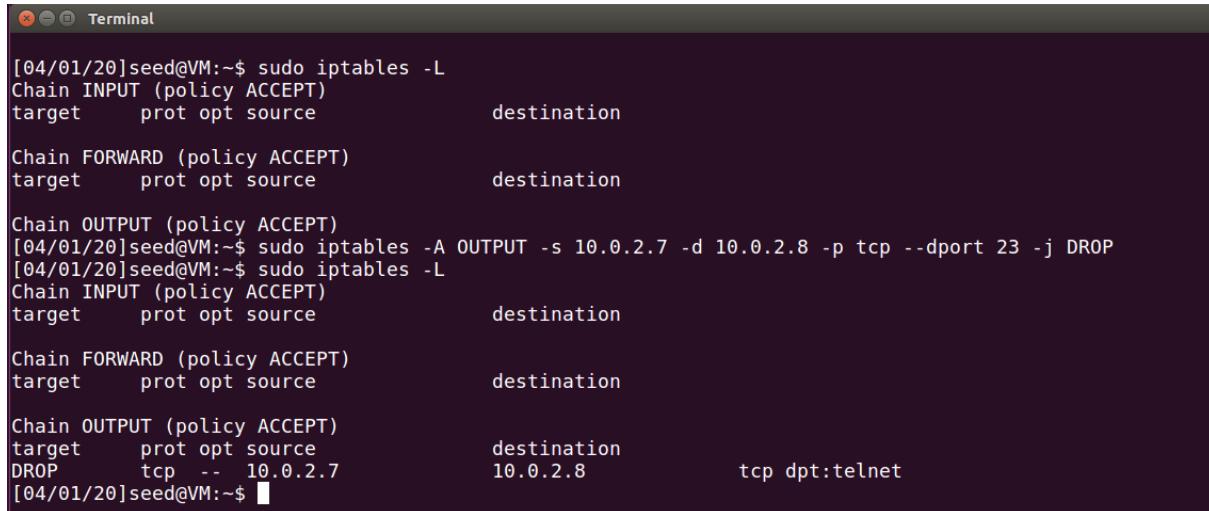
```
[04/01/20]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Mar 31 18:21:50 EDT 2020 from 10.0.2.7 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

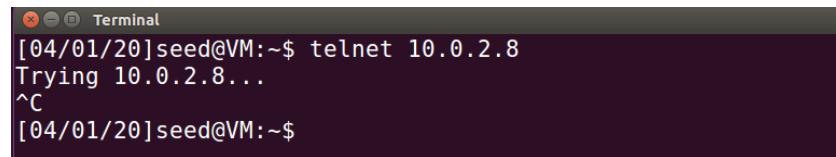
[04/01/20]seed@VM:~$ exit
logout
Connection closed by foreign host.
[04/01/20]seed@VM:~$
```

Now we set up a firewall rule to prevent Machine A from doing telnet to Machine B:



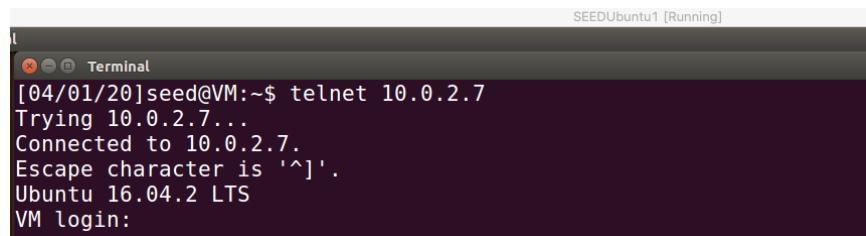
```
[04/01/20]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
[04/01/20]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.7 -d 10.0.2.8 -p tcp --dport 23 -j DROP
[04/01/20]seed@VM:~$ sudo iptables -L
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
DROP      tcp   --  10.0.2.7            10.0.2.8          tcp  dpt:telnet
[04/01/20]seed@VM:~$
```

On trying to telnet to Machine B, we see that the telnet connection is unsuccessful:



```
[04/01/20]seed@VM:~$ telnet 10.0.2.8
Trying 10.0.2.8...
^C
[04/01/20]seed@VM:~$
```

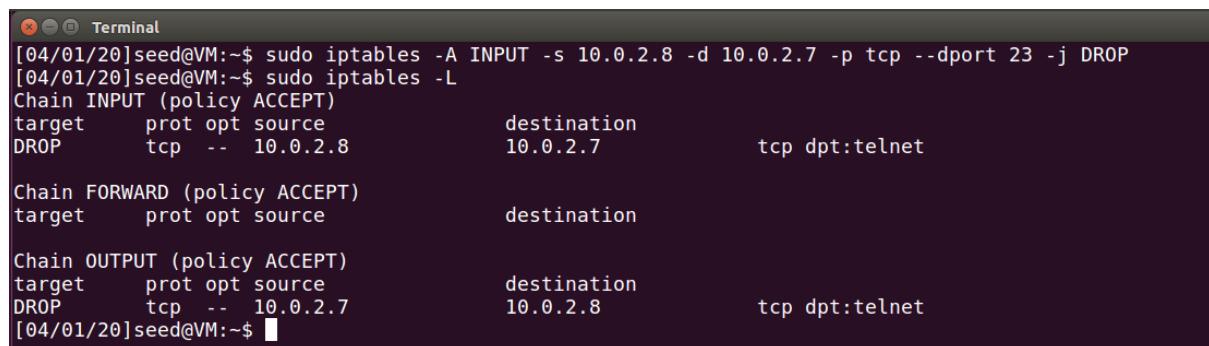
However, on telnetting from Machine B to A, we see that the telnet connection is successful:



```
[04/01/20]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login:
```

This verifies that only Machine A is prevented from doing telnet to Machine B.

Now we set up a firewall rule to prevent Machine B from doing telnet to Machine A:



```
[04/01/20]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.8 -d 10.0.2.7 -p tcp --dport 23 -j DROP
[04/01/20]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP      tcp   --  10.0.2.8            10.0.2.7          tcp  dpt:telnet
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP      tcp   --  10.0.2.7            10.0.2.8          tcp  dpt:telnet
[04/01/20]seed@VM:~$
```

The following shows that Machine B is no more able to telnet to Machine A:

```
[04/01/20]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
```

This verifies that Machine B is prevented from doing telnet to Machine A.

Now we set up a firewall rule to prevent Machine A from visiting an external web site. We use the following website for demonstration purposes: www.djsce.ac.in. We obtain an IP address for this website:

```
[04/01/20]seed@VM:~$ dig www.djsce.ac.in

; <>> DiG 9.10.3-P4-Ubuntu <>> www.djsce.ac.in
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17308
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.djsce.ac.in.           IN      A

;; ANSWER SECTION:
www.djsce.ac.in.       14400    IN      CNAME   djsce.ac.in.
djsce.ac.in.          14400    IN      A       148.251.191.4

;; Query time: 427 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed Apr 01 15:36:01 EDT 2020
;; MSG SIZE  rcvd: 74
```

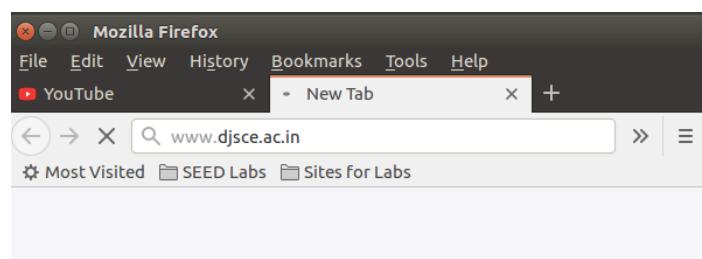
Now, we set up a rule to block any traffic going to the website's IP address. I chose to block the entire IP and not just http/https traffic. This helps in blocking the entire website completely:

```
Terminal
[04/01/20]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.7 -d 148.251.191.4 -j DROP
[04/01/20]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP      tcp  --  10.0.2.8            10.0.2.7             tcp dpt:telnet

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP      tcp  --  10.0.2.7            10.0.2.8             tcp dpt:telnet
DROP      all   --  10.0.2.7            vps.svkm.ac.in
```

We see that we are not able to load the website anymore (YouTube confirms the network connectivity):



Task 2: Implementing a Simple Firewall

In this task, we use LKM and Netfilter to implement the packet filtering module. We add the following policies to decide whether packets should be blocked or not:

- Block telnet from A to B
- Block telnet from B to A
- Block external website access from A
- Block SSH from A to B
- Block SSH from B to A

The following is the C code for this task:

```

1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/netfilter.h>
4 #include <linux/netfilter_ipv4.h>
5 #include <linux/ip.h>
6 #include <linux/tcp.h>
7 #include <linux/inet.h>
8
9 #define NIPQUAD(addr) ((unsigned char *)&addr)[0], ((unsigned char *)&addr)[1], ((unsigned char *)&addr)[2], ((unsigned char *)&addr)[3]
10
11 static struct nf_hook_ops nfho;
12 static struct nf_hook_ops nfho1;
13 static struct nf_hook_ops nfho2;
14 static struct nf_hook_ops nfho3;
15 static struct nf_hook_ops nfho4;
16
17 unsigned int telnet_outgoing(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
18 {
19     struct iphdr *iph;
20     struct tcphdr *tcph;
21
22     iph = ip_hdr(skb);
23     tcph = (void *)iph+iph->ihl*4;
24
25     if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && iph->saddr == in_aton("10.0.2.7") && iph->daddr==in_aton("10.0.2.8")) {
26         printk(KERN_INFO "Dropping Telnet Packet to destination address: %d.%d.%d.%d\n",NIPQUAD(iph->daddr));
27         return NF_DROP;
28     } else {
29         return NF_ACCEPT;
30     }
31 }
32
33 unsigned int ssh_outgoing(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
34 {
35     struct iphdr *iph;
36     struct tcphdr *tcph;
37
38     iph = ip_hdr(skb);
39     tcph = (void *)iph+iph->ihl*4;
40
41
42     if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && iph->saddr == in_aton("10.0.2.7") && iph->daddr==in_aton("10.0.2.8")) {
43         printk(KERN_INFO "Dropping SSH Packet to destination address: %d.%d.%d.%d\n",NIPQUAD(iph->daddr));
44         return NF_DROP;
45     } else {
46         return NF_ACCEPT;
47     }
48 }
49
50 unsigned int telnet_incoming(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
51 {
52     struct iphdr *iph;
53     struct tcphdr *tcph;
54
55     iph = ip_hdr(skb);
56     tcph = (void *)iph+iph->ihl*4;
57
58     if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && iph->saddr == in_aton("10.0.2.8") && iph->daddr==in_aton("10.0.2.7")) {
59         printk(KERN_INFO "Dropping Telnet Packet from source address: %d.%d.%d.%d\n",NIPQUAD(iph->saddr));
60         return NF_DROP;
61     } else {
62         return NF_ACCEPT;
63     }
64 }
65
66 unsigned int ssh_incoming(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
67 {
68     struct iphdr *iph;
69     struct tcphdr *tcph;
70
71     iph = ip_hdr(skb);
72     tcph = (void *)iph+iph->ihl*4;
73
74     if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && iph->saddr == in_aton("10.0.2.8") && iph->daddr==in_aton("10.0.2.7")) {
75         printk(KERN_INFO "Dropping SSH Packet from source address: %d.%d.%d.%d\n",NIPQUAD(iph->saddr));
76         return NF_DROP;
77     } else {
78         return NF_ACCEPT;
79     }
80 }
```

```

79     }
80 }
81 unsigned int web_block(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
82 {
83     struct iphdr *iph;
84     struct tcphdr *tcp;
85
86     iph = ip_hdr(skb);
87     tcp = (void *)iph+iph->ihl*4;
88
89     if (iph->protocol == IPPROTO_TCP && iph->saddr == in_aton("10.0.2.7") && iph->daddr==in_aton("148.251.191.4") && (tcp->dest == htons(80) || tcp->dest == htons(443)))
90         printk(KERN_INFO "Dropping Web Packet to web page on address: %d.%d.%d.%d\n",NIPQUAD(iph->daddr));
91     return NF_DROP;
92 } else {
93     return NF_ACCEPT;
94 }
95 }
96 }
97
98 int init_module()
99 {
100     nfho.hook = telnet_outgoing; /* Handler function */
101     nfho.hooknum = NF_INET_LOCAL_OUT;
102     nfho.pf = PF_INET;
103     nfho.priority = NF_IP_PRI_FIRST; /* Make our function first */
104     nf_register_hook(&nfho);
105
106     nfho1.hook = telnet_incoming; /* Handler function */
107     nfho1.hooknum = NF_INET_LOCAL_IN; /* First hook for IPv4 */
108     nfho1.pf = PF_INET;
109     nfho1.priority = NF_IP_PRI_FIRST; /* Make our function first */
110     nf_register_hook(&nfho1);
111
112     nfho2.hook = web_block; /* Handler function */
113     nfho2.hooknum = NF_INET_LOCAL_OUT; /* First hook for IPv4 */
114     nfho2.pf = PF_INET;
115     nfho2.priority = NF_IP_PRI_FIRST; /* Make our function first */
116     nf_register_hook(&nfho2);
117
118     nfho3.hook = ssh_outgoing; /* Handler function */
119     nfho3.hooknum = NF_INET_LOCAL_OUT; /* First hook for IPv4 */
120     nfho3.pf = PF_INET;
121     nfho3.priority = NF_IP_PRI_FIRST; /* Make our function first */
122     nf_register_hook(&nfho3);
123
124     nfho4.hook = ssh_incoming; /* Handler function */
125     nfho4.hooknum = NF_INET_LOCAL_IN; /* First hook for IPv4 */
126     nfho4.pf = PF_INET;
127     nfho4.priority = NF_IP_PRI_FIRST; /* Make our function first */
128     nf_register_hook(&nfho4);
129
130     return 0;
131 }
132 /* Cleanup routine */
133 void cleanup_module()
134 {
135     nf_unregister_hook(&nfho);
136     nf_unregister_hook(&nfho1);
137     nf_unregister_hook(&nfho2);
138     nf_unregister_hook(&nfho3);
139     nf_unregister_hook(&nfho4);
140 }
141

```

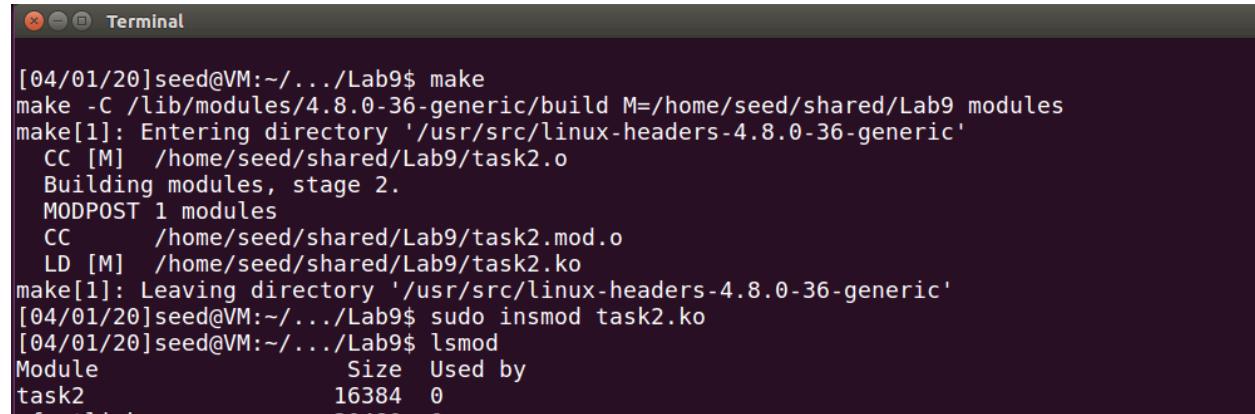
This is the Makefile that compiles the above C program as a loadable kernel module:

```

obj-m += task2.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

```

The following shows that the module is successfully compiled and loaded in the kernel:



```

[04/01/20]seed@VM:~/.../Lab9$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/shared/Lab9 modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/shared/Lab9/task2.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /home/seed/shared/Lab9/task2.mod.o
  LD [M]  /home/seed/shared/Lab9/task2.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[04/01/20]seed@VM:~/.../Lab9$ sudo insmod task2.ko
[04/01/20]seed@VM:~/.../Lab9$ lsmod
Module           Size  Used by
task2           16384  0

```

Before moving ahead, we make sure to remove the firewall rules configured in Task 1 so that the program's firewall rules' effect is visible.

```
[04/01/20]seed@VM:~/.../Lab9$ sudo iptables -L
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
[04/01/20]seed@VM:~/.../Lab9$
```

Now we test the firewall policies configured in the program. On performing telnet and SSH from Machine A to B, we see that the packets are dropped, and the connection is not established:

```
[04/01/20]seed@VM:~/.../Lab9$ telnet 10.0.2.8
Trying 10.0.2.8...
^C
[04/01/20]seed@VM:~/.../Lab9$ ssh 10.0.2.8
^C
[04/01/20]seed@VM:~/.../Lab9$ dmesg | tail -10
[ 2457.481989] device enp0s3 left promiscuous mode
[10514.445180] task2: module license 'unspecified' taints kernel.
[10514.445182] Disabling lock debugging due to kernel taint
[10565.639651] Dropping Telnet Packet to destination address: 10.0.2.8
[10566.645947] Dropping Telnet Packet to destination address: 10.0.2.8
[10625.009060] Dropping Telnet Packet to destination address: 10.0.2.8
[10626.037977] Dropping Telnet Packet to destination address: 10.0.2.8
[10632.431788] Dropping SSH Packet to destination address: 10.0.2.8
[10633.462369] Dropping SSH Packet to destination address: 10.0.2.8
[10635.478445] Dropping SSH Packet to destination address: 10.0.2.8
[04/01/20]seed@VM:~/.../Lab9$
```

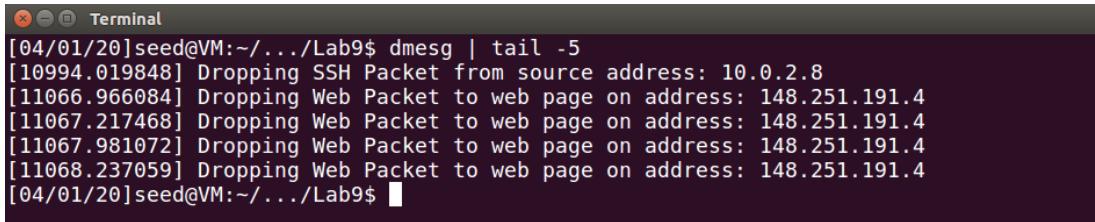
On initiating a telnet or SSH connection from Machine B to A, we see a similar result. The following on Machine B shows that the connection was not established:

```
[04/01/20]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
^C
[04/01/20]seed@VM:~$ ssh 10.0.2.7
^C
[04/01/20]seed@VM:~$
```

On looking at the logs in the kernel, we see that the telnet and SSH packets are dropped. This indicates that the telnet and SSH traffic is blocked between Machine A and B by the program:

```
[04/01/20]seed@VM:~/.../Lab9$ dmesg | tail -5
[10635.478445] Dropping SSH Packet to destination address: 10.0.2.8
[10984.576882] Dropping Telnet Packet from source address: 10.0.2.8
[10985.603321] Dropping Telnet Packet from source address: 10.0.2.8
[10993.003083] Dropping SSH Packet from source address: 10.0.2.8
[10994.019848] Dropping SSH Packet from source address: 10.0.2.8
[04/01/20]seed@VM:~/.../Lab9$
```

Now, to confirm that the website is blocked, we try to load the website in the Firefox browser, but it never loads. On looking at the kernel logs, we see that the packets going to this website is also dropped:



```
[04/01/20]seed@VM:~/.../Lab9$ dmesg | tail -5
[10994.019848] Dropping SSH Packet from source address: 10.0.2.8
[11066.966084] Dropping Web Packet to web page on address: 148.251.191.4
[11067.217468] Dropping Web Packet to web page on address: 148.251.191.4
[11067.981072] Dropping Web Packet to web page on address: 148.251.191.4
[11068.237059] Dropping Web Packet to web page on address: 148.251.191.4
[04/01/20]seed@VM:~/.../Lab9$
```

This concludes that the firewall rules configured in the program are effective.

Task 3: Evading Egress Filtering

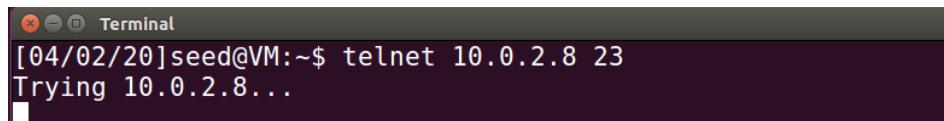
In this task, we evade the egress filtering by using SSH tunnels. On Machine A, we configure the following rules so that www.djsce.ac.in website and outgoing traffic to external telnet servers is blocked:



```
[04/02/20]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.7 -p tcp --dport 23 -j DROP
[04/02/20]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.7 -d 148.251.191.4 -j DROP
[04/02/20]seed@VM:~$ sudo iptables -L
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
DROP      tcp   --  10.0.2.7        anywhere            tcp dpt:telnet
DROP      all   --  10.0.2.7        vps.svkm.ac.in
```

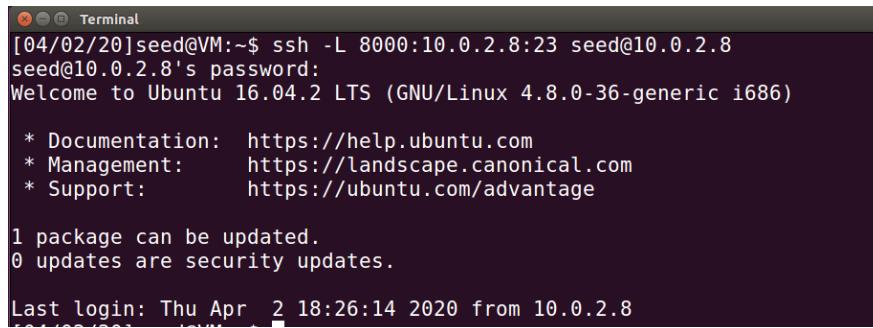
Task 3.a: Telnet to Machine B through the firewall

On trying to telnet to Machine B, we see that it is unsuccessful, as expected:



```
[04/02/20]seed@VM:~$ telnet 10.0.2.8 23
Trying 10.0.2.8...
```

To bypass the firewall, we establish an SSH tunnel between Machine A and B. The following command establishes an SSH tunnel between the localhost (port 8000) and Machine B (using the default port 22) and when packets come out of B's end, it will be forwarded to Machine B's port 23:



```
[04/02/20]seed@VM:~$ ssh -L 8000:10.0.2.8:23 seed@10.0.2.8
seed@10.0.2.8's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

Last login: Thu Apr  2 18:26:14 2020 from 10.0.2.8
```

On Machine A, the tunnel receives TCP packets from the telnet client (when we do telnet localhost 8000). On receiving this packet, the tunnel forwards the TCP packets to Machine B – port 22. Here, the received data is put into another TCP packet and sent to Machine B's port 23 (as mentioned in the SSH connection). The firewall only sees the SSH traffic and not the telnet traffic, and hence this SSH tunnel can be used to evade the firewall rule of blocking telnet connections.

So, on doing telnet to localhost 8000, we are basically telnetting to host 10.0.2.8 – port 23 via the SSH tunnel. This evades the firewall rule of blocking outgoing telnet connections.

```
[04/02/20]seed@VM:~$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Apr  2 18:26:58 EDT 2020 from 10.0.2.7 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

[04/02/20]seed@VM:~$
```

The following Wireshark capture shows the same. We see only SSH packets and no telnet packets in the packet capture, proving the point.

No.	Time	Source	Destination	Protocol	Length	Info
130	2020-04-01 18:41:36.660659723	10.0.2.8	10.0.2.7	SSHv2	102	Server: Encrypted packet (len=36)
131	2020-04-01 18:41:36.660683724	10.0.2.7	10.0.2.8	TCP	66	44168 - 22 [ACK] Seq=1724729912 Ack=1063966289 ...
132	2020-04-01 18:41:36.863181493	10.0.2.7	10.0.2.8	SSHv2	102	Client: Encrypted packet (len=36)
133	2020-04-01 18:41:36.863691719	10.0.2.8	10.0.2.7	SSHv2	102	Server: Encrypted packet (len=36)
134	2020-04-01 18:41:36.863710747	10.0.2.7	10.0.2.8	TCP	66	44168 - 22 [ACK] Seq=1724729948 Ack=1063966325 ...
135	2020-04-01 18:41:37.064943265	10.0.2.7	10.0.2.8	SSHv2	102	Client: Encrypted packet (len=36)
136	2020-04-01 18:41:37.066442887	10.0.2.8	10.0.2.7	SSHv2	102	Server: Encrypted packet (len=36)
137	2020-04-01 18:41:37.066467368	10.0.2.7	10.0.2.8	TCP	66	44168 - 22 [ACK] Seq=1724729984 Ack=1063966361 ...
138	2020-04-01 18:41:38.288720285	10.0.2.7	10.0.2.8	SSHv2	102	Client: Encrypted packet (len=36)
139	2020-04-01 18:41:38.288755974	10.0.2.8	10.0.2.7	SSHv2	102	Server: Encrypted packet (len=36)
140	2020-04-01 18:41:38.288774291	10.0.2.7	10.0.2.8	TCP	66	44168 - 22 [ACK] Seq=1724730020 Ack=1063966397 ...
141	2020-04-01 18:41:39.940240388	10.0.2.7	10.0.2.8	SSHv2	102	Client: Encrypted packet (len=36)
142	2020-04-01 18:41:39.941414851	10.0.2.8	10.0.2.7	SSHv2	102	Server: Encrypted packet (len=36)
143	2020-04-01 18:41:39.941488813	10.0.2.7	10.0.2.8	TCP	66	44168 - 22 [ACK] Seq=1724730056 Ack=1063966433 ...

Frame 135: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
 ▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
 ▶ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
 ▶ Transmission Control Protocol, Src Port: 44168, Dst Port: 22, Seq: 1724729948, Ack: 1063966325, Len: 36
 ▶ SSH Protocol
 ▶ SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
 Packet Length (encrypted): f0b47c71
 Encrypted Packet: 5122896d06c6030176662cdfb60e20e
 MAC: 871c3637367c1f5017bed87c4e93ef49

Task 3.b: Connect to Facebook using SSH Tunnel.

Another way of evading firewall rule is using Dynamic Port forwarding instead of the static one used in the previous task. In this, we only specify the local port number and not the destination. On the receiver's end, the packet is forwarded dynamically based on the destination information in the packet.

We achieve this using the following command. This command establishes an SSH tunnel between localhost port 9000 and Machine B port 22:

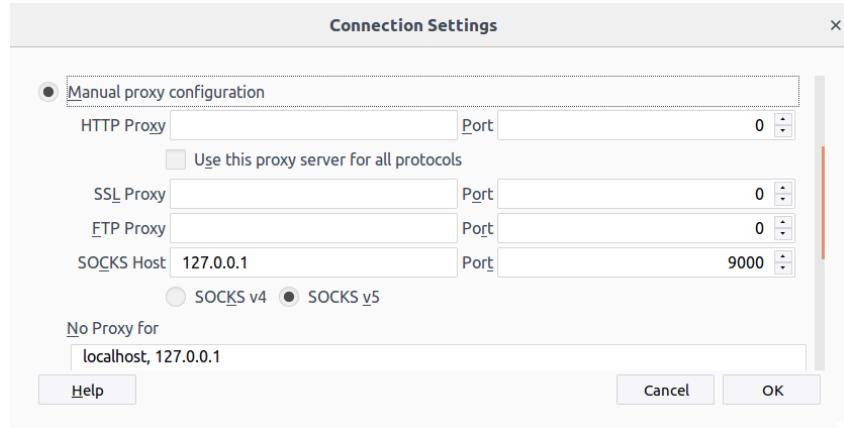
```
[04/01/20]seed@VM:~$ ssh -D 9000 -C seed@10.0.2.8
seed@10.0.2.8's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

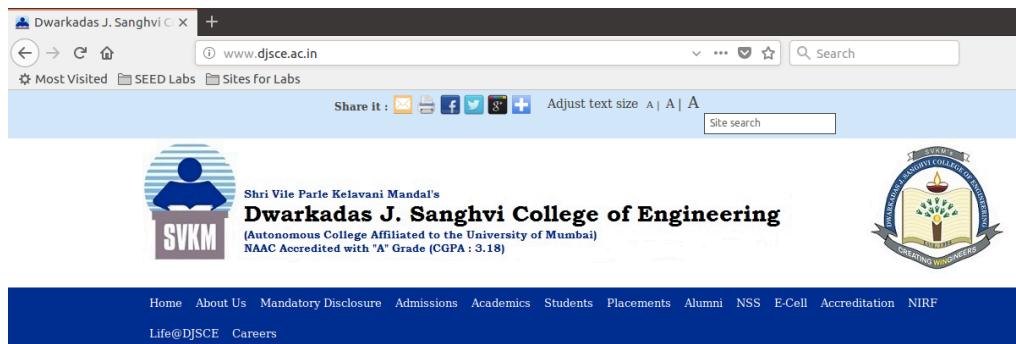
1 package can be updated.
0 updates are security updates.

Last login: Wed Apr  1 18:41:20 2020 from 10.0.2.7
[04/01/20]seed@VM:~$
```

Now, in order to evade the firewall, we need the browser to connect to localhost:9000 whenever it needs to connect to the web server. This will ensure that the traffic goes through the SSH tunnel. To do that, we use localhost:9000 as Firefox's proxy and set the same as follows:

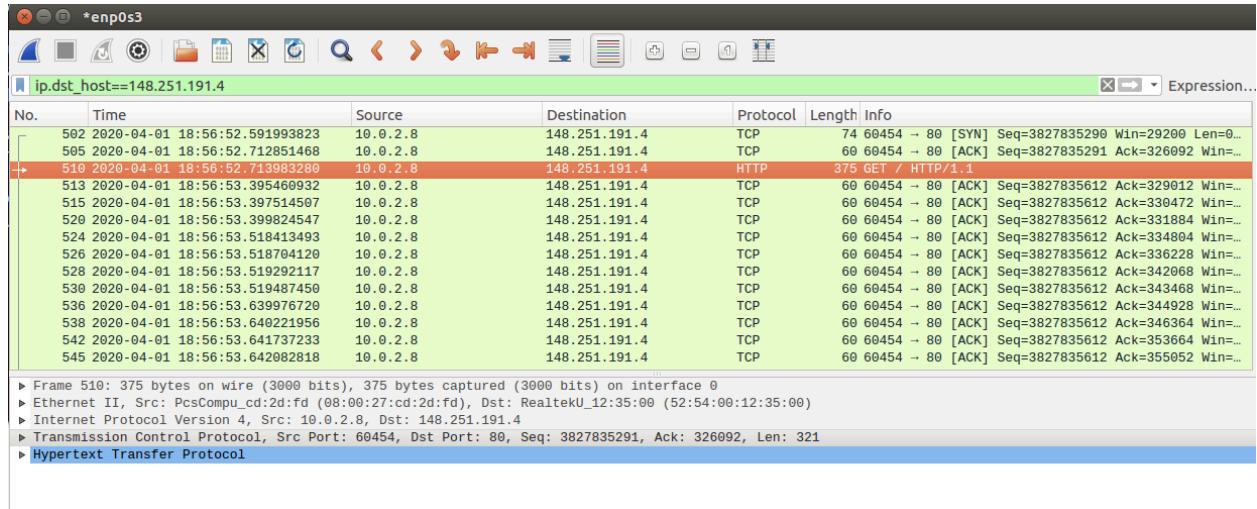


We use the SOCKS proxy because we need to let the browser tell the proxy about the destination in order to dynamically forward the traffic, since this proxy does not have the capability of detecting the destination on its own. After this is set, we visit the blocked web page and see that the website loads without any issues:

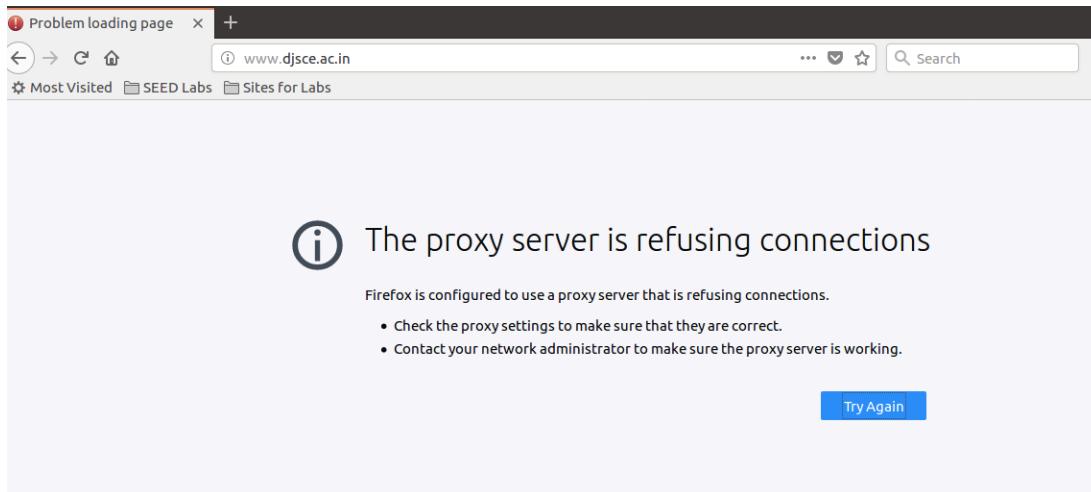


On looking at the Wireshark traffic, we see that the traffic to the website goes through Machine B and not Machine A. This is because the web traffic goes via the SSH tunnel between Machine A and Machine B,

and Machine B then forwards the traffic to the destination (web server). Since the traffic goes via SSH and not HTTP anymore, the firewall rule of blocked website is evaded.



Now, on breaking the SSH tunnel and reloading the page, we see that we are no more able to load the webpage. We see the error that the proxy server is refusing connections. This is because Machine A and B are no more connected via SSH, and this SSH tunnel was the proxy server for the browser.



On establishing the SSH tunnel again and reloading the webpage, we see the web page again as before. Since we have the established SSH tunnel, that is acting as the proxy server, the website was loaded again.

In this task, the browser connects to the SSH proxy at port 9000 on the localhost, and the SSH sends the TCP data over the tunnel to Machine B, which connects to the blocked website (based on the destination). This SSH tunnel responds back via the same tunnel and since all the traffic is SSH and not web traffic, the firewall does not block anything. We see the same in the following Wireshark, where the traffic goes from Machine A to B via SSH and Machine B to the web server on TCP/HTTP.

No.	Time	Source	Destination	Protocol	Length	Info
71	2020-04-01 19:03:14.559429307	10.0.2.7	10.0.2.8	SSHv2	150	Client: Encrypted packet (len=84)
72	2020-04-01 19:03:14.560156349	10.0.2.8	148.251.191.4	TCP	74	60492 → 80 [SYN] Seq=1148876095 Win=29200 Len=0 MSS=...
73	2020-04-01 19:03:14.602682863	10.0.2.8	10.0.2.7	TCP	66	22 → 44360 [ACK] Seq=1371910396 Ack=3181302980 Win=3...
74	2020-04-01 19:03:14.682796158	148.251.191.4	10.0.2.8	TCP	66	80 → 60492 [SYN, ACK] Seq=381044 Ack=1148876096 Win=...
75	2020-04-01 19:03:14.683175880	10.0.2.8	148.251.191.4	TCP	66	60492 → 80 [ACK] Seq=1148876096 Ack=381045 Win=29200...
76	2020-04-01 19:03:14.683971541	10.0.2.8	10.0.2.7	SSHv2	119	Server: Encrypted packet (len=44)
77	2020-04-01 19:03:14.684001911	10.0.2.7	10.0.2.8	TCP	66	44360 → 22 [ACK] Seq=3181302980 Ack=1371910440 Win=3...
78	2020-04-01 19:03:14.684642224	10.0.2.7	10.0.2.8	SSHv2	350	Client: Encrypted packet (len=284)
79	2020-04-01 19:03:14.685131970	10.0.2.8	10.0.2.7	TCP	66	22 → 44360 [ACK] Seq=1371910440 Ack=3181303264 Win=3...
80	2020-04-01 19:03:14.685393127	10.0.2.8	148.251.191.4	HTTP	375	GET / HTTP/1.1
81	2020-04-01 19:03:14.812583566	148.251.191.4	10.0.2.8	TCP	66	80 → 60492 [ACK] Seq=381045 Ack=1148876417 Win=32447...
82	2020-04-01 19:03:15.133473144	148.251.191.4	10.0.2.8	TCP	2974	[TCP segment of a reassembled PDU]
83	2020-04-01 19:03:15.134322344	10.0.2.8	148.251.191.4	TCP	66	60492 → 80 [ACK] Seq=1148876417 Ack=383965 Win=35040...
84	2020-04-01 19:03:15.135428864	148.251.191.4	10.0.2.8	TCP	1514	[TCP segment of a reassembled PDU]

Frame 71: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
 ► Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
 ► Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
 ▼ Transmission Control Protocol, Src Port: 44360, Dst Port: 22, Seq: 3181302896, Ack: 1371910396, Len: 84
 Source Port: 44360
 Destination Port: 22
 [Stream index: 0]
 [TCP Segment Len: 84]

This concludes that firewall rules can be evaded by using different protocol – such as SSH and perform the blocked activities without the firewall's knowledge unless deep packet inspection is used. Even with deep packet inspection, firewall rules can be evaded using encryption-based protocols – SSH instead of telnet.

Task 4: Evading Ingress Filtering

On Machine A, we run the web server which is protected by blocking any incoming HTTP/SSH traffic from the external network (here just Machine B). This Machine can also be configured to block all the SSH traffic from anywhere. The results will be similar. The following is the web page:

Internal Network

localhost

Most Visited SEED Labs Sites for Labs

This is an example of a simple HTML page in the Internal Network - Megha.

The following show the firewall rules configured on Machine A:

```
[04/02/20]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.8 -d 10.0.2.7 -p tcp --dport 80 -j DROP
[04/02/20]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.8 -d 10.0.2.7 -p tcp --dport 22 -j DROP
[04/02/20]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
DROP      tcp  --  10.0.2.8        10.0.2.7          tcp dpt:http
DROP      tcp  --  10.0.2.8        10.0.2.7          tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
[04/02/20]seed@VM:~$
```

On trying to access this web server from the outside, we see that it is not possible since the http traffic is blocked. We also cannot use the previous SSH tunnel mechanism for port forwarding that would have

provided us with the access to the web server. Since the Machine blocks only incoming SSH tunnel, we set up a reverse SSH tunnel on Machine A which is not blocked by the firewall. This SSH tunnel will be used to access the protected web server.

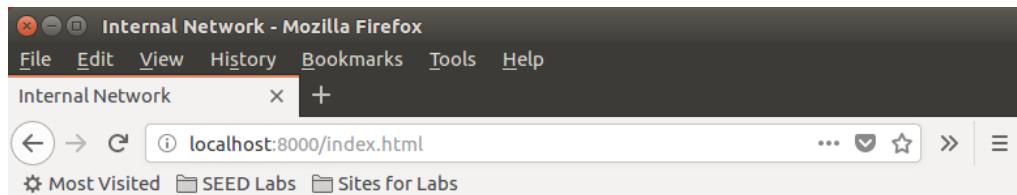
```
[04/02/20]seed@VM:~/html$ ssh -R 8000:10.0.2.7:80 seed@10.0.2.8
seed@10.0.2.8's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

Last login: Thu Apr  2 16:29:14 2020 from 10.0.2.7
[04/02/20]seed@VM:~$
```

After establishing the above SSH tunnel, we can access the web server from the outside (Machine B – 10.0.2.8) by simply going to localhost:8000 on the external Machine. This is because the established SSH tunnel forwards the request to SSH client on Machine A, which further forwards the request to port 80 of the Machine A – 10.0.2.7 i.e. the web server. The following shows that we can successfully access the web server from the outside:



This is an example of a simple HTML page in the Internal Network - Megha.

Network Setup for the Task 5 – 7 is as following:

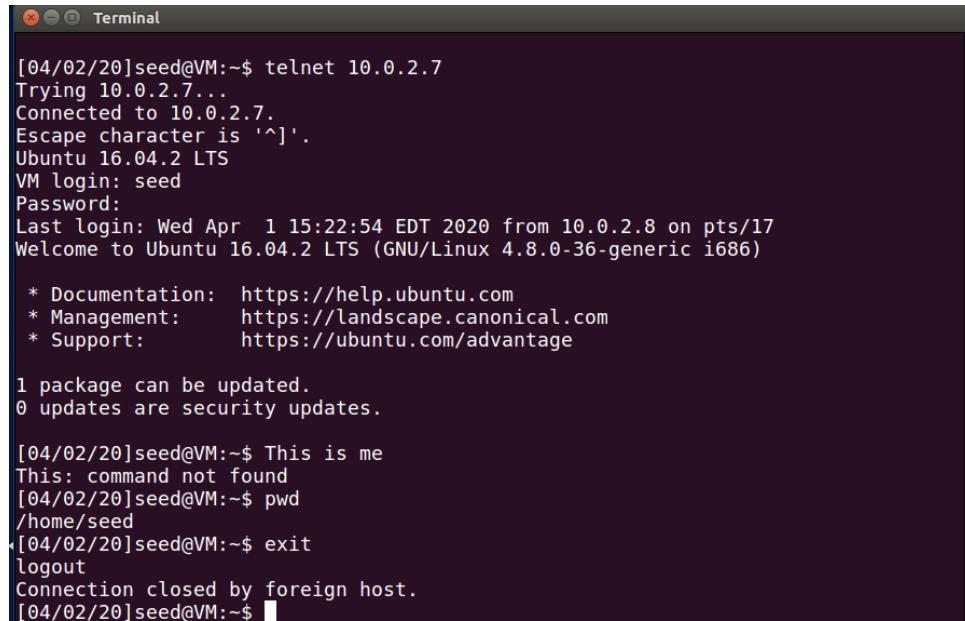
Name	Role	IP Address	MAC Address
SEEDUbuntu	Machine B (External)	10.0.2.7	08:00:27:b7:ba:af
SEEDUbuntu1	Router/Firewall	10.0.2.8 192.168.60.1	08:00:27:cd:2d:fd 08:00:27:50:12:67
SEEDUbuntu2	Machine A (Internal)	192.168.60.101	08:00:27:98:60:5e

Task 5: SNAT

We set up SNAT on the Firewall/Router as following:

```
[04/02/20]seed@VM:~$ sudo iptables -t nat -A POSTROUTING -o enp0s3 -j SNAT --to-source 10.0.2.8
[04/02/20]seed@VM:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[04/02/20]seed@VM:~$
```

This rule states that for any packet going out of the interface enp0s3, set the source IP address as 10.0.2.8. Now, on telnetting from Machine A to Machine B (internal to external network), we see that the connection is successfully established:



```
[04/02/20]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Apr  1 15:22:54 EDT 2020 from 10.0.2.8 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

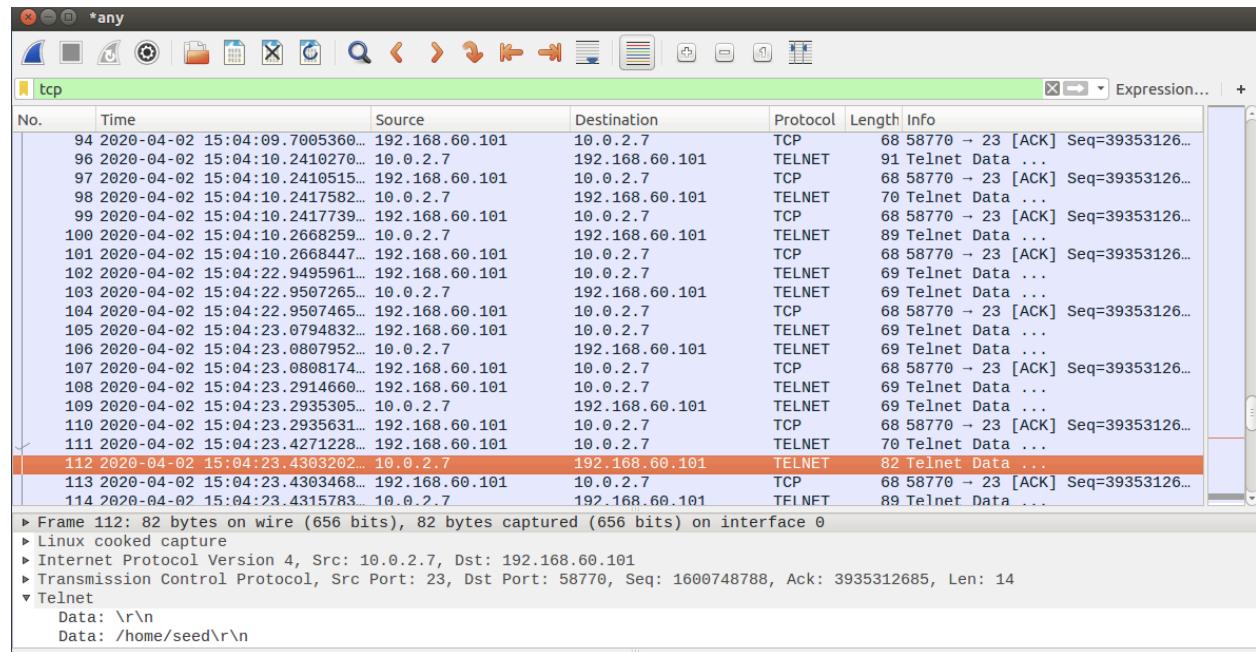
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[04/02/20]seed@VM:~$ This is me
This: command not found
[04/02/20]seed@VM:~$ pwd
/home/seed
[04/02/20]seed@VM:~$ exit
logout
Connection closed by foreign host.
[04/02/20]seed@VM:~$ ]
```

Now, on looking at the Wireshark trace on all the Machines, we see the following:

On Machine A, we see that the traffic goes between Machine A to Machine B, but since the destination is of a different network, the packet goes to the gateway – our firewall:



On the router, we see that the packet comes from Machine A to the firewall and then the firewall forwards this packet to Machine B using 10.0.2.8 as the source IP. For every packet sent, we see a response from 10.0.2.7 (Machine B) to the firewall and not directly to Machine A. This is because, for Machine B, the source IP becomes the firewall and not Machine A due to the SNAT rule. The firewall then forwards the packet to Machine A.

No.	Time	Source	Destination	Protocol	Length	Info
214	2020-04-02 15:04:23.1277643...	192.168.60.101	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312682 Ack=1600748...
215	2020-04-02 15:04:23.1277891...	10.0.2.8	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312682 Ack=1600748...
216	2020-04-02 15:04:23.3385332...	192.168.60.101	10.0.2.7	TELNET	69	Telnet Data ...
217	2020-04-02 15:04:23.3386600...	10.0.2.8	10.0.2.7	TELNET	69	Telnet Data ...
218	2020-04-02 15:04:23.3400116...	10.0.2.7	10.0.2.8	TELNET	69	Telnet Data ...
219	2020-04-02 15:04:23.3400383...	10.0.2.7	192.168.60.101	TELNET	69	Telnet Data ...
220	2020-04-02 15:04:23.34006376...	192.168.60.101	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312683 Ack=1600748...
221	2020-04-02 15:04:23.3406688...	10.0.2.8	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312683 Ack=1600748...
222	2020-04-02 15:04:23.4741613...	192.168.60.101	10.0.2.7	TELNET	70	Telnet Data ...
223	2020-04-02 15:04:23.4742806...	10.0.2.8	10.0.2.7	TELNET	70	Telnet Data ...
224	2020-04-02 15:04:23.4768729...	10.0.2.7	10.0.2.8	TELNET	82	Telnet Data ...
225	2020-04-02 15:04:23.4768972...	192.168.60.101	10.0.2.7	TELNET	82	Telnet Data ...
226	2020-04-02 15:04:23.4773005...	192.168.60.101	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312685 Ack=1600748...
227	2020-04-02 15:04:23.4773253...	10.0.2.8	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312685 Ack=1600748...
228	2020-04-02 15:04:23.4781384...	10.0.2.7	10.0.2.8	TELNET	89	Telnet Data ...
229	2020-04-02 15:04:23.4781625...	10.0.2.7	192.168.60.101	TELNET	89	Telnet Data ...
230	2020-04-02 15:04:23.4785533...	192.168.60.101	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312685 Ack=1600748...
231	2020-04-02 15:04:23.4785839...	10.0.2.8	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312685 Ack=1600748...
232	2020-04-02 15:04:26.8280644...	192.168.60.101	10.0.2.7	TELNET	69	Telnet Data ...
233	2020-04-02 15:04:26.8280924...	10.0.2.8	10.0.2.7	TELNET	69	Telnet Data ...
234	2020-04-02 15:04:26.8287393...	10.0.2.7	10.0.2.8	TELNET	69	Telnet Data ...
235	2020-04-02 15:04:26.8287617...	10.0.2.7	192.168.60.101	TELNET	69	Telnet Data ...
236	2020-04-02 15:04:26.8292272...	192.168.60.101	10.0.2.7	TCP	68	58770 → 23 [ACK] Seq=3935312686 Ack=1600748...

▶ Frame 224: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
 ▶ Linux cooked capture
 ▶ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
 ▶ Transmission Control Protocol, Src Port: 23, Dst Port: 58770, Seq: 1600748788, Ack: 3935312685, Len: 14
 ▶ Telnet
 Data: \r\n
 Data: /home/seed\r\n

On Machine B, we see that the communication is purely between Machine B and the firewall:

No.	Time	Source	Destination	Protocol	Length	Info
169	2020-04-02 15:04:23.295605532	10.0.2.7	10.0.2.8	TELNET	67	Telnet Data ...
110	2020-04-02 15:04:23.296475696	10.0.2.8	10.0.2.7	TCP	66	58770 → 23 [ACK] Seq=3935312681 Ack=1600748786 Win=30336 Len=...
111	2020-04-02 15:04:23.425228641	10.0.2.8	10.0.2.7	TELNET	67	Telnet Data ...
112	2020-04-02 15:04:23.425484842	10.0.2.7	10.0.2.8	TELNET	67	Telnet Data ...
113	2020-04-02 15:04:23.426492146	10.0.2.8	10.0.2.7	TCP	66	58770 → 23 [ACK] Seq=3935312682 Ack=1600748787 Win=30336 Len=...
114	2020-04-02 15:04:23.637785862	10.0.2.8	10.0.2.7	TELNET	67	Telnet Data ...
115	2020-04-02 15:04:23.638026549	10.0.2.7	10.0.2.8	TELNET	67	Telnet Data ...
116	2020-04-02 15:04:23.639277674	10.0.2.8	10.0.2.7	TCP	66	58770 → 23 [ACK] Seq=3935312683 Ack=1600748788 Win=30336 Len=...
117	2020-04-02 15:04:23.772798756	10.0.2.8	10.0.2.7	TELNET	69	Telnet Data ...
118	2020-04-02 15:04:23.774941736	10.0.2.7	10.0.2.8	TELNET	80	Telnet Data ...
119	2020-04-02 15:04:23.775825521	10.0.2.8	10.0.2.7	TCP	66	58770 → 23 [ACK] Seq=3935312685 Ack=1600748802 Win=30336 Len=...
120	2020-04-02 15:04:23.776222665	10.0.2.7	10.0.2.8	TELNET	87	Telnet Data ...
121	2020-04-02 15:04:23.777094315	10.0.2.8	10.0.2.7	TCP	66	58770 → 23 [ACK] Seq=3935312685 Ack=1600748823 Win=30336 Len=...
122	2020-04-02 15:04:27.124954113	10.0.2.8	10.0.2.7	TELNET	67	Telnet Data ...
123	2020-04-02 15:04:27.125159537	10.0.2.7	10.0.2.8	TELNET	67	Telnet Data ...
124	2020-04-02 15:04:27.126093244	10.0.2.8	10.0.2.7	TCP	66	58770 → 23 [ACK] Seq=3935312686 Ack=1600748824 Win=30336 Len=...
125	2020-04-02 15:04:27.348742262	10.0.2.8	10.0.2.7	TELNET	67	Telnet Data ...
126	2020-04-02 15:04:27.349048980	10.0.2.7	10.0.2.8	TELNET	67	Telnet Data ...

▶ Frame 118: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
 ▶ Ethernet II, Src: PcsCompu_b7:baf (08:00:27:b7:baf), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
 ▶ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
 ▶ Transmission Control Protocol, Src Port: 23, Dst Port: 58770, Seq: 1600748788, Ack: 3935312685, Len: 14
 ▶ Telnet
 Data: \r\n
 Data: /home/seed\r\n

This proves that the SNAT is successful since on the receiver's end, we do not see the original IP address but the configured NAT address (10.0.2.8). The contents of each of the highlighted packet in the Wireshark trace show the trace of a single packet.

Task 6: DNAT for Port Forwarding

We set up DNAT on the Firewall/Router as following to perform port forwarding:

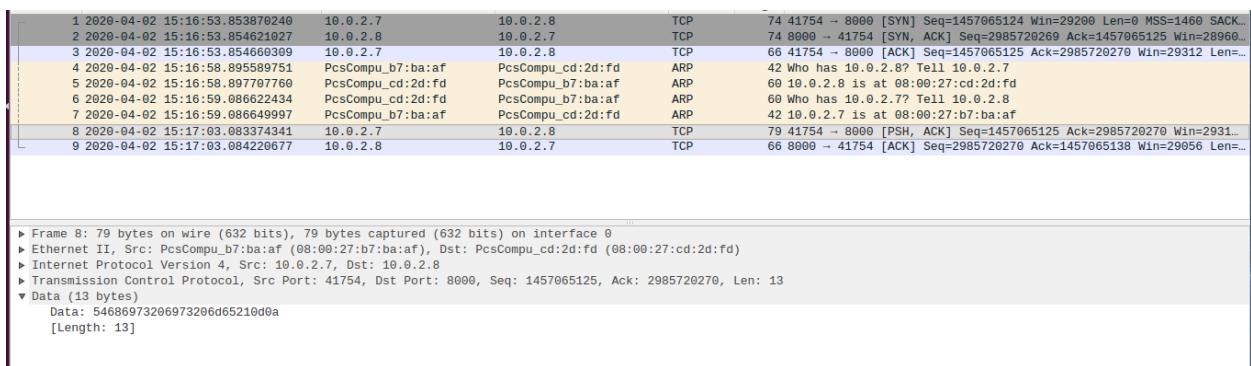
```
[04/02/20]seed@VM:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8000 -j DNAT --to-destination 192.168.60.101:9000
[04/02/20]seed@VM:~$
```

This rule will route any traffic received on Firewall port 8000 to Machine A's port 9000.

Now to test the configuration, we telnet to the Firewall on port 8000 from Machine B. Before telnetting, we start a server on Machine A listening to port 9000. We see that the telnet is successful:

```
[04/02/20]seed@VM:~$ telnet 10.0.2.8 8000
Trying 10.0.2.8...
Connected to 10.0.2.8.
Escape character is '^]'.
This is me!
```

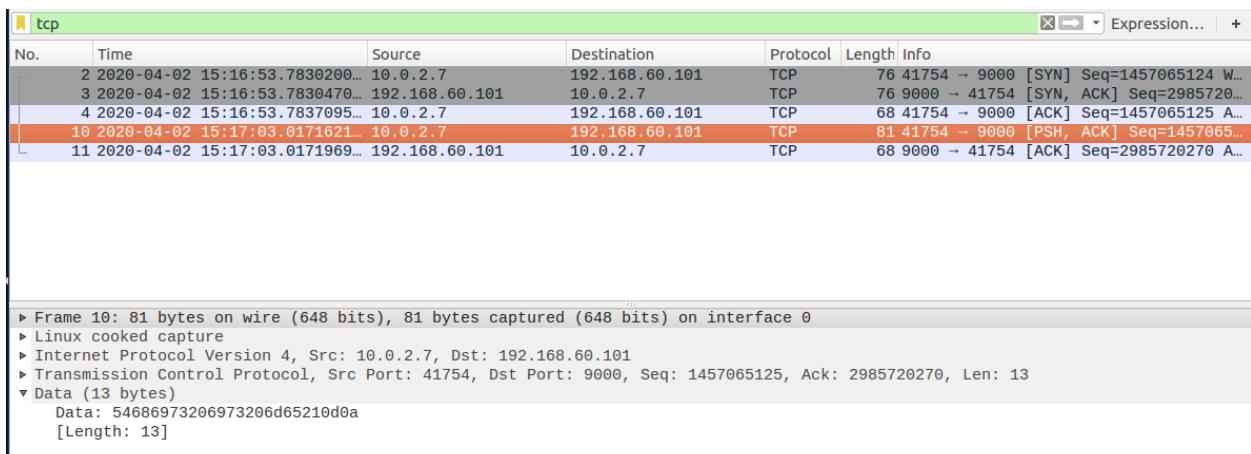
On looking at the Wireshark trace on Machine B, we see that the traffic goes between Machine B and Firewall's port 8000.



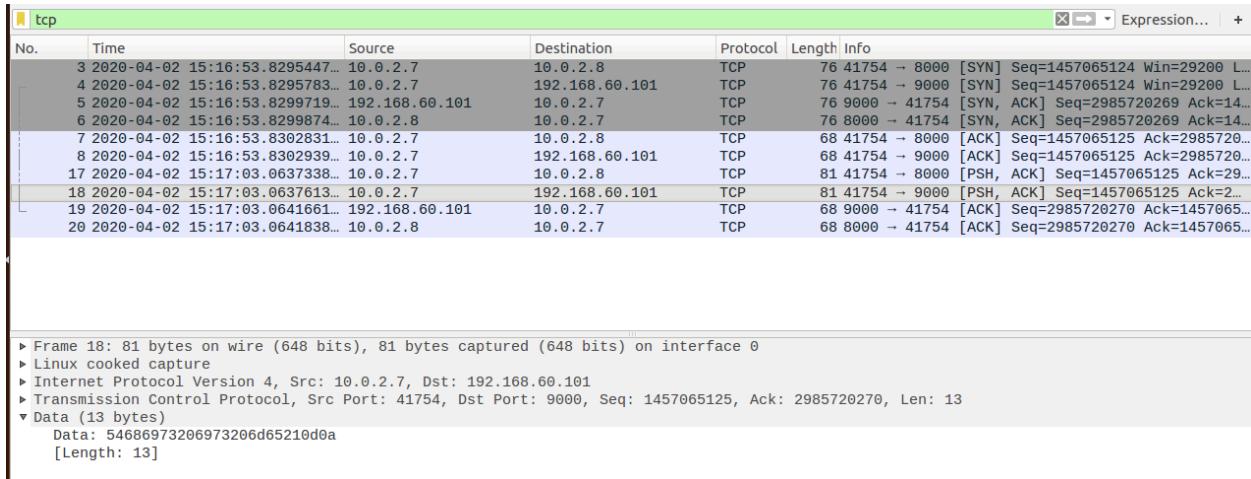
Now on looking at the connection on Internal Machine A, we see that the connection is the established with Machine A:

```
[04/02/20]seed@VM:~$ nc -l -v 9000
Listening on [0.0.0.0] (family 0, port 9000)
Connection from [10.0.2.7] port 9000 [tcp/*] accepted (family 2, sport 41754)
This is me!
```

On looking at the Wireshark trace on Machine A, we see that the traffic actually comes from Machine B:



This proves that Machine A and B are connected using the telnet connection. On looking at the Wireshark trace on the Firewall, we see that the traffic comes from Machine B to the firewall and then the firewall forwards this packet to Machine A (port 9000) as configured.



This proves that the DNAT port forwarding is successful.

Task 7: DNAT for Load Balancing

Now, we use the DNAT to perform Load Balancing. The following two rules are configured on the firewall which will forward the traffic with certain randomness to different Machines (here port):

```

[04/02/20]seed@VM:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.101:9001
[04/02/20]seed@VM:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -m statistic --mode nth --every 2 --packet 1 -j DNAT --to-destination 192.168.60.101:9002
[04/02/20]seed@VM:~$ 

```

On Machine A, we start two servers accepting multiple connections on port 9001 and 9002. In order to test the load balancing, we then run the following commands on Machine B that sends traffic to the firewall on port 8000:

```

[04/02/20]seed@VM:~$ echo "hello 1" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ 

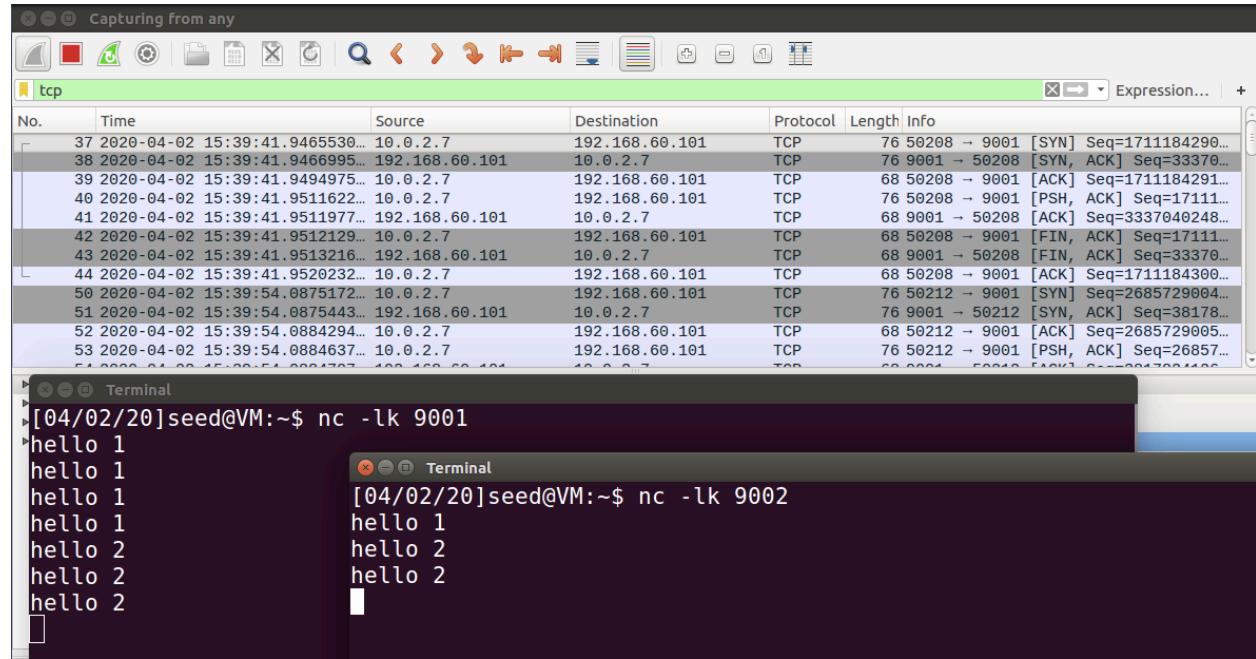
```

```

[04/02/20]seed@VM:~$ echo "hello 2" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ 

```

The following screenshot on Machine A indicates that even though the traffic was sent from Machine B to the firewall, it was forwarded to Machine A's servers running on different ports:



On the firewall, we see that the traffic comes from Machine B to the firewall (packet 70, 90) and then the firewall sends these packets to Machine A – either on port 9001 (packet 71) or port 9002 (packet 91).

No.	Time	Source	Destination	Protocol	Length	Info
70	2020-04-02 15:39:54.084444...	10.0.2.7	10.0.2.8	TCP	76	50212 → 8080 [SYN] Seq=2685729004 Win=29200...
71	2020-04-02 15:39:54.084475...	10.0.2.7	192.168.60.101	TCP	76	50212 → 9001 [SYN] Seq=2685729004 Win=29200...
72	2020-04-02 15:39:54.0849890...	192.168.60.101	10.0.2.7	TCP	76	9001 → 50212 [SYN, ACK] Seq=3817824125 Ack=...
73	2020-04-02 15:39:54.0850067...	10.0.2.8	10.0.2.7	TCP	76	8080 → 50212 [SYN, ACK] Seq=3817824125 Ack=...
74	2020-04-02 15:39:54.0854419...	10.0.2.7	10.0.2.8	TCP	68	50212 → 8080 [ACK] Seq=2685729005 Ack=38178...
75	2020-04-02 15:39:54.0854549...	10.0.2.7	192.168.60.101	TCP	68	50212 → 9001 [ACK] Seq=2685729005 Ack=38178...
76	2020-04-02 15:39:54.0855148...	10.0.2.7	10.0.2.8	TCP	76	50212 → 8080 [PSH, ACK] Seq=2685729005 Ack=...
77	2020-04-02 15:39:54.0855185...	10.0.2.7	192.168.60.101	TCP	76	50212 → 9001 [PSH, ACK] Seq=2685729005 Ack=...
78	2020-04-02 15:39:54.0855253...	10.0.2.7	10.0.2.8	TCP	68	50212 → 8080 [FIN, ACK] Seq=2685729013 Ack=...
79	2020-04-02 15:39:54.0855276...	10.0.2.7	192.168.60.101	TCP	68	50212 → 9001 [FIN, ACK] Seq=2685729013 Ack=...
80	2020-04-02 15:39:54.0858619...	192.168.60.101	10.0.2.7	TCP	68	9001 → 50212 [ACK] Seq=3817824126 Ack=26857...
81	2020-04-02 15:39:54.0858760...	10.0.2.8	10.0.2.7	TCP	68	8080 → 50212 [ACK] Seq=3817824126 Ack=26857...
82	2020-04-02 15:39:54.0861505...	192.168.60.101	10.0.2.7	TCP	68	9001 → 50212 [FIN, ACK] Seq=3817824126 Ack=...
83	2020-04-02 15:39:54.0861611...	10.0.2.8	10.0.2.7	TCP	68	8080 → 50212 [FIN, ACK] Seq=3817824126 Ack=...
84	2020-04-02 15:39:54.0863294...	10.0.2.7	10.0.2.8	TCP	68	50212 → 8080 [ACK] Seq=2685729014 Ack=38178...
85	2020-04-02 15:39:54.0863398...	10.0.2.7	192.168.60.101	TCP	68	50212 → 9001 [ACK] Seq=2685729014 Ack=38178...
86	2020-04-02 15:39:55.3388223...	10.0.2.7	10.0.2.8	TCP	76	50214 → 8080 [SYN] Seq=3319361801 Win=29200...
87	2020-04-02 15:39:55.3388583...	10.0.2.7	192.168.60.101	TCP	76	50214 → 9002 [SYN] Seq=3319361801 Win=29200...
88	2020-04-02 15:39:55.3393076...	192.168.60.101	10.0.2.7	TCP	76	9002 → 50214 [SYN, ACK] Seq=1691232388 Ack=...
89	2020-04-02 15:39:55.3393358...	10.0.2.8	10.0.2.7	TCP	76	8080 → 50214 [SYN, ACK] Seq=1691232388 Ack=...
90	2020-04-02 15:39:55.3396455...	10.0.2.7	10.0.2.8	TCP	68	50214 → 8080 [ACK] Seq=3319361802 Ack=16912...
91	2020-04-02 15:39:55.3396641...	10.0.2.7	192.168.60.101	TCP	68	50214 → 9002 [ACK] Seq=3319361802 Ack=16912...
92	2020-04-02 15:39:55.3403975...	10.0.2.7	10.0.2.8	TCP	76	50214 → 8080 [PSH, ACK] Seq=3319361802 Ack=...
93	2020-04-02 15:39:55.3404176...	10.0.2.7	192.168.60.101	TCP	76	50214 → 9002 [PSH, ACK] Seq=3319361802 Ack=...
94	2020-04-02 15:39:55.3404889...	10.0.2.7	10.0.2.8	TCP	68	50214 → 8080 [FIN, ACK] Seq=3319361810 Ack=...
95	2020-04-02 15:39:55.3404948...	10.0.2.7	192.168.60.101	TCP	68	50214 → 9002 [FIN, ACK] Seq=3319361810 Ack=...
96	2020-04-02 15:39:55.3407699...	192.168.60.101	10.0.2.7	TCP	68	9002 → 50214 [ACK] Seq=1691232389 Ack=33193...
97	2020-04-02 15:39:55.3407824...	10.0.2.8	10.0.2.7	TCP	68	8080 → 50214 [ACK] Seq=1691232389 Ack=33193...
98	2020-04-02 15:39:55.3409643...	192.168.60.101	10.0.2.7	TCP	68	9002 → 50214 [FIN, ACK] Seq=1691232389 Ack=...
99	2020-04-02 15:39:55.3410181...	10.0.2.8	10.0.2.7	TCP	68	8080 → 50214 [FIN, ACK] Seq=1691232389 Ack=...
100	2020-04-02 15:39:55.3412153...	10.0.2.7	10.0.2.8	TCP	68	50214 → 8080 [ACK] Seq=3319361811 Ack=16912...

The following shows that on Machine B, the traffic is purely communicated between Machine B and the firewall and Machine B has no idea about the traffic being distributed and forwarded to different ports on a different Machine.

No.	Time	Source	Destination	Protocol	Length	Info
19	2020-04-02 15:39:54.079642019	10.0.2.7	10.0.2.8	TCP	74	50212 → 8080 [SYN] Seq=2685729004 Win=29200 Len=0 MSS=1460 SA..
20	2020-04-02 15:39:54.088582897	10.0.2.8	10.0.2.7	TCP	74	8080 → 50212 [SYN, ACK] Seq=3817824125 Ack=2685729005 Win=289..
21	2020-04-02 15:39:54.088612462	10.0.2.7	10.0.2.8	TCP	66	50212 → 8080 [ACK] Seq=2685729005 Ack=3817824126 Win=29312 Le..
22	2020-04-02 15:39:54.088737980	10.0.2.7	10.0.2.8	TCP	74	50212 → 8080 [PSH, ACK] Seq=2685729005 Ack=3817824126 Win=293..
23	2020-04-02 15:39:54.088819327	10.0.2.7	10.0.2.8	TCP	66	50212 → 8080 [FIN, ACK] Seq=2685729013 Ack=3817824126 Win=293..
24	2020-04-02 15:39:54.08156585	10.0.2.8	10.0.2.7	TCP	66	8080 → 50212 [ACK] Seq=3817824126 Ack=2685729013 Win=29056 Le..
25	2020-04-02 15:39:54.081739949	10.0.2.8	10.0.2.7	TCP	66	8080 → 50212 [FIN, ACK] Seq=3817824126 Ack=2685729014 Win=290..
26	2020-04-02 15:39:54.081759659	10.0.2.7	10.0.2.8	TCP	66	50212 → 8080 [ACK] Seq=2685729014 Ack=3817824127 Win=29312 Le..
27	2020-04-02 15:39:55.334048473	10.0.2.7	10.0.2.8	TCP	74	50214 → 8080 [SYN] Seq=3319361801 Win=29200 Len=0 MSS=1460 SA..
28	2020-04-02 15:39:55.334969916	10.0.2.8	10.0.2.7	TCP	74	8080 → 50214 [SYN, ACK] Seq=1691232388 Ack=3319361802 Win=289..
29	2020-04-02 15:39:55.335069695	10.0.2.7	10.0.2.8	TCP	66	50214 → 8080 [ACK] Seq=3319361802 Ack=1691232389 Win=29312 Le..
30	2020-04-02 15:39:55.335595112	10.0.2.7	10.0.2.8	TCP	74	50214 → 8080 [PSH, ACK] Seq=3319361802 Ack=1691232389 Win=293..
31	2020-04-02 15:39:55.335714548	10.0.2.7	10.0.2.8	TCP	66	50214 → 8080 [FIN, ACK] Seq=3319361810 Ack=1691232389 Win=293..
32	2020-04-02 15:39:55.336369257	10.0.2.8	10.0.2.7	TCP	66	8080 → 50214 [ACK] Seq=1691232389 Ack=3319361810 Win=29056 Le..
33	2020-04-02 15:39:55.336692464	10.0.2.8	10.0.2.7	TCP	66	8080 → 50214 [FIN, ACK] Seq=1691232389 Ack=3319361811 Win=290..
34	2020-04-02 15:39:55.336624176	10.0.2.7	10.0.2.8	TCP	66	50214 → 8080 [ACK] Seq=3319361811 Ack=1691232390 Win=29312 Le..
35	2020-04-02 15:39:56.112024720	10.0.2.7	10.0.2.8	TCP	74	50216 → 8080 [SYN] Seq=3152820537 Win=29200 Len=0 MSS=1460 SA..
36	2020-04-02 15:39:56.116272764	10.0.2.8	10.0.2.7	TCP	74	8080 → 50216 [SYN, ACK] Seq=3736134138 Ack=3152820538 Win=289..
37	2020-04-02 15:39:56.116314347	10.0.2.7	10.0.2.8	TCP	66	50216 → 8080 [ACK] Seq=3152820538 Ack=3736134139 Win=29312 Le..
38	2020-04-02 15:39:56.117644126	10.0.2.7	10.0.2.8	TCP	74	50216 → 8080 [PSH, ACK] Seq=3152820538 Ack=3736134139 Win=293..
39	2020-04-02 15:39:56.117705840	10.0.2.7	10.0.2.8	TCP	66	50216 → 8080 [FIN, ACK] Seq=3152820546 Ack=3736134139 Win=293..
40	2020-04-02 15:39:56.118447611	10.0.2.8	10.0.2.7	TCP	66	8080 → 50216 [ACK] Seq=3736134139 Ack=3152820546 Win=29056 Le..

Frame 22: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 ▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
 ▶ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
 ▶ Transmission Control Protocol, Src Port: 50212, Dst Port: 8080, Seq: 2685729005, Ack: 3817824126, Len: 8

This indicates that we have achieved Load balancing using DNAT. In the above configuration, we see that the load is not completely balanced, and packets are lost. In order to achieve perfect load balance, we change the rules as follows:

```
[04/02/20]seed@VM:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -m statistic --mode random --probability .50 -j DNAT --to-destination 192.168.60.101:9091
[04/02/20]seed@VM:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT --to-destination 192.168.60.101:9092
[04/02/20]seed@VM:~$ sudo iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target    prot opt source          destination
DNAT      tcp  --  anywhere       anywhere        tcp dpt:http-alt statistic mode random pr
obability 0.500000000000 to:192.168.60.101:9091
DNAT      tcp  --  anywhere       anywhere        tcp dpt:http-alt to:192.168.60.101:9092

Chain INPUT (policy ACCEPT)
target    prot opt source          destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source          destination

Chain POSTROUTING (policy ACCEPT)
target    prot opt source          destination
[04/02/20]seed@VM:~$
```

Now, we send some traffic to the firewall on port 8000 from Machine B:

```
[04/02/20]seed@VM:~$ echo "Hello 2" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ echo "Hello 2" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ echo "Hello 2" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ 
```



```
[04/02/20]seed@VM:~$ echo "Hello 1" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ echo "Hello 1" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ echo "Hello 1" | nc 10.0.2.8 8080
[04/02/20]seed@VM:~$ 
```

On Machine A, we see that the packets are not lost anymore, and we achieved a perfect load balance:

The image shows two terminal windows side-by-side. The left terminal window has the title 'Terminal' and displays the command '[04/02/20]seed@VM:~\$ nc -lk 9091'. It receives three 'Hello 1' messages from the right terminal window. The right terminal window also has the title 'Terminal' and displays the command '[04/02/20]seed@VM:~\$ nc -lk 9092'. It receives three 'Hello 2' messages from the left terminal window. This indicates a balanced load where both hosts are receiving traffic.

Task 8: Connection Track

Considering the network connection as in Task 1. Before configuring any firewall rules on the Machine B, we run the following command on Machine A to telnet to Machine B and see that it is successful:

The terminal window shows the command '[04/02/20]seed@VM:~\$ telnet 10.0.2.8'. It prints 'Trying 10.0.2.8...', 'Connected to 10.0.2.8.', 'Escape character is '^]'.', 'Ubuntu 16.04.2 LTS', and 'VM login:'. This indicates a successful connection to the remote host.

Now we configure the following connection tracking rules that will not allow any externally initiated connections but only allow connections that were initiated from Machine B (internally). This is logical because we might want only the traffic that was initiated by the internal network.

The terminal window shows the configuration of iptables rules. It includes rules for the INPUT, FORWARD, and OUTPUT chains. The INPUT chain has rules for ACCEPTing established and related connections and REJECTING all other incoming connections. The FORWARD and OUTPUT chains both have an ACCEPT policy. The output shows the resulting table structure with target, protocol, options, source, destination, and ctstate information.

```
[04/02/20]seed@VM:~$ sudo iptables -A INPUT -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
[04/02/20]seed@VM:~$ sudo iptables -A INPUT -p tcp -j REJECT
[04/02/20]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
ACCEPT     tcp  --  anywhere        anywhere        ctstate RELATED,ESTABLISHED
REJECT    tcp  --  anywhere        anywhere        reject-with icmp-proto-unreachable

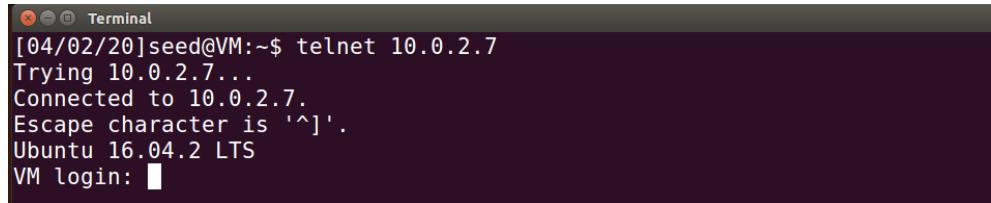
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
[04/02/20]seed@VM:~$
```

Now, on telnetting from Machine A (external) to B, we see that the telnet is no more successful:

The terminal window shows the command '[04/02/20]seed@VM:.../html\$ telnet 10.0.2.8'. It prints 'Trying 10.0.2.8...' and 'telnet: Unable to connect to remote host: Connection refused'. This indicates that the connection attempt failed due to the configured iptables rules.

On initiating the telnet connection from Machine B to A, we see that the connection is successful:



```
[04/02/20]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: ■
```

A screenshot of a terminal window titled "Terminal". The window shows the command "telnet 10.0.2.7" being run, followed by the output of the connection attempt. It shows the connection being established to "10.0.2.7", the escape character as '^]', and the operating system as "Ubuntu 16.04.2 LTS". The prompt "VM login:" is visible at the end of the output.

This proves that the firewall is now working as a stateful firewall that looks at the connection states and allow only those connections that were not initiated from the external network.