Internal Routing

COSC301 Laboratory Manual

Note

We use Vyatta as our routing engine. It consists of several different parts. In 2013 a company called Brocade bought Vyatta, and have since discontinued the open source version of Vyatta, turning it into a commercial product. Prior to this, the community was quite large, now it's a shadow of its former self. Fortunately, some of the community have made a fork of the last remaining version of the open source version called VyOS. Most of the commands are the same, but there are some differences. Because of this we have decided to stick with Vyatta for the time being. The purpose of the lab does not depend on the version of the software we are using.

There are five or six parts to this lab; there is plenty to do but it is not difficult; perhaps a little time consuming. Unlike other labs, if you only get half the lab completed, you can get partial marks. The first part is to aquaint yourself with the Vyatta router platform by looking at the helpful video provided by Vyatta; you could prepare by watching this outside of class time. The second part is learning about Virtual LANs, which we shall use in the third part using VirtualBox to implement a particular network topology and boot the machines using the materials provided. Investigating static routing and the use of the RIP routing protocol will take up the last two parts and should take the majority of the time.

One thing worth noting in this lab is that you don’t have to do anything with IPv6. However, because IPv6 would be interesting to do, there is an optional section at the end which looks at making a similar addressing structure and using RIPng.

For this lab, and particularly the following lab on subnetting and firewalls, you may well prefer to work in pairs. Take turn about configuring the machine, and doing the research as to which commands you will need. We strongly suggest you to keep the topology map for this lab within sight at all times.

On your Windows PC, you should find that the Live CD images, as `*.iso` files, for Vyatta and Ubuntu should be available from the COSC301 class resources folder in the K: drive (Coursework), along with all the documentation you might need. As far as the Vyatta documentation is concerned, you should find available an electronic copy of the Vyatta Command Reference and Quickstart Guide in `Lab resources`.

1. Watch Vyatta Demonstration Video

Start this section by navigating to Youtube [https://www.youtube.com/watch?v=2IvDzvHB858] and viewing the video *An Introduction to Vyatta*, by Kevin Barton. This is approximately 12 minutes, and will give you some useful background information about Vyatta and the product.

2. Virtual LANs (VLANs)

One of the common, though somewhat advanced technologies that make designing and maintaining a LAN easier today than in previous years is the advent of Virtual LANs. Before we look at VLANs, let’s first have a brief look at the motivation for them, and cover some technological background.

This material is also available from the course website and in the class resource server as the “Back of the Envelope Guide to Virtual LANs” video.

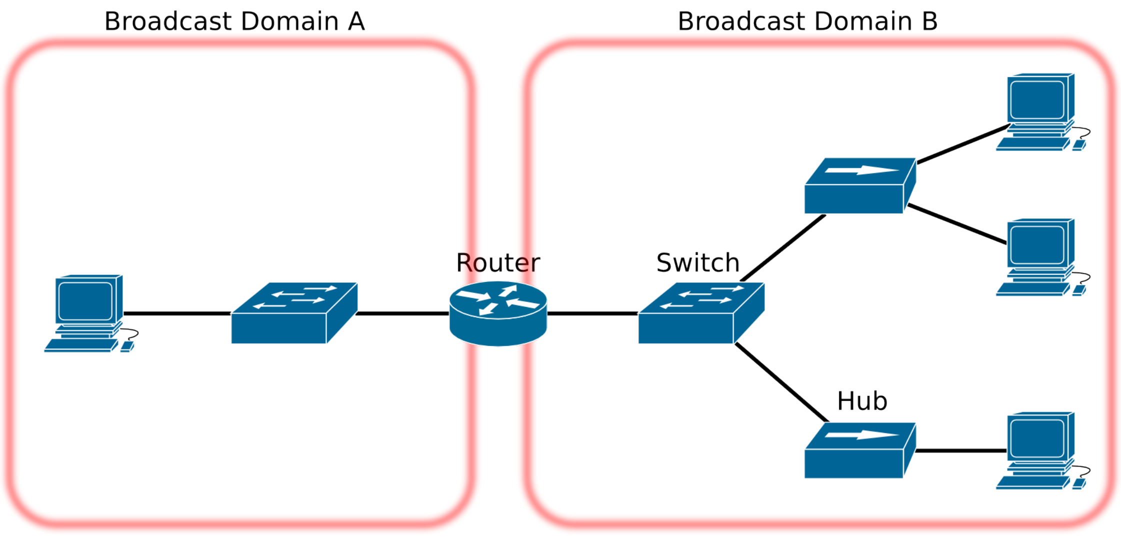
2.1. Broadcast Domains

Before we begin, we want to make clear that we are *not* talking about a *collision domain*. This is because the terms could be easily confused, and we want to make a clear distinction for you. A collision domain is what you have in a mixed ethernet segment, such as on a repeating hub. On a switching hub, the collision domain is simply the basic link between the host and the switch, typically a single cable; because the basic link is not shared between hosts, there are no other stations to contend for access to the physical medium.

A *broadcast domain* is everywhere a data-link level (eg. ethernet) broadcast frame would propogate to[1]. This area is demarcated by routers, which signal the end of a layer-2 (data-link layer) network; to go further requires support at a higher layer, such as layer-3 (eg. IP) to *route* the *packets* through the *inter-network*.

So, looking at the network below, we can see that there are two broadcast-domains, which are labelled as A and B. As an extra exercise, we suggest you also identify the various collision domains.

Figure 1. Broadcast Domains



A network showing two broadcast domains, and how they are connected using routers, switches and hubs. Try to identify the collision domains as well, to appreciate the difference between collision domains and broadcast domains.

2.2. A Virtual LAN

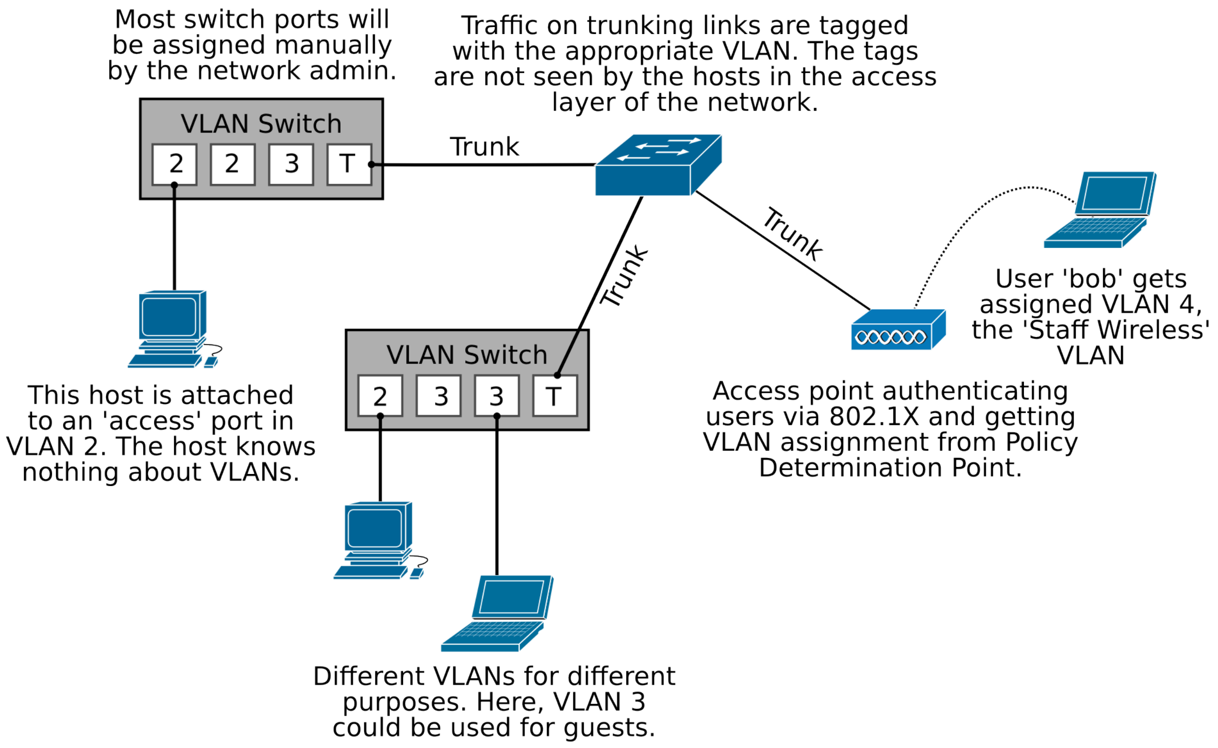
A Virtual LAN (VLAN) is, quite simply, the ability to segregate a switch into seperate broadcast-domains. This means that in order to get between the different VLANs, a router must be used. In the older days, when VLANs were still new, a *one-armed router* was used, which had an interface on both VLANs; today such a configuration would be more likely to

[1]This is assuming that the data-link layer being used supports broadcast. There are a number of Non-Broadcast Multiple Access (NBMA) network technologies; one example would be Frame Relay, which is used in a Wide Area Network (WAN) environment.

be called a “router on a stick”. Today however, a high-speed router is embedded as part of the switch; this switch is then referred to as a *layer-3 switch*.

VLANs are identified by a 12-bit number (4096 different VLAN IDs are possible). A switch-port may be a member of a number of VLANs; in the case of multiple VLAN assignments to a port, *trunking* must be used, which *tags* that the frames their VLAN identifier, so the next device (typically a switch or a router) can know which (virtual) LAN it belongs to. Figure 2, “VLAN Assignment and Trunking” should make this clear.

Figure 2. VLAN Assignment and Trunking


Most switch ports will be assigned manually by the network admin.
Traffic on trunking links are tagged with the appropriate VLAN. The tags are not seen by the hosts in the access layer of the network.
VLAN Switch
2
2
3
T
Trunk
Trunk
Trunk
User 'bob' gets assigned VLAN 4, the 'Staff Wireless' VLAN
VLAN Switch
2
3
3
T
This host is attached to an 'access' port in VLAN 2. The host knows nothing about VLANs.
Access point authenticating users via 802.1X and getting VLAN assignment from Policy Determination Point.
Different VLANs for different purposes. Here, VLAN 3 could be used for guests.

How VLANs are specified, including static assignment by switch-port, assignment by 802.1X authentication (“port” based authentication) and trunking. Not shown is the routing support and access control to allow traffic to flow between VLANs. VLAN 1 is generally reserved for management traffic and all ports generally default to being in VLAN 1. In particular, if the switch has an IP address for management purposes, it starts off in VLAN 1.

Also not shown is the Policy Determination Point, which is generally some server that tells the access-point (acting as a Policy Enforcement Point) what VLAN to assign a client to, as well as access-control data. These terms are particular to the field of Network Access Control, and are not talked about any further in this lab.

There are three layers to a standard Enterprise network design. The *Access layer* of a network is where clients connect to the switches. Traffic that needs to go to somewhere else on the network goes through the *uplink* to a *Distribution layer* switch (commonly there would be at least two, for redundancy). The Distribution switches aggregate a number of Access switches, and on their uplinks connect to the Core switches. As we move into the core, the switches get more and more powerful.

Clients, which are at the access layer of the network, will not have any idea that VLANs are in use, which is what we want, because it means the client doesn’t have any extra configuration to deal with. Thus, in this case, we say the switch-port is an “access” port, rather than a “trunk” port. In this case, the access layer device (typically a switch or wireless access point) will determine the VLAN based on the switch-port (typical for ethernet) or authentication data (typical for enterprise wireless access using 802.1X and RADIUS.

Take a moment to refer again to the picture above. Can you see the benefits we get from using VLANs when we have different classes of device? (business, staff wireless, etc.) Hint: think about the maintainance activities in a network (Moves, Adds and Changes).

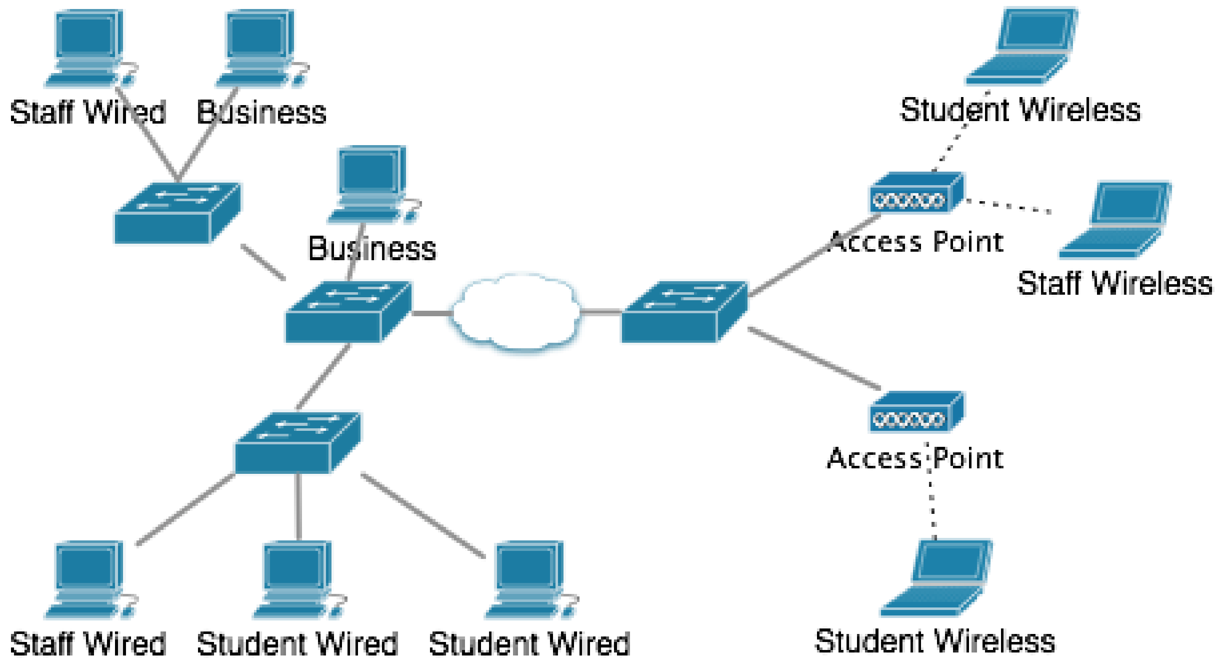
Can you imagine how much more complex the network would have to be if we *didn’t* have VLANs? We would loose a lot of flexibility, and cost would be very much higher. We would at least need many more switches and access-points, routers and cable. Running extra cable would be the most expensive part. We investigate this further in the next section.

2.3. The Motivation for Virtual LANs

Briefly, a VLAN gives us three major benefits: traffic control by prioritising traffic in particular VLANs or reducing broadcast traffic by making the broadcast domains smaller; security, by controlling traffic between different VLANs (subnets); and flexibility in network design without extra equipment.

We like to have flexibility in a network to move clients and servers into different subnets depending on their role and security level; firewalls are one-such tool that can help us here, which we cover in the following lab. Consider the network shown below, which is representative of a university campus where students can have their own laptops on the network. In this network, there are different security classes of device: student wireless, student wired, staff wireless, staff wired, and business (corporate) devices as distinct from academic staff. We want each of these to have their own subnet so we can control traffic going between them.

Figure 3. Using Virtual LANs to divide a network.



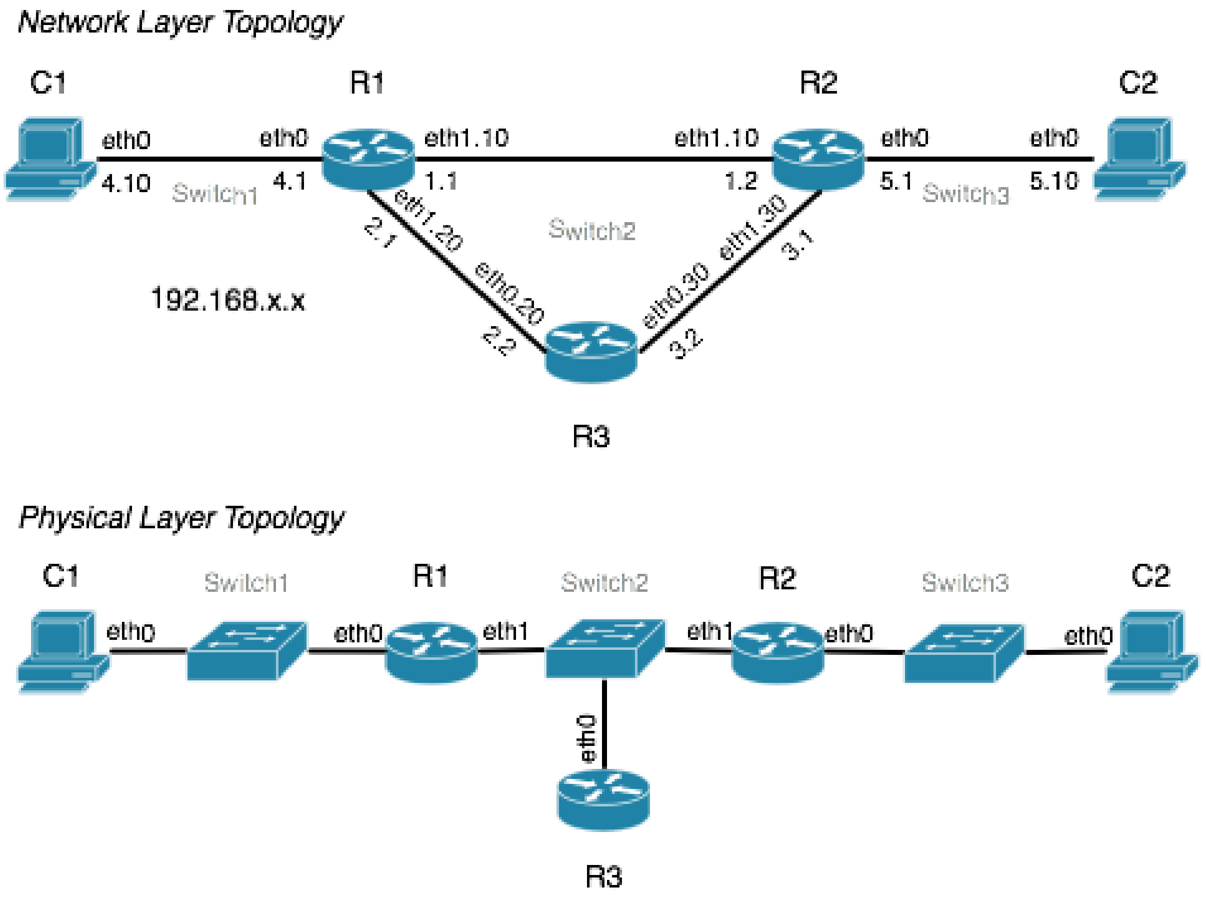
Using Virtual LAN technology gives us much greater flexibility and simplicity in how we design and implement switched networks.

With the use of VLANs in this network, we can have machines in different subnets that are physically dispersed within the network. That is something that would otherwise be quite impractical.

There is more to be said about VLAN management, most notably about how a VLAN database mapping between VLAN indentifiers and a name can be shared amongst the various switches, using the VLAN Trunking Protocol (VTP). More difficult is how you can automatically assign a VLAN based on protocols such as 802.1X, but that is outside the scope of this lab.

3. Configure VirtualBox with the Topology

Figure 4. Interior Routing Network Topology



The top part of the figure shows the Layer 3 network topology diagram showing address interfaces (VLAN is used on most interfaces) and the switch they are plugged into. This is not a typical deployment of VLANs because the redundancy we achieve at the IP layer is undone by the single point-of-failure of Switch 2. The bottom part of the figure shows the Layer 1 physical topology diagram. You can use this diagram to appreciate how the network equipment would be physically connected in this lab.

In this section you will be using VirtualBox to create, configure and connect the devices in the network:

1. You will define a virtual machine for each host and router in the network, connecting them appropriately to the switches, which will be created for you by Virtualbox. This defines the Physical layer topology (layer 1).

2. After booting the devices in the network, you will configure the software inside them to build the Network layer toplology (layer 3).

Figure 4, “Interior Routing Network Topology” shows the topology at layer 3 (the Network layer: IP) and also at layer 1 (the Physical layer: the cables), to help you to appreciate how the devices would physically connect to each other.

Each of the switches will need to be created in our virtual environment, so they are given simple names Switch1, Switch2 and Switch3 so we can distinguish between them; better names might have been Core, AccessEast and AccessWest as one example of many.

Start up VirtualBox. We shall begin by adding the Router R1. As we can see from the network map R1 has three logical interfaces[2] but two physical interfaces. We need to say what physical interfaces (corresponding to ethernet cards on a real machine) we want each machine to have.

Create a machine, call it IntRoute_R1 or something suitable so you don’t get it confused with other machines in the lab and other machines you might create in a later lab. The routers are command-line only and don’t need much memory, but since they won’t have any swap, we don’t want to underestimate the amount of memory; 128 MB should be plenty. Because we are using a Live CD we don’t need a hard-disk, so don’t configure one. Have it boot only from CD/DVD-ROM, taking care to remove the floppy drive from the boot order.

Note that Vyatta requires that the virtualised system has a CPU feature called *Physical Address Extensions*, so you will also require the Enable PAE/NX feature in the Processor section of the System tab.

In the Storage tab you will need to check Mount the CD/DVD Drive and use the ISO Image File. If the vyatta-livecd-virt_VC6.1-2010.10.16_i386.iso option is not present, click on the folder icon next to the drop-down box; click Add to register the ISO file with Vyatta so it now appears as a choice. Then you can click on Select to use the VC6.1 (Vyatta Community-edition version 6.1) ISO image.

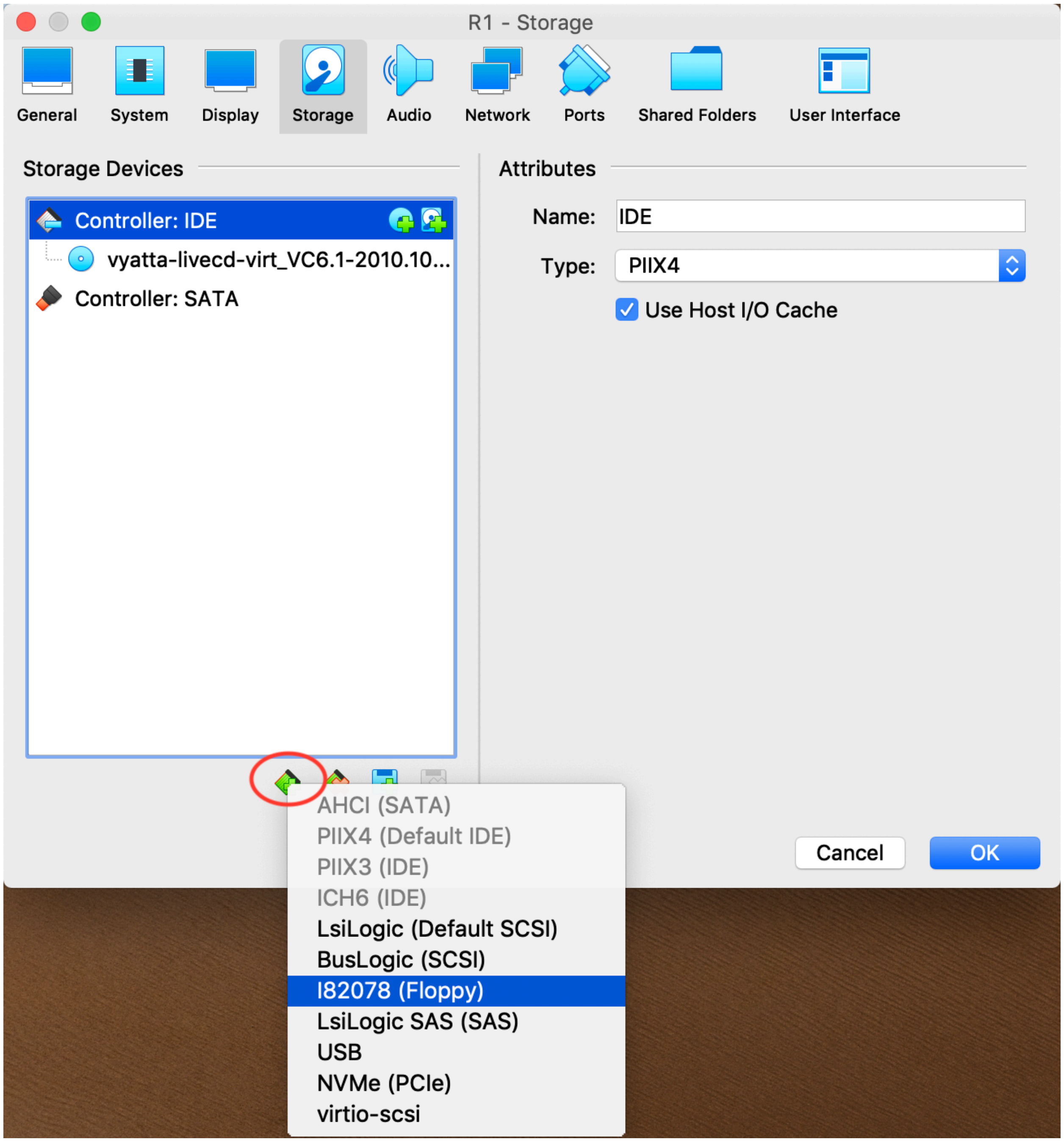
For each router we will need to create a virtual floppy disk (another image file) to store our router configuration. To do this open `Git Bash` app on your Windows PC and type the following command to create a 1.44 MB (actually 2 MB, but close enough) file full of empty space. You should then move these image files to your `myvms` directory in the J: drive.

```
$ for R in R1 R2 R3; do
> dd if=/dev/zero of=IntRoute_${R}_floppy.img bs=1k count=1440
> done
```

Back in VirtualBox, add a floppy controller by clicking on the Add Controller icon (first button from the left under the Storage Tree) and selecting I82078 (Floppy). The sequence of steps for adding a floppy controller and disk is illustrated in Figure 5, “Adding a Floppy Controller and Disk to VirtualBox (1)”, Figure 6, “Adding a Floppy Controller and Disk to VirtualBox (2)”, and Figure 7, “Adding a Floppy Controller and Disk to VirtualBox (3)”. The final state of the storage tree should look like Figure 8, “Adding a Floppy Controller and Disk to VirtualBox (4)”. Remember that each machine should use a *different* floppy image you made previously.

[2]Four if you count the loopback interface, but we don’t care about that at the moment.

Figure 5. Adding a Floppy Controller and Disk to VirtualBox (1)



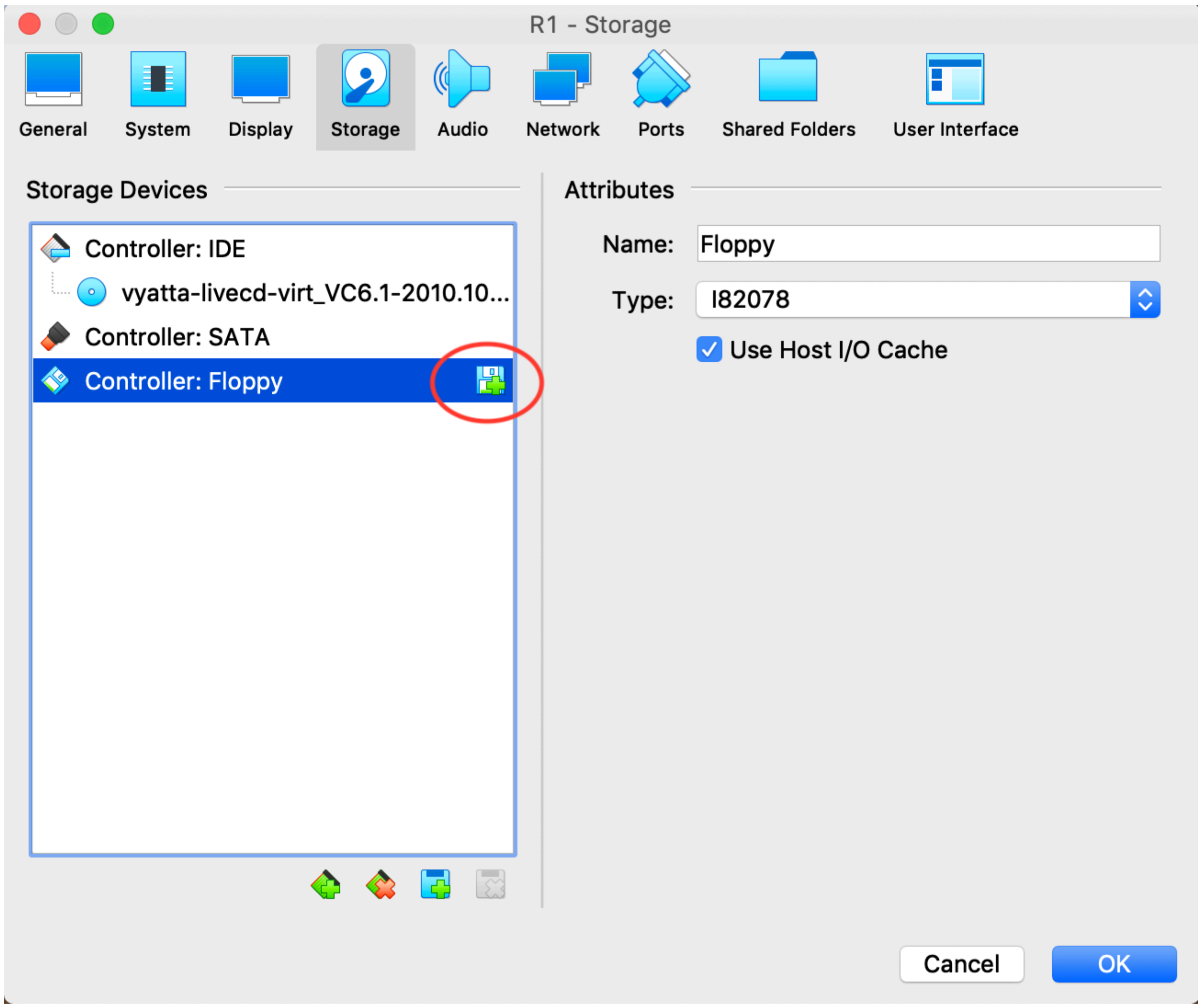
Name: IDE

Type: PIIX4

[✓] Use Host I/O Cache

The sequence of steps for adding a floppy controller to a Virtualbox virtual machine, and assigning a floppy disk image to it (Step 1).

Figure 6. Adding a Floppy Controller and Disk to VirtualBox (2)



Storage Devices

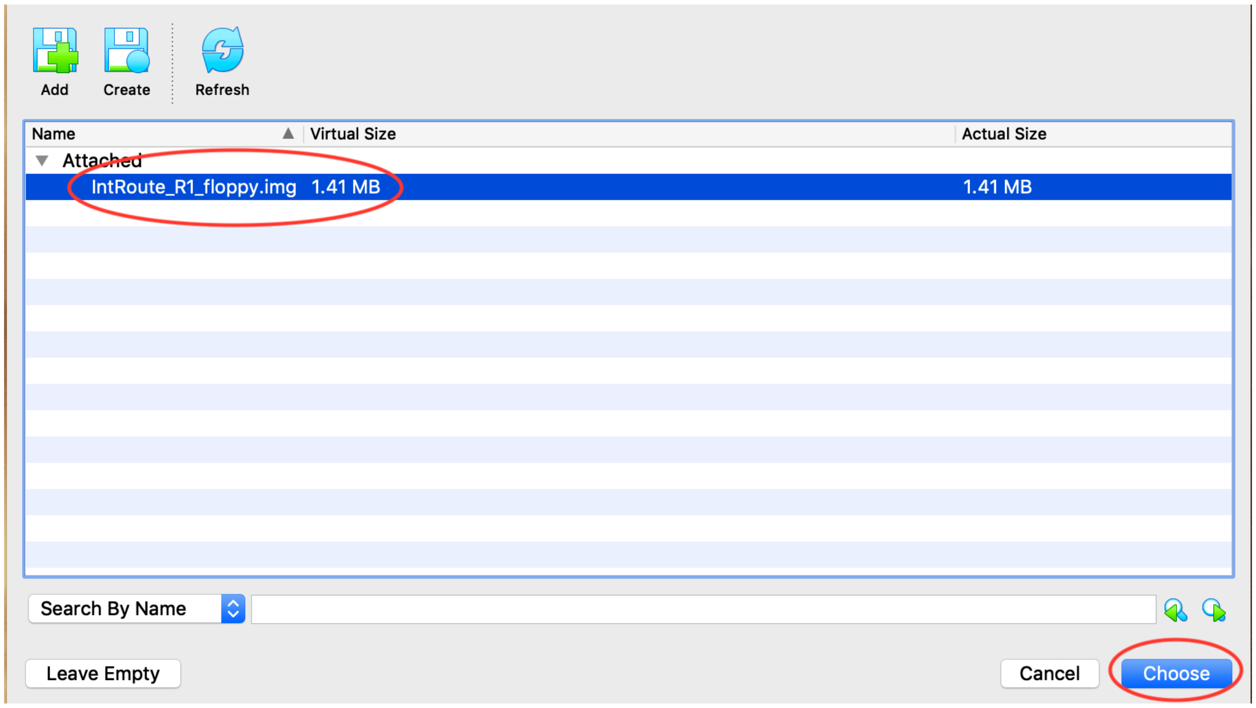
Controller: IDE
vyatta-livecd-virt_VC6.1-2010.10...
Controller: SATA
Controller: Floppy

Attributes

Name: Floppy
Type: I82078
✓ Use Host I/O Cache

The sequence of steps for adding a floppy controller to a Virtualbox virtual machine, and assigning a floppy disk image to it (Step 2).

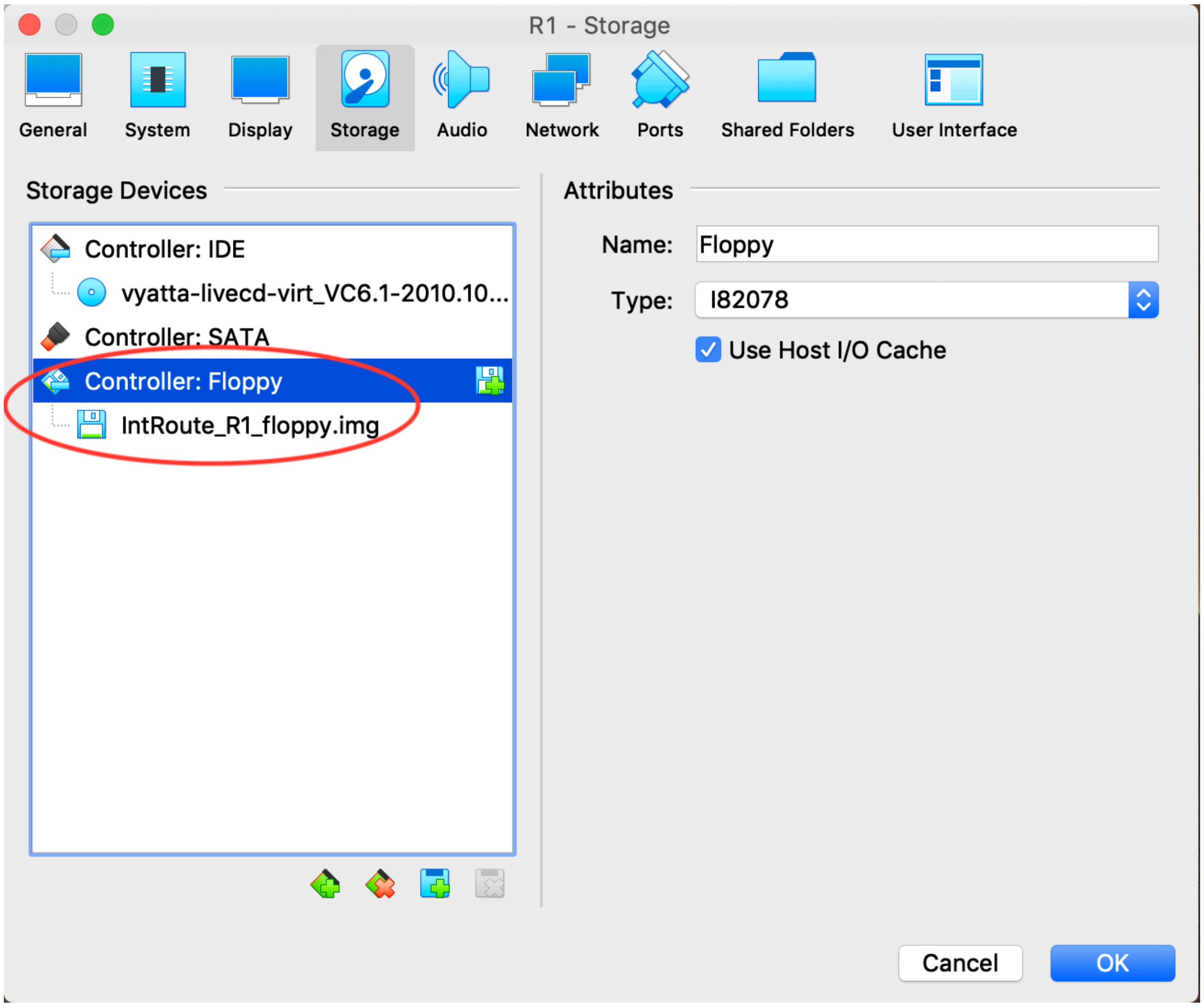
Figure 7. Adding a Floppy Controller and Disk to VirtualBox (3)



Name	Virtual Size	Actual Size
Attached		
IntRoute_R1_floppy.img	1.41 MB	1.41 MB

The sequence of steps for adding a floppy controller to a Virtualbox virtual machine, and assigning a floppy disk image to it (Step 3).

Figure 8. Adding a Floppy Controller and Disk to VirtualBox (4)



Controller: IDE
vyatta-livecd-virt_VC6.1-2010.10...
Controller: SATA
Controller: Floppy
IntRoute_R1_floppy.img

Name: Floppy
Type: I82078
✓ Use Host I/O Cache

Final state of the storage tree after the sequence of steps for adding a floppy controller to a Virtualbox virtual machine, and assigning a floppy disk image to it.

In the Network tab, we need to enable the particular virtual ethernet adaptors we need. R1 has two “physical” Ethernet adaptors: eth0 and eth1 (eth1 will be split into two logical adaptors when using VLAN). For all adaptors in this lab we want to use Attached to: Internal Network. Set the Network Name: field for R1’s Adaptor 1 (eth0) to the string `IntRoute-Switch1`. Enable R1’s Adaptor 2, and likewise connect it to `IntRoute-Switch2`; note that the first switch is now available in the drop-down box (we’ll use this later when configuring the other nodes, i.e., routers and clients, in the network).

Don’t use the Intel PRO/1000 family of adaptors

For every adaptor used by the routers, particularly those connected to a VLAN trunk (ie. connected to the second switch), go into the Advanced adaptor properties, and set the Adaptor Type to either the Paravirtualised adaptor or the AMD PCNet Fast III.

This is because the Intel PRO/1000 family of adaptors will strip off the VLAN tags as they leave the machine, causing your routers not be able to communicate, which is highly annoying. There is no supported way inside Vyatta (or Linux in general) for changing this.

In the Ports tab, disable USB (routers don't typically have USB ports); under the System tab, use USB Tablet as a pointing device[3].

Back in in the Ports tab, enable the Serial port COM1 and set the Port Mode to Disconnected. Vyatta expects a COM1 (`/dev/ttyS0` in Linux-speak) to be present so it can offer a serial-terminal, just like a real router. If this device is not present, it will occassionally complain, but it will still work. We shall use this later on in the next lab.

Screenshot

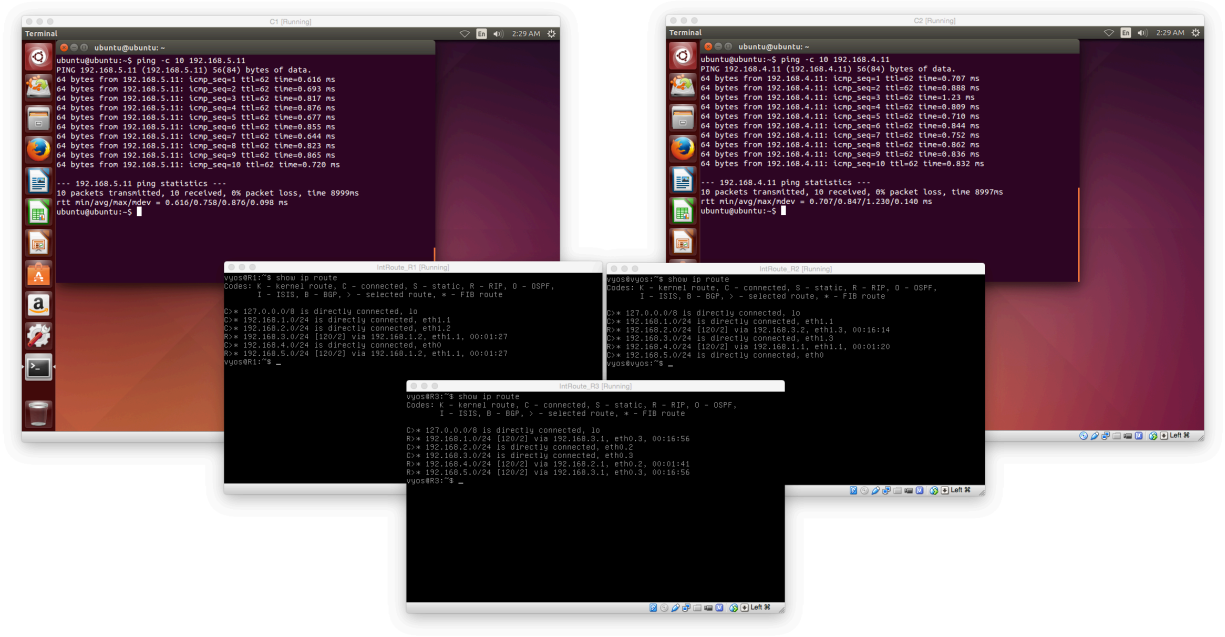
Now go on and configure all of the other routers and hosts. Remember that for the hosts C1 and C2 no floppy is needed and they will be booting from an Ubuntu Desktop Live CD, except you need to increase their RAM to more than 2GB. Also, don't forget to put the CDROM at the top of the boot list, or VirtualBox may look at other existing drives first, which cannot boot. You can leave all other configurations according to their defaults, which should be reasonable if you selected Ubuntu Linux as the operating system type in the wizard. Take a screenshot of each of the configuration summaries shown by VirtualBox.

When you have configured virtual machines for R1, R2, R3, C1 and C2 you are ready to boot your routers and hosts. When powering on a network of machines, we need to consider the order of which machines are started. Clients will typically (though not in this particular lab) be configured to use various network services such as DHCP and DNS, so the network and the servers need to be available first. Therefore, start the routers first then the hosts when the routers have finished booting.

When the routers boot they *might* complain about the floppy disks, because we have essentially given them a floppy disk each with no filesystem on it, and it's expecting to find a filesystem with configuration data on it. This is unlikely to be a problem for you initially, but you may still run into it if you reboot the router later. The initial login and password for all routers are **vyatta**

Once you have them all running you may like to arrange them on your desktop into roughly the same shape as the topology; you may find that doing this helps you to trace how traffic moves through your network.

[3]Which can be preferable because it means it doesn't need to capture the mouse.



```
ubuntu@ubuntu:~$ ping -c 10 192.168.5.11
PING 192.168.5.11 (192.168.5.11) 56(84) bytes of data.
64 bytes from 192.168.5.11: icmp_seq=1 ttl=62 time=0.616 ms
64 bytes from 192.168.5.11: icmp_seq=2 ttl=62 time=0.693 ms
64 bytes from 192.168.5.11: icmp_seq=3 ttl=62 time=0.817 ms
64 bytes from 192.168.5.11: icmp_seq=4 ttl=62 time=0.876 ms
64 bytes from 192.168.5.11: icmp_seq=5 ttl=62 time=0.677 ms
64 bytes from 192.168.5.11: icmp_seq=6 ttl=62 time=0.855 ms
64 bytes from 192.168.5.11: icmp_seq=7 ttl=62 time=0.644 ms
64 bytes from 192.168.5.11: icmp_seq=8 ttl=62 time=0.823 ms
64 bytes from 192.168.5.11: icmp_seq=9 ttl=62 time=0.865 ms
64 bytes from 192.168.5.11: icmp_seq=10 ttl=62 time=0.720 ms

--- 192.168.5.11 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 0.616/0.758/0.876/0.098 ms
ubuntu@ubuntu:~$
```

```
ubuntu@ubuntu:~$ ping -c 10 192.168.4.11
PING 192.168.4.11 (192.168.4.11) 56(84) bytes of data.
64 bytes from 192.168.4.11: icmp_seq=1 ttl=62 time=0.707 ms
64 bytes from 192.168.4.11: icmp_seq=2 ttl=62 time=0.888 ms
64 bytes from 192.168.4.11: icmp_seq=3 ttl=62 time=1.23 ms
64 bytes from 192.168.4.11: icmp_seq=4 ttl=62 time=0.809 ms
64 bytes from 192.168.4.11: icmp_seq=5 ttl=62 time=0.710 ms
64 bytes from 192.168.4.11: icmp_seq=6 ttl=62 time=0.844 ms
64 bytes from 192.168.4.11: icmp_seq=7 ttl=62 time=0.752 ms
64 bytes from 192.168.4.11: icmp_seq=8 ttl=62 time=0.862 ms
64 bytes from 192.168.4.11: icmp_seq=9 ttl=62 time=0.836 ms
64 bytes from 192.168.4.11: icmp_seq=10 ttl=62 time=0.832 ms

--- 192.168.4.11 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8997ms
rtt min/avg/max/mdev = 0.707/0.847/1.230/0.140 ms
ubuntu@ubuntu:~$
```

```
vyos@R1:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.1.0/24 is directly connected, eth1.1
C>* 192.168.2.0/24 is directly connected, eth1.2
R>* 192.168.3.0/24 [120/2] via 192.168.1.2, eth1.1, 00:01:27
C>* 192.168.4.0/24 is directly connected, eth0
R>* 192.168.5.0/24 [120/2] via 192.168.1.2, eth1.1, 00:01:27
vyos@R1:~$ _
```

```
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.1.0/24 is directly connected, eth1.1
R>* 192.168.2.0/24 [120/2] via 192.168.3.2, eth1.3, 00:16:14
C>* 192.168.3.0/24 is directly connected, eth1.3
R>* 192.168.4.0/24 [120/2] via 192.168.1.1, eth1.1, 00:01:20
C>* 192.168.5.0/24 is directly connected, eth0
vyos@vyos:~$ _
```

```
vyos@R3:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
R>* 192.168.1.0/24 [120/2] via 192.168.3.1, eth0.3, 00:16:56
C>* 192.168.2.0/24 is directly connected, eth0.2
C>* 192.168.3.0/24 is directly connected, eth0.3
R>* 192.168.4.0/24 [120/2] via 192.168.2.1, eth0.2, 00:01:41
R>* 192.168.5.0/24 [120/2] via 192.168.3.1, eth0.3, 00:16:56
vyos@R3:~$ _
```

4. Configure System Basics

The first thing to do in each Vyatta router is to format (initialise) the (virtual) floppy disk, so we have somewhere to permanently save our configuration when we run **save**. Otherwise, we lose our configuration when we reboot. Initialise the floppy using **init-floppy**. When you do, you will notice that it says that it is writing the configuration file to `/media/floppy/config/config.boot` (later, when saving, it might say `/opt/vyatta/…`, which doesn't seem much like a floppy disk, but its the same location). If you look at the floppy-disk icon at the bottom of the Virtualbox VM window, you will notice it flashing orange as it gets written to.

In configuration mode (**configure**), **set** the **system host-name** to the name of the router: R1, R2 or R3. Type **commit** to apply the new configuration (you will not see any changes until you log out) and then use **save** to make the change permanent.

Secure the router's authentication by adding a user for yourself with a password.

```
edit system login user theauthor
set full-name "The Author"
set level admin
set authentication plaintext-password new_password
show authentication
# note that the password appears in the clear
commit
show authentication
# note that the password has now been hashed
top
```

commit, **save**, **exit** from configuration mode, and **logout** of Vyatta. Login as the user you just created.

It's always nice to have your logs close to hand, and we can ask Vyatta to put a copy of important logs onto our console:

```
set system syslog console facility all level warning
commit
```

```
logger -p warning HELLO WORLD
```

Note that the last line is not a Vyatta command at all, but a standard command that you could type into a shell; the Vyatta shell implements what Vyatta calls “Fusion”: commands that are not recognised as Vyatta commands are run as shell commands. Vyatta’s shell is a modified version of the Bash shell.

Repeat these basic configurations on R2 and R3.

Silence irritating but innocuous error messages

You may often see such an error message "INIT: Id "T0" respawning too fast: disabled for 5 minutes". You can silence it by commenting out the line **T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100** with a "#" at the front of the line. Then run **sudo init q** to make the change effective.

5. Static Routing

Routers R1-3 will (in the next section) participate in dynamic routing using RIPv2 (Routing Information Protocol version 2). Clients C1 and C2 do not participate in RIPv2; instead they have a default route set to their respective router. The clients run Ubuntu Linux. Figure 4, “Interior Routing Network Topology” shows the addressing (the entire network is numbered out of the 192.168.x.x range of private addresses) and the interface device (“eth0” etc.) on each link.

Each switch here is labelled with a name, which is not particularly used for Ethernet at all, but is used more for management purposes. In particular, VirtualBox will create and manage the switches as needed.

This laboratory uses Virtual LANs (VLANs) in the middle part of the network. An interface such as `eth0.10` means VLAN 10 on the `eth0` interface, and is presented as a separate interface, which can have its own address, etc. When configuring interfaces inside the Vyatta shell, you would refer to this particular virtual interface as “interface ethernet eth0 vif 10”. The VLAN is created automatically, no special commands are required.[4]

Complete the following tasks:

- Consult the relevant parts of the Vyatta Command Reference; you will find them in the `Lab Resources/Vyatta/Documentation` folder, which are colour-tagged to make you find them easily (Hint: search commands like **set interfaces** and **set protocol static**). You will be well advised to make good use of Tab completion. For example, using Tab completion, you would find a useful command **set interfaces ethernet eth*0* vif *10*** for creating a new VLAN (with an ID 10).

- Configure the basic system for the routers; hostname, accounts (change the default passwords) and interface configuration. Do not enable access via SSH or HTTP, although you are welcome to try it later.

- You will also need to configure the interfaces on the clients C1 and C2. Because any configuration will perish upon boot, you can just use **ifconfig** and **route** to set up the network interfaces. Refer back to the earlier lab on Basic Interface Management where these commands were practiced. Note that you will quickly find out you need to install `net-`

[4]Which is to say that Vyatta calls the Linux command **vconfig** for you.

`tools` with **apt-get**. So you need to enable the second network adaptor connecting to NAT in order to have Internet access.

- Configure static routes for all routers. Use the **traceroute** command to verify reachability throughout your network. If you are in configuration mode, use **run traceroute ip**, otherwise use **traceroute ip** in operational mode. Note that the Vyatta shell has its own **traceroute** command; there is also the system command (**/usr/bin/traceroute**). The major difference is that the Vyatta shell command doesn't take extra arguments, such as `-n` which would otherwise be used to prevent lengthy DNS lookups. Thankfully, the Vyatta shell version of **traceroute** doesn't do DNS lookups anyway.

Screenshot

If **traceroute** is not available try **mtr** instead. Take a screenshot of your routing table and testing on all machines. You should verify that you can get from any interface in the network to any other interface.

- Remove all static routes on R1-3 after you have verified all your work and recorded suitable evidence; you can remove them easily using the configuration command **delete protocols static** and then **commit**ing your configuration. Once you have the static routes removed, start working on your RIPv2 configuration.

Partial Marks Available

If you only complete up to here (getting Static Routing completed and tested), then you are eligible for partial marks.

6. TCPDump

Important

Read this section, but do not try to do this until you have completed the RIP configuration task in the next assessment.

In the following section you will be required to use a *network sniffer* (traffic capture) utility called **tcpdump**, which is a widely known program for seeing what traffic is going through a network interface.

Note

Although we say "traffic capture" occasionally, we do not prevent the packet from reaching its destination. In this way, we are capturing a *copy* of the packet.

Because **tcpdump** requires root privilege, you will need to reinstate the ability to login as root. You can do this with the configuration command **set system login user root authentication plaintext-password `roots_new_password`** and then using **commit**. Now if you logout you will be able to login as root. Root gets a slightly different shell to standard Vyatta users which allows you to use standard system commands, such as **ls** or **tcpdump**.

- **tcpdump** is a very well known network sniffer in the Unix world. It is a command-line tool that takes a Packet CAPture (PCAP) expression and watches all the packets coming into, or going out of, an interface on the machine. For those packets that match the expression, a brief description of the packet header is printed.

- Have a look at the manual page for tcpdump(8), where you will find a rich number of examples near the bottom.

  RIP traffic uses the “router” port (UDP port 520), and if you tell **tcpdump** to output in verbose mode, it will decode the RIP advertisements. You will want to grab (“snarf”) all the packet (`-s0`), otherwise you will only get a small part of the body of the packet, and not all of the advertisement will be printed.

```
# tcpdump -i ethXXX -v -s0 udp and port 520
```

Typically, an Ethernet interface will only pass up to the operating system those ethernet *frames* that are either addressed to its own MAC address, or are a broadcast or multicast frame.

To enable functionality for traffic sniffing or packet forwarding (ie. acting as a router) the interface needs to pick up all frames. Accepting all packets in this way is called the *promiscuous* mode. Enabling promiscous mode is done automatically by **tcpdump** and similar tools on most platforms.

All these tools can commonly understand the PCAP (Packet CAPture) format, so you can use **tcpdump** to capture packets from a remote machine to a file (the `-w` option), copy it to your local machine, and then analyse it further using tools such as Wireshark.

7. RIP Assessment

Using RIP is actually very simple. Most of the time will likely be spent waiting for the routing tables to *converge* to a stable solution.

Important

Don’t forget that you need to remove any static routes from the routers R1-3 before you start on this section.

1. Configure the routers for use with RIP. You must refer to the Vyatta documentation to find out how to do this. Hint: search commands like **set protocols rip** and make sure redistribution of RIP information is enabled.

Tip

You will find you have a choice between advertising a “network” or an “interface” when setting the RIP protocol. In this case, advertising the “interface” is preferable, because it means that it will use the address assigned to a particular interface. This could reduce typing mistakes, but could also make it harder to check if you have many interfaces, as Vyatta doesn’t allow for easily renaming interfaces. If you set up RIP correctly, everything should work by default. Show the routing table of each router with **show ip route** or **run show ip route** in the configuration mode and test the routing functions with **ping** in the clients.

2. Observe RIP traffic using **tcpdump**. Listen on R3’s eth0.20 or eth0.30 interface. You will notice that some routes are missing in the advertisements. Explain why this happens. Hint: check RIP optimisations like *split horizon* in the lecture notes.

3. Go back into configuration mode, and enable poison-reverse using the configuration command **set interfaces ethernet ethX vif Y ip rip split-horizon poison-reverse** for each interface on each router.

Using **tcpdump** as before, what has changed? Hint: you will see some routes have a hop-count of 16, which means "unreachable". check RIP optimisations like *poison reverse* in the lecture notes.

4. Rather than using the system command **tcpdump**, as was done in the previous questions, we can get a lot more debugging information from Vyatta itself. Investigate which debugging commands might be useful for showing what advertisements are being received. Hint: **debug rip packet** and **show log tail**.

Note that **show log tail** is a particularly important command for actually seeing the logs, but it is unfortunately not at all obvious when reading the documentation for the debug commands related to RIP and others.

Be aware that in order to stop debugging output being produced, you will want to use **no debug …** instead of **debug …**. The arguments need to be the same. You can use **show debugging rip** to see what debugging options are enabled.

5. In VirtualBox, we can connect and disconnect adaptors from a machine reasonably easily. In the bottom part of a VirtualBox VM window, you can click on the network icon to unplug or plug an interface. This is the same as plugging or unplugging a cable. We’re going to use this feature to disconnect the 192.168.4.0/24 network from R1, and see how long it takes for R2 to learn of the outage, by following the instructions below.

To see changes of R2's routing table, we shall use the system command **ip monitor** (look back to the early lab where tools such as **ifconfig** were introduced). In modern versions of Vyatta that feature the FusionCLI interface, we can enter shell commands as well as Vyatta commands; the Tab completion shows only the Vyatta commands, but never shell commands, as you might have noticed.

1. In operator mode (ie. not configuration mode) run **show log tail** on R1; we should see a notification here when we unplug the interface.

2. On R2, run **ip -timestamp monitor**. This will start reporting for changes to the routing table.

3. On R1, un-tick the Adaptor 1 interface so it becomes unplugged, a log will be printed on-screen.

4. On R2, there will shortly be a message reporting the change. Note the time it took; the clocks will be synchronised, so use the timestamps from both machines.

5. That was fail-over. Re-enable the interface on R1. Measure how long it takes to fail-back.

What were the fail-over and fail-back times? What improvement to basic RIP is being used here?

The fail-over and fail-back times should be very quick, basically, about a second. This shows R1 is sending a *triggered update* about the change, rather than waiting for its next time period of advertisement.

6. The previous exercise simulated a situation whereby the isolation occured on the *far* side of a router, and the router could therefore tell us about it. Let’s now simulate a fault on the *near* side of the router where the router R1 is cut-off from the routers R2 and R3.

Consider the *physical* topology diagram shown earlier. If you disconnect Apaptor 2 (ie. eth1) on R1, *both* the 192.168.1.0/24 and 192.168.2.0/24 network links on R1 would be useless (which are built from VLANs that go over the same cable). Therefore, if Adaptor 2 goes down, it completely separates the network.

Now take down Adaptor 2 on R1, and measure how long it takes for R2 to respond by removing routes for those prefixes which are no-longer reachable. For example, how long does this take? what prefixes are removed from R2’s routing table?

Put the interface back, how long does fail-back take?

7. In the previous exercise, it took R2 a while to realise that it hadn’t received a message from R1 for a while and to place it in the “hold-down” state, which allows it to ignore updates about this route for a while in order to let any old routing information expire from the network. In this scenario, we are going to simulate a routing problem whereby this time the routers can route around the problem.

On R2, quit out of **ip monitor** using Control-C and use the operational command **show ip route** to observe the current state of the routing table (our “steady-state”). Then start **ip -timestamp monitor** to watch for R2’s reaction to the change we are about to make.

Simulate the network failure by withdrawing RIP on the eth1.10 interface of the router R1. You can do this using the command **delete protocols rip interface eth1.10** and then **commit**ting your change. Use the command **date** immediatly after to determine when you made the change. Remember that the clocks on R1 and R2 are running on the same host, and are implicitly synchronised.

When you see R2’s reaction, quit the monitor and observe the routing table again. How has it changed (look closely to which next-hop is being used for each network)? Ensure you can describe the meaning of any changes. Was an alternative route used immediately by R2, or was there some time gap between the deletion of the failover route and the setup of the alternative route? How long was the gap? Why? Hint: check RIP optimisations like *hold down timer* in the lecture notes.

Start the monitor again and allow R1 to participate in RIP on eth1.10 again. Don’t forget to commit the change and run **date**. How long does it take for R2 to fail back to the optimal path?

8. [Optional] Connecting to the Outside World

When you are done, and looking for something else to spark your interest, you might try these optional activities.

In this optional section, you’re going to add another adaptor to R1, connected via Virtualbox NAT, and advertise this as a default route to the rest of the network.

Add an extra adaptor to one of the routers with a VirtualBox NAT type interface and originate a default route using RIP. The interface will have to be configured via DHCP (as a client). You should now be able to access the network beyond the host using TCP services such as SSH; note that **ping** won't work, as the VirtualBox NAT adaptor can't support ICMP.

Note that it is not our router that is performing the NAT, but rather some VirtualBox device just beyond. We shall be investigating doing NAT ourselves in the next lab.

Because we're going to need VirtualBox's NAT system to know more about our network (in order to figure out where to route traffic coming back into the network), we're going to require something a little more complex with regard to how we configure VirtualBox to do the NAT. To do some of this, we have to use the command-line management tool **VBoxManage** instead of the GUI.

Make sure you have **save**d your configuration on R1, and **shutdown** the system gracefully. Then, on a terminal window on your Mac workstation, run the following commands:

```
Add Adaptor 3 (eth2), which is a NAT adaptor
$ VBoxManage -q modifyvm IntRoute_R1 \
>   --nic3 nat \              Make Adaptor 3 connected via VirtualBox NAT
>   --nictype3 virtio \       Setting adaptor type [performance]
>   --natnet3 "192.168/16" \  Use 192.168.x.x instead of 10.0.x.x
>   --natdnsproxy3 on \       Advertise DNS server at 192.168.0.3
>   --natdnspassdomain3 off   Don't advertise DNS search path of host
```

Now start R1, and configure the new interface:

```
$ configure
$ set interfaces ethernet eth2 address dhcp
$ commit
$ run show dhcp client leases
interface  : eth2
ip address : 192.168.0.15         [Active]
subnet mask: 255.255.0.0
router     : 192.168.0.2
name server: 192.168.0.3
dhcp server: 192.168.0.2
…
```

Note that the subnet our interface is placed in is 192.168.0.0/**16**, which is good for us because it means all of our addresses in our network fall under that. In essence, we have hierarchical addressing. As far as our gateway to the outside world (VirtualBox NAT) is concerned, anything addressed to something in 192.168.x.x will be sent onto the local network which R1's eth2 interface will be attached to.

This means that traffic coming from the outside that *ought* to be routed *via* R1 as the next-hop, will not. Take for example traffic returning to C2 (192.168.5.10). VirtualBox will think that it should be a local delivery, so will ARP for C2's hardware address. We work-around this problem (solving the problem properly involves being able to add a route entry to VirtualBox's NAT engine) by enabling *Proxy-ARP* on R1's eth2 interface: **set interfaces ethernet eth2 ip enable-proxy-arp**.

The VirtualBox manual states that, given the configuration used above, the only addresses that are used are 192.168.0.15 for the guest, 192.168.0.2 for the router and 192.168.0.3 for the DNS name server. Thus, we can set aside 192.168.0.0/**24** in our addressing plan as being used for the network between R1 and the VirtualBox NAT, which conveniently leaves the rest of our network unchanged.

After you've added and tested the interface, test connectivity. Remember though, that we cannot use **ping** to test because the VirtualBox NAT doesn't support ICMP. R1 was configured with DNS settings via DHCP, so we should be able to resolve hostnames. We can also try using test-by-doing.

```
$ host vertex.otago.ac.nz
… (or anything else). Should work fine.
$ ssh user@vertex.otago.ac.nz
… (or to anywhere else). Should work fine.
```

On all other hosts and routers you will want to set the DNS nameserver to 192.168.0.3. On Vyatta, this can be done using **set system name-server 192.168.0.3**. On Linux guests, just put `nameserver 192.168.0.3` into `/etc/resolv.conf`.

Now all you need to do is advertise the default route via RIP. Because default routes catch a lot of traffic, and can be problematic with regard to routing loops, there are protection measures in place to ensure they are not accidentally advertised.

On Vyatta, we tell Vyatta that we want to originate default route information using the configuration command **set protocols rip default-information originate** (this is the sanity check), and advertise the default route using **set protocols rip network 0.0.0.0/0**. There are no special configuration elements you need to put into the receiving routers, just use the operational command **show ip route** to verify that the other routers now have a default route, learned via RIP.

9. [Optional] Configure RIPv2 Authentication

In this optional section you're going to authenticate your dynamic routing advertisements using RIPv2 MD5 authentication. This will help prevent attackers from subverting your routing infrastructure by easily advertising bogus information, which could be make it easy to launch attacks such as man-in-the-middle (MitM) and denial-of-service (DoS).

The original version of RIP didn't support authentication (or variable-length subnet masks, known as VLSM, which we also think of as class-less addressing), and so should never be used.

RIPv2 supports at least two forms of authentication. The first is a plain-text password, so is only useful for protecting against accidental misconfiguration, not deliberate attacks. You should instead use the MD5 form of authentication. To allow keys to be changed, a number of different passwords can be entered simultaneously, identified by an index. We're just going to use a single password, so our index will be 1 (the lowest index allowable).

RIPv2 Authentication is enabled on a per-interface basis, so on R1 we shall need to enable authentication on two interfaces: eth1.10 and eth1.20, as those are the interface which are connected to other RIP routers. The configuration command is **set interfaces `ethernet eth1 vif 10` ip rip authentication md5 1 password `OurSharedSecret`**. The shared secret has to be the same on each link, but does not need to be the same throughout the RIP routing domain. When you have made the configuration change to all the routers, ensure that it is all working. Use **run show ip rip status** to inspect the status of the RIP agents. Refer to the Vyatta RIP Reference for further commands.

10. [Optional] IPv6 and RIPng

Don't worry if you don't have much time to do this. The parts regarding subnetting and addressing will be covered as a class-exercise in the lab on Subnetting. Students wanting *to do this section should not find it difficult, but will benefit greatly from watching the 20-minute video on IPv6 Subnetting in the Lab Resources.*

RIP (either version 1 or 2) doesn't have support for IPv6. Instead, we have RIPng (RIP Next Generation)[5], which is an extension of RIPv2, although some features, such as authentication, are not supported in RIPng[6].

10.1. Design the Subnetting and Addresses

Let's begin by doing a little bit of addressing. This is starting to get into the up-and-coming subnetting lab, but this is rather easy (easier than IPv4 with public IP addresses).

If we want to subnet our network, we first need a network allocation. We'll create one using Unique-local addresses, and then subnet that. A Unique-local address can easily be generated using random numbers, but we can generate fewer clashes by basing it on something that is already reasonably globally unique, such as an Ethernet MAC address. Point your web browser to Generate Unique Local Address [http://www.kame.net/~suz/gen-ula.html] and put in the MAC address of any of your interfaces, such as any one of the interfaces on R1.

Write down the network allocation that has been generated; this is what we shall be subnetting. For this example, I'm using fd49:59ab:879f::/48; you should use the one you generated.

The allocation that you generate will contain 48 bits for the network ID. We typically want subnets no smaller than a /64, so we have 16 bits to use for subnetting. If we were to only allocate subnets of size /64, then we have up to 2^{16} subnets, and a routing entry for each!

If, however, we wanted to provide some more hierarchy in our addressing plan, and allow some of these divisions to be further subnetted, then we should use some intermediate size. For example, if we allocate subnets of size "/56", then that means we've used 56-48=8 bits for our subnet IDs. This means we can have 2^8=256 subnets of size "/56".

IPv4 conditioned us to like subnetting on 8-bit boundaries. This is because IPv4 addresses are dotted decimal, and each decimal was 8-bit integer (0—255). IPv6, on the other hand, uses hexadecimal notation, so we can naturally work easily with 4-bit boundaries.

In the example above, if 2^{16}=65,536 subnets is too many, and 2^8=256 is worryingly few, we could compromise and use subnets of size "/60", which would give us $2^{60-48}=2^{12}$=4096 subnets, and each subnet could have 4 bits of subnet ID remaining which could be used for further subnetting. This is good, because dealing with subnets smaller than a /64 is not good for hosts, as you can't use SLAAC in that case.

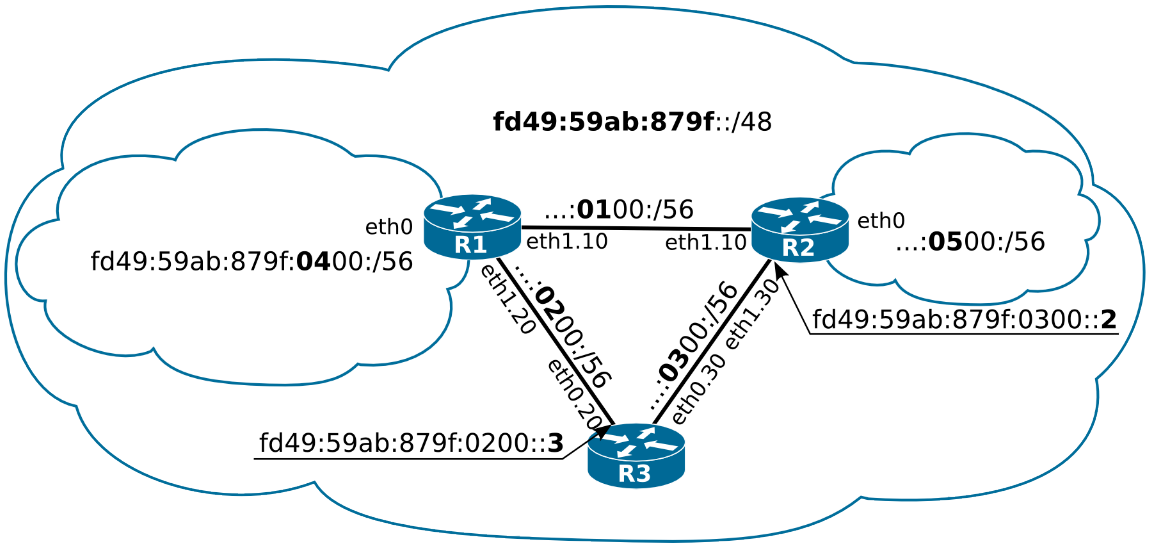
Since this is our first network, let's just assume for now that we want each subnet in our network to be a /56. Bear in mind, though, that we could have a mixture of different subnet sizes if we needed, but we can keep things simple for now.

So, looking at the network map, give each of the five subnets a number; because we've already got a nicely organised IPv4 network running in parallel, we shall save some confusion and use the same subnet IDs. Allocate a /56 to each subnet in your network. You might end up with something like the following. This is also shown on the map in Figure 9, "IPv6 Subnetting and Addressing".

[5]During early development, IPv6 was refered to as IPng, for Next Generation.
[6]Such functionality is supposed to be supplied by IPSec support in IPv6.

```
Network allocation:   fd49:59ab:879f::/48
   subnet 1:                       …:01│00::/56
   subnet 2:                       …:02│00::/56
   subnet 3:                       …:03│00::/56
   subnet 4:                       …:04│00::/56
   subnet 5:                       …:05│00::/56
                                       ↑ /56 boundary
```

Figure 9. IPv6 Subnetting and Addressing



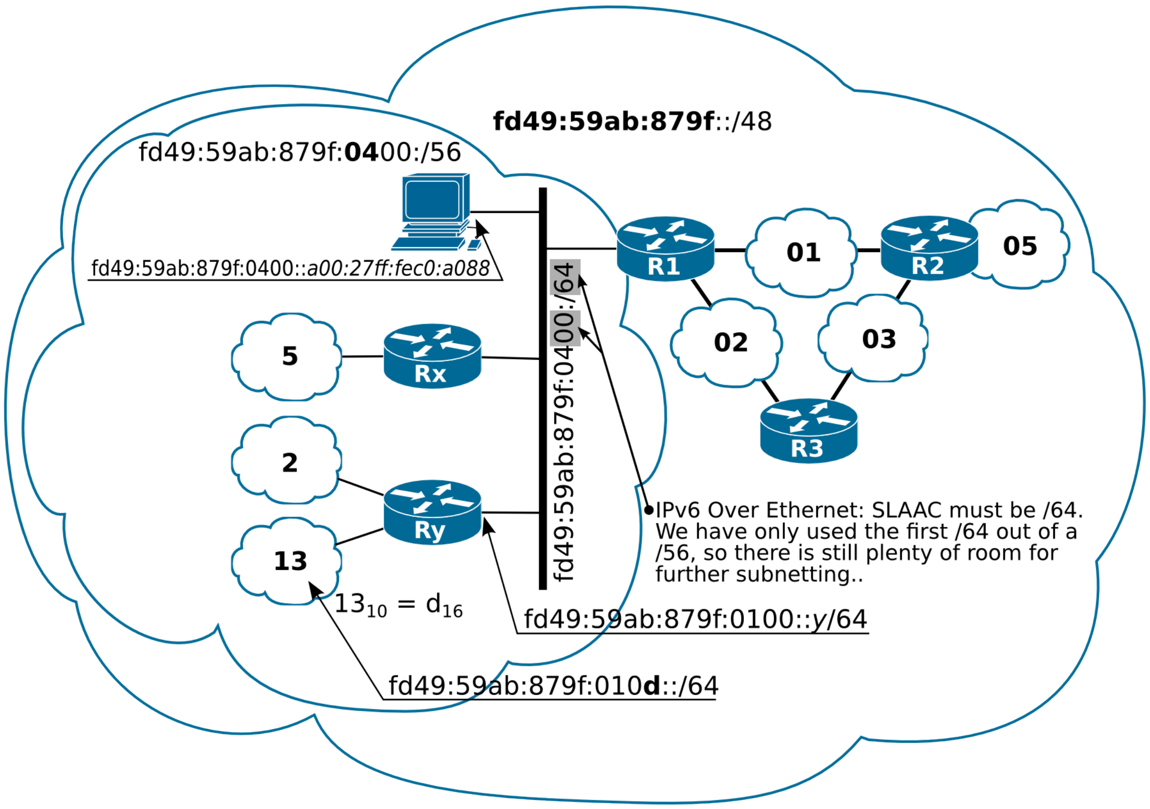
The subnetting and addressing plan for our network.

Note that we have shown the network identifier as ommitted in the various subnets, to try and aid your understanding of how the addressing scheme is structured. The full prefix of subnet 3, for example, is fd49:59ab:879f:0300::/56.

Please also note that we have shown the subnet ID as 0300, and *NOT* as 03, because then you might be tempted to write fd49:59ab:879f:03::/56, which is *NOT THE SAME*, as the latter is actually fd49:59ab:879f:0003::/56. Only the first two digits (8 bits) of 0300, for example, are being used here as the subnet ID, the remaining bits up to the 64th bit would typically be zero and available for further subnetting.

But we're not quite done yet. We want to use SLAAC to advertise a prefix onto our ethernet segments. According to RFC 2464 — "Transmission of IPv6 Packets over Ethernet Networks" — the prefix length must be exactly 64 bits. So in our case we've given ourselves a prefix of /56, but we only offer a smaller /64 out of that. What happens with the left-over bits? They remain available for further subnetting, should we desire to subnet one of the networks further. Figure 10, "IPv6 Subnetting Allowing for Growth" shows one possible way in which the network might grow. Note that by reserving some bits for subnetting, we allow for more hierarchical addressing which leads to more efficient routing: the number of routes that the other routers need to know about remain the same bacause they can easily be summarised.

Figure 10. IPv6 Subnetting Allowing for Growth


fd49:59ab:879f::/48
fd49:59ab:879f:0400:/56
fd49:59ab:879f:0400::a00:27ff:fec0:a088
fd49:59ab:879f:0400::/64
R1
01
R2
05
02
03
R3
5
Rx
2
Ry
13
$13_{10} = d_{16}$
IPv6 Over Ethernet: SLAAC must be /64. We have only used the first /64 out of a /56, so there is still plenty of room for further subnetting..
fd49:59ab:879f:0100::y/64
fd49:59ab:879f:010d::/64

The subnetting of our IPv6 network, showing why it is useful to reserve some bits for subnetting.

Routers don't use address autoconfiguration, so each will need a static address. We'll be simple and just use a host ID of 1, 2 and 3 for routers R1, R2 and R3. There is no requirement to do it quite like this, it's just to make addresses easier to recognise. Figure 9, "IPv6 Subnetting and Addressing" shows the addressing structure of our network, as we have discussed it. Write down the address for each interface on R1, R2 and R3. Note that clients C1 and C2 still use address autoconfiguration.

```
R1  eth0     fd49:59ab:879f:0400::1
R1  eth1.10  fd49:59ab:879f:0100::1
R1  eth1.20  fd49:59ab:879f:0200::1
R2  eth0     fd49:59ab:879f:0500::2
R2  eth1.10  fd49:59ab:879f:0100::2
R2  eth1.30  fd49:59ab:879f:0300::2
R3  eth0.20  fd49:59ab:879f:0200::3
R3  eth0.30  fd49:59ab:879f:0300::3
```

10.2. Configure Interfaces

Now that we have designed the addressing and have assigned addresses for each interface, let's now affect those assignments. You should consult the Vyatta IPv6 documentation for the full details. Here is an example of configuring just the eth1 VLAN 10 interface on R1:

```
configure
set interfaces ethernet eth1 vif 10 address fd49:59ab:879f:0100::1/56
commit
```

```
run show interfaces
```

Use **ping6 `address`** (if you’re still in configuration mode, use **run ping6 `address`**) and test that you can contact all of your neighbours.

Use **show ipv6 route** to view the IPv6 routing tables.

You’ll also want to enable router advertisements on at least R1’s eth0 interface, and similarly on R2:

```
set interfaces ethernet eth0 ipv6 router-advert prefix fd49:59ab:879f:0400::/64
commit
```

Use **ifconfig** or similar on the clients to ensure they have an address.

10.3. Configuring RIPng

Configuring RIPng is much like the IPv4 equivalent in Vyatta. The first thing to do is ensure that IPv6 forwarding is turned on, or perhaps it would be better to ensure it hasn’t been disabled:

```
delete system ipv6 disable-forwarding  Not needed now, was in earlier versions of Vyatta
commit
```

Enable RIPng on the interfaces that need to participate in RIPng. In our case, this is at least those interfaces towards the inside of our network. Using R1 as an example:

```
set protocols ripng interface eth1.10
set protocols ripng interface eth1.20
commit
```

Advertise (redistribute) connected networks:

```
set protocols ripng redistribute connected
commit
```

Verify that it is working:

```
run show ipv6 ripng status
run show ipv6 route
```

Repeat the above on all other routers, then test reachability using **ping6** and/or **traceroute6**.