Experimenting Internetworking using Linux Virtual Machines – Part II

Hui Chen

October 27, 2014

Contents

1 Overview

> This document is Part II of the series of experiments to plan and implement internetworks on Linux platforms. To make this document self-contained, portions of Part I are reproduced in this docoument.
>
> Sections are largly duplicated from the documentation for Part I are, Sections 1, 2.1, 2.2, and **??**. You may skip those sections if you are familiar with the documentation for Part I. However, if you did complete the steps in Part I, make sure that you undo all the changes in the Linux virtual machines since this document assumes that you have a fresh Linux virtual machines.

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).

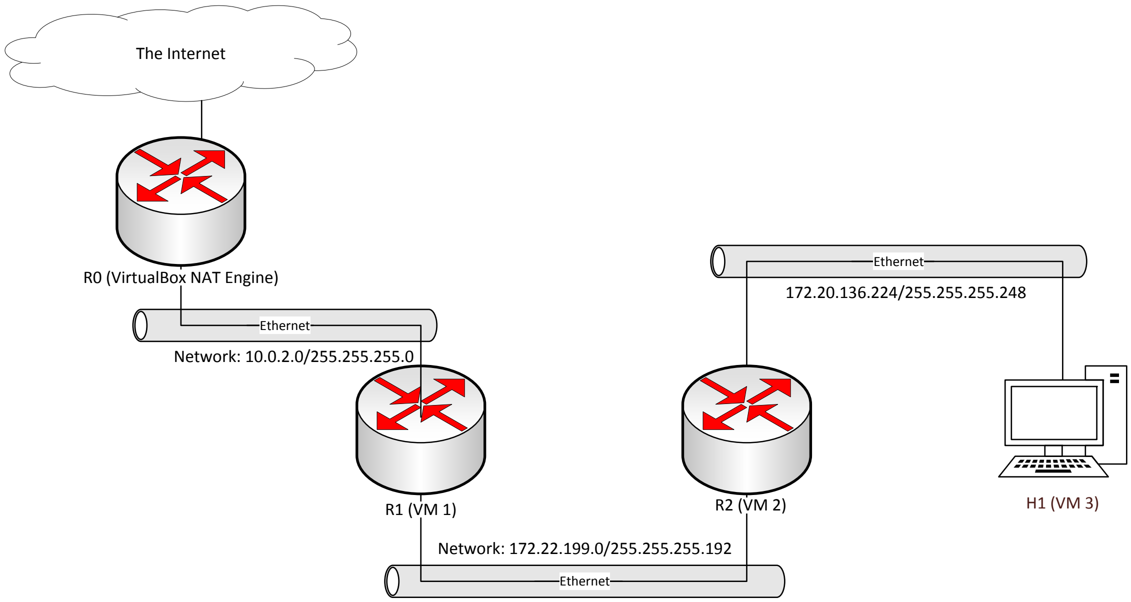

Figure 1: Layout of planned network and existing network infrastructure. The existing network infrastructure consists of VirtualBox NAT engine and host network setup.

2 Part II: Connecting IPv4 Internetwork to the Internet

In Part I of this series, we design an intenetwork as shown in Figure 1 and demonostrates the implementation using 3 VirtualBox virtual machines as shown in Figure 2. Although the hosts in the internetwork can communication with each other, the implementation has a few shortcomings. Below are two most important shortcomings,

- When we ping VM 3 from VM 1 or ping VM 1 from VM 3, we 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 this experiment of the series, we will divide the single Ethernet into two VLANs.

- We find that *ping* will not be successful if we ping adapter *eth0*'s address on VM 1. Moreover, we 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 this part.

The main objectives of this experiment are,

- to plan and implement Virtual LANs (or VLANs) on Linux platforms; and
- to plan and implement IP Masquerading.

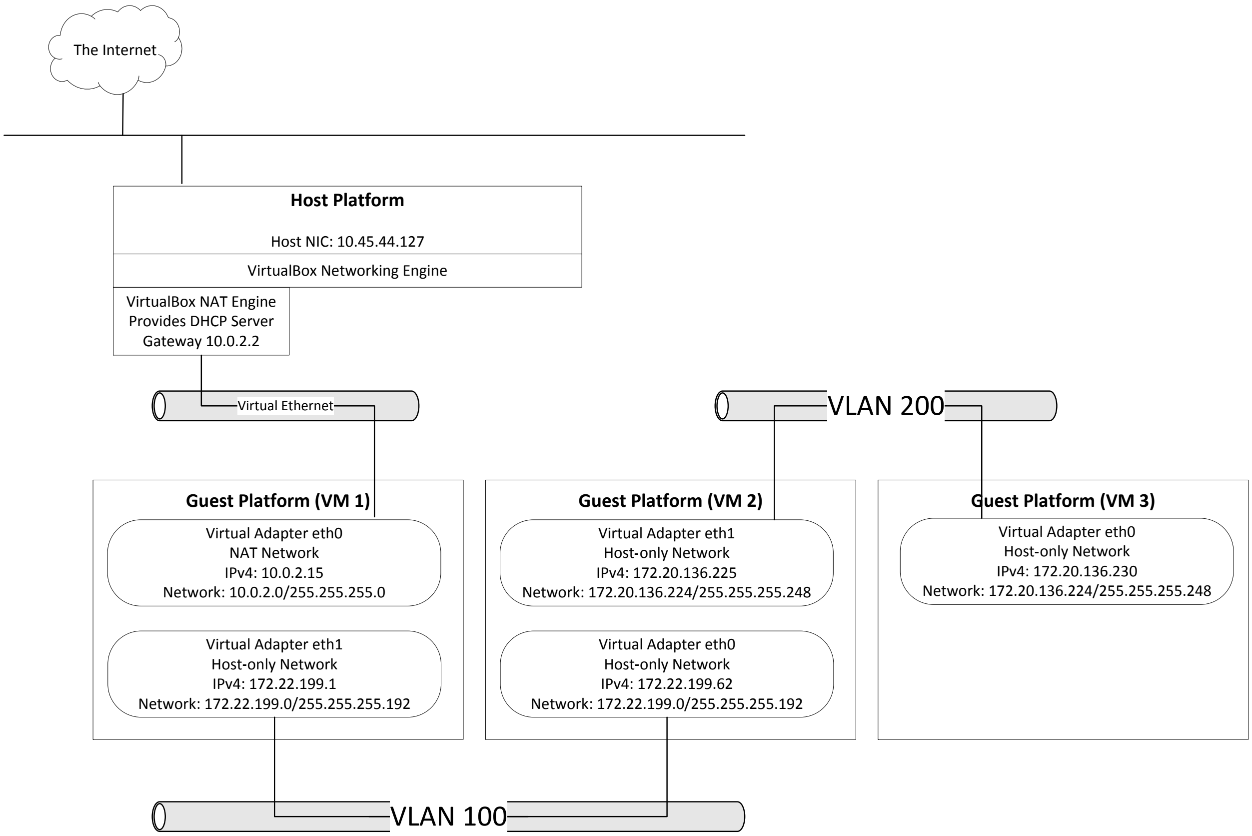

Figure 2: An internetwork consisting of 3 virtual Linux hosts.

2.1 Software

Although the principal 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 on the following software,

- Oracle VM VirtualBox version 4.2.16 and above
- Ubuntu Linux 14.04 (Trusty)

The Linux commands will be used in this experiment are, *ifconfig*, *route*, *ip*, *ping*, *sysctl*, *tcpdump*, *vconfig*, *modprobe*, and *lsmod*.

The Ubuntu Linux system configuration files that of our concern are */etc/network/interfaces*, */etc/sysctl.conf*, and */etc/NetworkManager/NetworkManager.conf*.

2.2 Preparation

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

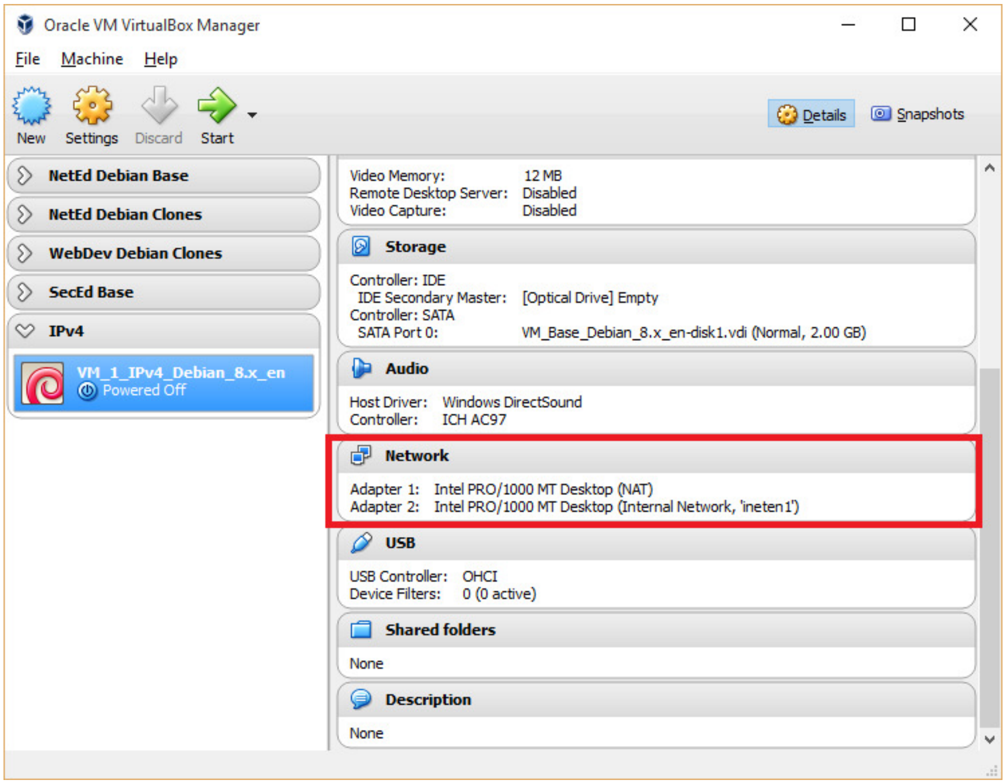

Figure 3: Configurations of VirtualBox virtual machine 1 (VM 1)

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 [**?**],

> "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

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
```

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.,

[frame=single] sudo apt-get install tcpdump

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 **??**,

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 VLAN Setup

VirtualBox appears to make its internal network to be a single Ethernet. We can create two logically separated LANs using VLAN out of the single Ethernet. We plan to have 2 VLANs, *VLAN 100* and *VLAN 200*, as shown in Figure 2.

- *VLAN 100*. Adapter *eth1* in VM 1 and adapter *eth0* in VM 2 belong to VLAN 100.
- *VLAN 200*. Adapter *eth1* in VM 2 and adapter *eth0* in VM 3 belong to VLAN 200.

To set up VLAN on Ubuntu Linux hosts, we need to install the *vlan* package as shown below from the command line on each Linux virtual machine.

```
sudo apt-get install vlan
```

Note that the host must have the Internet access to install the package. You can temporary set up one network adapter to the NAT mode and switch it back to the internal network mode after you finish installing the package.

You will set up the Linux hosts to load *8021q* kernel module. IEEE 802.1Q historically has been the standard specifying Virtual LANs and VLAN Bridges. See [?] for more detail.

First, we check if the module has been been loaded using *lsmod*.

Listing 2: Output Showing Module *8021q* is Loaded

```
user@VM-1:~$ lsmod | grep 8021q
8021q                  23920  0
garp                   14019  1 8021q
mrp                    18357  1 8021q
user@VM-1:~$
```

Listing 3: Adding Adapter *eth1* in VM 1 to *VLAN 100*

```
user@VM-1:~$ sudo vconfig add eth1 100
Added VLAN with VID == 100 to IF -:eth1:-
user@VM-1:~$ 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 08:00:27:83:11: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
4: eth1.100@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    link/ether 08:00:27:6f:2c:53 brd ff:ff:ff:ff:ff:ff
```

```
    lsmod | grep 8021q
```

If nothing is returned, module *8021q* was not loaded. We then load the module using *modprobe*.

```
    sudo modprobe 8021q
```

You can now use *lsmod* shown as before to check if module *8021q* is loaded. Listing **??** is a sample output that shows the module is loaded.

We can now assign adapter *eth1* in VM 1 to *VLAN 100* using *vconfig* and check if the assignment is successful. The steps and the results are shown in Listing **??**, In the result, you will find a new Ethernet adapter called *eth1.100@eth1*, that indicates that *eth1* is a member of *VLAN 100*.

We now move to assign adapter *eth0* in VM 2 to *VLAN 100* and assign adapter *eth1* in VM 2 and adapter *eth0* in VM 3 to *VLAN 200*. Listing **??** shows the steps and the results for assigning adapter *eth0* to *VLAN 100* and assigning adapter *eth1* to *VLAN 200* in VM 2. Listing **??** is the steps and the results for assigning *eth0* to *VLAN 200* in VM 3.

2.4.2 IPv4 Address Setup

We now consider the two items, (1) to assign IPv4 addresses to the adapters on the Linux hosts, i.e., the 3 virtual machines; and (2) to configure routing tables and necessary packet forwarding for IPv4.

As we pointed out in Part I, commands *ifconfig* and *route* are considered to be deprecated and provide a minimum exposure on how one may configure IP networks using the *ip* command. In

Listing 4: Adding Adapters *eth0* & *eth1* in VM 2 Respectively to *VLANs 100* & *200*

```
user@VM-2:~$ lsmod | grep 8021q
user@VM-2:~$ sudo modprobe 8021q
user@VM-2:~$ lsmod | grep 8021q
8021q                  23920  0
garp                   14019  1 8021q
mrp                    18357  1 8021q
user@VM-2:~$ sudo vconfig add eth0 100
Added VLAN with VID == 100 to IF -:eth0:-
user@VM-2:~$ sudo vconfig add eth1 200
Added VLAN with VID == 200 to IF -: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
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
4: eth0.100@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    link/ether 08:00:27:c9:2d:43 brd ff:ff:ff:ff:ff:ff
5: eth1.200@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    link/ether 08:00:27:84:f8:e3 brd ff:ff:ff:ff:ff:ff
```

Listing 5: Adding Adapter *eth0* in VM 3 to *VLAN 200*

```
user@VM-3:~$ lsmod | grep 8021q
user@VM-3:~$ sudo modprobe 8021q
user@VM-3:~$ lsmod | grep 8021q
8021q                  23920  0
garp                   14019  1 8021q
mrp                    18357  1 8021q
user@VM-3:~$ vconfig add eth0 200
WARNING:  Could not open /proc/net/vlan/config.  Maybe you need to load the 8021q module, or
     maybe you are not using PROCFS??
ERROR: trying to add VLAN #200 to IF -:eth0:-  error: Operation not permitted
user@VM-3:~$ sudo vconfig add eth0 200
Added VLAN with VID == 200 to IF -: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
3: eth0.200@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    link/ether 08:00:27:c9:2d:43 brd ff:ff:ff:ff:ff:ff
```

this part, we will use the *ip* command only with intention to introduce more on the usage of *ip*.

VM 1. According to the design in Figure 2, we assign address 172.22.199.1 to network interface

Listing 6: Setting IPv4 Address in VM 1

```
user@VM−1:~$ sudo ip addr add 172.22.199.1/255.255.255.192 broadcast 172.22.199.63 dev eth1
    .100 label eth1.100:1
user@VM−1:~$ ip addr show eth1.100:1
4: eth1.100@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    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 eth1.100:1
       valid_lft forever preferred_lft forever
```

Listing 7: Setting IPv4 Addresses in VM 2

```
user@VM−2:~$ sudo ip addr add 172.22.199.62/255.255.255.192 broadcast 172.22.199.63 dev eth0
    .100 label eth0.100:1
user@VM−2:~$ ip addr show eth0.100:1
4: eth0.100@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    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 eth0.100:1
       valid_lft forever preferred_lft forever
user@VM−2:~$ ip route show
172.22.199.0/26 dev eth0.100  proto kernel  scope link  src 172.22.199.62
user@VM−2:~$ sudo ip addr add 172.20.136.225/255.255.255.248 broadcast 172.20.136.232 dev
    eth1.200 label eth1.200:1
user@VM−2:~$ ip addr show eth1.200:1
5: eth1.200@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    link/ether 08:00:27:84:f8:e3 brd ff:ff:ff:ff:ff:ff
    inet 172.20.136.225/29 brd 172.20.136.232 scope global eth1.200:1
       valid_lft forever preferred_lft forever
user@VM−2:~$ ip route show
172.20.136.224/29 dev eth1.200  proto kernel  scope link  src 172.20.136.225
172.22.199.0/26 dev eth0.100  proto kernel  scope link  src 172.22.199.62
```

eth1.100, a VLAN 100's interface in VM 1. The address 172.22.199.1 is an address in Network 172.22.199.1/255.255.255.192 whose broadcast address is 172.22.199.63. We can now use *ip* to make the IPv4 address assignment in VM 1 as shown in Listing **??**.

VM 2. In VM 2, we have configured has two network interfaces that belong to VLAN 100 and VLAN 200, resepctively.

As illustrated in Figure 2, we assign address 172.22.199.62 to network interface *eth0.100*, a VLAN 100's interface in VM 2. The address 177.22.199.62 is in Network 172.22.199.1/255.255.255.192 whose broadcast address is 172.22.199.63.

Likewise, we assign address 172.20.136.225 to network interface *eth1.200*, a VLAN 200's interface in VM 2. The address 172.20.136.225 is in Network 172.20.136.224/255.255.255.248 whose broadcast address is 172.20.136.232.

The steps and the results of assigning the two addresses are shown in Listing **??**.

Since network interface *eth1.100* in VM 1 and network interface *eth0.100* in VM 2 are in the same Ethernet, a direct-link network, the two hosts should reach other via the direct-link network. It is a good practice to verify the connectivity between the two hosts via the direct-link network because in turn we can check if we make any mistake during the set up as described before. Listing **??** shows such an example using *ping*. The example shows that the two hosts can reach each other via

Listing 8: Pinging VM 1 from VM 2 via VLAN 100

```
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=0.362 ms

--- 172.22.199.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.362/0.362/0.362/0.000 ms
user@VM-2:~$
```

Listing 9: Setting IPv4 Address for VM 3

```
user@VM-3:~$ sudo ip addr add 172.20.136.230/255.255.255.248 broadcast 172.20.136.232 dev
    eth0.200 label eth0.200:1
user@VM-3:~$ ip addr show eth0.200:1
3: eth0.200@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
    default
    link/ether 08:00:27:c9:2d:43 brd ff:ff:ff:ff:ff:ff
    inet 172.20.136.230/29 brd 172.20.136.232 scope global eth0.200:1
       valid_lft forever preferred_lft forever
user@VM-3:~$ ip route show
172.20.136.224/29 dev eth0.200  proto kernel  scope link  src 172.20.136.230
user@VM-3:~$
```

Listing 10: Testing 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=0.435 ms

--- 172.20.136.225 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.435/0.435/0.435/0.000 ms
user@VM-3:~$
```

the direct-link network, i.e., *VLAN 100*.

VM 3. Finally, we assign the planned IPv4 address to network interface *eth0.200* in VM 3. The address is 172.20.136.230, an address in network 172.20.136.224/255.255.255.248. The network's broadcast address is 172.20.136.232. Listing **??** shows the assignment and the result.

Similar as before, we can now test the connectivity between VM 2 and VM3 via *VLAN 200*, a direct-link network. Listing **??** shows the test result using *ping*.

2.4.3 Routing Table Setup

Similar to Part I, to allow hosts in network 172.22.199.1/255.255.255.192 and those in network 172.20.136.224/255.255.255.248 communicate with each other, we must update routing tables on the network gateway, i.e., VM 2 and the hosts, i.e., VM 1 and VM 3. Since VM 2 is a network gateway and is reponsible for forwarding IPv4 packets on behalf of the other network, we will enable packet forwarding for IPv4 on VM 2 as well. The steps and the resulting routing tables are shown Listings **??**, **??**, and **??**.

Listing 11: Steps and Results for Routing Table Update in VM 1

```
user@VM-1:~$ ip route show
default via 10.0.2.2 dev eth0  proto static
10.0.2.0/24 dev eth0  proto kernel  scope link  src 10.0.2.15  metric 1
172.22.199.0/26 dev eth1.100  proto kernel  scope link  src 172.22.199.1
user@VM-1:~$ sudo ip route add 172.20.136.224/29 via 172.22.199.62
user@VM-1:~$ ip route show
default via 10.0.2.2 dev eth0  proto static
10.0.2.0/24 dev eth0  proto kernel  scope link  src 10.0.2.15  metric 1
172.20.136.224/29 via 172.22.199.62 dev eth1.100
172.22.199.0/26 dev eth1.100  proto kernel  scope link  src 172.22.199.1
user@VM-1:~$
```

Listing 12: Enabling Packet Forwarding for IPv4 in VM 2

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

Listing 13: Steps and Results for Routing Table Update in VM 3

```
user@VM-3:~$ sudo ip route show
172.20.136.224/29 dev eth0.200  proto kernel  scope link  src 172.20.136.230
user@VM-3:~$ sudo ip route add 172.22.199.0/255.255.255.192 via 172.20.136.225
user@VM-3:~$ sudo ip route show
172.20.136.224/29 dev eth0.200  proto kernel  scope link  src 172.20.136.230
172.22.199.0/26 via 172.20.136.225 dev eth0.200
user@VM-3:~$
```

Listing 14: Testing Connectivity between VM 1 and VM 2 using *ping*

```
user@VM-3:~$ ping -c 2 172.22.199.1
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=63 time=0.696 ms
64 bytes from 172.22.199.1: icmp_seq=2 ttl=63 time=1.92 ms

--- 172.22.199.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.696/1.312/1.929/0.617 ms
user@VM-3:~$
```

To test the setting, we can ping the hosts in network 172.20.136.224/255.255.255.248 from a host in 172.22.199.1/255.255.255.192. We then ping 172.22.199.1, the address allocated to network interface *eth1.100* in VM 1 from VM 3. The result is shown in Listing **??**

2.4.4 Connecting to VirtualBox Internal Network

Although the hosts in the two networks 172.22.199.1/255.255.255.192 and 172.20.136.224/255.255.255.248 can communicate with each other, you will quickly find that the networks cannot reach network 10.0.2.0/255.255.255.0, the network that adapter *eth0* in VM 1 belongs to. For instance, Listing **??** shows that 10.0.2.15, a host on network 10.0.2.0/255.255.255.0 is not reachable from VM 3.

Listing 15: Showing Network 10.0.2.0/255.255.255.0 Not Reachable on VM 3

```
user@VM-3:~$ ping 10.0.2.15
connect: Network is unreachable
user@VM-3:~$
```

Listing 16: Adding Network 10.0.2.0/255.255.255.0 in Routing Table in VM 2

```
user@VM-2:~$ sudo ip route add 10.0.2.0/255.255.255.0 via 172.22.199.1
user@VM-2:~$ ip route show
10.0.2.0/24 via 172.22.199.1 dev eth0.100
172.20.136.224/29 dev eth1.200  proto kernel  scope link  src 172.20.136.225
172.22.199.0/26 dev eth0.100  proto kernel  scope link  src 172.22.199.62
```

Listing 17: Adding Network 10.0.2.0/255.255.255.0 in Routing Table in VM 3

```
user@VM-3:~$ sudo ip route add 10.0.2.0/255.255.255.0 via 172.20.136.225
user@VM-3:~$ sudo ip route show
10.0.2.0/24 via 172.20.136.225 dev eth0.200
172.20.136.224/29 dev eth0.200  proto kernel  scope link  src 172.20.136.230
172.22.199.0/26 via 172.20.136.225 dev eth0.200
```

Listing 18: Testing Connectivity to Network 10.0.2.15/255.255.255.0 in VM 3

```
user@VM-3:~$ ping -c 2 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=63 time=1.08 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=63 time=2.18 ms

--- 10.0.2.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.081/1.632/2.184/0.552 ms
user@VM-3:~$
```

Examining the routing tables in VM 2 and VM 3 as included in Listings **??** and **??**, we know that no entry for network 10.0.2.0/255.255.255.0 in the routing tables. Therefore, we shall add the entries for network 10.0.2.0/255.255.255.0 in the routing tables, as illustrated in Listings **??** and **??**.

Now we can show that host 10.0.2.15 is reachable from either VM 2 or VM 3 as in Listing **??**.

2.4.5 Connecting to Outside Networks

Next stage is to connect to host platform and outside network. If the host has connectivity to the Internet, we would like to connect to the Internet from the two networks we created, i.e., networks 172.22.199.1/255.255.255.192 and 172.20.136.224/255.255.255.248 as well. Since VM 1 is the single gateway between the two networks we created, we shall add an entry to routing tables in VM 2 and VM 3 to forward traffic to any networks other than the two networks and the VirtualBox Internal Network, i.e., networks 172.22.199.1/255.255.255.192 and 172.20.136.224/255.255.255.248 and 10.0.2.15/255.255.255.0 to VM 1. To achieve the above, we will add a *default* network entry to the routing tables as shown in Listings **??** and **??**.

Listing 19: Adding Default Gateway in VM 2

```
user@VM-2:~$ sudo ip route add 0.0.0.0/0.0.0.0 via 172.22.199.1
user@VM-2:~$ ip route show
default via 172.22.199.1 dev eth0.100
10.0.2.0/24 via 172.22.199.1 dev eth0.100
172.20.136.224/29 dev eth1.200  proto kernel  scope link  src 172.20.136.225
172.22.199.0/26 dev eth0.100  proto kernel  scope link  src 172.22.199.62
```

Listing 20: Adding Default Gateway in VM 3

```
user@VM-3:~$ sudo ip route add 0.0.0.0/0.0.0.0 via 172.20.136.225
user@VM-3:~$ ip route show
default via 172.20.136.225 dev eth0.200
10.0.2.0/24 via 172.20.136.225 dev eth0.200
172.20.136.224/29 dev eth0.200  proto kernel  scope link  src 172.20.136.230
172.22.199.0/26 via 172.20.136.225 dev eth0.200
```

Listing 21: Enabling Packet Forwarding for IPv4 in VM 2

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

Since VM 1 is now serving as a network gateway and forwarding packets for the two networks, i.e., networks 172.20.136.224/255.255.255.248 and 172.22.199.1/255.255.255.192, we shall enable packet forwarding for IPv4 in VM 1 as shown in Listing **??**.

However, the above steps are insufficient — the observation shown in Listing **??** indicates that although the packets from networks 172.22.199.1/255.255.255.192 and 172.20.136.224/255.255.255.248 are indeed being forwarded into adapter *eth0*, VM 1 does not appear to know how to forward packets back to the two networks, i.e., when we ping IPv4 address 10.0.2.15 from any of the two networks (any hosts on the two networks) (such as, run *ping* 10.0.2.15 *in either VM 2 or VM 3*), we do observe ICMP echo requests being transmitted to the outside networks; however, no ICMP echo reply requests were forwarded back.

The above result is not surprising, giving that all IPv4 addresses are private internet addresses. Private internet addresses are specified in RFC 1918 [**?**] and private internet addresses are not globally unique. We must use Network Address Translation (NAT) to enable the private networks to connect to outside networks. For more information on NAT, readers are referred to RFC 3022 [**?**], [**?**, p. 439–442].

In Ubuntu Linux, we can configure IPv4 Masquerading in VM 1 and VM 1 becomes a NAT gateway. IPv4 Masquerading is a functionality of Ubuntu network firewall, called *Uncomplicated Firewall* or *ufw* [**?**].

Following the instruction in [**?**], we make changes to *ufw*.

- Packet forwarding needs to be enabled in *ufw*. We shall change *DEFAULT_FORWARD_POLICY* to "ACCEPT" from "DROP" in */etc/default/ufw*. We then uncomment *net.ipv4.ip_forward=1* in */etc/ufw/sysctl.conf*

Listing 22: Testing Connectivity to Outside Networks Showing No Return Packets

```
user@VM-1:~$ sudo tcpdump -i eth1.100
tcpdump: WARNING: eth1.100: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1.100, link-type EN10MB (Ethernet), capture size 65535 bytes
20:42:29.948343 IP 172.22.199.62 > www.vsu.edu: ICMP echo request, id 2776, seq 21, length
    64
20:42:34.948391 IP 172.22.199.62 > www.vsu.edu: ICMP echo request, id 2776, seq 26, length
    64
^C
2 packets captured
4 packets received by filter
0 packets dropped by kernel
user@VM-1:~$ 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
20:43:38.134916 IP 172.22.199.62 > pilot.vsu.edu: ICMP echo request, id 2776, seq 89, length
     64
20:43:38.137005 ARP, Request who-has 172.22.199.62 (Broadcast) tell 10.0.2.2, length 46
20:43:38.445532 IP 10.0.2.15.47326 > ldap2.vsu.edu.domain: 42636+ PTR? 15.33.174.150.in-addr
    .arpa. (44)
20:43:38.445857 IP 10.0.2.15.47326 > ldap1.vsu.edu.domain: 42636+ PTR? 15.33.174.150.in-addr
    .arpa. (44)
20:43:38.446061 IP 10.0.2.15.47326 > 150.174.7.85.domain: 42636+ PTR? 15.33.174.150.in-addr.
    arpa. (44)
20:43:38.446249 IP 10.0.2.15.47326 > external.vsu.edu.domain: 42636+ PTR? 15.33.174.150.in-
    addr.arpa. (44)
20:43:38.448039 IP external.vsu.edu.domain > 10.0.2.15.47326: 42636* 3/0/0 PTR pilot.vsu.edu
    ., PTR vsu.edu., PTR www.vsu.edu. (103)
20:43:38.448973 IP 150.174.7.85.domain > 10.0.2.15.47326: 42636* 3/0/0 PTR pilot.vsu.edu.,
    PTR www.vsu.edu., PTR vsu-webcs-02v.vsu.edu. (117)
20:43:38.449014 IP 10.0.2.15 > 150.174.7.85: ICMP 10.0.2.15 udp port 47326 unreachable,
    length 153
20:43:38.449252 IP 10.0.2.15.55885 > external.vsu.edu.domain: 33668+ PTR? 62.199.22.172.in-
    addr.arpa. (44)
20:43:38.451702 IP external.vsu.edu.domain > 10.0.2.15.55885: 33668 NXDomain 0/1/0 (96)
.....
```

- We will add rules to the */etc/ufw/before.rules* file, for which, we add the following to the top of the file just after the header comments,

```
# nat Table rules
*nat
:POSTROUTING ACCEPT [0:0]

# Forward traffic from eth1 through eth0.
-A POSTROUTING -s 172.22.199.0/26 -o eth0 -j MASQUERADE
-A POSTROUTING -s 172.20.136.224/29 -o eth0 -j MASQUERADE

# don't delete the 'COMMIT' line or these nat table rules won't be processed
COMMIT
```

- Then, we shall restart *ufw*.

```
sudo ufw disable && sudo ufw enable
```

We now test the connectivity to outside network by pinging Google's webserver as in Listing **??**.

Listing 23: Testing Connectivity to Outside Network on VM 3

```
user@VM-3:~$ ping -c 5 74.125.131.106
PING 74.125.131.106 (74.125.131.106) 56(84) bytes of data.
64 bytes from 74.125.131.106: icmp_seq=1 ttl=44 time=107 ms
64 bytes from 74.125.131.106: icmp_seq=2 ttl=44 time=110 ms
64 bytes from 74.125.131.106: icmp_seq=3 ttl=44 time=107 ms
64 bytes from 74.125.131.106: icmp_seq=4 ttl=44 time=98.2 ms
64 bytes from 74.125.131.106: icmp_seq=5 ttl=44 time=106 ms

--- 74.125.131.106 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 98.275/106.273/110.812/4.242 ms
user@VM-3:~$
```

Listing 24: Testing Name Resolution in VM 3

```
user@VM-3:~$ nslookup
> www.yahoo.com
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
www.yahoo.com   canonical name = fd-fp3.wg1.b.yahoo.com.
Name:   fd-fp3.wg1.b.yahoo.com
Address: 98.139.180.149
Name:   fd-fp3.wg1.b.yahoo.com
Address: 98.139.183.24
> www.espn.com
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
www.espn.com    canonical name = redir.espn.gns.go.com.
Name:   redir.espn.gns.go.com
Address: 68.71.212.159
> exit
user@VM-3:~$
```

The last, not the least, is to add domain name server to VM 2 and VM 3. We can add the following lines to */etc/resolv.conf* in both VM 2 and VM 3.

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Note that hosts 8.8.8.8 and 8.8.4.4 are Google's public domain name servers (DNS) [**?**]. To test the name resolution, you can use either *dig* or *nslookup*. Listing **??** is an example of using *nslookup* to look up *www.yahoo.com* and *www.espn.com*.

2.5 Making Changes Permanent

The configuration changes we have made are not permanent and do 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.

Different Linux distributions may have different layout of configuration files. Ubuntu Linux can

Listing 26: Enabling IPv4 Packet Forwarding in */etc/sysctl* on VM 1

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

be considered as a derivative of Debian Linux distribution [?]. The changes to the configuration files referred in this section are tested on Ubuntu 14.04 and are mostly applicable to Debian Linux distributions and other Linux distributions derived from Debian Linux.

The loadable kernerl modules can be specified in */etc/modules*. 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*.

2.5.1 Configuration in VM 1

We first summarize the configuration changes as follows,

- loading kernel module *8021q* [?, ?],
- enabling packet forwarding for IPv4 and item disabling IPv6 [?, ?],
- creating desired VLAN, assigning IPv4 address to the VLAN interface *eth1.100* [?], and
- setting NAT gateway by enabling IP Masqerading [?].

When the configuration, described in detail below is completed, *reboot the Linux virtual machine*.

Loading Kernel Module *8021q* We modifiy */etc/modules* to load module *8021q*. The content of */etc/modules* on VM 1 is shown as Listing **??**. The content of the configuration file may be different on your machine; however, it must contains a line *8021q* that informs Linux to load module *8021q*— during system startup.

Listing 25: Content of */etc/modules* on VM 1

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

lp
8021q
```

Enabling Packet Forwarding for IPv4 To enable pacekt forwarding for IPv4 on VM 1, we will uncomment the line *net.ipv4.ip_foward=1* in */etc/sysctl.conf*. The line should resemble Listing **??** after the change.

Disabling IPv6 To disable IPv6 on VM 1, we will add a few lines */etc/sysctl.conf*. Conveniently, we add the lines to the end of the configuration file. The lines are shown in Listing **??**

Listing 27: Enabling IPv4 Packet Forwarding in */etc/sysctl* on VM 2

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

Listing 28: 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 eth1.100
iface eth1.100 inet static
  vlan-raw-device eth1
  address 172.22.199.1
  netmask 255.255.255.192
  post-up ip route add 172.20.136.224/255.255.255.248 via 172.22.199.62
  pre-down ip route del 172.20.136.224/255.255.255.248 via 172.22.199.62
```

Configuring VLAN and Network Interface To configure VLAN, IPv4 address assignment, and domain name service, we resort to */etc/network/interfaces*. The content of */etc/network/interfaces* on VM 1 is shown as Listing **??**.

2.5.2 Configuration in VM 2

Similarly, we will make the following configuration changes,

- loading kernel module *8021q* [**?**, **?**],
- enabling packet forwarding for IPv4 and item disabling IPv6 [**?**, **?**], and
- creating desired VLAN, assigning IPv4 address to the VLAN interface *eth1.100* [**?**], and setting up name resoultion [**?**].

Note that VM 2 is not a NAT gateway and no configuration change is necessary for the network firewall. When the configuration, described in detail below is completed, *reboot the Linux virtual machine.*

Loading Kernel Module *8021q* This is identical to Section **??**, i.e., we need to add the line *8021q* to /etc/modules. The content of the file is identical to that for VM 1.

Enabling Packet Forwarding for IPv4 This is identical to Section **??**, i.e., we need to uncomment the line net.ipv4.ip_forward=1 in */etc/sysctl.conf*.

Disabling IPv6 This is identical to Section **??**.

Listing 29: 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 eth0.100
iface eth0.100 inet static
  address 172.22.199.62
  netmask 255.255.255.192
  dns-nameservers 8.8.8.8 8.8.4.4
  post-up ip route add 0.0.0.0/0.0.0.0 via 172.22.199.1
  pre-down ip route del 0.0.0.0/0.0.0.0 via 172.22.199.1
  post-up ip route add 10.0.2.0/255.255.255.0 via 172.22.199.1
  pre-down ip route del 10.0.2.0/255.255.255.0 via 172.22.199.1

auto eth1.200
iface eth1.200 inet static
  address 172.20.136.225
  netmask 255.255.255.248
```

Configuring VLAN, Network Interface, and Name Resolution The difference bwteen the configuration in VM 2 and that in VM 1 lies here since we need to confgiure two VLANs and assign different addresses to the two network interfaces. The same as in Section **??**, the configuration is defined in */etc/network/interfaces*. However, the content of the file is different. The content of */etc/network/interfaces* on VM 2 is shown as Listing **??**.

Note that in the configuration file, we specify two domain name servers. The two are Google's public domain name servers [**?**]. We may use others, such as those made avaiable by a nearby network.

2.5.3 Configuration in VM 3

Similarly, we will make the following configuration changes,

- loading kernel module *8021q* [**?**, **?**],
- diabling IPv6 [**?**],
- creating desired VLAN, assigning IPv4 address to the VLAN interface *eth1.100* [**?**], and setting up name resolution [**?**].

Note that VM 2 is not a NAT gateway and no configuration change is necessary for the network firewall. In addition, VM 3 is not serving as a network gateway and there is no need to enable packet forwarding for IPv4. When the configuration, described in detail below is completed, *reboot the Linux virtual machine.*

Loading Kernel Module *8021q* This is no different from Section **??**, i.e., we need to add the line *8021q* to /etc/modules. The content of the file is identical to that for VM 1.

Disabling IPv6 This is the same as Section **??**.

Listing 30: 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 eth0.200
iface eth0.200 inet static
  address 172.20.136.230
  netmask 255.255.255.248
  dns-nameservers 8.8.8.8 8.8.4.4
  post-up ip route add 172.22.199.0/255.255.255.192 via 172.20.136.225
  pre-down ip route del 172.22.199.0/255.255.255.192 via 172.20.136.225
  post-up ip route add 10.0.2.0/255.255.255.0 via 172.20.136.225
  pre-down ip route del 10.0.2.0/255.255.255.0 via 172.20.136.225
  post-up ip route add 0.0.0.0/0.0.0.0 via 172.20.136.225
  pre-down ip route del 0.0.0.0/0.0.0.0 via 172.20.136.225
```

Listing 31: Example: Unassign IPv4 Address Using *ifconfig*

```
    sudo ifconfig eth1 0
```

Listing 32: Example: Unassign IPv4 Address Using *ifconfig*

```
    sudo ifconfig eth1.100 0
```

Configuring VLAN and Network Interface We need to configure a VLAN and assign an IPv4 address to the network interface. The configuration is specified in */etc/network/interface*. The content of */etc/network/interfaces* on VM 3 is shown as Listing **??**.

Note that in the configuration file, we specify two domain name servers. The two are Google's public domain name servers [**?**]. We may use others, such as those made avaiable by a nearby network.

2.6 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 either *ifconfig* and *ip*. If we created a wrong routing table entry, we can undo the change using either *route* and *ip*. As we mentioned before, many consider that *ifconfig* and *route* are deprecated.

Listings **??** and **??** show two examples that you can use *ifconfig* to unset IPv4 address by setting the IPv4 address to 0. You may have to repeat the command for a few times if more than one wrong IPv4 address was assigned to the adapter. You must run the command and verify the setting of the network adapter, and repeat the command until no IPv4 address is present.

Listings **??** and **??** are two examples of deleting a previously added IPv4 address using the *ip* command. The essence is that to delete an address, simply replace the "add" keyword in the command that you use to add the address by "del" and run the new command. Similar as before, you may have to repeat the command with different IPv4 addresses if more than one IPv4 addresses

Listing 33: Example: Deleting an IPv4Address using *ip*

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

Listing 34: Example: Deleting an IPv4Address using *ip*

```
sudo ip addr del 172.22.199.1/255.255.255.192 dev eth0.200
```

Listing 35: Exmaple: Deleting an Entry from a Routing Table using *route*

```
sudo route del -net 172.22.199.0 netmask 255.255.255.192 gw 172.20.136.225
```

Listing 36: Example: Deleting an Entry from a Routing Table using *ip*

```
sudo ip route del 172.22.199.0 255.255.255.192 via 172.20.136.225
```

were assigned to the adapter.

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

3 Remaining Issues

Although we have completed the configuration for the internetwork, the network appears to be working. One minor issue that we may address is to merge unnecessary routing table entries — the objective is to make the routing table as small as possible while the routing table is in effect the same, which speeds up routing table lookup when a IPv4 packet arrarive. For instance, in VM 3, regardless which network the packet's destination may be, the only possible network gateway is 172.20.136.225. The task to make the routing table smaller will be an exercise of the readers.

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 **??**.
- Connect the internetwork to the outside network.

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

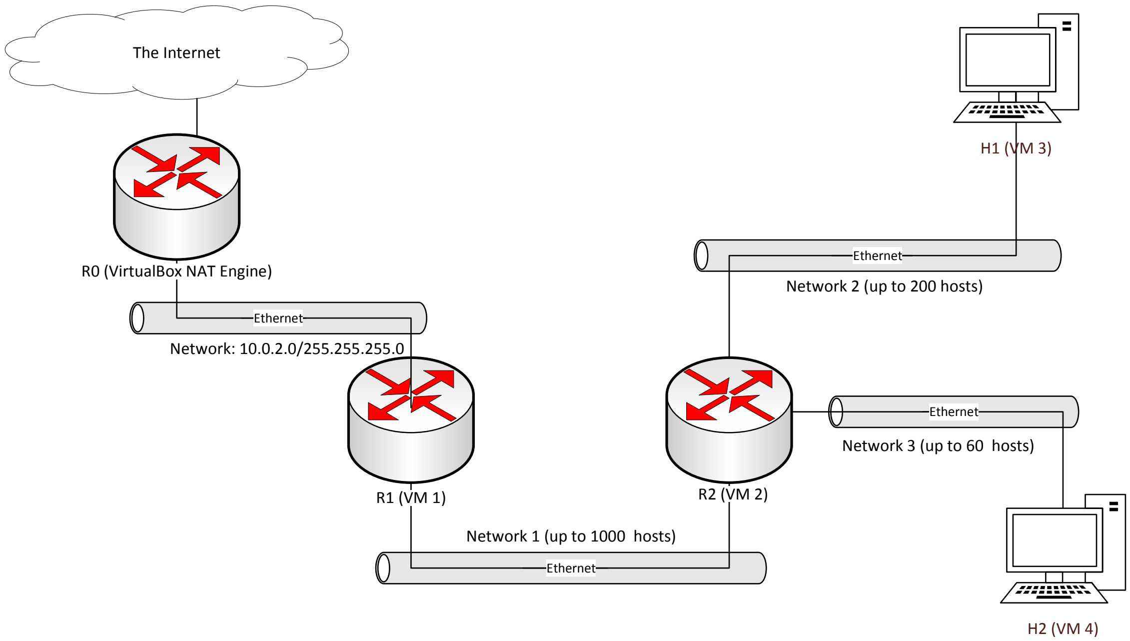

Figure 4: Layout of a planned network and existing network infrastructure. The existing network infrastructure consists of VirtualBox NAT engine and host network setup.

References

[1] Debian Linux Developers. Network configuration: Multiple ip addresses on one interface. `https://wiki.debian.org/NetworkConfiguration#Multiple_IP_addresses_on_One_Interface`, retrieved on October 27, 2014.

[2] Google Developers. Google public dns. `https://developers.google.com/speed/public-dns/docs/using`, retrieved on October 27, 2014.

[3] Ubuntu Linux Developers. https://wiki.ubuntu.com/uncomplicatedfirewall. `https://wiki.ubuntu.com/UncomplicatedFirewall`, retrieved on October 27, 2014.

[4] Ubuntu Linux Developers. Ipv6. `https://wiki.ubuntu.com/IPv6`, retrieved on October 27, 2014.

[5] Ubuntu Linux Developers. Loadable modules. `https://help.ubuntu.com/community/Loadable_Modules`, retrieved on October 27, 2014.

[6] Ubuntu Linux Developers. Network configuration: Name resolution. `https://help.ubuntu.com/14.04/serverguide/network-configuration.html#name-resolution`, retrieved on October 27, 2014.

[7] Ubuntu Linux Developers. Ubuntu 14.04: Ubuntu server guide: Security: Firewall. `https://help.ubuntu.com/lts/serverguide/firewall.html`, retrieved on October 27, 2014.

[8] Ubuntu Linux Developers. Ubuntu and debian. `http://www.ubuntu.com/about/about-ubuntu/ubuntu-and-debian`, retrieved on October 27, 2014.

[9] Ubuntu Linux Developers. Vlan. `https://wiki.ubuntu.com/vlan`, retrieved on October 27, 2014.

[10] 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.

[11] IEEE. 802.1q – virtual lans. `http://www.ieee802.org/1/pages/802.1Q.html`, retrieved on October 27, 2014.

[12] Charles Kozierok. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. No Starch Press, San Francisco, CA, USA, 2005.

[13] Mark0978. How to disable ipv6 on ubuntu? `http://askubuntu.com/a/381623`, retrieved on October 27, 2014.

[14] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996.

[15] P. Srisuresh and K. Egevang. Traditional ip network address translator (traditional nat). RFC 3022, RFC Editor, January 2001. `http://www.rfc-editor.org/rfc/rfc3022.txt`.