Experimenting Internetworking using Linux Virtual Machines – Part I

Hui Chen

Previous Release on October 27, 2014
Lastly revised on November 4, 2015
Revision:

Contents

1 Overview **2**

2 Part I: Building an IPv4 Internetwork **2**
2.1 Software . . . 3
2.2 Preparation . . . 4
2.2.1 Linux Virtual Machines Settings . . . 4
2.2.2 Installing Tcpdump . . . 4
2.2.3 Making Clones . . . 5
2.2.4 Disabling IPv4 . . . 5
2.3 Network Planning . . . 6
2.4 Implementation using Linux Virtual Machines . . . 6
2.4.1 Configuration on R1 (VM 1) . . . 6
2.4.2 Configuration on R2 (VM 2) . . . 8
2.4.3 Configuration on H1 (VM 3) . . . 9
2.5 Reexamining Routing Table Update . . . 9
2.6 Making Changes Permanent . . . 13
2.7 Dealing with Regrets . . . 14

3 Remaining Issues **14**

4 Practice Assignment **15**

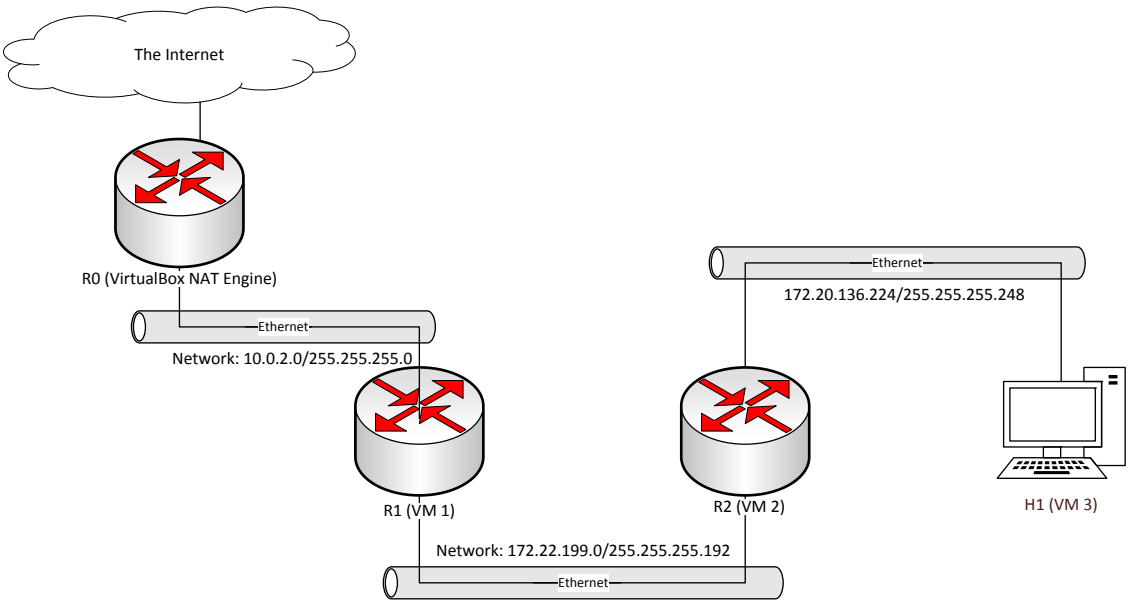

Figure 1: The layout of a planned network and its existing network infrastructure. The existing network infrastructure consists of VirtualBox NAT engine and host network setup.

1 Overview

Virtualization software such as Oracle VM VirtualBox and VMware Player makes it conveniently for one to experiment with internetworks, i.e., experiment with network planning and net implementation.

Through a series of experiments, we will practice to plan a few networks, to connect the networks to form an internetwork, and to implement the internetwork using a few Linux virtual machines. The series of experiments, consisting of Part I, Part II, Part III, and Part IV, has the following main objectives, respectively,

1. plan and implement a *standalone* IPv4 internetwork of linear topology where *standalone* means that the internetwork is not configured to connected to the Internet and can be considered as an isolated *intranet* (in Part I);

2. connect the IPv4 internetwork to the Internet using IPv4 Masquerading (in Part II);

3. repeat experiment 1, however, in IPv6 (in Part III); and

4. repeat experiment 2, however, in IPv6 (in Part IV).

2 Part I: Building an IPv4 Internetwork

In this experiment, we will plan and implement a simple internetwork as illustrated in Figure 1, and implement it using Linux virtual machines as shown in Figure 2.

The main objectives of this experiment are,

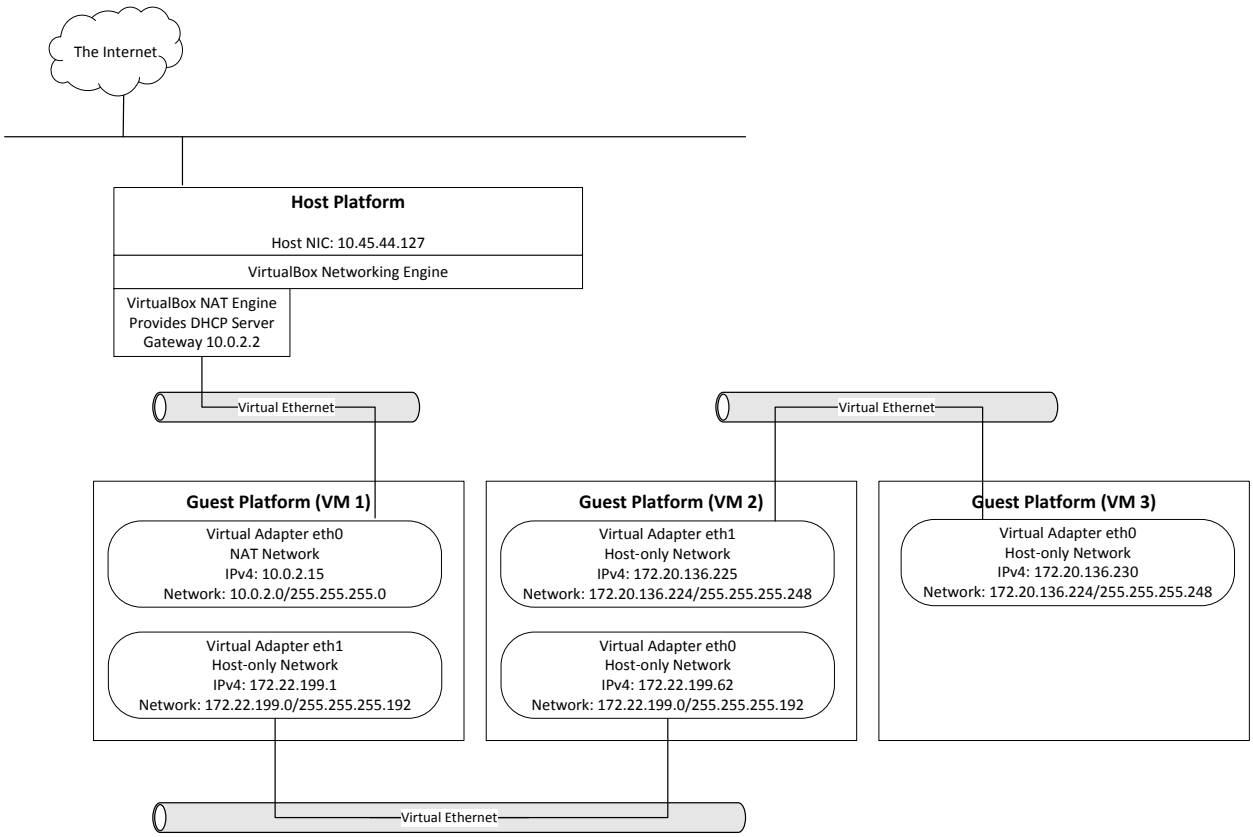

Figure 2: An internetwork consisting of 3 virtual Linux hosts.

- to plan networks by determining IPv4 address spaces including network number and network mask;
- to assign IPv4 addresses to Linux hosts on the networks using Linux commands;
- to plan and implement routing tables on the Linux hosts using Linux commands;
- to configure packet forwarding for IPv4 on hosts serving network gateways (i.e., routers); and
- to diagnose and test the networks.

2.1 Software

Although the principle of this instruction is applicable to different virtualization software, e.g. VMware Player and different Linux distributions, e.g., Fedora Linux, this instruction is tested used the following components,

- Windows 8.1 as host operating systems
- Oracle VM VirtualBox version 4.2.16 and above runing on the host
- Debian Linux 8.x as guest operating systems

The Linux commands will be used in this experiment are, `route`, `ip`, `ping`, `sysctl`, and `tcpdump`.

The Linux system configuration files that of our concern are `/etc/network/interfaces` and `/etc/sysctl.conf`.

2.2 Preparation

Before we begin actual work, we will prepare and configure 3 virtual machines.

2.2.1 Linux Virtual Machines Settings

The 3 virtual machines will run on a single host computer. Therefore, you host computer must have sufficient RAM and hard drive space. Each virtual machine is configured with *64MB* RAM and this instruction is tested on a Windows 8.1 host with *4GB* RAM.

The network settings of the 3 Linux virtual machine images in VirtualBox are as follows,

- *VM 1* has two Ethernet adapters, one in the NAT mode, and the other in the Internal Network mode;
- *VM 2* has two Ethernet adapters and both are in the Internal Network mode and the name of the Ethernet the two adapters are on is `ineten1`; and
- *VM 3* has one Ethernet adapter that is in the Internal network mode and the name of the Ethernet of the adapter is on is `ineten1`.

Figure 3 shows the settings of the two adapters in *VM 1*.

We choose the Internal Network mode because as indicated in [1],

> "The internal network (in this example `ineten1`) is a totally isolated network and so is very 'quiet'. This is good for testing when you need a separate, clean network, and you can create sophisticated internal networks with vm's that provide their own services to the internal network. (e.g. Active Directory, DHCP, etc). Note that not even the Host is a member of the internal network, but this mode allows vm's to function even when the Host is not connected to a network (e.g. on a plane)."

Note that it also implies that you cannot reach the host, let alone the outside network from using the adapters put on the internal network mode without some "additional" help from other nodes. An objective of this experiment is to connect these adapters on Ethernet `ineten1` via a gateway node, i.e., *VM 1*.

We install `Debian Linux 8` on each of the virtual machines. You can download the base virtual machine image from from either Dropbox or OneDrive.

2.2.2 Installing Tcpdump

Note that the base virtual machine image has not had `tcpdump` installed. Before you make any clones, install `tcpdump`, e.g.,

```
sudo apt-get install tcpdump
```

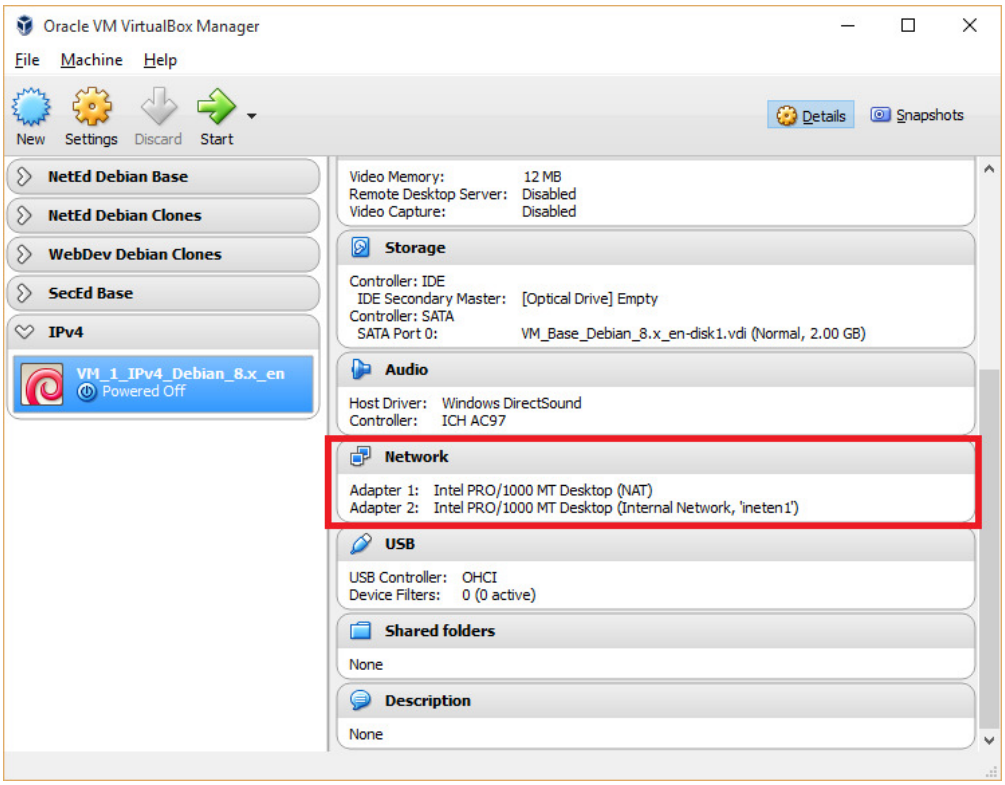

Figure 3: Configurations of VirtualBox virtual machine 1 (VM 1)

Listing 1: Statements in */etc/sysctl.conf* that Disables IPv6

```
# disable IPv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

2.2.3 Making Clones

You must change their settings in Oracle Virtual Box to match the required settings. It is recommended that you create linked clones from the base images and use the lined clones for this experiment. See the insturctor's VM Setup document for more information.

2.2.4 Disabling IPv4

Since we are to experiment on IPv6 in later experiments, we would desire a clean setup by configuring the network adapters to be IPv4 only, for which, we add the following statements to the end of */etc/sysctl.conf* as shown in Listing 1,

To make it effective without a reboot, execute the following command,

```
sudo sysctl -p /etc/sysctl.conf
```

To check if IPv6 is disabled, execute `ip address show` command and the result should not contain any reference to `inet6`.

2.3 Network Planning

Let us assume that we would like to set up two new networks for an organization. One needs to support about 60 hosts on the network and the other needs about 6 hosts.

- *Network 1.* It can support about 60 hosts on the network. Since $2^6 = 64$, the network mask can be $0xffffffc0$, or 255.255.255.192 in dot-decimal notation. Examining the available blocks of IPv4 address within the organization, you may conclude that the network can be 172.22.199.0/255.255.255.192, or 172.22.199.0/26, i.e., network 172.22.199.0 with network mask 255.255.255.192. The network can actually suport $2^6 - 2 = 64 - 2 = 62$ hosts. Two addresses must be excluded from the count as explained below,
  - 172.22.199.0 must be excluded from the available addresses to be allocated to hosts to avoid a confusion because 172.22.199.0 is reserved as the network number.
  - 172.22.199.63 must be exluced from the available addresses to be allocated to hosts because 172.22.199.63 whose bits for host numbers are all 1's is reserved as the broadcast address for the network.
- *Network 2.* It can support about 6 hosts on the network. Since $2^3 = 8$, the network mask can be $0xfffffff8$, or 255.255.255.248 in dot-decimal notation. Examining the available blocks of IPv4 addresses within the organization, you may conclude that the network can be 172.20.136.224/255.255.255.248, or 172.20.136.224/29. Similarly as *Network 1*, The number of hosts on the network can be $2^3 - 2 = 8 - 2 = 6$.

Based on the above, we plan the internet as in Figure 1. Network 1 and Network 2 are connected by Router $R2$; and Network 1 and the existing infrastructure network is connected by Router $R1$.

2.4 Implementation using Linux Virtual Machines

The implementation of the above internetwork design can be demonstrated using Linux virtual machines. As described in subsection 2.2, we prepare 3 Ubuntu 14.04 virtual machines, i.e., VM 1, VM 2, and VM 3. VM 1 is serving as $R1$, VM 2 is serving as $R2$, and VM 3 a host on Network 2, i.e., $H1$. The result will be Figure 2.

2.4.1 Configuration on R1 (VM 1)

The configuration on VM 1 consists of two parts, (1) IPv4 address assignment, and (2) routing table update. We will discuss item 2 in a greater detail in Section 2.5. This subsection discusses item 1. Inevitably, we make mistakes when we attempt to implement the network design. Section 2.7 is dedicated to the discussion on how to undo changes and correct mistakes.

IPv4 address assignment. We will assign an IPv4 address on Network 1, i.e., an address belonging to Network 172.22.199.0/255.255.255.192 to the adapter that is supposed to be on the network, i.e., *eth1* in VM 1. We choose IPv4 address 172.22.199.1. You can also use the *ip* command as shown in Listing 2 or shown in Listing 3.

Listing 2: Assign IPv4 Address to Adapter *eth1* on VM 1

```
sudo ip addr add 172.22.199.1/255.255.255.192 brd + dev eth1
```

Listing 3: Assign IPv4 Address to Adapter *eth1* on VM 1

```
sudo ip addr add 172.22.199.1/26 brd + dev eth1
```

Listing 4: Showing Settings of Network Interfaces at VM 1 Using *ip*

```
user@VM-1:~$ ip addr show eth1
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    qlen 1000
    link/ether 08:00:27:83:11:cc brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    qlen 1000
    link/ether 08:00:27:6f:2c:53 brd ff:ff:ff:ff:ff:ff
    inet 172.22.199.1/26 brd 172.22.199.63 scope global eth1
       valid_lft forever preferred_lft forever
user@VM-1:~$
```

Listing 5: Showing Routing Table in VM 1

```
user@VM-1:~$ route -n
Kernel IP routing table
Destination   Gateway   Genmask         Flags Metric Ref Use Iface
0.0.0.0       10.0.2.2  0.0.0.0         UG    0      0   0   eth0
10.0.2.0      0.0.0.0   255.255.255.0   U     1      0   0   eth0
172.22.199.0  0.0.0.0   255.255.255.192 U     0      0   0   eth1
user@VM-1:~$
```

It is important that you always verify the result of your actions, for which you can display the result of the IPv4 address assignment. You can use *ip* as shown in Listing 4.

When you add an IPv4 address to an adapter, the machine's routing table will also be updated. You can use either the *route* command or the *ip* command to show and manipulate routing tables. *Be aware that many consider that the* route *command to be deprecated, although this document only shows examples using the* route *command when we show and manipulate routing tables.* Listing 5 shows that an entry, i.e., Line 6 is inserted for Network 172.22.199.0/255.255.255.192. The line indicates that the next-hop router or the network gateway is 0.0.0.0, i.e., this network is the network that the network adapter belongs to, as such, all the hosts in the network would be on the direct-link network and can reach each other directly without packet forwarding.

Listing 6: Assign IPv4 Addresses to Two Adapters Using *ip* on VM 2

```
user@VM-2:~$ sudo ip addr add 172.22.199.62/255.255.255.192 brd + dev eth0
user@VM-2:~$ sudo ip addr add 172.20.136.225/255.255.255.248 brd + dev eth1
user@VM-2:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    qlen 1000
    link/ether 08:00:27:c9:2d:43 brd ff:ff:ff:ff:ff:ff
    inet 172.22.199.62/26 brd 172.22.199.63 scope global eth0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    qlen 1000
    link/ether 08:00:27:84:f8:e3 brd ff:ff:ff:ff:ff:ff
    inet 172.20.136.225/29 brd 172.20.136.231 scope global eth1
       valid_lft forever preferred_lft forever
user@VM-2:~$
```

Listing 7: Testing Network Connectivity between VM 1 and VM 2

```
user@VM-2:~$ ping -c 1 172.22.199.1
PING 172.22.199.1 (172.22.199.1) 56(84) bytes of data.
64 bytes from 172.22.199.1: icmp_seq=1 ttl=64 time=0.420 ms

--- 172.22.199.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.420/0.420/0.420/0.000 ms
user@VM-2:~$
```

Listing 8: Showing Routing Table on VM 2 After IPv4 Addresses were Assigned

```
user@VM-2:~$ route -n
Kernel IP routing table
Destination     Gateway Genmask         Flags Metric Ref Use Iface
172.20.136.224  0.0.0.0 255.255.255.248 U     0      0      0 eth1
172.22.199.0    0.0.0.0 255.255.255.192 U     0      0      0 eth0
user@VM-2:~$
```

2.4.2 Configuration on R2 (VM 2)

Similar as before, the configuration on VM 2 consists of two parts, (1) IPv4 address assignment, and (2) routing table update.

IPv4 Address Assignment. We need to assign addresses to the two adapters, one on Network 172.22.199.0/255.255.255.192 and the other on Network 172.20.136.224/255.255.255.248. Listing 6 shows the steps to assign the addresses.

If you assign IPv4 addresses correctly, you should be able to reach VM 1 since adapter *eth1* on VM 1 and adapter *eth0* on VM 2 are on the same direct link network. We can verify it by using *ping* as shown in Listing 7.

Two entries were also inserted to the routing table as shown in Listing 8. The same as before, the gateways' being 0.0.0.0 indicates that the adapters belong to the two networks, respectively.

Listing 9: Assign IPv4 Addresses to Adapters Using *ip* on VM 3

```
user@VM-3:~$ sudo ip addr add 172.20.136.230/255.255.255.248 brd + dev eth0
user@VM-3:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    qlen 1000
    link/ether 08:00:27:c9:2d:43 brd ff:ff:ff:ff:ff:ff
    inet 172.20.136.230/29 brd 172.20.136.231 scope global eth0
       valid_lft forever preferred_lft forever
```

Listing 11: Testing Network Connectivity between VM 2 and VM 3

```
user@VM-3:~$ ping -c 1 172.20.136.225
PING 172.20.136.225 (172.20.136.225) 56(84) bytes of data.
64 bytes from 172.20.136.225: icmp_seq=1 ttl=64 time=0.476 ms

--- 172.20.136.225 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.476/0.476/0.476/0.000 ms
user@VM-3:~$
```

2.4.3 Configuration on H1 (VM 3)

We assign an IPv4 address on Network 172.20.136.224/255.255.255.248 to adapter *eth0* on VM 3 as shown in Listing 9.

One entry was also inserted to the routing table as shown in Listing 10. The same as before, the gateway's being 0.0.0.0 indicates that the adapter belong to the network. You can now verify that you can reach VM 2 using *ping* as shown in Listing 11.

Listing 10: Showing Routing Table on VM 3 After IPv4 Address was Assigned

```
user@VM-3:~$ route -n
Kernel IP routing table
Destination     Gateway Genmask         Flags Metric Ref Use Iface
172.20.136.224 0.0.0.0 255.255.255.248 U     0      0      0 eth0
user@VM-3:~$
```

2.5 Reexamining Routing Table Update

Previously, we show that you can reach VM 2 from VM 3 and vice versa. We can make the same argument for VM 1 and VM 2 as well. Note that the connectivty is established via the two adapters on the two machines that are part of a direct-link network, i.e., eth1 on VM 2 and eth0 on VM 3 belong to a direct-link network. *Can you reach VM 2 from VM 3 using the IPv4 address belonging to the other adapter on VM 2, i.e., eth0 on VM 2?* The answer is *No* as shown in List 12.

Why? Examining the routing table shown in Listing 10, we realize that the table does not contain an entry for Network 172.22.199.62/255.255.255.192. Therefore, VM 3 does not know how to forward packets to the network, i.e, *the network is indeed unreachable*. How to fix this problem? We can fix it by adding the network into the routing table. Listing 13 shows that we added the

Listing 12: Testing Network Connectivity between VM 2 and VM 3

```
user@VM-3:~$ ping -c 1 172.22.199.62
connect: Network is unreachable
user@VM-3:~$
```

Listing 13: Updating Routing Table on VM 3

```
user@VM-3:~$ sudo route add -net 172.22.199.0 netmask 255.255.255.192 gw 172.20.136.225
user@VM-3:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
172.20.136.224  0.0.0.0         255.255.255.248 U     0      0      0 eth0
172.22.199.0    172.20.136.225  255.255.255.192 UG    0      0      0 eth0
user@VM-3:~$ ping -c 1 172.22.199.62
PING 172.22.199.62 (172.22.199.62) 56(84) bytes of data.
64 bytes from 172.22.199.62: icmp_seq=1 ttl=64 time=0.339 ms

--- 172.22.199.62 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.339/0.339/0.339/0.000 ms
user@VM-3:~$
```

network to the routing table and checked the network connectivity. Note that the gateway is IPv4 address of adapter *eth1* on VM 2.

You may now ask another question. *Can we reach VM 1, in particular, the IPv4 address assigned to adapter* eth1 *on VM 1?* The answer appears to be *positive* by examining the routing table in VM 3 and that in VM 2. Both tables have an entry for the network.

Listing 14: Testing Network Connectivity between VM 1 and VM 3

```
user@VM-3:~$ ping -c 1 172.22.199.1
```

Unfortunately, this is not the case as shown in Listing 14. The *ping* apparently waits indefinitely. This, in fact, is not about routing tables. Rather, it is about how Linux kernel is configured. By default, Linux kernel does not forward packets on behalf of other networks unless explictly requested. We can verify this by using *tcpdump* as shown Listing 15. We first run *tcpdump* for adapter *eth1* on VM 2, we observe that many ICMP echo requests arrive from 172.20.136.230 and are destined to 172.22.199.1. However, when we then run *tcpdump* for adapter *eth0* on VM 2, we observe no ICMP echo requests at all, which proves that VM 2 does not forward packets on behalf of host 172.20.136.230.

We can use *sysctl* to enable Linux kernel packet forwarding. On VM 2, we run the *sysctl* command as shown in Listing 16. When we run *tcpdump* for adapter *eth0* on VM 2, we now observe many ICMP echo requests, which indicates that VM 2 now forwards packets received from adapter *eth1* to adapter *eth0*, as shown in Listing 17. However, *ping* remains waiting indefinitely on VM 3, the same as Listing 14.

To diagnose the problem, we now run *tcpdump* for *eth1* on VM 1. As shown in Listing 19, we do observe many ICMP echo requests coming in; however, no ICMP echo replies were sent. Examining the routing table on VM 1, at present whose content is shown in Listing 5, in which there is no entry for Network 172.20.136.224/255.255.255.248. Therefore, the solution is to add the entry to

Listing 15: Result of Running *tcpdump* on VM 2 before Enabling IPv4 Packet Forwarding on VM 2

```
user@VM-2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
22:34:33.900705 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 471,
    length 64
22:34:34.909553 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 472,
    length 64
22:34:35.908829 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 473,
    length 64
^C
3 packets captured
4 packets received by filter
0 packets dropped by kernel
user@VM-2:~$
user@VM-2:~$ sudo tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Listing 16: Enabling IPv4 Packet Forwarding on VM 2

```
user@VM-2:~$ sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
user@VM-2:~$
```

Listing 17: Result of Running *tcpdump* on VM 2 before Enabling IPv4 Packet Forwarding on VM 2

```
user@VM-2:~$ sudo tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:43:04.024621 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 980,
    length 64
22:43:05.023262 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 981,
    length 64
^C
2 packets captured
3 packets received by filter
0 packets dropped by kernel
user@VM-2:~$
```

the routing table as shown in Listing 18.

We now run *tcpdump* for adapter *eth1* on VM 1 before we ping VM 1 from VM 3. As shown in Listing 20, we can now observe that VM 1 now forwards ICMP echo replies to 172.20.136.230 when we ping VM 1 from VM 3. As shown in Listing 21, *ping* on VM 3 now happily reports that it reached VM 1.

Listing 18: Updating Routing Table on VM 1

```
user@VM-1:~$ sudo route add -net 172.20.136.224 netmask 255.255.255.248 gw 172.22.199.62
user@VM-1:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     1      0        0 eth0
172.20.136.224  172.22.199.62   255.255.255.248 UG    0      0        0 eth1
172.22.199.0    0.0.0.0         255.255.255.192 U     0      0        0 eth1
```

Listing 19: Result of Running *tcpdump* on VM 1 Before Routing Table Update

```
user@VM-1:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:02.260057 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 1457,
    length 64
22:51:03.259809 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 1458,
    length 64
22:51:04.258984 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2202, seq 1459,
    length 64
^C
3 packets captured
6 packets received by filter
0 packets dropped by kernel
user@VM-1:~$
```

Listing 20: Result of Running *tcpdump* on VM 1 After Routing Table Update

```
user@VM-1:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
23:00:42.677493 IP 172.20.136.230 > 172.22.199.1: ICMP echo request, id 2218, seq 1, length
    64
23:00:42.677537 IP 172.22.199.1 > 172.20.136.230: ICMP echo reply, id 2218, seq 1, length 64
23:00:42.686687 IP 172.22.199.62 > 172.22.199.1: ICMP time exceeded in-transit, length 92
^C
3 packets captured
3 packets received by filter
0 packets dropped by kernel
user@VM-1:~$
```

Listing 21: Testing Network Connectivity between VM 1 and VM 3

```
user@VM-3:~$ ping -c 1 172.22.199.1
PING 172.22.199.1 (172.22.199.1) 56(84) bytes of data.
64 bytes from 172.22.199.1: icmp_seq=1 ttl=64 time=0.609 ms

--- 172.22.199.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.609/0.609/0.609/0.000 ms
user@VM-3:~$
```

Listing 22: Content of */etc/network/interfaces* on VM 1

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth1
iface eth1 inet static
  address 172.22.199.1
  netmask 255.255.255.192
  post-up route add -net 172.20.136.224 netmask 255.255.255.248 gw 172.22.199.62
  pre-down route del -net 172.20.136.224 netmask 255.255.255.248 gw 172.22.199.62
```

Listing 23: Content of */etc/network/interfaces* on VM 2

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
  address 172.22.199.62
  netmask 255.255.255.192

auto eth1
iface eth1 inet static
  address 172.20.136.225
  netmask 255.255.255.248
```

Listing 24: Enabling IPv4 Packet Forwarding in */etc/sysctl.conf* on VM 2

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

2.6 Making Changes Permanent

The configuration we have made does not survive a reboot. To make the configuration changes permanent, i.e., to survive a reboot, we need to make changes to a few Linux configuration files.

The IPv4 address assignment and other configuration settings can be manually added in configuration file */etc/network/interfaces*. The Linux kernel packet forwarding can be enabled by modifying configuration file */etc/sysctl.conf*.

Be aware that to make the changes described below effective, you must reboot the virtual machines once you complete the changes.

VM 1. The content of */etc/network/interfaces* on VM 1 is shown as Listing 22.

VM 2. The content of */etc/network/interfaces* on VM 2 is shown as Listing 23. In */etc/sysctl.conf*, only one line is uncommented to enable IPv4 packet forwarding. The uncommented line is shown in Listing 24.

VM 3. The content of */etc/network/interfaces* on VM 3 is shown as Listing 25

Listing 25: Content of */etc/network/interfaces* on VM 3

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
  address 172.20.136.230
  netmask 255.255.255.248
  post-up route add -net 172.22.199.0 netmask 255.255.255.192 gw 172.20.136.225
  pre-down route del -net 172.22.199.0 netmask 255.255.255.192 gw 172.20.136.225
```

Listing 27: Deleting an Entry from a Routing Table using *route*

```
user@VM-3:~$ sudo route del -net 172.22.199.0 netmask 255.255.255.192 gw 172.20.136.225
user@VM-3:~$
```

2.7 Dealing with Regrets

Everyone makes mistakes. We will make mistakes when we configure the networks. It is important that we know how to undo a change that is considered a mistake or is not shown to be working. If we assigned a wrong IPv4 address, we can undo the change using *ip*.

Listing 26 shows that you can delete an IPv4 address you assigned using the *ip* command. Similar as before, you may have to repeat the command with different IPv4 addresses if more than one IPv4 addresses were assigned to the adapter.

Listing 26: Deleting an IPv4 Address using *ip*

```
sudo ip addr del 172.22.199.1/26 dev eth1
```

To delete an entry from a routing table, you may use the *route* command as shown in Listing 27.

3 Remaining Issues

Although we have completed the configuration for the internetwork, the network still has a few issues.

- When you ping VM 3 from VM 1 or ping VM 1 from VM 3, you will observe large amount of duplicated ICMP echo reply packets, which appears to be the result that VirtualBox internal networking mode has only one Ethernet while we are trying to divide it into a few IPv4 networks.

  To address the issue, one solution is to divide the virtual Ethernet into a few Virutual Local Area Networks (VLANs). In next experiment of the series, i.e., Part II, we will do just that.

- You will find that *ping* will not be successful if you ping adapter *eth0*'s address on VM 1. Moreover, you cannot connect to any hosts other than the three hosts from any of the three hosts. This is also an important item that we will address in next experiment (i.e., Part II).

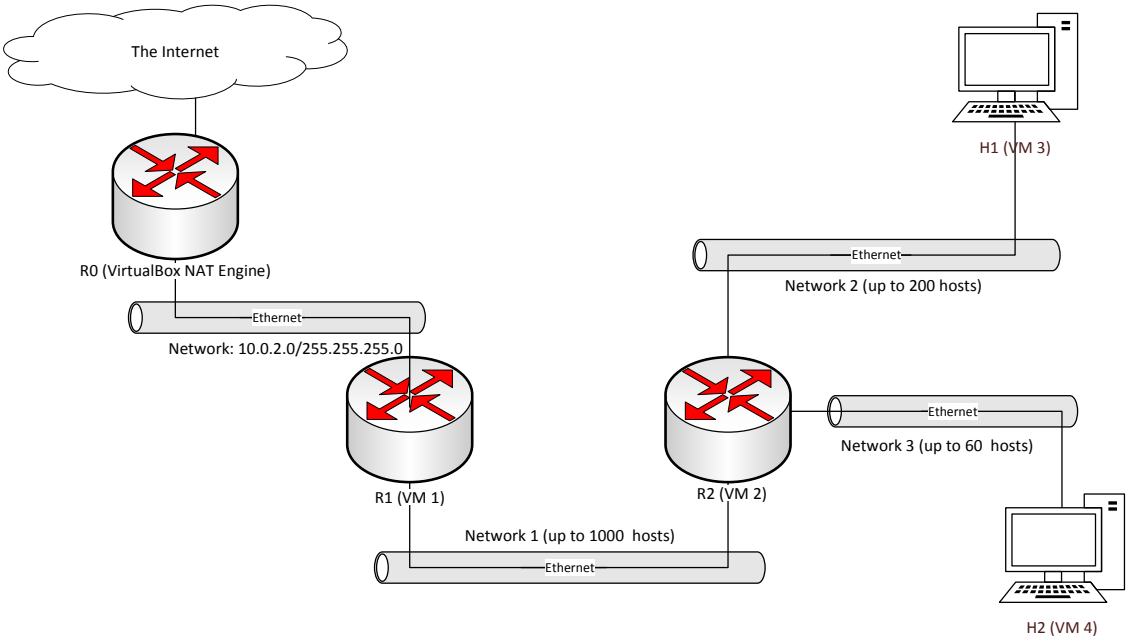

Figure 4: Layout of a planned network and existing network infrastructure. The existing network infrastructure consists of VirtualBox NAT engine and host network setup.

4 Practice Assignment

You are required to complete the following items.

- Following the instruction in this document, implement the internetwork as illustrated in Figure 2 in 3 virtual machines.
- Design and implement an internetwork using 4 virtual machines as shown in Figure 4.

Show steps, the results of configuration, and testing results in a brief report.

References

[1] Blog: The Fat Bloke Sings: Thoughts from a Fat Bloke. Networking in virtualbox. https://blogs.oracle.com/fatbloke/entry/networking_in_virtualbox1, posted on October 15, 2013 and retrieved on October 27, 2014.