Interface Management

COSC301 Laboratory Manual

Required Reading Prior to Lab

To ensure you get plenty of time to ask for any help during the lab, please ensure you have read at least Section 1, “A Map, Notation and a bit of Theory” before coming to the lab, as this includes important revision material and does not require you be in the lab.

If you have already done so, congratulations! You can directly jump to Section 2, “Prepare Client1”.

Today is a fairly exciting lab; you get to create a network topology of (virtual) hosts and (virtual) Ethernet switches, which is a great first step to network management. In order to understand what it is that we’re creating, we will need to indulge in a little theory (don’t worry, its quite practical).

Then we’ll be adding another virtual machine, called Client2, to our network; add Client1 and Client2 into a separate network of their own, give them some addresses and ensure they can communicate with each other.

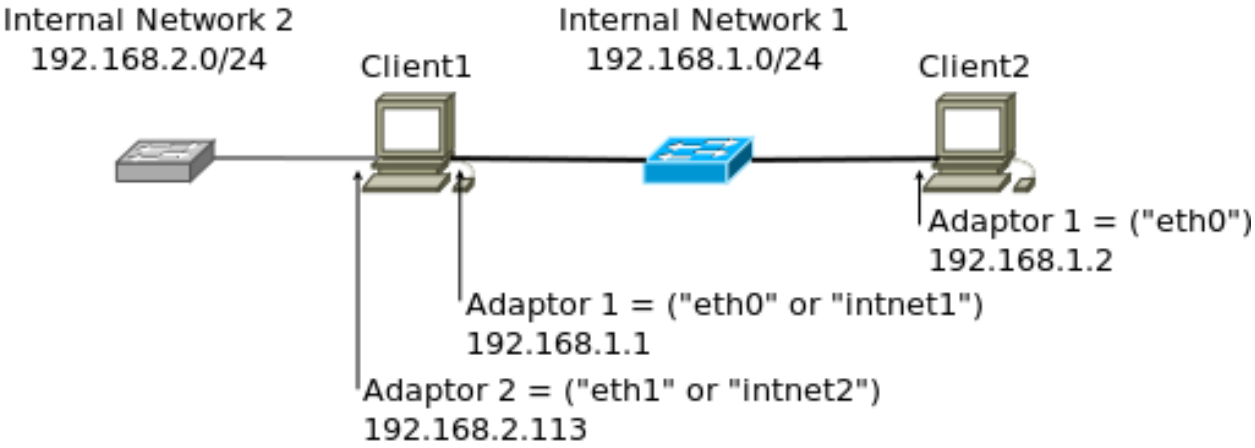
Naturally, there is a lot of ways in which network can go wrong, so we need some diagnostic tools to help us figure out what’s actually happening. We’ll look at some management and diagnostic tools that will be useful here.

Since prevention is better than cure, we’ll also show you how you can usefully name your interfaces which will help prevent mistakes later on.

1. A Map, Notation and a bit of Theory

Figure 1, “Topology for Interface Management Lab” shows the layout of the machines and switches in todays virtual environment. The smaller boxes connecting the PCs shown in the diagram are Ethernet switches, and will be managed automatically by VirtualBox. The switch to the left of Client1 we will ignore until much later in the lab. Client1 will have two interfaces, and will be connected to two separate networks, which we shall simply call “COSC301 Internal Network 1” and “COSC301 Internal Network 2”. VirtualBox calls these virtual Ethernet “Network Interface Cards” (“NICs”) as an “Adaptor”. By default each VirtualBox virtual machine will have one adaptor, so we will have to enable a second adaptor.

Figure 1. Topology for Interface Management Lab



The topology of the virtual network we shall be creating today.

Inside each virtual machine (aka. “VM” or just “guest”), the operating system – Ubuntu Linux in our case – will give each interface a name. By default, Linux will give all of its Ethernet interfaces names such as `ethN`, where `N` starts at 0. In recent versions of Ubuntu, there has been a move to a predictable naming scheme. Traditionally the `ethN` style names have been subject to race conditions. The new names are based around how the adapters appear on the internal busses.

> **Don't Panic**
> If you see `enp0s3` as the name of your interface. The instructions below will sort it out.

Other OSes have different naming conventions, although they don’t really tell you a lot, such as what it is that they are connecting to: `intnet1` and `intnet2` might be better names, and we’ll rename those interfaces later.

We can recognise that Client1 is different from Client2 in that it attached to multiple networks at one time. You could imagine Client1 as being a notebook computer with one interface being a wired Ethernet port, and another being on a WiFi Wireless Ethernet network. We’ll ignore its second interface until later in this lab.

Each network, of which we have two, must have a different “network address” (we could also just as easily say “subnet address”, as the two terms are identical for our purposes currently). Thus, everything in “COSC301 Internal Network 1” has an IP address starting with 192.168.1.x, while everything in “COSC301 Internal Network 2” has an IP address starting with 192.168.2.x. This is important, because Client1 must be able to determine (via its “routing table”) which interface it should send traffic out of in order to reach a particular IP address.

Inside each network, each interface must have a different address, or else we wouldn’t be able to uniquely specify a particular host. Notice we said “interface”, and not “host”. An IP address is assigned to an interface, not a host; Client1 has two interfaces, and thus two addresses, each on a different network.

Client1’s interface on “COSC301 Internal Network 1”, Adaptor 1 has an IP address of 192.168.1.1, while Client2’s interface on the same network has an IP address of 192.168.1.2. Assume that Client1 wants to send a packet to 192.168.1.2: how does it determine which interface to send out of?

To answer that, note that each network has a network address such as `192.168.1.0/24`. What does that `/24` mean? It is the "prefix-length" and specifies the number of bits from the start ("left") of the address where we stop talking about the network (the "network" or "subnet" ID) and start talking about the host ID. Recall an IPv4 address has 32 bits, and each number in a "dotted-quad", `a.b.c.d` takes up eight of those bits:

```
      192    .    168    .     1     .     0     / 24       Network address
   1111 1111   1111 1111   1111 1111   0000 0000           Binary bitmask
      255    .    255    .    255    .     0               Netmask
             8           16          24                    Number of bits from start
```

Let's explain what this shows:–

Line 1
: This is a network address and prefix-length. The prefix-length is specified here in "CIDR" (pronounced "cider") notation, which is a convenient shorthand for the older "netmask" (aka. "network mask" or "subnet mask") on the third line.

Line 2
: Shows the 24 (from `/24`) binary bits from the start of the 32-bit network address, nicely spaced out.

  The lower this number, the fewer networks we can represent, but each network can have more hosts inside it. So, if every network on the Internet were to have a /8, we could only have 255 networks on the Internet, but each network could have 2^{32-8} or over 16 million hosts[1]. Alternatively, if every network on the Internet had a /24, you could have over 16 million networks, but each network could have less than 256 hosts.

  In practice, we have a mixture of different network sizes in use, and larger networks are "subnetted" into smaller networks, but we don't need to concern ourselves with that for now.

  Note that you should only ever have a string of ones at the beginning, then a string of zeros until the end.[2]

  What this shows is that the entire first three octets specify the network ID portion of the address.

Line 3
: Shows the "dotted quad" (or "dotted decimal") notation, where every eight bits ("octet", or "byte") is converted into decimal. For our purposes today, we only need to recognise that binary "1111 1111" makes decimal 255, and binary "0000 0000" makes decimal 0.

Line 4
: Notice that each number is at a location of a dot. Since every decimal number on the first line is specified in eight bits (hence, a maximum of 255, or 2^5-1) we can easily recognise /8 /16 and /24. So, for example, suppose we specify 10.0.0.0/8, then everything matching `10.x.y.z` would be in that network..

  We can actually specify "sub-octet" prefix-lengths, but these are harder to work with and we ignore them for now until much later in the paper.

[1] Although we would have to subtract two reserved addresses, but that's not important right now.
[2] Unless you're working with Cisco "wildcards", in which case it is inverted, so a string of zeros followed by a string of ones.

So, you should be able to see, given the map above, that 192.168.1.2 is in the same network as 192.168.1.1, and in a different network to 192.168.2.113. If you have trouble understanding that, please ask a demonstrator.

One last, tiny little thing before we go on. Later on you will see that every machine has a “loopback” interface, often called “lo” or “lo0”, having an IP address of 127.0.0.1. Please note that every machine has one of these, and so anything sent to 127.0.0.1[3] gets sent to the local machine, which we call “localhost”.

2. Prepare Client1

Before we start the practical work for this lab, we’ll pause and first check that everything is as we expect it:–

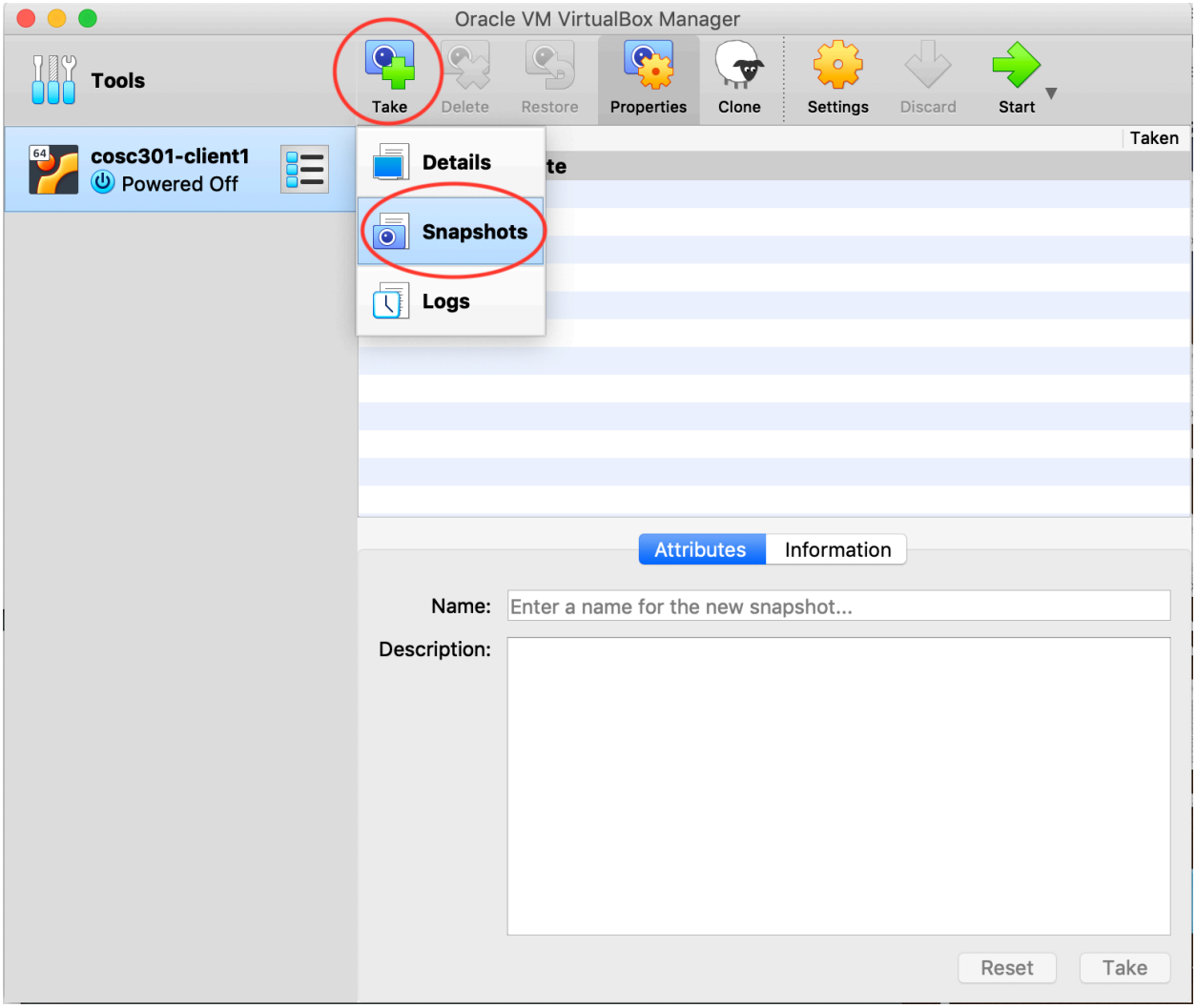
• Ensure Client1 is shutdown; its status should be shown as “Powered off” in VirtualBox.

Now we’re going to make it easy for ourselves to undo all the changes that we shall be making in Client1 today. To do this, we shall be using a feature of VirtualBox called “snapshots”, which is not related to screenshots, but simply provides a way to “roll-back” your changes inside a virtual machine to a particular point in time: ie, the beginning of this lab.

Snapshots are useful, but they come at a significant cost for us in terms of performance, so we won’t make much use of them. There are a number of ways to take a snapshot, and they can also be taken while a virtual machine is running. Since our virtual machine is not running, in the main VirtualBox window, click on the “COSC301-client1” virtual machine, then click on the machine's extension menu icon and you will find the Snapshots menu as shown in Figure 2, “VirtualBox Snapshot”. Then you will find the Take icon at the top of the window, which has a little icon of a camera, as shown in Figure 2, “VirtualBox Snapshot”. Click on the Take icon, you will be asked to give a name to the snapshot. Give the snapshot a suitable name, such as “Prior to Interface Management Lab”. Figure 3, “VirtualBox after Snapshot at Beginning of Lab” shows the GUI interface after taking the snapshot.

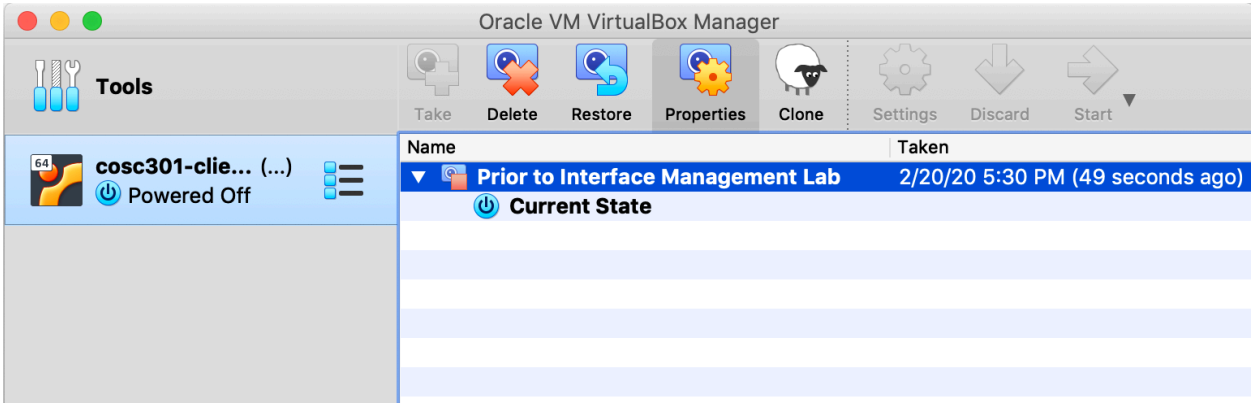
[3] Actually, anything in 127.0.0.0/8.

Figure 2. VirtualBox Snapshot



The VirtualBox interface for taking a snapshot.

Figure 3. VirtualBox after Snapshot at Beginning of Lab



The VirtualBox interface after having taken a snapshot recording the state of the machine before making the changes for this lab.

Exercise

Inside VirtualBox's main window, as an exercise click on the "COSC301-client1" machine and then click on Settings. Click on the Network icon. On Adaptor 1, set Attached to: Internal Network with a Name of "COSC301 Internal Network 1" (make sure to type the name correctly which is case sensitive). Do not click OK yet. Click on the Advanced label to show the advanced settings. Record the Ethernet hardware address ("MAC") by taking a screenshot. You'll want to refer to it later.

Exercise

Another exercise is, in Adaptor 2, tick the box to Enable Network Adaptor. Just as you did with Adaptor 1, attach it to the internal network "COSC301 Internal Network 2". Take another screenshot showing the Ethernet hardware address of this interface as well. Click OK when complete. You have now completed the "hardware" changes for Client1.

Start Client1 and proceed to the next section.

3. Create Client2

In this section, we're going to create the virtual machine Client2. It will run simply from an Ubuntu Live CD, and will take very little time to complete. Because it runs from a Live CD, and has no useful permanent storage, it will only be used a couple of times in the course for temporary machines.

In the VirtualBox main window, click on the New button, which will launch the Create New Virtual Machine wizard. Give the machine a name of "COSC301-client2", choose "Linux" as the type of the machine and "Ubuntu (64-bit)" as the version, then click Continue. Since it is Ubuntu 18.04 LTS "Bionic" desktop amd64, it needs at least 4096MB of memory. *Do not* create a virtual hard-disk: choose Do not add a virtual hard disk. Because this is little unusual, VirtualBox will prompt you if this is what you really want to do: answer Continue to complete the wizard.

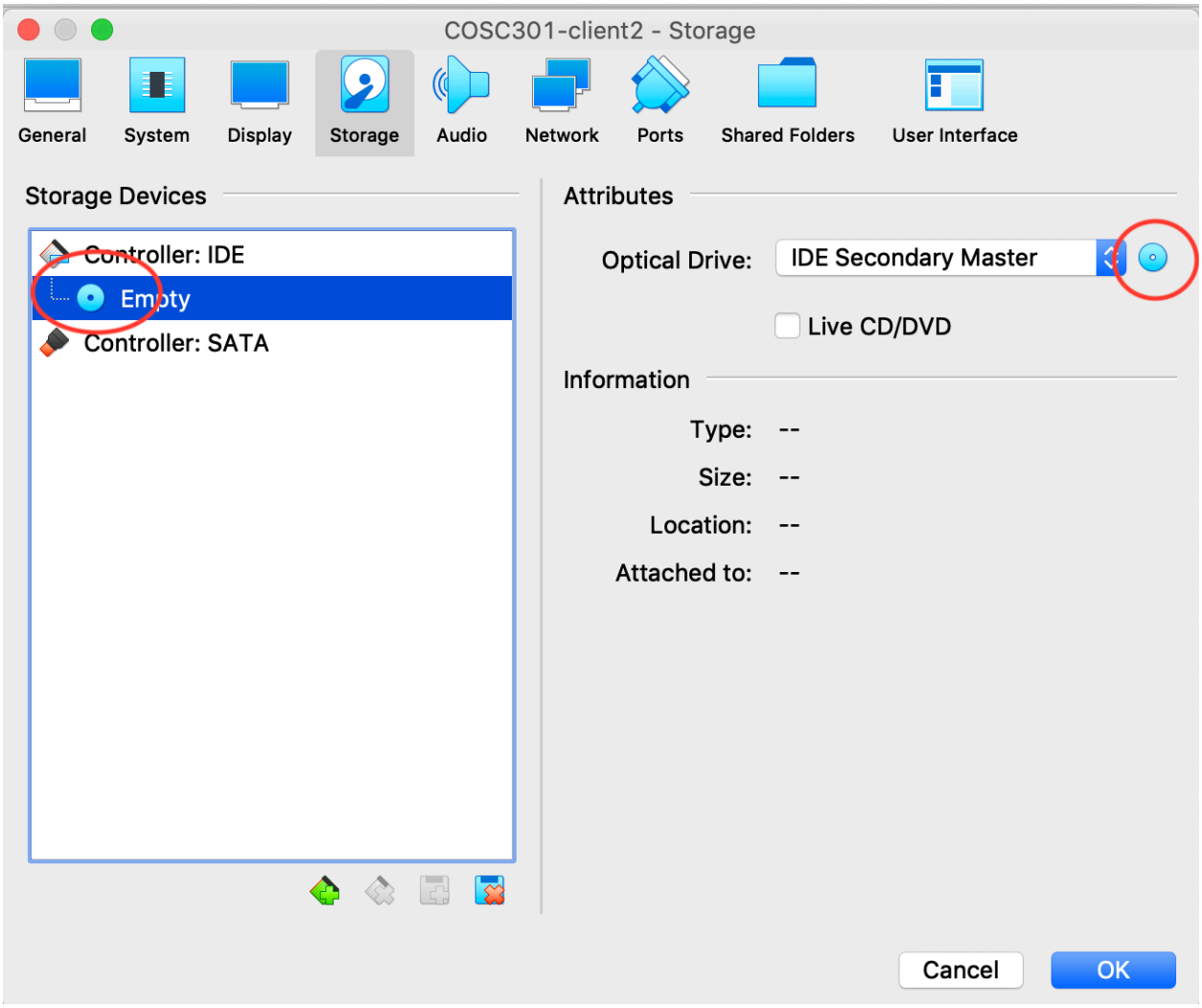
Now choose a proper graphics controller. Click on the “COSC301-client2” VM and then click on Settings. Click on the Display tab. Choose VBoxVGA as the graphics controller. You may find out an "Invalid settings detected" warning, which is ok.

Now set up network adaptors. Click on the “COSC301-client2” VM and then click on Settings. Click on the Network tab. For Adaptor 1, make it attached to Internal Network with the network name “COSC301 Internal Network 1”. Record the Ethernet hardware address of the adaptor as done previously. Then click on Adaptor 2, enable the adaptor and make it attached to NAT. The reason for us to have the second adaptor is to allow Client2 to have Internet access, which is essential for Client2 to have essential software packages installed later.

Now you need to virtually “insert” the Ubuntu CD that the machine will boot from. Click on the “COSC301-client2” VM and then click on Settings. Click on the Storage tab, and then click on the empty CD-ROM icon. Figure 4, “Client2’s Storage Configuration in VirtualBox” shows how the interface should appear at this point. Then click on the small folder icon on the right beside Empty to access the Virtual Media Manager. If the disc called `ubuntu-18.04.4-desktop-amd64.iso` is not present in the list, click on the Choose a disk file menu and find it from the `ISOs/Ubuntu` folder in the “resources” folder, then Select it.

Tick the Live CD/DVD box to keep the ISO image in the virtual optical drive everytime the machine is shutdown.

Figure 4. Client2's Storage Configuration in VirtualBox



The storage settings of VirtualBox for Client2. We want to associate an CD-ROM image "ISO" to insert it into the virtual CD-ROM drive of the virtual machine.

Now Start "cosc301-client2". When asked if you want to either try Ubuntu or install Ubuntu, just click on Try Ubuntu, because you don't have a disk to install into.

Note

The screen of Client2 will be smaller than Client1, because Client1 has the Guest Additions installed, which allows it to have a larger virtual display adaptor, among other enhancements.

After "cosc301-client2" started, find a Terminal window, type the following command to install the net-tools package.

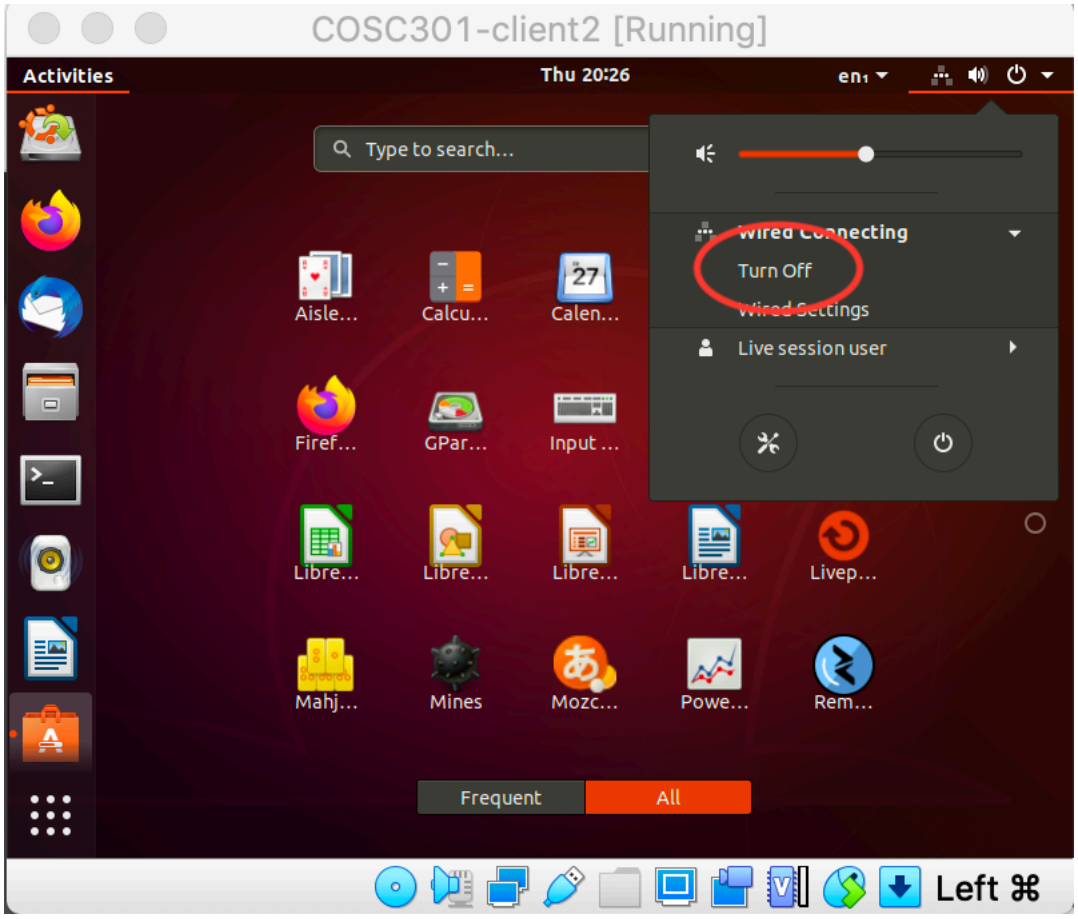
```
$ sudo apt install net-tools
```

4. Affect a Temporary Configuration and Test

In this section, we're going to configure the interfaces on Client1 and Client2 that are connected to the network "COSC301 Internal Network 1" using first principles. These changes are temporary in nature; a reboot would remove any changes. In the next section, we look at how we can make these changes permanent. Once we have configured both interfaces, we will verify IP-level connectivity between them by "pinging" one machine from the other. Then we shall briefly look at other common tools that are useful for learning more about the current state of the host and network.

First though, we need to prevent Ubuntu from automatically managing our interfaces for us, because then it would override the temporary changes we're wanting to make. This is a common feature found on most modern systems. Ubuntu, and other Gnome-based desktop environments, perform this task using something called NetworkManager. It is designed to automatically configure network interfaces when, for example, a cable gets plugged in or a wireless network is joined, etc. Because we want to statically configure the interfaces – without using the typical permanent configuration facilities – we need to turn it off to prevent it from getting in our way. Figure 5, "Disabling Network Manager" shows where you can find the NetworkManager icon, and what you see when you click on it.

Figure 5. Disabling Network Manager



Network Manager by default can be found in the top panel in a Ubuntu desktop environment. This figure shows how to disable Network Manager.

On Client1, disable Network Manager for both adaptors.

On Client1, open a terminal window (if you don't have a shortcut already on your top panel, find it from Applications or search Terminal from Applications) and type the **ifconfig** "interface configuration" command:

```
$ /sbin/ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  eth0 is UP but no IP address configured…
        ether 08:00:27:b6:3d:eb  txqueuelen 1000  (Ethernet)
        RX packets 24  bytes 11979 (11.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 30  bytes 8075 (8.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: …    Ignore for now. Only on Client1
…
…

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0         or 127.0.0.1/8 in CIDR notation
        inet6 ::1  prefixlen 128  scopeid 0x10<host>   IPv6: ignore for now…
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4  bytes 356 (356.0 B)
```

```
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 356 (356.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Like many administrative commands, **ifconfig** lives in the directory `/sbin` ("system binaries"), and not `/bin` ("user binaries"), and because of this, users generally won't be able to find it if they just type **ifconfig** as `/sbin` may not be in their `PATH`. To find out if `/sbin` is in your `PATH`, use the following command.

```
$ echo $PATH
```

If `PATH` includes `/sbin`, then you can simply use **ifconfig** without adding `/sbin`[4]. Also, note that even though **ifconfig** is an administrative command, it is not being used to make any changes or access restricted information at the moment, so we don't need root privileges.

What is that `lo` interface? Hopefully, you should be able to notice that the `lo` interface has an IPv4 address of 127.0.0.1, which you should recognise as being the loopback address which every machine will have, so this is the "loopback" interface (so called because if we send to it, we end up talking to the same host, which is still quite useful but we'll see it more later on).

By default, **ifconfig** only shows those interfaces that have the `UP` flag set. There are other flags as well, such as `RUNNING`. `UP` basically means the interface is configured, and `RUNNING` basically means that interface has layer-2 ("Datalink layer") connectivity. So if you saw an interface that was `UP` but not `RUNNING`, that might indicate that the cable was unplugged somewhere.[5] For interfaces like eth0, it is UP and RUNNING, but has no IP address configured. This is because we have turned off the network manager.

We can ask **ifconfig** to show all interfaces including those that are not UP (if any) by using the option `-a`:

```
$ /sbin/ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  eth0 is UP but no IP address configured…
        ether 08:00:27:b6:3d:eb  txqueuelen 1000  (Ethernet)
        RX packets 24  bytes 11979 (11.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 30  bytes 8075 (8.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: …    Ignore for now. Only on Client1
…
…

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0         or 127.0.0.1/8 in CIDR notation
        inet6 ::1  prefixlen 128  scopeid 0x10<host>   IPv6: ignore for now…
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4  bytes 356 (356.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 356 (356.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The output could be the same as before. Notice that your Ethernet hardware address ("MAC address") will be different to that shown in the above listing. Look at the screenshot you took when configuring the VirtualBox network settings, and just ensure that the MAC address of eth0 is the same as Adaptor 1.

[4] We will learn more about this in the scripting lab.
[5] In Cisco-parlance, we would instead say "up and up". Knowledge that the data-link layer is working in particularly useful in serial interface such as WAN network links.

Repeat the above procedure on Client2, but no need to disable the Network Manager. You should find two interfaces named "enp0s3" and "enp0s8". Try to find out which one is for Adaptor 1 and which one for Adaptor 2.

Now we shall give Client1's eth0 and Client2's Adaptor 1 (maybe named "enp0s3") IPv4 addresses, and ensure they can talk to each other. This time, we shall be using **ifconfig** to make administrative changes to the system, so we shall require administrative powers (ie. use **sudo** as we practiced in the previous lab). Here is the information, taken from the topology map at the beginning of this lab, but shown in tabular form:

	Client1's eth0	**Client2's Adaptor 1**
Address	192.168.1.1	192.168.1.2
Netmask	/24 = 255.255.255.0	/24 = 255.255.255.0
Network[a]	192.168.1.0	192.168.1.0
Broadcast[b]	192.168.1.255	192.168.1.255

[a]Automatically determined from Address and Netmask
[b]Automatically determined from Address and Netmask

Notice that the Address of each is different, but the Network and Broadcast addresses are identical, which means they are in the same network. The Network and Broadcast addresses are, by default, determined automatically from the Address and Netmask. In the case of the Network address, all of the host bits are 0 (ie. the very first possible host address), while in the Broadcast address, all the host bits are 1 (ie. the very last possible host address). You don't need to worry about the Network and Broadcast addresses for the present time; the default behaviour is desirable and correct.

Okay, so here's how we can configure Client1:

```
# /sbin/ifconfig eth0 192.168.1.1 netmask 255.255.255.0 up
```

Verify what we have done by querying the interface configuration, much as we did before. I've highlighted the parts that you should check.

```
$ /sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:99:c2:7d
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe99:c27d/64 Scope:Link
          UP BROADCAST  RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1710 (1.7 KB)  TX bytes:5693 (5.6 KB)
```

Exercise

As an exercise repeat on Client2's Adaptor 1 using the very same procedure, but with a different address and interface name. Make sure the **ifconfig** output after configuration is expected.

Go back to Client1, and use the **ping** command to see if you can reach Client2. This sends a message called an "ICMP Echo Request" ("ping") to a target host. Upon receiving the request, the target would reply by sending back an "ICMP Echo Response" ("pong"). The requestor would then look at the time difference and print out the "round-trip time", which is the time taken to send the message and get a response.

Tip

A lot of networks will prevent pings into their networks from outside, so if you try to ping well-known sites, such as Microsoft, you'll probably find you do not get a reply. Thus, not getting a response doesn't necessarily mean the target host is not up, as it could be filtered somewhere.

Let's ping Client2 from Client1. We could also ping Client1 from Client2, but generally that is not needed, because if we get a response we know that traffic can flow in both directions.[6] The default behaviour of **ping** on Unix-like systems is to continue pinging, once per second, until stopped by the user typing **Ctrl+C**, which is generally shown as **^C**:

```
$ ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=5.96 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.418 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.324 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.314 ms
^C
--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.314/1.755/5.966/2.431 ms
```

Exercise

As an important exercise make sure you see something like the above, which means the configuration of both Client1 and Client2 worked. This will be an important test for your first assignment.

Didn't work?

If you instead saw **ping** wait for a while and then output "Destination Host Unreachable", that would indicate that Client2 is either mis-configured with an IPv4 address on its eth0 interface, or you configured its VirtualBox settings wrongly, perhaps connecting it to the wrong network.

You've just configured your first network. For a real beginner, working on their own on their first network, the hardest thing to understand is what you should put as the IP address.

Running late?

The rest of this section is optional. If you run out of time, you may skip over to Section 5, "Affect a Permanent Configuration".

Have a look at those round-trip times reported by **ping**, the first is rather larger than the subsequent pings; if instead you don't see this, don't panic, just read on. In order to find out why we have a longer initial ping, we need to remind ourselves about how the "ARP cache" fits in to packet delivery.

In order to for Client1 to send a packet to Client2 on the local network, it needs to be able to address Client2's Ethernet interface using the appropriate MAC address. However, Client1

[6] At least, this is true at the IP layer, but as we will see much later, devices such as Firewall and NAT boundaries can cause problems here, so pinging Client1 from Client2 as well could be useful when crossing such boundaries.

wouldn't know that MAC address corresponding to Client2's IP address, so it uses the Address Resolution Protocol (ARP) to find out. ARP basically asks everyone on the network "Yo! Whoever has IP address 192.168.1.2, please respond to me at MAC address `Client1's eth0 MAC address`", and, assuming a station with IP address 192.168.1.2 exists on that network, it will respond with "I have IP address 192.168.1.2, and my MAC address is `Client2's enp0s3 MAC address`". Client1 then remembers this in its ARP cache for a period of time; a common time-to-live is five minutes for most client systems, after which it will be expired from the cache.

So we can see that the first, more lengthy, ping is because it's had to do some ARP lookup before it could actually deliver the ping request. If you were to repeat the **ping** command before the ARP entry expires, you would find you don't get that first initial delay.

Tip

There can be other reasons for an initial delay, such as the system having to "page in" some of the operating system from memory if it hasn't been used recently. Commonly there will be some combination of reasons that contribute to round-trip-time fluctuations, including those relating to the destination host and the intervening network.

We can inspect the ARP cache using the **arp** command. We suggest that you run it with the `-n` "numeric" option, which prints out IP addresses instead of trying to find names for them, which makes the output easier to recognise:

```
$ /usr/sbin/arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.2              ether   08:00:27:93:32:8b   C                     eth0
```

If instead you see no output, or no entry for 192.168.1.2, it probably indicates that the entry expired before you ran the **arp** command. Run the **ping** command again and try again.

So far, we've seen enough to learn about the deliveries to hosts on the local network ("local deliveries"). But in real-life, there are many different networks that we might wish to communicate with. In fact, that's the very reason IP was created. So how does it know whether something is a local delivery or whether it has to first be sent off to some other "router" to be sent towards its destination. That's where the "routing table" comes in, which you should have some rememberance about from your previous studies. Here is how we can inspect the routing table on Client1:

```
$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

As you can see, that's a very short routing table, as Client1 only has one network it is currently connected to (Client1's eth1 is not yet UP, but we're ignoring that for now anyway). Most real systems with a connection to the Internet, or any other wider network, will also have at least a default route (aka. "route of last resort"), which basically says "if no other route matches, send it towards the router with IP address X".

Now, we're not going to be playing with routing until much later in the paper, but we would at least like to show you a couple of things while we're talking about basic network interface management: how to add a (default) route, and to determine how something would get routed through a network, typically in order to find out where something is broken.

We don't actually have anything operating as a router in this network, but we can simulate a misconfiguration, which is perhaps more interesting. We're going to *pretend* that there *should* be a router at the address 192.168.1.254, but that the router is currently present on the network. We shall only do this activity for Client1.

On Client1, run the following commands to first inspect the routing table before you make the change, then add a default route to its routing table, then inspect the routing table to verify the change:

```
$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
$ route add default gw 192.168.1.254
SIOCADDRT: Operation not permitted   Whoops, forgot to use sudo
# route add default gw 192.168.1.254
No output, no problem!
$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
0.0.0.0         192.168.1.254   0.0.0.0         UG    0      0        0 eth0
```

Notice that the **gw** part of the command specifies the "gateway" (meaning "nearest router" in this case) that Client1 is going to use to try and get to anywhere else. The **default** is shorthand for **-net 0.0.0.0 netmask 0.0.0.0** and the G in the Flags field indicates that this route sends via a gateway, as opposed to a local delivery.

Since we've spent a bit of effort breaking the network, let's spend a little bit of effort to recognise how we've broken it, and then fix it. Imagine you're currently drinking your morning coffee, of whatever, you get a call from a user saying that they can't get to anywhere on the Internet. Pretend for now you don't know it's because the router is down.

At Client1, you do a few little tests: you could start on the inside (eg. "Does this machine have a valid IP address?") and work outwards toward the Internet (eg. "Can this machine get to sites on the Internet?"), but it can be faster to start outwards, often because you might recognise various error messages more easily than a user, and be able to come to a resolution faster (eg. "Oh, its a DNS issue", or "Hmmm, what about other sites?"). In this case, let's say you start by pinging a machine you know the IP address of, one that happens to be outside your network:

```
$ ping -c2 192.168.12.34
PING 192.168.12.34 (192.168.12.34) 56(84) bytes of data.
From 192.168.1.1 icmp_seq=1 Destination Host Unreachable
From 192.168.1.1 icmp_seq=2 Destination Host Unreachable

--- 192.168.12.34 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, …
```

If you want to delete an entry in the routing table or if you want to add the entry back later, use the following commands.

```
$ sudo route del -net 192.168.1.0 gw 0.0.0.0 netmask 255.255.255.0
$ sudo route add -net 192.168.1.0 netmask 255.255.255.0 dev eth0
```

Network is unreachable

If you instead get a message about "Network is unreachable", that is a standard networking error that indicates that there is no route to the network, and you

> therefore haven't added the default route as instructed above, or you missed some entries in the routing table that you should have added.

Notice it says "Destination Host Unreachable". This is a standard networking error that indicates an ARP lookup failed. "Aha!" you say, so somewhere in the path between here and 192.168.12.34 (which we're imagining is on a wider network for now), "there is a host down somewhere, but where?". This is when we use **traceroute**, to find out where in the path the packets are being "dropped".

> **Note**
>
> You may need to install **traceroute** with **apt** before you can use these commands.

```
$ traceroute -n 192.168.12.34
It will hang for about ten seconds…
traceroute to 192.168.12.34 (192.168.12.34), 30 hops max, 60 byte packets
 1  192.168.1.1  3004.111 ms !H  3004.116 ms !H  3004.116 ms !H
```

Take a screenshot. What are we seeing here? Before we explain it, let's see an example in a larger network: a traceroute to google.com from my office workstation, which is running Mac OS X:

```
$ traceroute -n google.com
traceroute to google.com (66.102.11.104), 64 hops max, 52 byte packets
 1  139.80.96.1  0.736 ms  0.474 ms  0.424 ms
 2  139.80.244.2  0.812 ms  0.757 ms  0.721 ms
 3  210.7.32.2  9.676 ms  9.802 ms  9.621 ms
 4  210.7.36.227  20.069 ms  20.113 ms  20.047 ms
 5  210.7.36.182  56.378 ms  44.905 ms  44.851 ms
 6  202.167.228.73  44.910 ms  45.060 ms  44.893 ms
 7  66.249.95.232  45.587 ms  45.525 ms  45.614 ms
 8  64.233.174.242  52.161 ms  53.708 ms  54.021 ms
 9  66.102.11.104  45.751 ms  45.758 ms  45.716 ms
```

In this example, which you probably won't be able to reproduce on your own student lab machines, we successfully reach google.com. Note that it is showing you the time taken to get to each hop from the query source: you can often see transitions between the local network to remote networks, or in the case of New Zealand, when it leaves the country on one of the long-haul links, although in this case the round-trip times would indicate the traffic is not leaving New Zealand (if it were, we would expect the trip-times to be on the order of 200ms). If you want other, more interesting examples, when you're at home, try google.com.cn, or any website if you have trouble getting to it.

But back to our earlier **traceroute -n 192.168.12.34** example, where we saw `!H` being output instead of a time. Have a look at traceroute(1) to find out what that means.

Okay, enough of that. Let's fix it by removing the route (in the real-world, if the router was supposed to exist, we would fix the router instead).

```
# route del default
```

5. Affect a Permanent Configuration

In this section, Client2 should left as it was at the end of the previous section. However, Client1 should be rebooted to ensure everything should still be at its defaults to minimise the

chance of errors. This is the section where we will be using the second Ethernet adaptor of Client1, which we added at the start of the lab.

We're first going to rename the interface on Client1, so instead of having eth0 and eth1, we will now have intnet1 and intnet2 respectively.

Historically, there have been a number of different ways that this has been done under Linux, so it pays to "know your system". Linux systems today have converged on a subsystem called "systemd", which is used for dealing with adding and configuring hardware, particularly for removable devices such as USB, but also for network interfaces and other things, which can all come and go through the course of the system. One of its key responsibilities is determining how to make the device available (ie. give it a well-known name). In the case of network interfaces, "systemd" can make this a naming decision most easily based on the Ethernet hardware address.

The `/etc/systemd/network/70-intnet.link` is used to rename interfaces. It uses the MAC address of the interface cards to do so[7]. If you want to replace any network interfaces on Linux systems, it is useful to know about this, otherwise you may find that the replaced card gets a different name than the one you were expecting, which can be annoying.

We have created an example file for your systems. Here is what you would find in that file, you may need to change the MAC address to suit your imported machine.

```
# If, when we add a device, we see the following MAC address
[Match]
MACAddress=08:00:27:7c:ea:6e

# set the name to 'eth0' from the predictable enp0s3
[Link]
Name=eth0
```

denotes a comment

Here we see a very common form of comments that you will see a lot in configuration files. Everything from # to the end of line is considered a comment. Note the # doesn't have to be at the start of a line, and inside strings it doesn't count.

Not all configuration files use the same sort of comments or language structures: this is one of the ugliest parts of Unix systems.

Make a copy of this file (called, `/etc/systemd/network/70-intnet2.link`) and adjust the name of the link as appropriate.

Being sure to edit the file with root privileges, consult the screenshots you took previously of your VirtualBox network configuration and change the NAME of each interface appropriately, using the MAC address to double-check that the adapters are connected to the correct interfaces.

Editing as root

If you're not sure how to open a file with root privileges, you can try using a command such as **sudo nano -w `filename`**. Later, you will learn about a more capable editor compared to the very simple **nano**.

[7] It can use a lot more than just the MAC address, check the documentation

Warning

Once the changes have been made, run **sudo update-initramfs -u**.

This command must be run every time changes are made to systemd configuration files.

Exercise

Now that you've run the **update-initramfs** command. Once done, as an exercise restart (reboot) Client1, then bring up a terminal and make sure you have **ifconfig -a** output similar to the following:

```
$ /sbin/ifconfig -a
intnet1   Link encap:Ethernet  HWaddr 08:00:27:99:c2:7d
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

intnet2   Link encap:Ethernet  HWaddr 08:00:27:19:3b:c2
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        … We've seen that before
```

Disable Network Manager if it has reactivated. In the next step, we will affect a permanent interface configuration.

Different Linux systems configure networking differently to each other, so Debian-based systems, such as Ubuntu, are different from Redhat-based systems, such as Fedora, which are different again from SuSE (Novell), Slackware, Gentoo, Arch, etc. It is completely impractical to cover all the possibilities. Even to cover just Debian and Redhat systems would be beyond the scope of this paper, which is about Systems and Network Administration, not necessarily about Linux administration. The previous section regarding temporary configuration is what you might call "first principles" material, and Debian, Redhat, and others use those as part of their own configuration system.

Since we have to ultimately deal with some sort of vendor-specific systems, we shall see how we do this in Debian-based systems, which is what Ubuntu is. On Debian-based systems, network interface configuration is stored in the file `/etc/network/interfaces`. Have a look at this file now on Client1. You can use **cat /etc/network/interfaces**.

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# The loopback network interface
auto lo
iface lo inet loopback

The following stanza may not appear
# The primary network interface
auto eth0
#iface eth0 inet dhcp
```

Notice that it is referring you to the manual page interfaces(5), which contains a lot of useful information, but unfortunately the initial example can be a little overwhelming. The `interfaces` file is read by two commands: **ifup** and **ifdown**, which take care of bringing an interface up or down, configuring them appropriately and optionally running commands before or after an operation, as specified in the `interfaces` file.

If we look at the first "stanza", which contains the `auto lo` and `iface lo` lines, we see that the loopback interface (`lo`) is brought up automatically when the system boots. Its IPv4 (`inet`) configuration is that of the `loopback` interface. That's not particularly interesting though.

Now look at the `eth0` stanza, which is now out-of-date because it is renamed as `intnet1` previously in `70-intnet1.link`. The interface comes up when the system boots, but by default is not configured according to the `interfaces` file (notice that it is commented out). The `interfaces` file is used by **ifup** and **ifdown**, if Network Manager is not running on a Ubuntu desktop system. However, Network Manager can override the `interfaces` file and only uses it for *static* interface configuration. That is, Network Manager is free to manage anything not otherwise managed in this file.

Open the `interfaces` file in an editor such as **nano**, being careful to use root privileges. Remove any lines that refer to the "eth0" stanza (if any). If they do not exist, that is fine. Then add the following content at the end of `interfaces`:

```
auto intnet1
iface intnet1 inet static
    address 192.168.1.1
    netmask 255.255.255.0

auto intnet2
iface intnet2 inet static
    address 192.168.2.113
    netmask 255.255.255.0
```

You could reboot once done, which would also test whether the interfaces "come up on boot" (meaning that the interfaces get configured when the operating system boots), or we could avoid a reboot and just use the **ifdown** and **ifup** commands:

```
# ifdown intnet1 intnet2
ifdown: interface intnet1 not configured  that's expected
ifdown: interface intnet2 not configured
# ifup intnet1 intnet2
ssh stop/waiting                          Run automatically in /etc/network/if-*.d/
ssh start/running, process 1641           Don't panic if it takes a bit longer than expected
ssh stop/waiting
ssh start/running, process 1700
```

Exercise

Now, as the final exercise look at the interfaces, and then verify connectivity by pinging Client2, as below.

```
$ /sbin/ifconfig
intnet1   Link encap:Ethernet  HWaddr 08:00:27:99:c2:7d
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe99:c27d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:3857 (3.8 KB)
```

```
intnet2   Link encap:Ethernet  HWaddr 08:00:27:19:3b:c2
          inet addr:192.168.2.113  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe19:3bc2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:3885 (3.8 KB)

lo        … we've seen this before
$ ping -c2 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=8.48 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=2.83 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 2.836/5.658/8.481/2.823 ms
```

Finally, complete! Now proceed to the self-assessment section and then we'll clean up after that.

6. Self-assessment

1. What is significant about the address 127.0.0.1?

2. Given the network 10.0.0.0/16, write down one address that is inside the same network, and another that is outside of this network. How many hosts could fit inside this network?

3. **[Optional challenge.]** This question is completely optional. It is included for those students who may already quite familiar with the tools we have used today. Completing this challenge will very likely improve your understanding of ARP and its role in mapping between IP and link-layer protocols such as Ethernet, but only do it if you feel that you weren't very challenged by today's lab.

Here is a problem and some solutions modelled on real-life. While each of the solutions would work, they each have a certain degree of suitability. Explain what happened to cause the problem to present itself. Also, rank the possible solutions in order from best to worst:

A computer is moved from one IP address to another on the same network. However, it cannot access some network resources, including its router to get out of the network and onto the wider inter-network. No other machines appear to be affected. The problem goes away after some minutes for some resources, but for some, such as the router, it can take up to two hours.

There are multiple possible solutions that would work: wait it out; reboot all the machines (typically these are shared resources such as servers and printers) that can't be accessed; flush the ARP cache on all the machines that can't be accessed; or send out a "Gratuitous ARP" to announce to others the new mapping between MAC and IP.

7. Cleanup

Shutdown both Client1 and Client2, preferably after you've checked everything is working. In the main VirtualBox window, click on the virtual machine called "COSC301-client1". Take

a snapshot of the current machine state (you may need the current state in future for testing purposes). Then, find the Snapshots tab, click on the snapshot we created before (you perhaps called it something like “Prior to starting interface management lab”), and click on the Restore button, which is a computer icon with a caret (^) above it. It won’t take long, and after the very brief operation, if you hover your mouse about the “Current state” line, it should say “This machine’s state is identical to the current snapshot”, or similar.

After starting Client1 again, if you start up a terminal, you should find that **ifconfig -a** only reports one Ethernet interface, which is eth0.

You are now ready to begin the next lab.