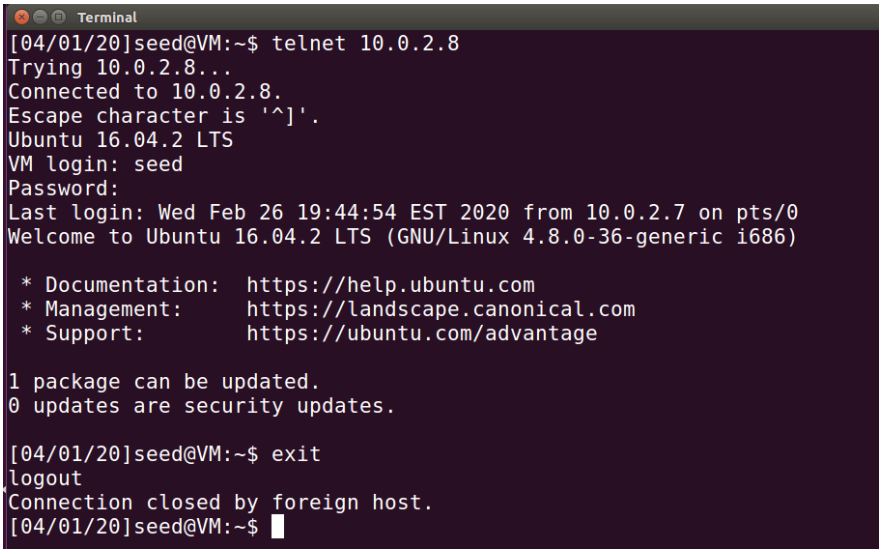
The following is the Network Setup in this lab for Task 1 through 4 and 8 (Same NAT network):

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

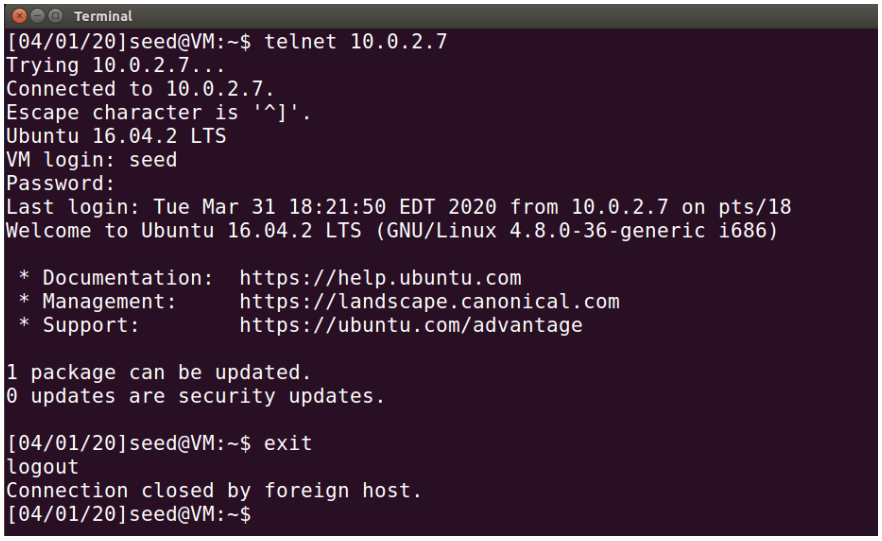
Task 1: Using Firewall

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:

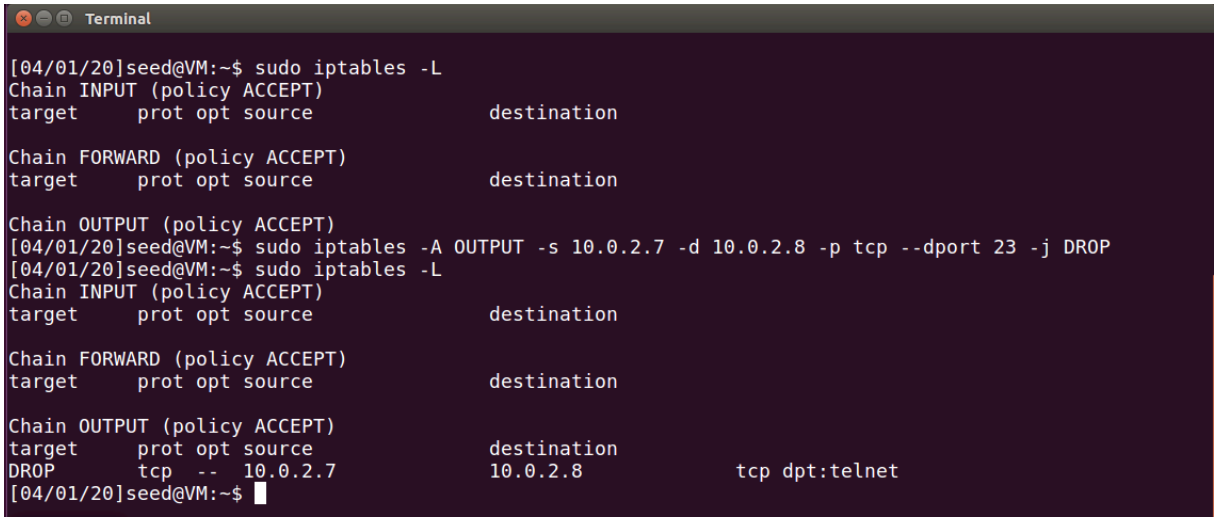
From Machine A to Machine B – Successful Telnet connection:



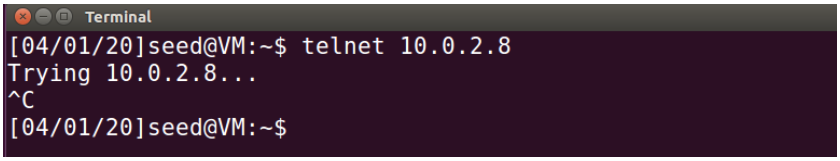
From Machine B to Machine A – Successful Telnet connection:



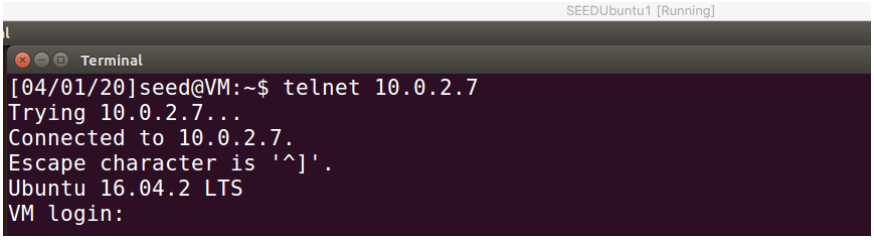
Now we set up a firewall rule to prevent Machine A from doing telnet to Machine B:



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

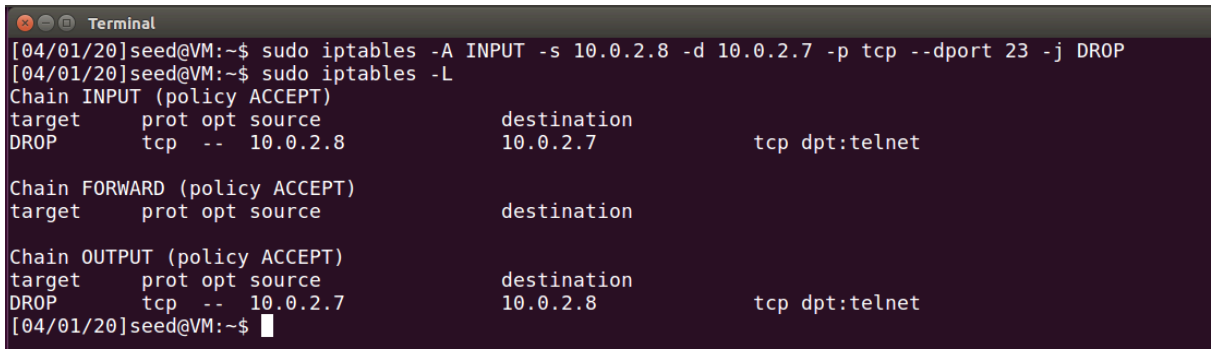


However, on telnetting from Machine B to A, we see that the telnet connection is successful:



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:



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.

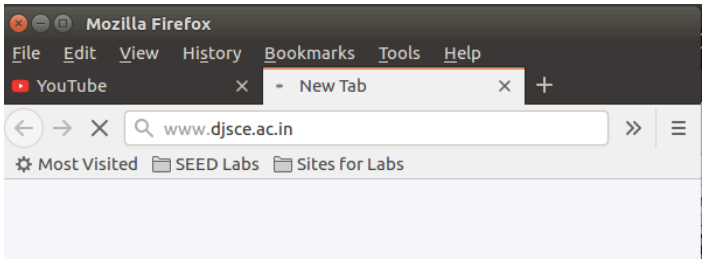
Now we set up a firewall rule to prevent Machine A from visiting an external web site. We use the following website for demonstration purposes: www.djsce.ac.in. We obtain an IP address for this website:

```
[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:

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/inet.h>

#define NIPQUAD(addr) ((unsigned char *)&addr)[0], ((unsigned char *)&addr)[1], ((unsigned char *)&addr)[2], ((unsigned char *)&addr)[3]

static struct nf_hook_ops nfho;
static struct nf_hook_ops nfho1;
static struct nf_hook_ops nfho2;
static struct nf_hook_ops nfho3;
static struct nf_hook_ops nfho4;

unsigned int telnet_outgoing(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph+iph->ihl*4;

    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")) {
        printk(KERN_INFO "Dropping Telnet Packet to destination address: %d.%d.%d.%d\n",NIPQUAD(iph->daddr));
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}

unsigned int ssh_outgoing(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph+iph->ihl*4;


    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")) {
        printk(KERN_INFO "Dropping SSH Packet to destination address: %d.%d.%d.%d\n",NIPQUAD(iph->daddr));
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}

unsigned int telnet_incoming(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph+iph->ihl*4;

    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")) {
        printk(KERN_INFO "Dropping Telnet Packet from source address: %d.%d.%d.%d\n",NIPQUAD(iph->saddr));
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}

unsigned int ssh_incoming(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph+iph->ihl*4;

    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")) {
        printk(KERN_INFO "Dropping SSH Packet from source address: %d.%d.%d.%d\n",NIPQUAD(iph->saddr));
        return NF_DROP;
    } else {
        return NF_ACCEPT;
```

```
    }
}

unsigned int web_block(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph+iph->ihl*4;

    if (iph->protocol == IPPROTO_TCP && iph->saddr == in_aton("10.0.2.7") && iph->daddr==in_aton("148.251.191.4") && (tcph->dest == htons(80) || tcph->dest == ht
        printk(KERN_INFO "Dropping Web Packet to web page on address: %d.%d.%d.%d\n",NIPQUAD(iph->daddr));
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}

int init_module()
{
    nfho.hook = telnet_outgoing; /* Handler function */
    nfho.hooknum = NF_INET_LOCAL_OUT;
    nfho.pf = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho);

    nfho1.hook = telnet_incoming; /* Handler function */
    nfho1.hooknum = NF_INET_LOCAL_IN; /* First hook for IPv4 */
    nfho1.pf = PF_INET;
    nfho1.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho1);

    nfho2.hook = web_block; /* Handler function */
    nfho2.hooknum = NF_INET_LOCAL_OUT; /* First hook for IPv4 */
    nfho2.pf = PF_INET;
    nfho2.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho2);

    nfho3.hook = ssh_outgoing; /* Handler function */
    nfho3.hooknum = NF_INET_LOCAL_OUT; /* First hook for IPv4 */
    nfho3.pf = PF_INET;
    nfho3.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho3);

    nfho4.hook = ssh_incoming; /* Handler function */
    nfho4.hooknum = NF_INET_LOCAL_IN; /* First hook for IPv4 */
    nfho4.pf = PF_INET;
    nfho4.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho4);

    return 0;
}
/* Cleanup routine */
void cleanup_module()
{
    nf_unregister_hook(&nfho);
    nf_unregister_hook(&nfho1);
    nf_unregister_hook(&nfho2);
    nf_unregister_hook(&nfho3);
    nf_unregister_hook(&nfho4);
}
```

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:

```
Terminal

[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.

```
Terminal
[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:

```
Terminal
[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:

```
Terminal
[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:

```
Terminal
[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:

```
Terminal
[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:

```
Terminal
[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:

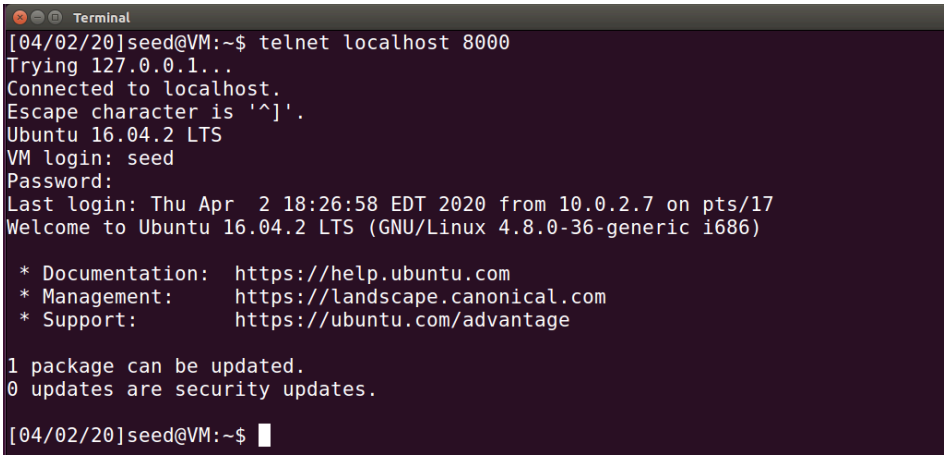
```
Terminal
[04/02/20]seed@VM:~$ telnet 10.0.2.8 23
Trying 10.0.2.8...
```

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:

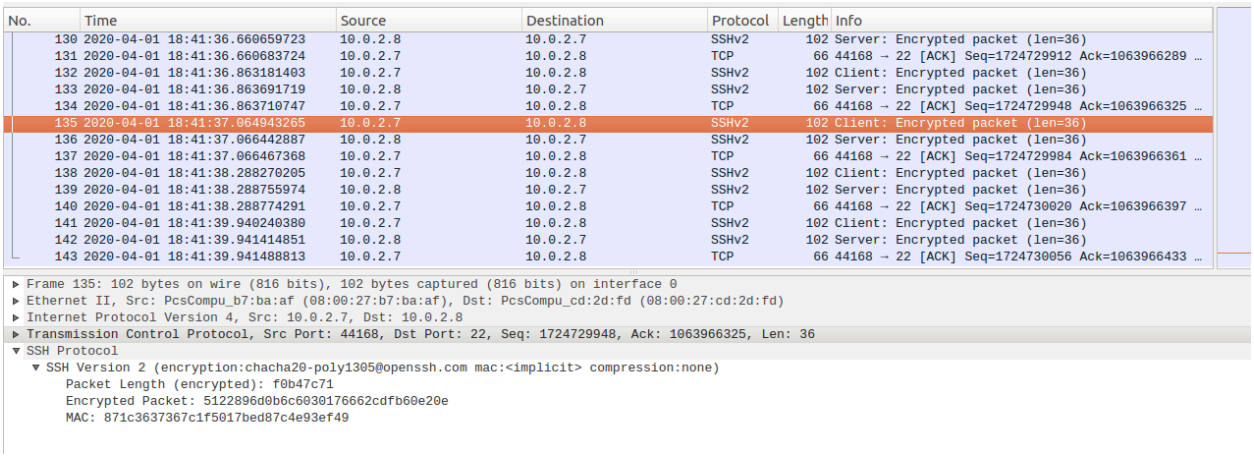
```
Terminal
[04/02/20]seed@VM:~$ ssh -L 8000:10.0.2.8:23 seed@10.0.2.8
seed@10.0.2.8's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * 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.



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

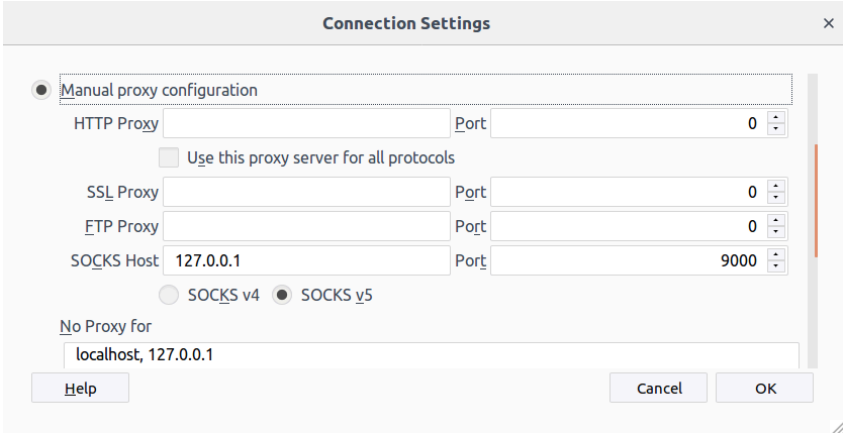


Task 3.b: Connect to Facebook using SSH Tunnel.

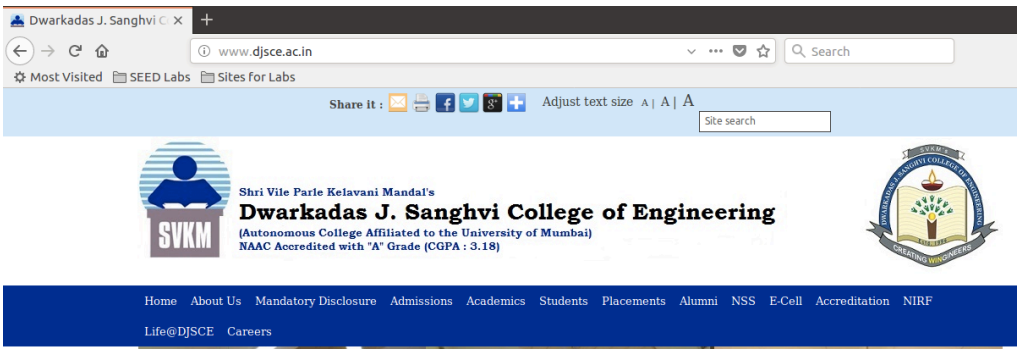
Another way of evading firewall rule is using Dynamic Port forwarding instead of the static one used in the previous task. In this, we only specify the local port number and not the destination. On the receiver's end, the packet is forwarded dynamically based on the destination information in the packet.
We achieve this using the following command. This command establishes an SSH tunnel between localhost port 9000 and Machine B port 22:

```
Terminal
[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:

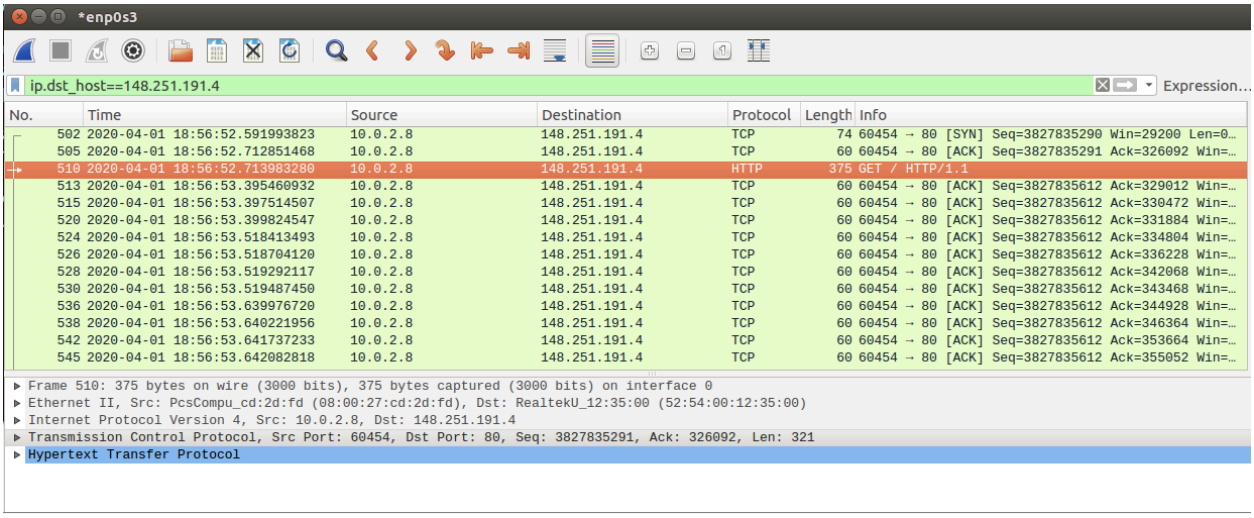


We use the SOCKS proxy because we need to let the browser tell the proxy about the destination in order to dynamically forward the traffic, since this proxy does not have the capability of detecting the destination on its own. After this is set, we visit the blocked web page and see that the website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

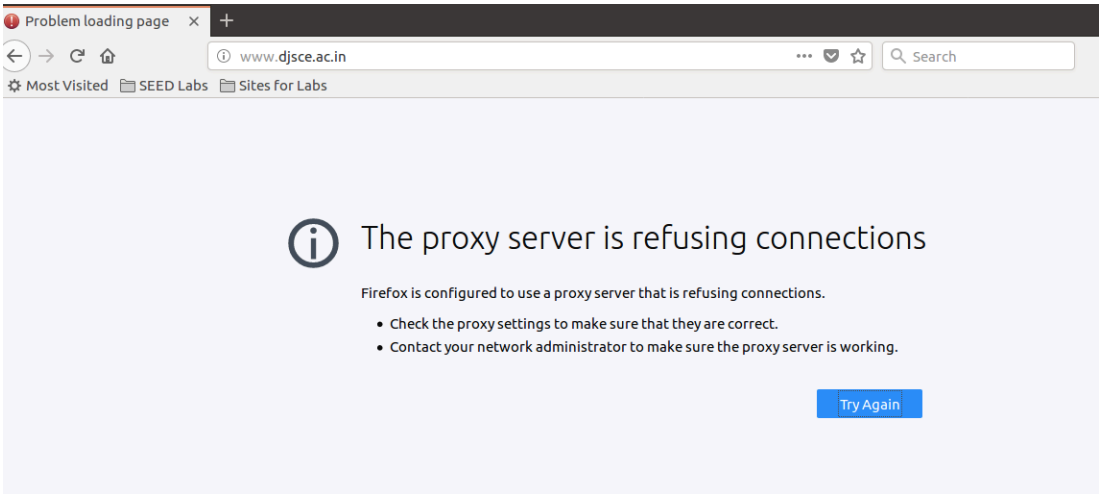


On looking at the Wireshark traffic, we see that the traffic to the website goes through Machine B and not Machine A. This is because the web traffic goes via the SSH tunnel between Machine A and Machine B,

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.

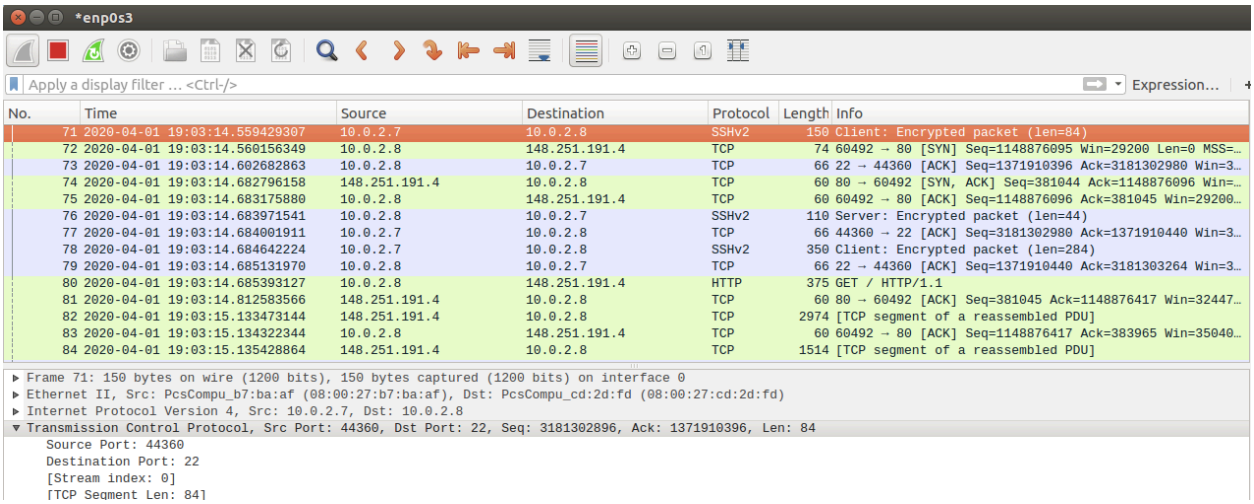


Now, on breaking the SSH tunnel and reloading the page, we see that we are no more able to load the webpage. We see the error that the proxy server is refusing connections. This is because Machine A and B are no more connected via SSH, and this SSH tunnel was the proxy server for the browser.



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.

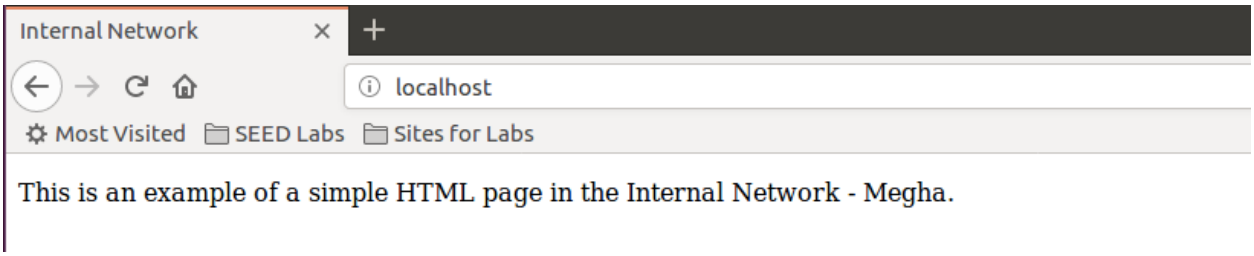
In this task, the browser connects to the SSH proxy at port 9000 on the localhost, and the SSH sends the TCP data over the tunnel to Machine B, which connects to the blocked website (based on the destination). This SSH tunnel responds back via the same tunnel and since all the traffic is SSH and not 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.



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:

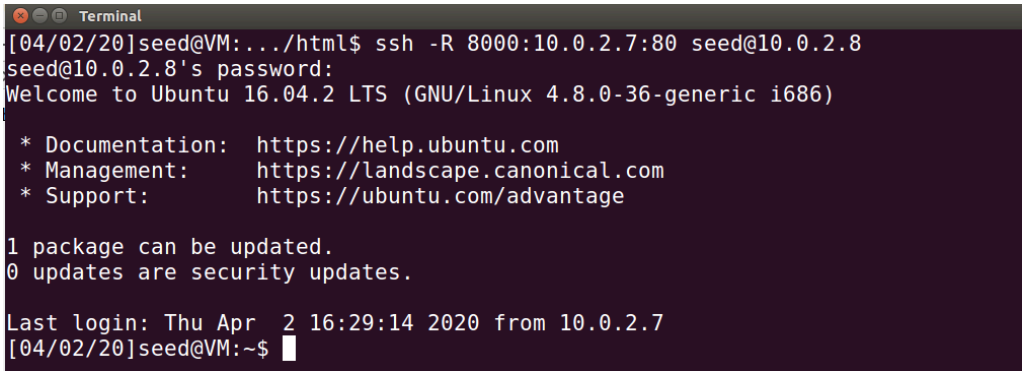


The following show the firewall rules configured on Machine A:

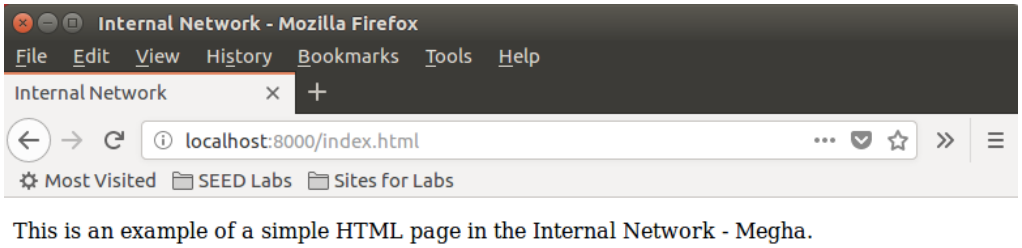
```
[04/02/20]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.8 -d 10.0.2.7 -p tcp --dport 80 -j DROP
[04/02/20]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.8 -d 10.0.2.7 -p tcp --dport 22 -j DROP
[04/02/20]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination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



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:

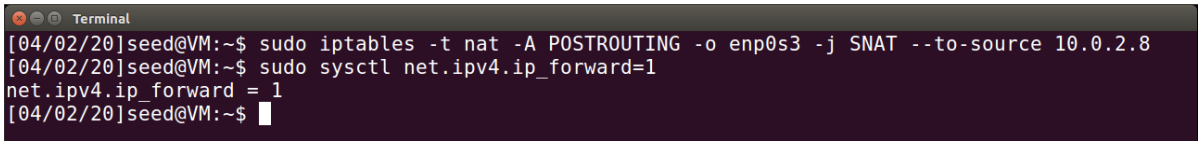


Network Setup for the Task 5 – 7 is as following:

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:

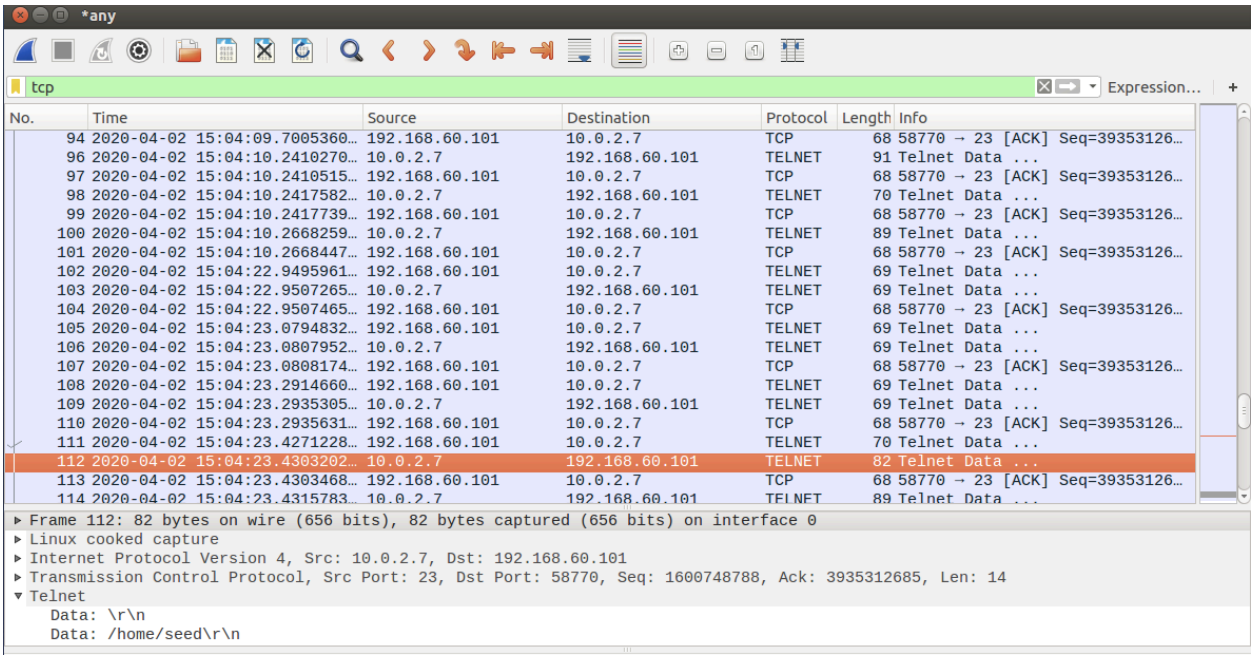


This rule states that for any packet going out of the interface enp0s3, set the source IP address as 10.0.2.8. Now, on telnetting from Machine A to Machine B (internal to external network), we see that the connection is successfully established:

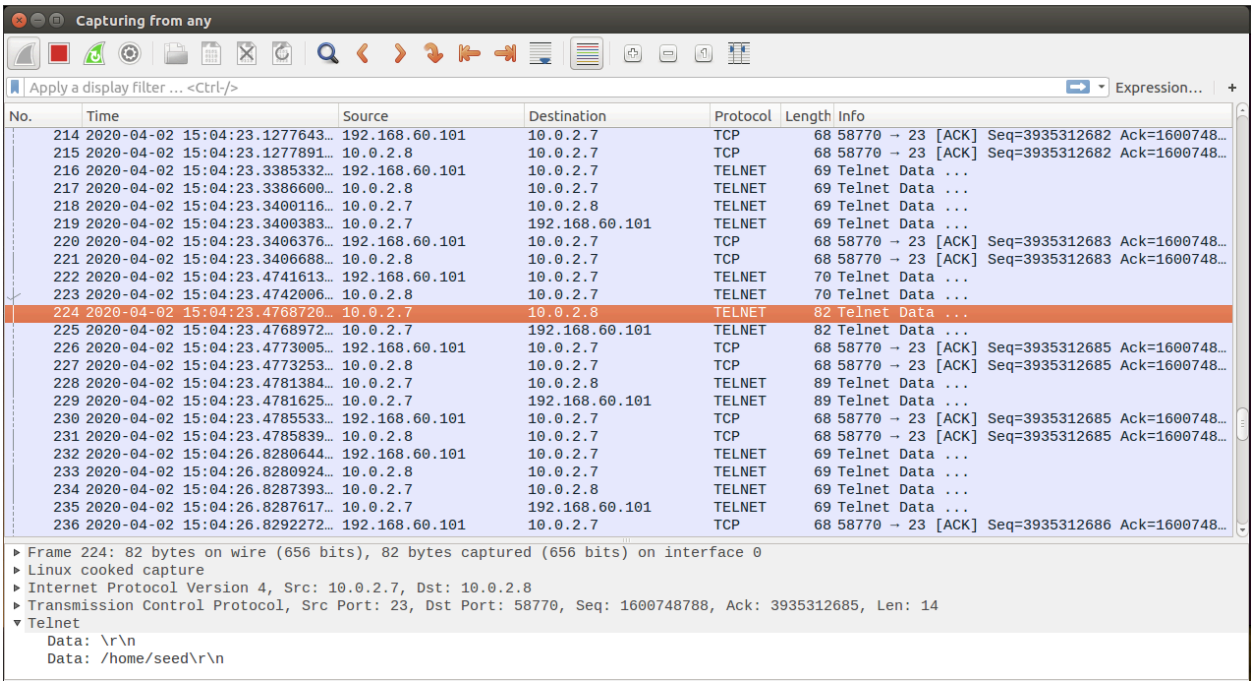
```
Terminal

[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:

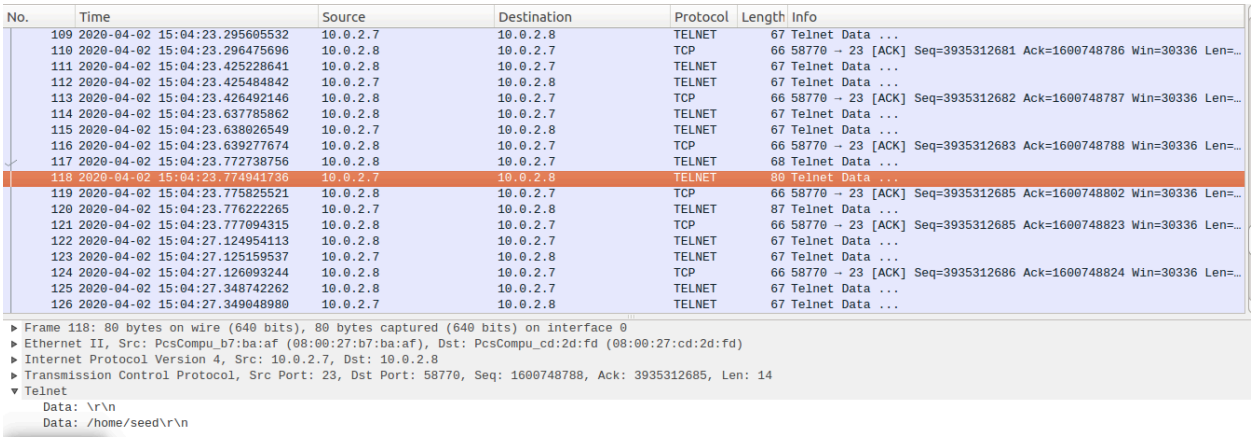
On Machine A, we see that the traffic goes between Machine A to Machine B, but since the destination is of a different network, the packet goes to the gateway – our firewall:



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.



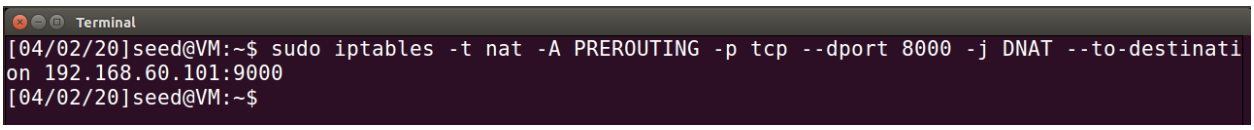
On Machine B, we see that the communication is purely between Machine B and the firewall:



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.

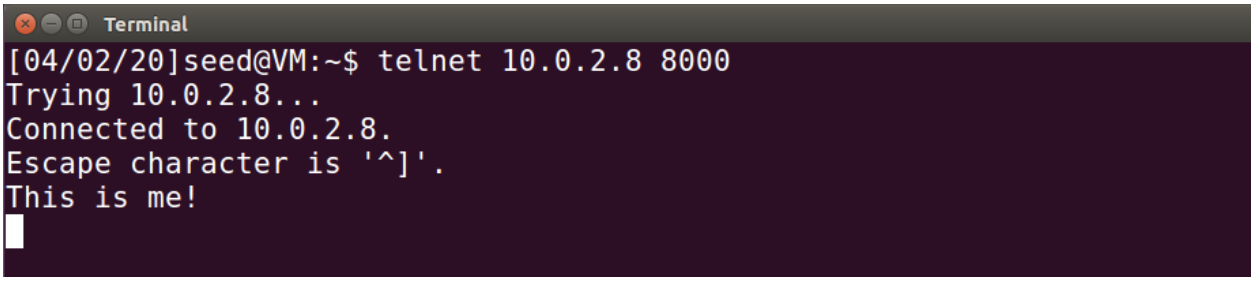
Task 6: DNAT for Port Forwarding

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

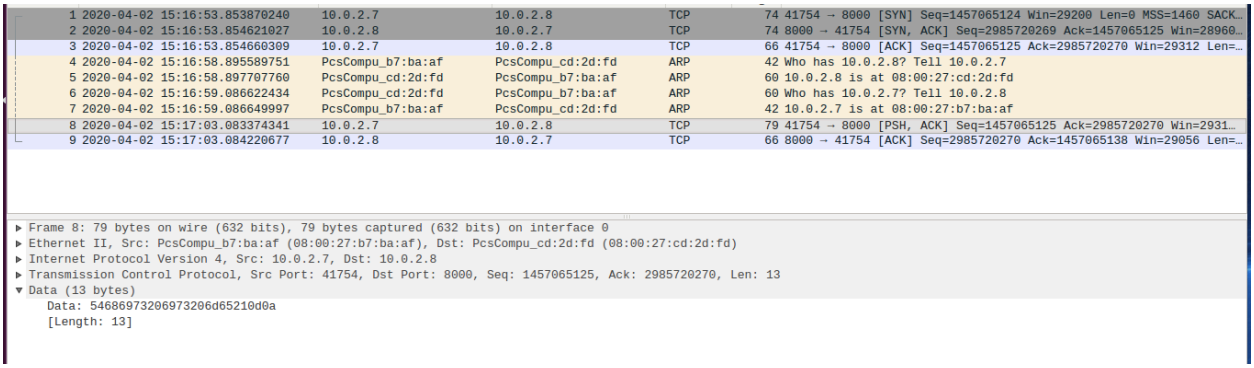


This rule will route any traffic received on Firewall port 8000 to Machine A's port 9000.

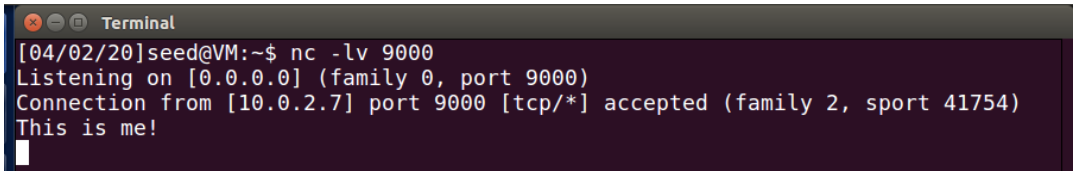
Now to test the configuration, we telnet to the Firewall on port 8000 from Machine B. Before telnetting, we start a server on Machine A listening to port 9000. We see that the telnet is successful:



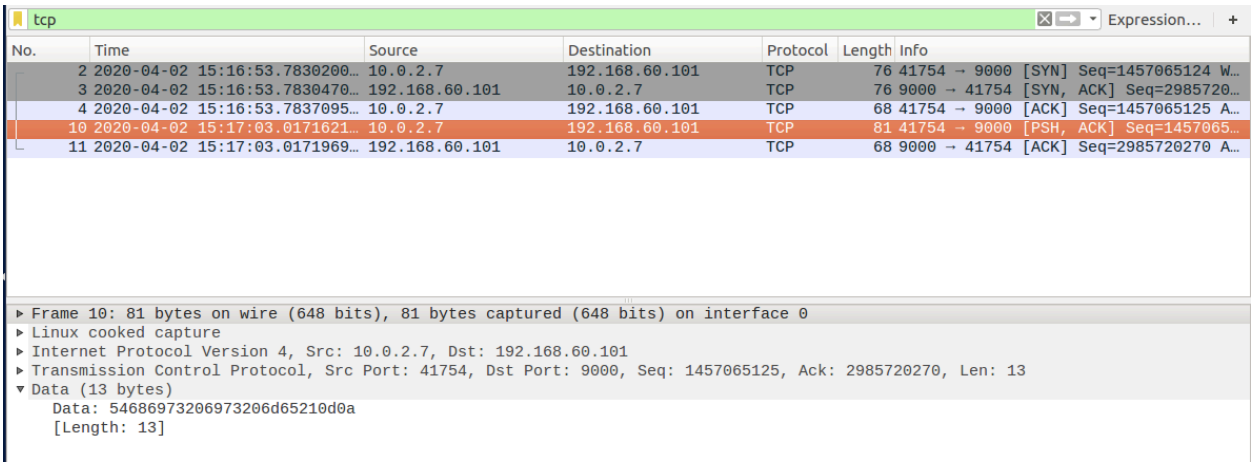
On looking at the Wireshark trace on Machine B, we see that the traffic goes between Machine B and Firewall's port 8000.



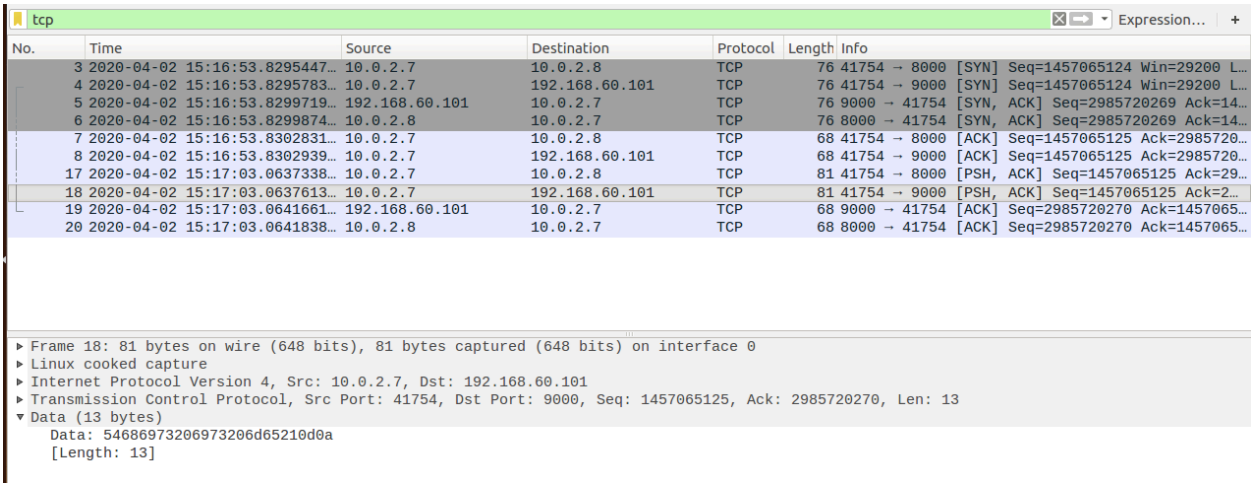
Now on looking at the connection on Internal Machine A, we see that the connection is the established with Machine A:



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



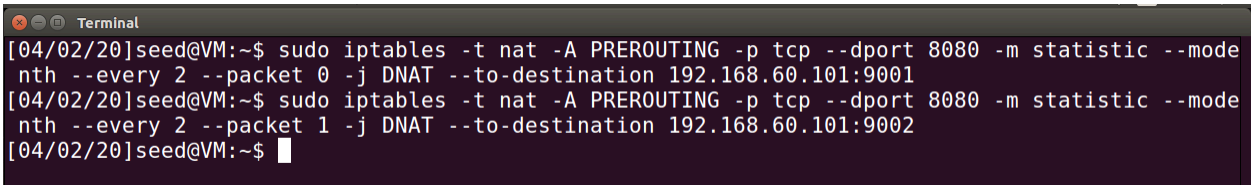
This proves that Machine A and B are connected using the telnet connection. On looking at the Wireshark trace on the Firewall, we see that the traffic comes from Machine B to the firewall and then the firewall forwards this packet to Machine A (port 9000) as configured.



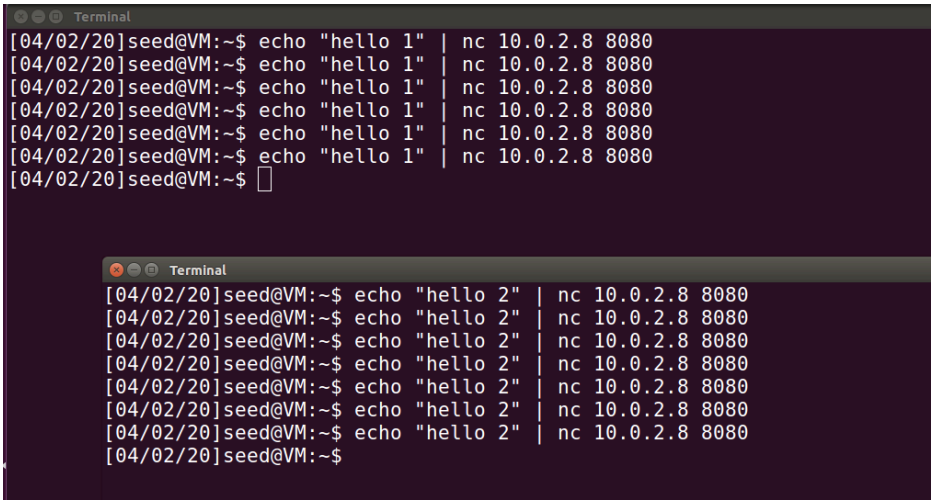
This proves that the DNAT port forwarding is successful.

Task 7: DNAT for Load Balancing

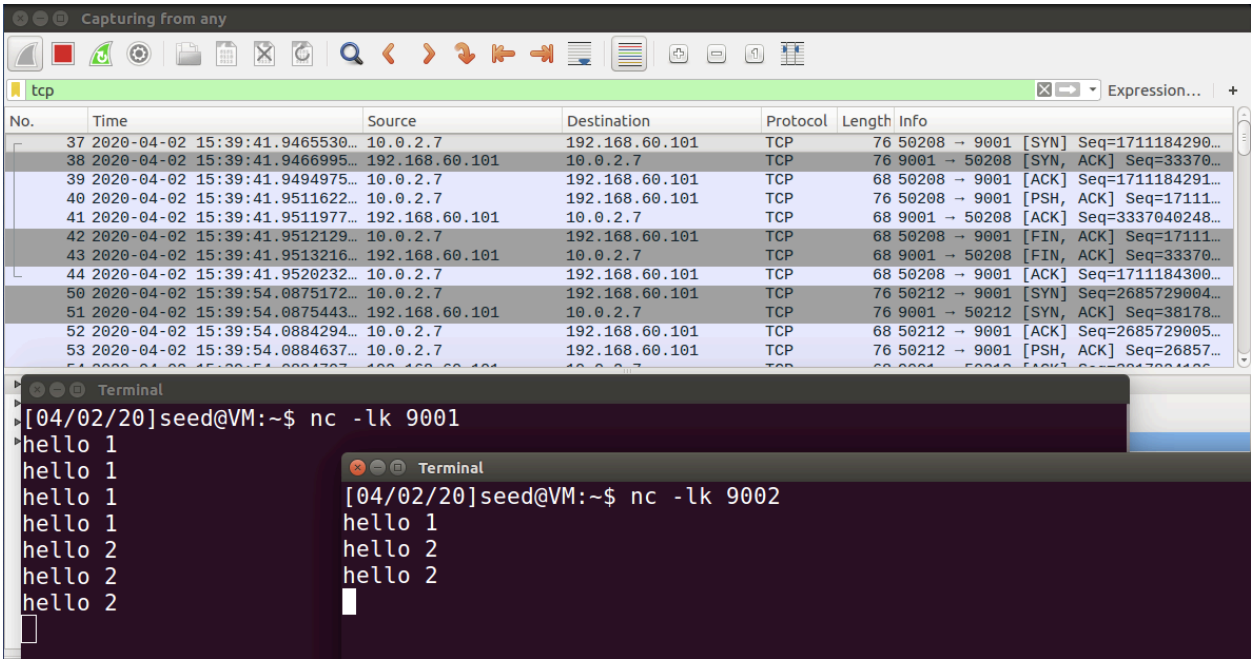
Now, we use the DNAT to perform Load Balancing. The following two rules are configured on the firewall which will forward the traffic with certain randomness to different Machines (here port):



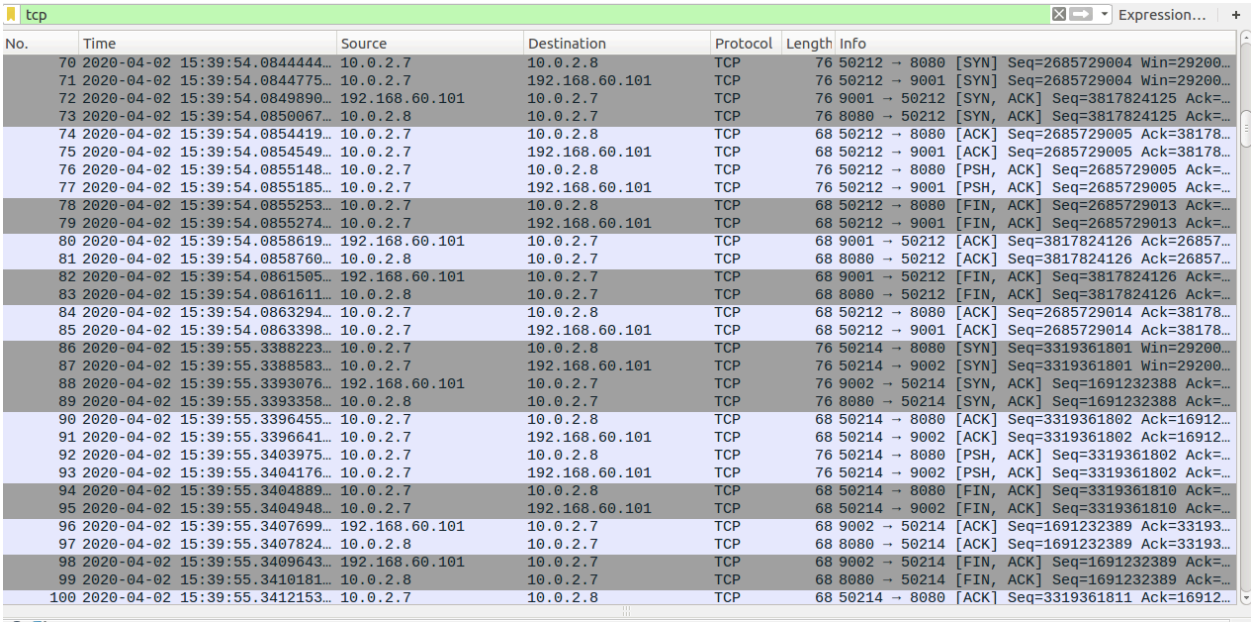
On Machine A, we start two servers accepting multiple connections on port 9001 and 9002. In order to test the load balancing, we then run the following commands on Machine B that sends traffic to the firewall on port 8000:



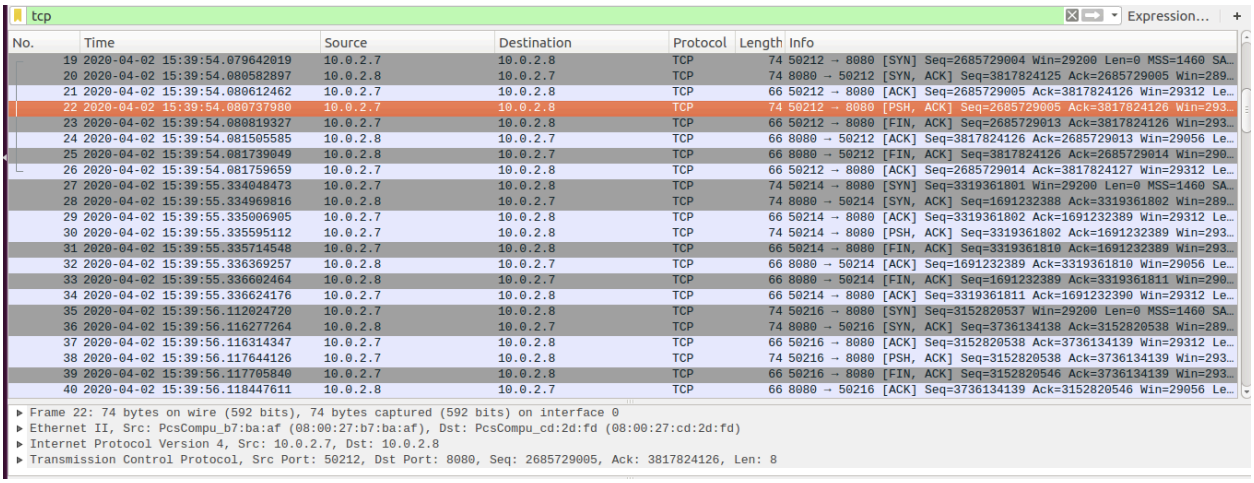
The following screenshot on Machine A indicates that even though the traffic was sent from Machine B to the firewall, it was forwarded to Machine A's servers running on different ports:



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).

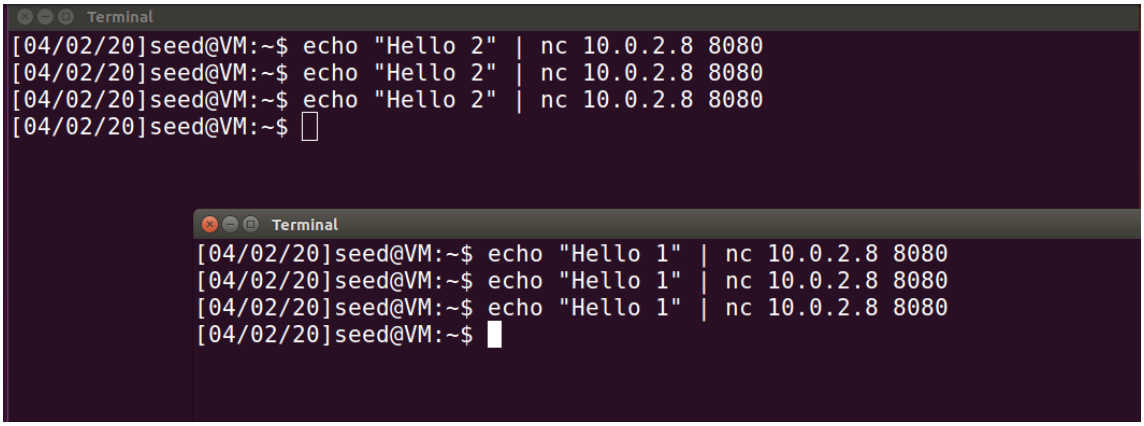


The following shows that on Machine B, the traffic is purely communicated between Machine B and the firewall and Machine B has no idea about the traffic being distributed and forwarded to different ports on a different Machine.

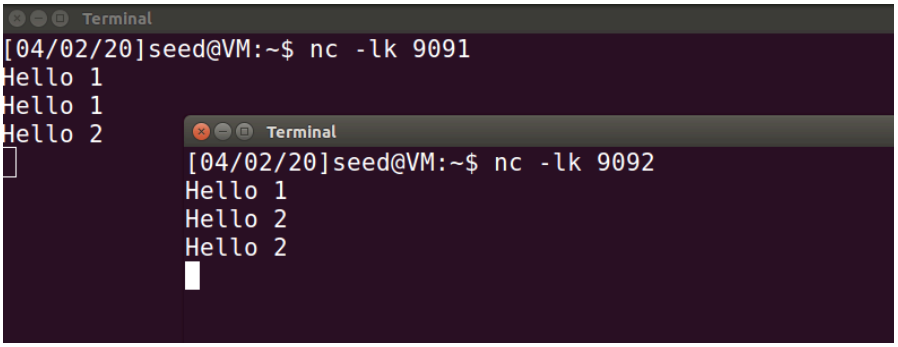


This indicates that we have achieved Load balancing using DNAT. In the above configuration, we see that the load is not completely balanced, and packets are lost. In order to achieve perfect load balance, we change the rules as follows:

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

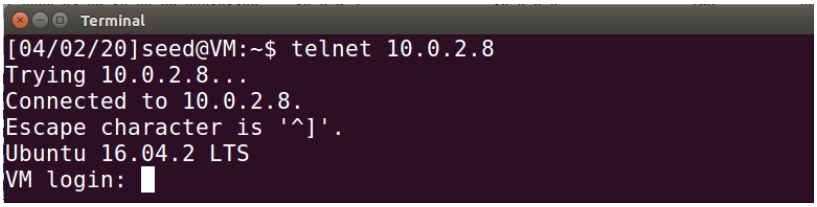


On Machine A, we see that the packets are not lost anymore, and we achieved a perfect load balance:

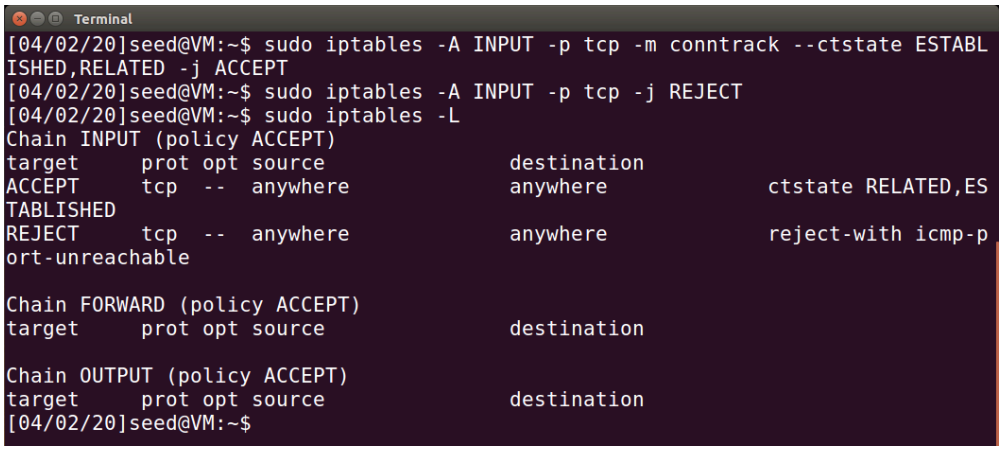


Task 8: Connection Track

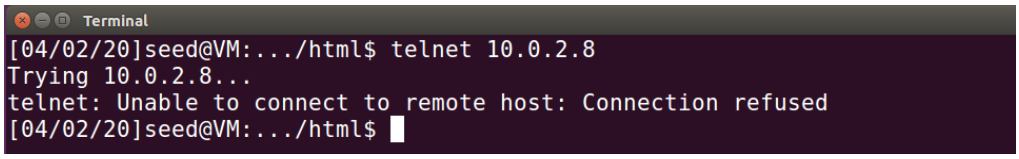
Considering the network connection as in Task 1. Before configuring any firewall rules on the Machine B, we run the following command on Machine A to telnet to Machine B and see that it is successful:



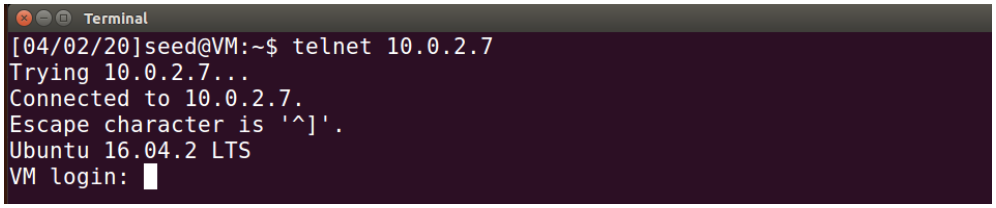
Now we configure the following connection tracking rules that will not allow any externally initiated connections but only allow connections that were initiated from Machine B (internally). This is logical because we might want only the traffic that was initiated by the internal network.



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



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



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.