COSC301 Laboratory Manual

Computer Science Department,
University of Otago

COSC301 Laboratory Manual:

Computer Science Department, University of Otago

Part I. Paper Administration Details

Quality Expectations

Part of the University's goal is preparing you for the professional world, and that includes an expectation of quality with regard to submitted work. Therefore, it is an implicit requirement that any submitted work look reasonably professional. Marks may be deducted for spelling, grammar and presentation for any work, or the work may be rejected and asked to be resubmitted.

University Policy on Plagiarism

> Students should make sure that all submitted work is their own. Plagiarism is a form of dishonest practice. Plagiarism is defined as copying or paraphrasing another's work and presenting it as one's own (University of Otago Calendar 2011 page 224). In practice this means plagiarism includes any attempt in any piece of submitted work (e.g. an assignment or test) to present as one's own work the work of another (whether of another student or a published authority). Any student found responsible for plagiarism in any piece of work submitted for assessment shall be subject to the University's dishonest practice regulations which may result in various penalties, including forfeiture of marks for the piece of work submitted, a zero grade for the paper, or in extreme cases exclusion from the University. The University of Otago reserves the right to use plagiarism detection tools.
>
> —http://www.otago.ac.nz/study/plagiarism.html

Academic Integrity and Academic Misconduct

Academic integrity means being honest in your studying and assessments. It is the basis for ethical decision-making and behaviour in an academic context. Academic integrity is informed by the values of honesty, trust, responsibility, fairness, respect and courage. Students are expected to be aware of, and act in accordance with, the University's Academic Integrity Policy [http://www.otago.ac.nz/administration/policies/otago116838.html].

Academic Misconduct, such as plagiarism or cheating, is a breach of Academic Integrity and is taken very seriously by the University. Types of misconduct include plagiarism, copying, unauthorised collaboration, taking unauthorised material into a test or exam, impersonation, and assisting someone else's misconduct. A more extensive list of the types of academic misconduct and associated processes and penalties is available in the University's Student Academic Misconduct Procedures [http://www.otago.ac.nz/administration/policies/otago116850.html].

It is your responsibility to be aware of and use acceptable academic practices when completing your assessments. To access the information in the Academic Integrity Policy and learn more, please visit the University's Academic Integrity website [http://www.otago.ac.nz/study/academicintegrity] or ask at the Student Learning Centre or Library. If you have any questions, ask your lecturer.

About this Paper

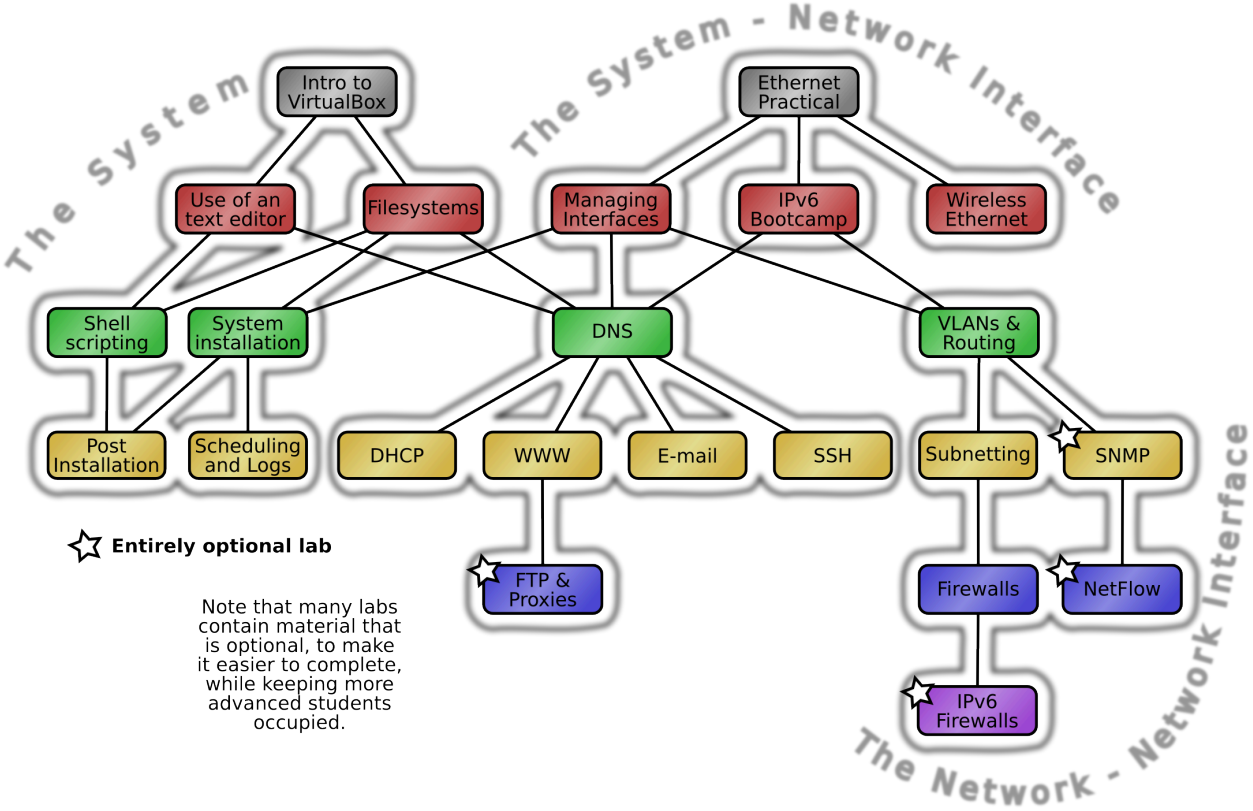
COSC301 is a paper about network management. We cover topics such as system administration using Linux, Ethernet, and network management using a typical router interface.

There are two labs per week, each two hours long. The reading for this paper is primarily to read the lab notes, which in total are large enough to consitute a usefully sized text-book, which you will likely find useful after the course as well.

Because of the diverse background of students coming into this paper, you should all make certain to work through the pre-lab material. This will ensure that you can log in, and have basic Linux/Unix experience; it is also important for getting everyone off to a smooth start without having login problems etc.

In order to stay up-to-date with events in the paper, you are expected to check your Computer Science e-mail once a day also. If you are not using your Computer Science address for your daily correspondence, it is your responsibility to have it forwarded. You can do this through the Computer Science webmail interface. If forwarding off-campus, you are required to ensure that a copy remains on-campus; that way if your off-campus e-mail becomes unreachable for whatever reason, you still have it available locally.

You will have been used to papers previously where the structure of the paper was very linear: each lab depends on all the labs previously. That cannot be the case in a paper such as this where there are many dependencies in knowledge. This is illustrated in the following diagram.



> Labs cannot be presented in a fully linear fashion, but there are dependencies. Students should not expect each lab to necessarily follow on from the immediately previous lab, but will use skills developed in some previous labs.

As you can see, many labs depend on DNS. This mirrors reality: many networks services, such as Web servers use DNS, but you don't need to know about FTP services in order to set up a SSH server. There are some broad themes however; you can break this paper into three parts: the first part introduces you to aspects of the individual system; the second part introduces you to network services run on servers for clients; and the third part looks at network services that run between devices such as routers, which clients have very little to do with.

So if you're feeling a bit lost, wondering how a particular lab relates to the previous lab, realise that it probably doesn't and that this mirrors real life, but that there are some labs that you will have needed to complete in order to progress onto others.

Typographical Conventions

As you find in any technical documentation, there are a number of typographical conventions you need to be familiar with to properly understand what is being presented to you.

To make computer text non-ambiguous from the surrounding prose angle quotes are used to show input text. So, for example, `localdomain.`. You can easily therefore tell that localdomain is to be entered with precisely one dot after it, and that the second dot is part of the surrounding text. The font is also changed to a monospace font.

The following is what you see when interacting with the shell or other program:

```
prompt$ you type this
Output from your command
A commentary about what is happening
Output from your command
prompt$ command filename
Question from running command? [y|n] y
Output from your command
```

When editing a file, you will see something similar to this. Note that when a file is being changed slightly your attention will be drawn to those areas that are highlighted.

```
…    the ellipsis indicate omitted content
option wpad code 252 = string;  bold indicates changes

subnet 192.168.1.0 netmask 255.255.255.0 {
    option wpad "http://webserver.example.com/wpad.dat";
}
```

The prompt may be different, depending on what you need to know. Sometimes, a directory might be shown, such as `/etc$`, in which case it is expected that the command be run from the directory `/etc`. The `$` is a common notation for **sh**-based shells (such as **bash** which you are all using by default) and means that the command is to be run as a normal user. A `#` would indicate that it is to be run as the root user, typically by using the **sudo** command.

Often you will need to enter something different than what the text is saying; the **`filename`** above for example. You are not expected to type in **`filename`**, but rather a filename, such as **`/etc/resolv.conf`**, as indicated by the surrounding text.

Beware though that different conventions may be used in on-line and printed documentation.

Exercise

To complete the lab work, you are reminded the important tasks as an exercise in order to make sure the system has worked. They will be indicated like this paragraph. Make sure you have completed them successfully!

Screenshot

For assessed labs, to show that you have completed the lab work, you are asked to take a screenshot, they will be indicated like this paragraph. They will sometimes contain instructions too --- make sure you read carefully!

We also want to draw your attention to different things in a variety of ways. For example:

Tip

This is used for a useful tid-bit to make your life easier.

Note

This is important as you may or may not need to take action.

Important

This is something that you ought to know or do.

Caution

This is something will cause you to make mistakes. Care should be taken.

Warning

HERE BE DRAGONS!

Pre-lab Material

> **Important**
>
> If you do nothing else in this lab, make sure you can at least log on to the lab machines at Lab F or E at the Computer Science department and familiarise yourself with the course website.

To prevent having to teach a large swath of introductory material in the first couple of weeks of the course, we have elected to create this pre-lab. You should do this (or at least ensure you are comfortable with the skills) before you come to the first lab, and make sure you are up to speed on the basics, such as Unix command-line basics. As many of the students in this course are Computer Science majors, many will already have the requisite knowledge covered in this pre-lab material. But please take the time to ensure you know what is covered here.

1. Logging On to the Mac

Your login for the Macs (including your home folder) are administered by the Computer Science department. If you need to have your password reset, you must go through them.

Once you have completed logging in, start up Safari (or another web browser if you prefer,) and navigate to the course homepage [http://www.cs.otago.ac.nz/cosc301/]. You should better drag the site icon in the URL bar to the dock or desktop to make a shortcut to it. Here you will find news, lab errata, and other class resources.

2. Reading Documentation

There is much documentation to be found on most Unix-like systems, such as Linux or Mac OS X. Most commands have a manual page, which you can get from the terminal prompt.

Lets say we want to know about a command called **ls**, which is a command for listing files (much like the **dir** command under DOS). Though there are **GUI** tools for this, where we're going in this course, there is no **GUI**. Manual pages are sorted under a number of sections, as described under the manual page for **man** manual page. If you see a notation like man(1), then that means to look under section 1. Most of the time you don't need to specify a section, it will take the first match it finds.

```
$ man 1 ls
The section is optional
$ man ls
```

You can use the arrow keys, space, page up/down to navigate. When you're done, type **q** to exit. Notice how each manual page describes a brief synopsis about what the command does, its options with their meanings, various notes and other bits and pieces.

Have a look at the man(1) manual page and have a quick look at what sections there are. Most of what you'll need will be found in sections 1, 5 and 8.

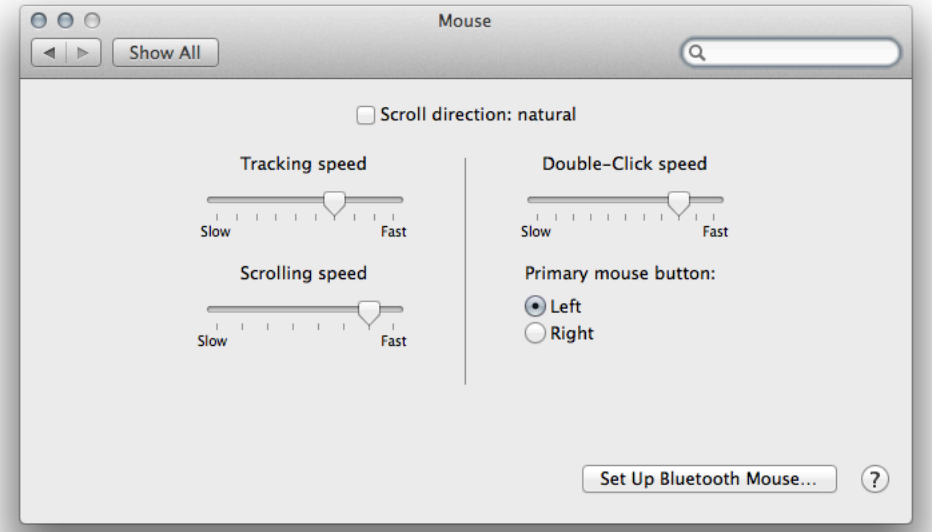
3. Natural Scrolling

In this laboratory, we are using Apple iMacs for our workstations. With the advent of touch displays, Apple have decided to change the way the scroll wheel works. Prior to Lion (version

10.7), if you wanted to scroll to the bottom of the page you would (logically) move the scroll wheel down (and vice-versa). However, this is counter-intuitive for touch displays as the content would end up going in the opposite direction. The net effect is that you now scroll up to move the content down when on a computer.

This option is entirely a personal preference, so you can enable or disable it as you see fit!

Figure 1. Turn-off Natural Scrolling



Natural scrolling, to scroll the content down, you move the window up --- which is modelled after touch displays. Unchecked you move the content underneath the window (and is the old behaviour).

4. Taking Screenshots on Mac OS X

The most basic is a full-screen shot. You can do this using **Command**-**Shift**-**3** (in Mac OS X documentation, this may be styled as ⌘⇧**3**).This will take a screenshot, and put it into a file on the Desktop called `Screen Shot timestamp.png`.

If you include the **Control** key as well, making it ⌘⌃⇧**3**, it will vary the result slightly by putting it on the clipboard instead. Typically, it is the **Option** key that generally modifies commands, which makes key combinations easier to remember.

If you just want a selection of the window, use ⌘⇧**4** instead. The cursor will change to a crosshair instead, allowing you to select the region on screen you want. Include **Control** (which has the symbol ⌃) to put it into the clipboard instead.

Finally, and very usefully, you can easily take a snapshot of a window or dialog by ⌘⇧**4** and then pressing **Space**, which will turn the crosshair into a camera, which you can click on a window to take a shot of the window.

5. Basic Unix Skills

If you don’t have any experience working with Unix systems (as a user), then please complete a basic Unix tutorial, such as this UNIX/Linux Tutorial for Beginners [http://info.ee.surrey.ac.uk/Teaching/Unix/].

6. Changing Your Password

You can change your password easily using the Macs; click on the blue Apple in the top-left, and select System Preferences.... In the Users and Groups preferences, you can find your way easily from there.

Industry-respected security expert Bruce Schneier has a good write-up on Real-World Passwords [http://www.schneier.com/blog/archives/2006/12/realworld_passw.html] and another on Choosing Secure Passwords [http://www.schneier.com/blog/archives/2007/01/choosing_secure.html] that you should read.

Part II. Laboratories on System-based Services

The System-Network Interface

Lab 1 Introduction, Operating Systems

In this introductory laboratory, we shall know how the labs will build on each other. We shall then practice some basic lab skills. We will practice these skills implicitly in later labs, meaning that the labbook will expect us to know how to do these, so please pay attention and ask questions whenever one is struggling to understand or perform any of the activities in this laboratory. It is *not* recommended to do this lab outside of the prescribed lab environment for this lab.

Please report any problems you have with the lab material; this will help students both in this year and subsequent years.

As an overview for this lab session, we will be practicing the following basic skills: seeking help and working as a supportive community, accessing class resources, changing your password, introduction and correct use of VirtualBox, and practicing some basic terminal skills. Mastery of these essential skills will help you complete the assigned labs with fewer problems, greater speed and gain a more thorough understanding of the lab material and its relation to the lectures.

> **Important**
>
> It is vitally important that you learn to cope intelligently with errors and learn to diagnose problems; being able to give a good diagnostic report is a very desirable outcome of this paper. Working on other labs can be a useful way of coping if you're stuck waiting for help.

1.1. Overview of the Laboratories

The laboratories in this paper build on some of the previous labs. Therefore, some labs will require you to have completed the work in some of the previous laboratories. This means you not only have to install and configure a service, but also maintain it, which involves some configuration management skills.

Our network will start off very small, and we shall slowly grow it as needed. However, we shan't have any more than about five virtual machines at most in any one lab, due to host resource requirements.

The network we shall build mirrors quite nicely what you might find in a Small Or Home Office (SOHO) network, so you should immediately be able to reproduce and build on what you learn in this course at home. This is an important way of solidifying and expanding your knowledge and skills, which is crucial for developing your experience in preparing for work in the industry.

We start building our network in this lab by importing a ready-made machine. This machine shall have a hostname of "client1", and in VirtualBox, we shall refer to as as "cosc301-client1", in order that all COSC301 related machines are grouped together. In later labs, we will add a second client using a Live CD, then quickly move on to installing our server virtual machine which will eventually become our gateway in our SOHO network to the wider network.

1.2. Accessing class resources

During much of this paper, you will be developing a network of computers. To accomplish this, you will sometimes require resources such as CD-ROM disk images ("ISO images") or entire machines which may have been prepared ("appliances"). It may also include documentation; whatever is needed. The exception would be the on-line edition of the labbook, which should be available from the course website, where this pdf was found.

Another resource which will be *very* important is where you need to store your virtual machines.

To easily find the resources like ISO images and appliances and to create a storage for your virtual machines, open a Terminal window from the MacOS dock and type the following commands:

```
$ ln -s /home/cshome/coursework/301/resources/ ~/Desktop/resources
$ mkdir ~/Desktop/myvms
$ cd ~/Desktop/myvms
$ pwd
/home/cshome/h/hzy/Desktop/myvms
This is the fully qualified path of myvms.
You may see something slightly different, but remember the output for later use.
```

Now you should have on your Desktop the two folders: `resources` and `myvms`, which you will need to access often while working on the labs. The output of the last command above will tell you the fully qualified path of the folder `myvms`, which you will need shortly.

Note that the contents of the `myvms` ("My Virtual Machines") folder is individual to your VirtualBox. You can read from or write to it like your home directory. However, the `resources` folder will be read-only to you.

1.3. Seeking help and working as a community

In this paper, you are never really alone when you are working on your lab material; there are always avenues of support readily available. One of the essential skills an administrator needs to be able to do is to know how to tap these support avenues.

But tapping these resources takes time, so you also have to try and solve or at least investigate the problem yourself to some extent, so you can give a good description of the problem. It is this description which is *essential* to getting a prompt and useful response. It is our continued experience that when one starts writing down what steps they have done to try and solve a problem – with a view to submitting a request for help or bug report – that they end up solving the problem themself.

Some support avenues that are available to you are described below. The best option will depend on your circumstances:

Classmates
: Much of the value of teaching this course is in learning from others mistakes, which has enriched our experience as well. There is a lot of truth in the saying "to teach is to learn twice". Therefore, you are allowed to share experiences with each other when trying to solve problems. Naturally, this does not mean giving each other the answer to a question

— that would be plagiarism — but rather helping each other to diagnose problems and evaluate solutions.

Teaching Team
This includes your Demonstrators as well as your Lecturers.

Class Resources
In this lab, you will learn how to access the "resources" folder for this paper, which contains various screen casts, scripts and other files that are useful or necessary for completing the labs quickly.

Class Website
The class website [http://www.cs.otago.ac.nz/cosc301/] (as you have already known) is where you can find important information including: an on-line version of the laboratory manual, which should always be the most up-to-date version; the course schedule with links to the lecture notes; the PDF version of the marksheet; assignment material and even some videos to help you to perform some of the lab activities.

Class E-mail list
Any class correspondence will be sent to the cosc301@cs.otago.ac.nz mailing list. This is where you will find lab notices such as updates and corrections. Please also use this as an avenue for asking questions outside of lab time. *Also, please feel free to post useful replies, so long as they do not give assessment answers directly.*

The Internet
A lot of help can be found on the internet; one of the more likely things you will want to use this for is to look for particular error messages. Note that Usenet News (available via such avenues as Google groups) is also a rich resource. There are many support avenues around which you should learn and share; it is a very useful way of increasing your experience and problem solving ability[1].

At your convenient time, skim-read Eric S. Raymond's "How To Ask Questions The Smart Way [http://catb.org/~esr/faqs/smart-questions.html]" and Simon Tatham's "How to Report Bugs Effectively [http://www.chiark.greenend.org.uk/~sgtatham/bugs.html]". In fact, you should probably consider the first to be required reading; it *will* save you a lot of time and face in your future student and professional career… Even some Post-Graduate students need to read this. In fact, it's so useful, you are required to at least skim-read "How To Ask Questions The Smart Way" for part of the self-assessment of this lab.

1.4. Correct use of VirtualBox

The "myvms" folder on your Desktop is where you will need to store your virtual machines.

You must first configure VirtualBox appropriately such that the location of the virtual machine files will be stored in the "myvms" folder.

Procedure 1.1. Initial Configuration of VirtualBox on Mac OS X

Following this procedure, you will configure VirtualBox to save your VM disk images on "myvms", which is regularly backed up by our IT support.

1. Start VirtualBox, which you will find in the `Applications` folder or the dock on your Mac OS X.

[1] It can also raise your Internet profile, which is useful when someone is looking to hire you.

2. From the menu, choose VirtualBox → Preferences….

3. Under the General section, set the Default Machine Folder field to `~/Desktop/myvms` by clicking the drop-down menu on the right of the field and finding the folder from the menu. Here "~" means your home directory such as `/home/cshome/t/tom`.

> ## Note
>
> If you want to type the folder into the field directly, make sure you type the fully qualified path (as shown before) into the field.

4. From the same VirtualBox → Preferences… window, under the Update section, untick Check for Updates. This can prevent VirtualBox from showing update notification.

We shall now use some simple features of VirtualBox, importing a simple virtual *appliance*, which is prepared for you, and practicing some basic skills related to VirtualBox.

> ## Note
>
> If interested, read this note box.
>
> To save you some time, we have created a client for you, and you can import it into your VirtualBox environment. This is basically the concept of a *virtual appliance* whereby an entire operating system and software stack has been configured for a particular task, ready to be imported into a virtualisation environment. You can learn more about Virtual Appliances with VirtualBox by looking at the “Importing and Exporting virtual appliances” episode of the Fat Bloke’s Shorts [http://www.virtualbox.org/wiki/VirtualBoxTV]. Client1 is a default installation of Ubuntu 18.04 LTS “Bionic” desktop amd64 installation, with updates applied and configured to use a particular Ubuntu software repository. Additionally, some software specific to VirtualBox (“Guest Additions”) has been installed to help the guest run faster and with greater capability. We shall discover more about the Guest Additions when we look at the lab concerning System Installation.
>
> We provide some local media containing the large installation files for the appliance. They are made locally available for performance reasons, otherwise it can be too slow.

From within the VirtualBox window (ie. not a window for a virtual machine, but the window that lists the available virtual machines), import Client1 using File → Import Appliance… and Choose the `cosc301-client1-latest-version.ova` file from the `~/Desktop/resources/Appliances` folder. Accept the settings in the following dialog by clicking “Continue”. Importing the appliance will take a while, perhaps 5-10 minutes.

> ## Schedule your time intelligently
>
> Whenever you have a lengthy wait on your hands, work on other things that aren’t blocking on the current task. This gets you thinking about project and time management. For example, while waiting for the virtual appliance to import, open up a terminal on your Mac OS X (or Linux workstation) and go through the Terminal Agility section of this lab, which doesn’t require the use of VirtualBox or Client1.

Once you have imported the virtual machine, Start it. Log in as the user “Miss A. Laneous”, who has the username `mal`. The password is `Quack1nce4^`. This user, with the same password, will be used throughout the machines in this paper; but do note that these are separate accounts that share neither password storage nor home directory contents.

Exercise

As an exercise make sure that the Client1 VM is running and that you have successfuly logged into the guest. Note that, this type of *Exercise* boxes is very important for you in the rest of the labbook. They indicate what are expected to work in the labs, which will be expected to work in the assignments of client and server in this paper. Some exercises are for you to have better understanding of the lab work.

Note that, after you log in, the virtual machine may automatically offer to update software packages, which may take a while. You may leave it until it finishes, or cancel it and choose to do so later.

A Password Mnemonic

A duck walks into an elevator, but is too short to reach the buttons, so an attendant says “quack once for up”. You may find such mnemonics useful to generate and remember strong passwords and pass-phrases.

1.5. Using sudo and su for Administration

In subsequent labs, you will need to regularly manage the system which will require administrator (the “root” user) rights. Because this affects system security, we’re briefly going to introduce you to it, and you’ll practice the skills very often throughout the rest of the paper.

On a standard Unix/Linux host, superuser privileges are generally attained through the use of the substitute user command **su**. This is considered better than logging in as root because it creates an audit-trail, meaning we can look at the login records and determine how root privileges were gained. For example, we might look at the login record and find that a user called “test” logged into the system at 3am in the morning from some remote system in the Russian Federation (all of which are likely very unusual) and got access to the root account using **su**.

On modern Unix-like hosts, you also have the option of using the **sudo** command. Whereas **su** gives you a session (sub-shell) as root, **sudo** gives you the ability to run a single command as root. **sudo -s** can be used to give you a shell also, much like **su**.

Don’t do everything as root!

If you do everything as root, you *will* eventually fall victim to yourself. Instead, use **sudo** where you can, and **sudo -s** or **su** only when you need to.

Actually, neither command requires that you end up as root. Indeed, **su** is short for *substitute user*, so you could change to any other user. This is perhaps useful if you need to change from root to a normal user, or from a normal user to a special account, such as a database administrator.

All Unix-like systems will have **su**, and many will have **sudo**, depending on administrator preference. It is installed by default on Mac OS X and Ubuntu, and is many other Linux systems. One of the things that make both Ubuntu and Mac OS X a bit different is that the root account is by default locked on both systems, and require the use of **sudo** to gain priveleges.

sudo has a different authentication model than **su** or practically any other authentication system you will have used. Instead of having to enter in the root password – assuming that account is not locked – you instead enter *your own* password. This is to authenticate yourself to the **sudo** command, as someone else may have started using your account if you had left your workstation unattended for a while. The system administrator (root) will have added you to a file called `/etc/sudoers` (which is edited using the command **visudo**, but don't try that now). This allows the system administrator to grant particular users the ability to run particular commands as a particular user. This allows the system administrator the ability to give partial root access without disclosing the root password.

Because the user "mal" was the user that was created when the system was installed, that user has already been added to the `sudoers` file.

sudo will remember when you last successfully authenticated. If you use **sudo** shortly thereafter, it will not ask you for your password. By default, this timeout is 15 minutes, and can be overridden in the `sudoers` file.

Try the following exercise to practice getting root access. Do this in a terminal window on Client1; there should be a Terminal icon in the Gnome panel at the left of your virtual machine; if not you can search for Terminal. Before you begin, follow the instructions below.

We're first going to use our normal user account to look at the permissions on a protected file that root can only read. We'll try to read it as a normal user, which will fail with "Permission denied". We'll then elevate our privileges using **sudo** and see that it works.

```
$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1144 2015-01-08 09:57 /etc/shadow
```

We'll see more about filesystem permissions in a later lecture/lab, but for now you'll just need to understand that this file is not readable by a normal user, only the "root" user (and members of the "shadow" group, which we can ignore) can do anything with this file. In case you're wondering, this file stores all the local users passwords in a secure manner, which is why it has restricted access. Let's just verify that a normal user can't access it:

```
$ whoami
mal    The command ran with the rights of the "mal" user.
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

We first used the **whoami** command to learn the usercode we were running under; actually, what we found out is what user the **whoami** command was running as. The **cat** command outputs a file to the terminal, but it failed because the user "mal" doesn't have access to read `/etc/shadow`. Okay, so how *do* we get access to that file? We use **sudo**, which is as simple as the following:

```
$ sudo whoami
[sudo] password for mal: enter mal's password, not root's!
root
$ sudo cat /etc/shadow
root:!:14621:0:99999:7:::        You see the commands output
daemon:*:14545:0:99999:7:::      which has many lines
```

```
bin:*:14545:0:99999:7:::          that are not important
…                                 so we'll use … to hide them.
```

Great! So we verified, using the **whoami** command, that we can change our privileges to the "root" user by **sudo**, and further verified that we could now do something we couldn't do with our regular "mal" user. Running commands as root is very common, so common that there is a common notation you will find in technical documentation[2]. Look closely at the prompt, in the prompts you will see in this labbook, all commands that run as a normal user have a prompt of $, while all the prompts for a root-user command will have #, so in the following transcript, the first line is asking the same as the second line:

```
# whoami
root
$ sudo whoami
root
```

Sometimes, the command you need to run is too awkward to easily run using **sudo**, and you really want to run it from a shell. To do that, use **sudo -s**. Try this now:

```
$ whoami
mal
$ sudo -s       Start a shell with root privileges
root# whoami    Yes, I'm root
root
root# exit      Leave our root shell
exit
$ whoami        Back to where we were in line 1
mal
```

Exit from your root shell by tying **exit** or **^D**. This will exit your inner, root shell, and return you to your previous shell, which was running as "mal".

1.6. Shutting down

When you are finished, temporarily, with a virtual machine, you have a few options to shut it down. To do this on Mac OS X, do the same as you would do to close a window: click on the red gem (of the red, yellow and green gems in the top left of a Mac OS X window). You can also use **Host+Q**, remembering that on a Mac, the host key is by default the left ⌘. When you do so, a dialog will slide out, allowing you to choose one of the three options:

- Save the machine state: this is like hibernating a laptop, the contents of the system and graphics memory is saved by VirtualBox to non-volatile storage (ie. somewhere on disk). The only thing that cannot be preserved is external state such as network connections, so these will most likely fail when the machine is woken up. However, this should be used most of the time in your day-to-day work.

- Send the shutdown signal: this is equilavent to pressing the power-switch on a modern computer. It will cause a power-management signal to be sent to the operating system, which will commonly do things like a) pop up a menu on-screen to ask the user what it should do; b) sleep, or c) shutdown.

- Power off the machine: *Never* choose this option unless you have to. This is like forcing a machine to turn off, such as by pulling out the power cord or holding in a power button for

[2] They come from the Bourne shell of which your shell is a derivitive. % is also sometimes used, which comes from the C-shell family of command-line shells.

five seconds. The operating system has no chance to clean up, so filesystem corruption can occur. This should be a last resort when the above two options do not work.

Exercise

As an exercise practise the first option to save the state of your virtual machine; its state should show as “Saved” if successful.

1.7. Terminal Agility

When you know a few tricks, working inside a terminal can be a very fast way of operating. Alas, even experienced people don’t know a lot of these ways, and end up not being as productive as they could be. So in this section, we aim to increase your agility in the terminal by giving you a few exercises for you to practice in the following labs.

You do not need to memorise this list; but if you never hear of them, you may never think of them. The most important ones are marked with a pencil icon; it would pay to memorise these few to begin with.

Most shells use a library called readline; since this is widely available on different systems, these keystroke sequences will work in many shells, on Linux as well as Mac.

Important

The following list uses the Emacs style of keystoke notation, so C-a means to type **a** while holding down the **Ctrl** key.

M-a means to hold down the **Meta** key, which is not a key you will find on modern keyboards; instead, use the Left Alt (or the Left Option key on a Mac).

Cursor movement

C-a, **C-e**
✎ Move to the beginning and end of the line. Type some text in your shell, use the keystrokes to move to the beginning and end of the line.

M-f, **M-b**
Move forward and back a word at a time, but note that this doesn’t always work; the keystroke might instead invoke menu functions instead. Thankfully, some Linux systems, Ubuntu included, define **Ctrl+LeftArrow** and **Ctrl+RightArrow** for the same task.

Editing commands

Backspace, **C-h**
✎ Delete character to the left of the cursor. Knowing about C-h can be useful, because Backspace is not always reliable on some systems.

Del, C-d
Delete character to the right of the cursor.

C-w
✎ Delete the word to the left of the cursor. Type a few words in the shell, then use C-w to erase them, word by word. This is much faster than using backspace. It can be faster to erase and retype a whole word than to use backspace.

C-c
✎ Abort the current command being entered. Begin typing a command, and then type C-c, to see it disappear.

C-l
Clear screen. This commonly also means redraw screen in various full-screen terminal applications, such as editors.

History commands

C-r
✎ Start searching backwords in your history, incrementally.

For example, assume you had in your command history a command that mentioned a file `dhcpd.conf`. You can then type **`C-r`** and then start typing **`dhcpd.conf`**, as you type each character, you will be shown items in your history that match. You can type **`C-r`** again to show matches earlier in your history.

You can use the standard editing keystrokes to edit the command if you like, or you could just type **Enter** to run the command again.

!!
✎ (pronouced “bang-bang”) is a sequence that you will probably use very often, as it expands to the last command line you used. This is useful when you forget to use **sudo**:

```
$ some long admin command
Operation not permitted       Oops, forgot to use sudo
$ sudo !!
sudo some long admin command
Now it works, and we saved a lot of typing or navigating our history.
```

Command Completion

Tab Completion
✎ One incredible time-saving device a lot of students *not* using is Tab completion. This makes it *much* faster to accurately navigate around the filesystem and commands.

To use Tab-completion, you simply type **Tab**, and it will complete as much your current word as it can before it gets ambiguous. If you type **Tab** again, it will present you with a list of choices. If there is a large number of choices, you will be prompted with a `--More--` prompt, and you can press **Space** to view the next page, or **q** to quit. The best way is simply to try it out. Don’t be afraid, you have a lot to gain.

C-x C-e
This one is perhaps less useful for a beginner, but if you remember it, it will make developing longer commands (often the precursor to a script) much easier. This keystroke sequence will open your current command in your preferred editor (which can be set using the `EDITOR` environment variable, which by default on Ubuntu is **nano**).

1.8. Self-assessment

Exercise
As an exercise do the following tasks.

1. You must consult the manual pages for sudo(8) and sudo_root(8) (only available on Ubuntu systems) to answer these questions.

List briefly the advantages and disadvantages of using **sudo** as set up on Ubuntu, and of leaving the root account disabled. Ensure you understand what is being said in the documentation.

2. List five major points of the “How To Ask Questions the Smart Way” that might be of particular relevance to a system administrator asking for help. (note that most are relevant, and make sure you’ve actually read the document)

Lab 2 Interface Management

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 2.1, “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 2.2, “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.

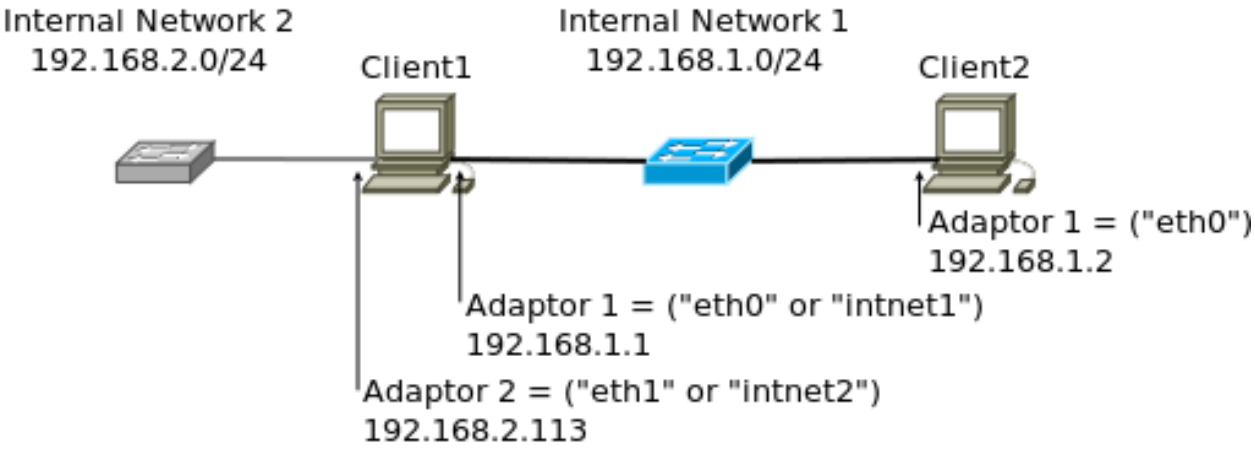
Naturally, there is a lot of ways in which network can go wrong, so we need some diagnostic tools to help us figure out what’s actually happening. We’ll look at some management and diagnostic tools that will be useful here.

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.

2.1. A Map, Notation and a bit of Theory

Figure 2.1, “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 2.1. 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”.

2.2. 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 2.2, “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 2.2, “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 2.3, “VirtualBox after Snapshot at Beginning of Lab” shows the GUI interface after taking the snapshot.

Figure 2.2. VirtualBox Snapshot

The VirtualBox interface for taking a snapshot.

Figure 2.3. 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.

[3] Actually, anything in 127.0.0.0/8.

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.

2.3. 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 2.4, “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 2.4. 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
```

2.4. 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 2.5, “Disabling Network Manager” shows where you can find the NetworkManager icon, and what you see when you click on it.

Figure 2.5. 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)
```

[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.

```
        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.

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 2.5, "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

[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.

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 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
```

2.5. 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.

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

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**.

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.

2.6. 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.

2.7. 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.

Lab 3 IPv6 Bootcamp

Reading guide

With the exception of Section 3.4, “Diagnostic and Query Tools”, you should be able to read most of this lab prior to coming to perform the lab. This will help you understand the material again (just like a text-book, the second reading is usually much more informative than the first).

There are many things you just can’t ignore, such as death and taxes; and there are things that you probably should ignore, such as requests from your users to install the latest greatest version of Mahjong on their workstation; but there are also things that you could ignore for now and learn later, which is what a lot of people have done with IPv6. This is a great and terrible illustration of the chicken and egg problem: most people aren’t bothering too much about IPv6 because not many are asking for it; and not many people are asking for it because not many people are offering it. Meanwhile, four of the five regional Internet registries (RIR) managed by the Internet Assigned Numbers Authority (IANA) have run out of IPv4 addresses. The RIRs are responsible for distributing publically routable IPv4 address blocks to Internet Service Providers (ISP). The Asia Pacific Network Information Center (APNIC), responsible for New Zealand, Australia, China, India, among others, has around 8 million IPv4 address remaining. These are distributed in blocks of 1024 addresses at a time. With approximately 300 new requests per year, this is expected to take a long time to run out completely[1]. The APNIC Labs [https://labs.apnic.net/?p=1107] has a report on the latest figures.

IPv6 is both simpler and more complex at the same time. Most of the complexity comes from general inexperience and ongoing development or misconceptions in most of the industry, but especially with regard to the various transition mechanisms to help IPv6 work in an IPv4 world, and IPv4 work in the coming IPv6 world. Some management areas become more complicated, such as DNS, while others ought to become simpler (such as address management).

We don’t have a lot of time in this lab to explore a lot of IPv6, so we shall cover only the very basics. We shall look at IPv6 enablement of various network services at the same time as we cover IPv4. In this lab, we shall look mostly at the client and how to manage IPv6 on the client side. Here is a brief overview of what we shall cover in this lab:

1. Practice enabling and disabling IPv6 on Linux, and how to control autoconfiguration etc.

2. Observe how IPv6 works with router advertisements using Wireshark, which is a network traffic analyser (“network sniffer”).

3. Practice the use of basic IPv6 diagnostic and query tools, namely **ping6** and **ip**.

3.1. Preparation

In the previous lab on basic network interface configuration, you should have restored to the snapshot you took at the beginning of that lab. In this section, we want to ensure that everything is still as we expect, and to add a new virtual appliance which we shall using just for today.

[1] See https://www.apnic.net/community/ipv4-exhaustion/graphical-information/ for more information.

In this lab we shall only be needing Client1 and a new virtual appliance from the `resources` folder. The new virtual appliance is called `COSC301-v6bootcamp-radv-version.ova`, which we shall call Radv. “Radv” is the virtual machine acting as our router advertiser. Radv is running the router-advertisement daemon (**radvd**).

On Client1, ensure that only one network adaptor is present, and that it is connected to the Internal Network “COSC301 Internal Network 1”. Start Client1, and ensure that **ifconfig -a** shows only one network interface (excluding the loopback interface), and that it is called “eth0”, and that it has no IPv4 address (no “inet” line, though it will have one “inet6” line). Basically, you should see this:

```
$ /sbin/ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:99:c2:7d
          inet6 addr: fe80::a00:27ff:fe99:c27d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1368 (1.3 KB)  TX bytes:1836 (1.8 KB)

lo        … Seen this before …
```

If you find an interface called “eth1” or “intnet*”, then you need to complete the Cleanup section from the previous lab. Reboot Client1 if you needed to make any modifications.

Next we need to start importing Radv. In VirtualBox’s main window, import the Radv appliance from the `Appliances` in the `resources` folder. If you cannot remember how, consult the notes from the "Introduction" lab. The filename will be named similarly to `COSC301-v6bootcamp-radv-version.ova`. Use the latest version available unless otherwise instructed. Accept the default settings as presented in the import dialogs.

The operation will take several minutes to complete. While you are waiting for the import operation to complete, go onto Section 3.2, “Enabling and Disabling IPv6”. We will come back to Radv in Section 3.3, “Observing Router Advertisements”.

3.2. Enabling and Disabling IPv6

In Linux, the act of enabling IPv6 is generally very trivial, as it is most often enabled by default. The tricky thing is disabling it if you don’t want it. Even trickier still is only enabling it on certain interfaces. In this section, we look at how support gets added to an IPv6 interface, how we can remove it if required, and how we can control some aspects of how IPv6 configures itself.

In Linux, the kernel support for IPv6 *might be* contained in a kernel module called `ipv6.ko` (the `.ko` extension meaning “kernel object”). Support for IPv6 is often made available as a loadable module; although in the case of Ubuntu 9.10 and later, it is built into the kernel, which complicates matters somewhat if you need to disable it.

One of the most common questions regarding IPv6 on Linux is how to turn it off, because like anything, if you don’t need it you don’t really want to be running with it. Many sites are not yet ready to support IPv6 and so it doesn’t make much sense to be using IPv6 because if you try using something that is unsupported in a production environment you can expect problems to occur. Running with IPv6 can cause transition mechanisms to be used which invites problems relating to tunneling, performance and security.

> **Note**
>
> Not all of the techniques below are applicable on Client1, which is running Ubuntu 18.04 LTS. Read all the sections, but only try the command shown in the sections with [applicable].

Removing IPv6 addresses when support is built-in… [applicable]

For us, as we are using Ubuntu 18.04 LTS, we don't have a kernel module we can unload as the support is built into the kernel. We could perhaps recompile the kernel and omit IPv6 support, but that is a lot of work and it doesn't help us if we just want to enable IPv6 on specific interfaces, which would be a handy thing to do on a gateway. Compiling your own kernel also creates a maintenance requirement that we could probably do without.

We can manage addresses on an interface using the **ip** tool. It can work with both IPv4 and IPv6 and is a Linux-specific tool, unlike the likes of **ifconfig**, which you can find on most Unix-like systems. Unlike IPv4 commands, which grew up with a shared heritage, IPv6 stacks have been developed more independently and have much different management interfaces on each platform.

Here is just one example of using the **ip** command to delete an address attached to an interface. Thinking back to the types of addresses that were covered in the lecture, what sort of IPv6 address is being deleted here?

```
$ ip -6 addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::a00:27ff:fe97:164a/64 scope link
       valid_lft forever preferred_lft forever
# ip -6 addr del fe80::a00:27ff:fe97:164a/64 dev eth0
```

If we wanted to bring it back, we *could* use **ip -6 addr add address**, as shown in the example below.

> **Note**
>
> To add or delete an IPv6 address, make sure to include the prefix length (/64) explicitly after the address; otherwise the prefix length is defaulted to /128

```
# ip -6 addr add fe80::a00:27ff:fe97:164a/64 dev eth0
```

However, because the above command is generally used for configuring addresses manually, we won't use it here for our link-local address, which should be assigned automatically. We can illustrate that by bringing the interface back up using **ifconfig**:

```
# /sbin/ifconfig eth0 down
# /sbin/ifconfig eth0 up
```

And now our IPv6 address is back, which can be shown by the following command.

```
$ ip -6 addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::a00:27ff:fe97:164a/64 scope link
       valid_lft forever preferred_lft forever
```

If you don't see this, repeat the above command.

Normally though, we would expect to bring an interface down or up using **ifdown** and **ifup**, as that includes the configuration of IPv4 addresses and other associated tasks. We can't do this at the moment on Client1, because we haven't configured `/etc/network/interfaces`, so it would not work presently.

```
This won't work on Client1 currently
# ifdown eth0
There may be output from any "hooks" that get run
# ifup eth0
There may be output from any "hooks" that get run
```

Passing arguments to the IPv6 subsystem… [applicable]

As soon as the IPv6 support is loaded into the kernel (in the case of built-in support this would be at boot-time) it will perform StateLess Address AutoConfiguration (SLAAC) in order to generate most of its addresses, so if we want to prevent that from happening, or configure how it is done, we need to pass it some arguments.

In the case of kernel modules, we can add arguments inside `/etc/modules` or similar, or by hand when we use **modprobe**. We can find out what parameters are available to a module using **modinfo**, but that assumes that we're dealing with loadable modules. However, for subsystems that are built into the kernel like IPv6, you need to add them to the kernel command-line, which is set by the bootloader (typically, this is GRUB on modern systems). So what parameters can we use, and how do we learn about them?

To find out what parameters are available, you could try looking in various manual pages such as ipv6(7) (more relevant to network programming) and ip(8) (interface management). But in this case, where we are wanting to know about kernel modules and such, the various documentation [http://www.kernel.org/doc/Documentation/] contained in the Linux kernel source is probably going to be more informative. In particular, the file `Documentation/networking/ip-sysctl.txt` documents the sort of thing we're after; here is the relevant extract; you are not expected to understand it just yet:

```
/proc/sys/net/ipv6/* Variables:

IPv6 has no global variables such as tcp_*.  tcp_* settings under ipv4/ also
apply to IPv6 [XXX?].     Hmm, how reliable is this documentation?

bindv6only - BOOLEAN
 Default value for IPV6_V6ONLY socket option,
 which restricts use of the IPv6 socket to IPv6 communication
 only.
  TRUE: disable IPv4-mapped address feature
  FALSE: enable IPv4-mapped address feature

 Default: FALSE (as specified in RFC2553bis)
…
conf/default/*:
 Change the interface-specific default settings.

conf/all/*:
```

```
 Change all the interface-specific settings.

conf/all/forwarding - BOOLEAN
 Enable global IPv6 forwarding between all interfaces.

    This would be important for creating a router, but we don't care about that just yet.
…
conf/interface/*:
 Change special settings per interface (where interface is the name of your network interface).


accept_ra - BOOLEAN
 Accept Router Advertisements; autoconfigure using them.

 Functional default: enabled if local forwarding is disabled.
       disabled if local forwarding is enabled.

accept_ra_defrtr, accept_ra_pinfo
    We can fine-tune what we accept in a router advertisement.

…
autoconf - BOOLEAN
 Autoconfigure addresses using Prefix Information in Router
 Advertisements.

 Functional default: enabled if accept_ra_pinfo is enabled.
       disabled if accept_ra_pinfo is disabled.
…
forwarding - INTEGER
    Configure interface-specific Host/Router behaviour.

    Note: It is recommended to have the same setting on all
    interfaces; mixed router/host scenarios are rather uncommon.

    Possible values are:
    0 Forwarding disabled
    1 Forwarding enabled

    FALSE (0):

    By default, Host behaviour is assumed.  This means:

    1. IsRouter flag is not set in Neighbour Advertisements.
    2. If accept_ra is TRUE (default), transmit Router
    Solicitations.
    3. If accept_ra is TRUE (default), accept Router
    Advertisements (and do autoconfiguration).
    4. If accept_redirects is TRUE (default), accept Redirects.

    TRUE (1):

    If local forwarding is enabled, Router behaviour is assumed.
    This means exactly the reverse from the above:

    1. IsRouter flag is set in Neighbour Advertisements.
    2. Router Solicitations are not sent unless accept_ra is 2.
    3. Router Advertisements are ignored unless accept_ra is 2.
    4. Redirects are ignored.

    Default: 0 (disabled) if global forwarding is disabled (default),
    otherwise 1 (enabled).


…
disable_ipv6 - BOOLEAN
 Disable IPv6 operation.  If accept_dad is set to 2, this value
 will be dynamically set to TRUE if DAD fails for the link-local
 address.
```

```
 Default: FALSE (enable IPv6 operation)

 When this value is changed from 1 to 0 (IPv6 is being enabled),
 it will dynamically create a link-local address on the given
 interface and start Duplicate Address Detection, if necessary.

 When this value is changed from 0 to 1 (IPv6 is being disabled),
 it will dynamically delete all address on the given interface.
…

IPv6 Update by:
Pekka Savola <pekkas@netcore.fi>
YOSHIFUJI Hideaki / USAGI Project <yoshfuji@linux-ipv6.org>
```

Also, in the `Documentation/kernel-parameters.txt` file, you are told that you can pass in arguments to built-in modules (in which case they are no-longer "modules" as such) using `modulename.parameter=value` syntax. So, for example, to run with the `disable_ipv6` option turned on (and hence disabling IPv6 addresses from being assigned), we can add the following argument to the command-line of the kernel:

```
ipv6.disable=1
```

But where do we put it? Assuming we are using the GRUB boot-loader, we need to edit the GRUB configuration. In Ubuntu 18.04 LTS, this is done using the file `/etc/default/grub` and those files in `/etc/grub.d/`. The two sources of files are combined into one using the command **update-grub**, which should be run after modifying either of these sources; the resultant file is stored in `/boot/grub/grub.cfg`, but you should not edit that as it will be overwritten on subsequent calls to **update-grub**. Find the part of `/etc/default/grub` that looks like the following:

```
GRUB_CMDLINE_LINUX=""
```

and change it to the following:

```
GRUB_CMDLINE_LINUX="ipv6.disable=1"
```

Run, with root privileges, the command **update-grub**; the final line of its output message should say `done`. Now reboot the guest operating system; you can do this easily from the command-line using **sudo shutdown -r now**, or from the GUI.

Exercise

As an exercise after you have restarted, make sure there is no IPv6 (`inet6`) address listed in **ifconfig**. If it does not work, double-check if you disabled the network manager.

Let's think about what we have just done. Previously, we had an IPv6 enabled kernel with IPv6 support enabled *by default*. Now, we have an IPv6 enabled kernel, but the interfaces are IPv6 *disabled* by default.

Regardless of whether IPv6 is by default enabled or disabled, we can enable or disable IPv6 interfaces at runtime by modifying the `disable_ipv6` parameter as follows:

```
$ sudo sh -c "echo 0 > /proc/sys/net/ipv6/conf/interface/disable_ipv6"
```

This is the general interface for tuning the networking stack behaviour. The contents of the `/proc` filesystem are special, as they are an interface into the running kernel. We shall revisit

this concept later in the paper when we look at filesystems in more detail. We will use `/proc` more frequently when we encounter firewalls, which are devices that control what traffic can enter or leave an interface.

Now *undo* your changes, because we actually *do* want IPv6 for the remainder of this paper. Verify that you get an IPv6 address when you boot the guest.

3.3. Observing Router Advertisements

In this section, you will practice using a network analyser called Wireshark[2] to take a close look at what happens when an interface is configured using StateLess Address AutoConfiguration (SLAAC) and to observe other fundamental IPv6 mechanisms.

By now Radv should have finished importing. Go into its Network settings and ensure that its Adaptor 1 is connected to the Internal Network "COSC301 Internal Network 1". Then Start it.

The purpose of this machine is send out IPv6 "router advertisements", so host on the network can learn about their local router(s) and determine a suitable set of addresses to use. Radv will not be used as a router for this lab, we only need it to offer us router advertisements, so we can study what they contain and how it influences Client1. You will not need to configure anything inside this machine.

When you have imported and started the new machine, which is called "Radv", it will boot to a console-login window ("radv login:"). You can minimise the machine, you won't need to interact with it.

Exercise

As an exercise on Client1, run **ifconfig** and **ip addr show** in a terminal window on Client1. You should see that you now have another address in addition to the link-local address (fe80::/64) you will typically always have. This address, which starts with fd6b:, is a "Unique-Local Unicast Address". If you see this address it means the router advertisements from Radv worked with Client1.

Wireshark needs to run with sufficient privileges to capture traffic, which generally—though not necessarily—implies root privilege. However, Wireshark is such a complicated program (due to all the various protocols it tries to understand) that experience has shown it to be something of a security risk, so ideally you would run it as a non-root user that has the appropriate permissions to capture traffic; but currently Wireshark doesn't support separation of privileges when you are capturing and analysing (displaying) the packets at the same time. So just run **wireshark** using a privilege elevator such as **sudo**.

Note

In the related video, we show an alternative, more secure way of doing the capture and analysis, using **tcpdump** for the capture, and **wireshark** for the analysis.

```
# wireshark
```

[2] Previously known as Ethereal.

Tip

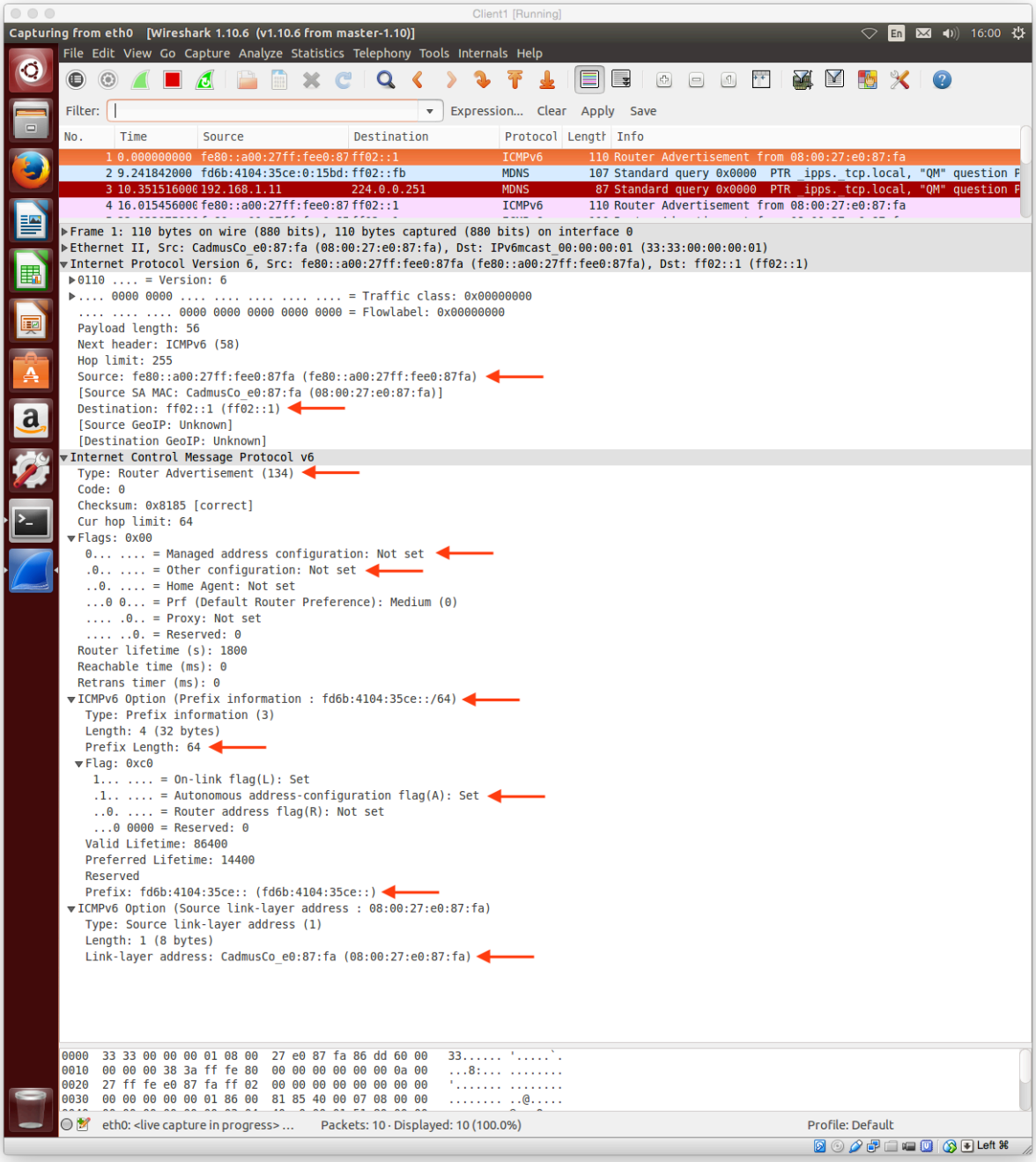
Wireshark will detect that it is running as root and pop up an alert box, which you will need to dismiss before you can continue running the application. Note that the alert box may be hidden behind the Wireshark window.

Start capturing on the first ethernet interface (it should be “eth0”). You should soon see ICMPv6 Routing Advertisements. The **radvd** software running on the machine Radv has been configured to send out routing advertisements rather more frequently than needed, to reduce your waiting time. Stop the capture when you have at least 10 or so. Using the View menu, un-check the Packet Bytes to give yourself more screen real-estate. Alternatively, you can just make the VirtualBox window larger and (due to the fact that the VirtualBox Guest Additions are installed inside the guest) it should change the screen resolution of the guest.

Exercise

As an exercise click on any one of the ICMPv6 Router advertisement packets. In the Packet Details area of the Wireshark window, click on the disclose (+ or little triangle) button recursively to open up the packet so you see that all of the fields and subfields for the “Internet Protocol Version 6” and “Internet Control Message Protocol v6”. Take a screenshot showing the entire contents, you will want to refer to it later.

Figure 3.1. Detail of IPv6 Router Advertisement in Wireshark



Wireshark showing the full details of a single IPv6 Router Advertisement.

Figure 3.1, “Detail of IPv6 Router Advertisement in Wireshark” shows the screenshot similar to what you should see, showing the details of a single router advertisement. Right now we’re going to discuss what is significant in this packet.

Here is a brief explanation of the most important parts of the router advertisement, from top to bottom in the packet details view.

IPv6 Source

In the packet shown, this is the "link-local unicast" address of Radv. We can confirm this by logging into Radv as `mal` and running **ifconfig**. (note that your Radv will have a different MAC address from ours) and seeing that the IPv6 address has a MAC address embedded inside of it[3].

IPv6 Destination

ff02::1 is a well-known address commonly called the "all-hosts link-local multicast address". Every host on the local network (the "local link", which is everything up to the first router) will be listening for traffic sent to this address.

Internet Control Message Protocol v6 (ICMPv6) Type

ICMPv6 is a management protocol that is very important to the running of IPv6. As such, there are many different types of messages it could transmit. To identify the type of message being transmitted, the Type field is used. Here, type 134 identifies this message as being a Router advertisement.

ICMPv6 Flags

The "Managed" address config flag specifies whether "Stateful configuration" is to be used. Stateful configuration would typically be understood to mean DHCPv6 (we learn about DHCP in a later lab, but we'll ignore DHCPv6 in this paper). On most networks, this flag would not be set, meaning that the host should use "StateLess Address AutoConfiguration" (SLAAC).

The "Other" stateful config flag means that nodes on this link (local network) should use Stateful configuration (DHCPv6) for things other than address assignment. Thus, if you don't use Stateful configuration for address assignment, you can still use it to advertise other information (such as information about DNS services, but we learn about DNS later in this course).

For example, in the capture we saw on our own network, we saw M=0 and O=0, meaning "Not managed" and "Not other". So we don't use stateful configuration (such as DHCPv6) to try and get an address (instead we just use SLAAC), and neither do we use stateful configuration to find out other, non-address information about the network (eg. DNS settings, which host to send log messages to, and a wide range of other possibilities, some of which we mention further in the lab on DHCP).

These "M" and "O" bits (Managed and Other) are important for understanding how IPv6 address assignment works. The remaining flags are not important for what we are aiming to understand today.

ICMPv6 Option (Prefix information)

There can be a number of optional components to a router advertisement. The most common would be an option advertising what "prefix(es)" are used on this link. A prefix is very much like a network address: in SLAAC, a set of addresses is formed by taking each prefix and adding the interface's EUI-64 host ID (typically formed by the MAC address).

Note that there can be multiple "Prefix information" options included in a router advertisement. An interface can use multiple prefixes to generate multiple IPv6 addresses.

ICMPv6 Prefix length

Somewhat self-explanatory, it tells us that this particular prefix is a /64 (remember that IPv6 addresses are 128-bit). With SLAAC, all advertised prefixes are /64, which makes this particular entry rather less interesting. We'll come back to it when we see the Prefix.

[3]This need not always be the case if you consider the "privacy" addresses.

ICMPv6 Flags
: The only flag we wish to draw to your attention here is the Autonomous Configuration flag (shown as “auto” in the screenshot). Recall that there can be multiple prefixes included in a router advertisement: only the ones marked with this “auto” flag get SLAAC addresses. This is on by default.

ICMPv6 Prefix
: This is the key piece of data in this announcement. Coupled with the prefix length we saw earlier, we see that this the prefix being advertised is fd6b:4104:35ce::/64. Viewed uncompressed, so you can see exactly how long the prefix is in relation to the whole address:

```
fd6b:4104:35ce:0000:0000:0000:0000:0000
     network bits ↤ ↦ host bits
```

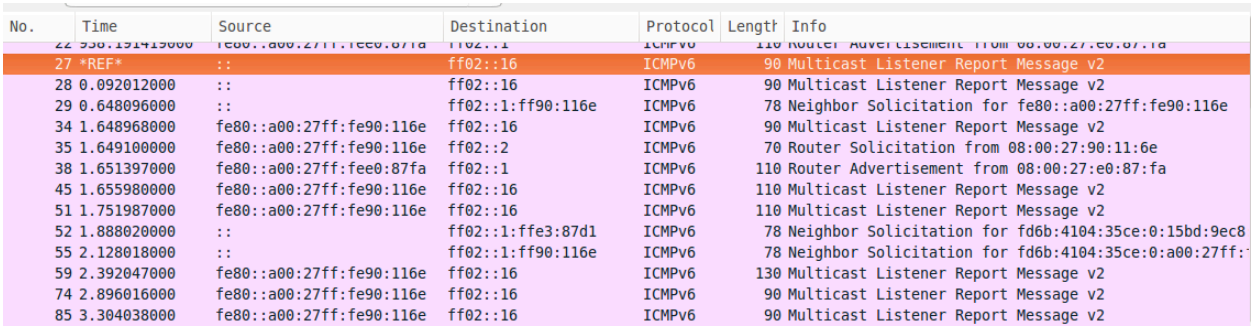
ICMPv6 Source link-layer address option
: This option simply communicates the router’s MAC address to the hosts on the link, so they can pre-cache it in their “neighbour cache” (this is like ARP, although IPv6 doesn’t use ARP but something similar which we don’t want to get into right now). Having it included here means we don’t have to have an extra round-trip on the network to figure out the link-layer (MAC) address of the router, which reduces delay.

So that’s a router advertisement, seen during steady-state whereby nothing particularly interesting is happening such as a new host appearing on the link. Let’s now repeat the capture and this time we shall see everything that happens when an interface comes up.

Exercise

As an exercise resize the Wireshark display so that the packet list area takes up most of the room. Start a new live capture in Wireshark; you may discard your previous capture. When it has started capturing, from a terminal window, as root, disable IPv6 with **echo 1 > /proc/sys/net/ipv6/conf/all/disable_ipv6**. Now remember what the last packet was that is currently shown in Wireshark’s packet list area. When you're ready, run change the 1 to a 0 in the previous command, you should see several packets arrive in the window. Then stop the capture and save the captured packets. Figure 3.2, “What Wireshark sees when an Interface “Comes Up”” is what we see on our own capture, which may likely be a bit different from yours; in particular, the ordering of some packets can vary.

Figure 3.2. What Wireshark sees when an Interface “Comes Up”



No.	Time	Source	Destination	Protocol	Length	Info
27	*REF*	::	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
28	0.092012000	::	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
29	0.648096000	::	ff02::1:ff90:116e	ICMPv6	78	Neighbor Solicitation for fe80::a00:27ff:fe90:116e
34	1.648968000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
35	1.649100000	fe80::a00:27ff:fe90:116e	ff02::2	ICMPv6	70	Router Solicitation from 08:00:27:90:11:6e
38	1.651397000	fe80::a00:27ff:fee0:87fa	ff02::1	ICMPv6	110	Router Advertisement from 08:00:27:e0:87:fa
45	1.655980000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	110	Multicast Listener Report Message v2
51	1.751987000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	110	Multicast Listener Report Message v2
52	1.888020000	::	ff02::1:ffe3:87d1	ICMPv6	78	Neighbor Solicitation for fd6b:4104:35ce:0:15bd:9ec8
55	2.128018000	::	ff02::1:ff90:116e	ICMPv6	78	Neighbor Solicitation for fd6b:4104:35ce:0:a00:27ff:
59	2.392047000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	130	Multicast Listener Report Message v2
74	2.896016000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
85	3.304038000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	90	Multicast Listener Report Message v2

Wireshark showing the packets that are seen when an interface comes up. The packet marked as *REF* is the first packet to come into the system after we brought up the interface.

Rather than having you read about what each packet is doing, we're going to show you with a video, which will walk you through what is happening in this sequence of packets; or rather, a similar sequence of packets. In this video, which lasts about half an hour, you will encounter various parts of IPv6 that form part of its basic mechanisms. We require that you have some understanding of what is going on, to an extent whereby you could look at such a listing, and with a bit of work, explain basically what is happening. In the self-assessment, you will be asked to summarise some of what you have seen.

The video can be found in the "resources" folder of your desktop, and is called *IPv6 SLAAC* under `Lab resources/IPv6`. However, you should watch the video later.

Watch the video later

This is because you won't need to be in the lab to complete the video-watching task, but you will want to be in the lab for Section 3.4, "Diagnostic and Query Tools". Doing this will help you use your time more effectively.

3.4. Diagnostic and Query Tools

This section is very simple; it's just learning to use some simple tools to see what we they can tell us and to get ourselves familiarised with the environment. Without this familiarity, we are blind and lame when we need to diagnose problems. We shall limit ourselves to a very small selection of commands.

Basic ip usage

With IPv6, we generally have a new set of tools, or system specific additions to old tools, such as **ifconfig**. On Linux, the new, preferred and much more capable tool that manages most of the network stack is simply called **ip**.

The **ip** command has extensive help built in, so you generally don't use the manual page for getting documentation. The help system is modal; it depends on what mode you are using. For example, try these commands on Client1 to see the different help you get.

```
$ ip help
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] [-batch filename
where  OBJECT := { link | addr | addrlabel | route | rule | neigh | ntable |
                   tunnel | maddr | mroute | monitor | xfrm }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -f[amily] { inet | inet6 | ipx | dnet | link } |
                    -o[neline] | -t[imestamp] }
$ ip addr help
Usage: ip addr {add|change|replace} IFADDR dev STRING [ LIFETIME ]
                                                      [ CONFFLAG-LIST]
       ip addr del IFADDR dev STRING
       ip addr {show|flush} [ dev STRING ] [ scope SCOPE-ID ]
                            [ to PREFIX ] [ FLAG-LIST ] [ label PATTERN ]
IFADDR := PREFIX | ADDR peer PREFIX
          [ broadcast ADDR ] [ anycast ADDR ]
          [ label STRING ] [ scope SCOPE-ID ]
SCOPE-ID := [ host | link | global | NUMBER ]
FLAG-LIST := [ FLAG-LIST ] FLAG
FLAG  := [ permanent | dynamic | secondary | primary |
           tentative | deprecated | CONFFLAG-LIST ]
CONFFLAG-LIST := [ CONFFLAG-LIST ] CONFFLAG
CONFFLAG  := [ home | nodad ]
LIFETIME := [ valid_lft LFT ] [ preferred_lft LFT ]
```

```
LFT := forever | SECONDS
```

In order to understand what is being shown, you simply need to understand that `[ … | … ]` denotes an optional choice, `{ … | … }` denotes a mandatory choice, `lowercase` are keywords (mandatory) and `UPPERCASE` are rules ("grammar productions" to use a Computer Science term) that expand (`:=`) further. You can also see that you just need to add **`help`** to the end of the command you are trying to complete; it's pretty simple, though a little bewildering at first. Most of the things you don't need to worry about for simple uses; the **ip** command can allow for quite advanced commands which makes the grammar more useful to have.

Let's use **ip** to look at some useful information. For each example, add **`help`** to the end to see what else you can do. Since we're only interested in IPv6 at present, we'll add the `-6` option so it limits its operation to IPv6. Also, we're not going to show the output of the commands; this makes you have to figure out what they do for yourself.

```
$ ip -6 addr
…
$ ip -6 link
…
$ ip -6 route
…
```

Okay, so that was pretty basic, we just looked at the Layer 3 network address configuration (for IPv6) and the Layer 2 datalink (Ethernet, in this case) information.

We want to briefly look at the glue the binds the IPv6 layer (network layer) to the Ethernet layer (datalink layer). One of the things that makes IPv6 different from IPv4 is that we no-longer use ARP to determine the Ethernet MAC address that is bound to a particular IPv4 address. Instead, IPv6 has its own protocol for this called Neighbour Solicitation. Likewise, the Neighbour Cache has replaced the ARP cache; that's a little simplified, but it will do for now.

```
$ ip -6 neigh help
…
$ ip -6 neigh
fe80:a00:27ff:fe69:9487 dev eth0 lladdr 08:00:27:69:94:87 router STALE
```

We use the above output to illustrate a few things (your output will look different). The basic pattern is an IPv6 address is associated to a link-layer address (in this case, an Ethernet MAC address) on a particular network device. This IPv6 address apparently belongs to a router (but at the moment we don't care about that). Each entry has a Neighbour Unreachability Detection (NUD) state; "stale" means the entry is valid but not particularly trustworthy because of its age. Let's ping Radv and see what changes in our neighbour cache.

Pinging with IPv6

Log in as mal on Radv (the password is the same) and find out Radv's enp0s3 IPv6 link-local address. We suggest you write it down. Now ping Radv from Client1; don't include the prefix length (`/64`) in the link-local address of the command below:

```
$ ping6 -c2 radv-link-local-ipv6-address%interface-id-of-the-local-host
…
```

If you get "connect: Invalid Argument", that indicates that you forgot to include the interface id of the local machine such as `eth0` or `intnet1`, or the interface id was wrong. If instead you got "unknown host", then it means you copied the address wrongly such that it is no longer a valid IPv6 address.

> **Note**
>
> Notice that because we are pinging a link-local address (fe80::/64), which every interface has and is therefore generally ambiguous, we need to use a special syntax to identify the interface we want to use `%eth0`. A number can be used for the interface ID on some systems instead, such as Windows.

What actually went on there? If you had been looking at what was happening on the network, using Wireshark or similar, you should likely see something similar to this, although it can differ depending on what each side already knows. So the packets used look like the following:

1. Client1 wants to send an IPv6 packet to Radv's IPv6 link-local address (layer 3), which is a local delivery. But it probably doesn't yet know Radv's link-layer address (layer 2), so it needs to look it up by using a **ICMPv6 Neighbour solicitation** from Client1's link-local IPv6 address to the Solicited node multicast address for the address that shall be pinged. It also carries with it a notification of Client1's link-layer (Ethernet) address, preventing Radv from having to look it up later and thus saving network traffic.

   This is analogous to an ARP request in IPv4.

2. Radv responds with a **ICMPv6 Neighbour advertisement**. It comes from Radv's link-local IPv6 address to Client1's link-local IPv6 address. It carries a notification of Radv's link-layer (Ethernet) address, which Client1 puts into its Neighbour Cache.

   This is analogous to an ARP reply and the result being cached in the ARP cache.

3. Now Client1 knows everything it needs to in order to make the local delivery of this IPv6 packet, so it sends the **ICMPv6 Echo request** ("ping"), from Client1's link-local IPv6 address to Radv's link-local IPv6 address (which is the address we pinged).

4. Radv has already learned of Client1's link-layer address, so it doesn't need to perform the Neighbour Solication process again, as it has it in its neighbour cache. So Radv can now send the **ICMPv6 Echo response** packet immediately, which is its response to the "ping" (Echo request) packet.

Notice that this is a *local*, or "on-link" delivery; it does not go through a router. We'll ignore the issue of routing until much later.

We've seen some multicast addresses being used; how do we find out which multicast addresses the host is currently interested in? **ip** can help here also:

```
$ ip -6 maddr
…
```

That's it, a few very simple commands to allow us to investigate the local link (layer 2) and IPv6 in a single network.

There is much we haven't covered in this lab, such as assigning static IPv6 addresses, which is really easy, but we'll cover that later.

3.5. Self-assessment

This lab has been very introductory, and we don't want to labour you with a lot of assessment, except to give you a little to help you check that you have understood some of the new concepts you have seen today.

You should be able to answer the following questions after attending the IPv6 lecture and watching the video as mentioned in Section 3.3, “Observing Router Advertisements”. You can work on the questions in small groups (to help everyone understand).

1. Describe multicast, as opposed to unicast and broadcast. Also, give an example of an IPv6 multicast address.

2. Give an example of a link-local IPv6 address. Briefly describe how it is generally formed from the Ethernet MAC address.

3. Give an example of a “unique-local” IPv6 address.

4. Describe a Solicited Node [multicast] address. What is its purpose? Give an example. How is it formed?

5. Multicast Listener Discovery version 2 (MLDv2) Snooping is a mandatory feature for any switch that wants to support IPv6[4]What is the motivation for MLDv2 snooping, which in the IPv4 world we might compare with “IGMP Snooping”? We want to see that you understand its purpose with respect to network switches (not routers). You may like to include a diagram with a little table showing real IPv6 multicast addresses (such as two solicited node addresses, and an all-routers address (ff02::2) and switch port numbers.

6. Briefly describe the most important components of a router advertisement.

7. Briefly describe Duplicate Address Detection (DAD). Why is it needed and how is it performed? What do you think the difference is between a “Tentative” address and a “Preferred” address, with regard to DAD?

[4]Or at least, mandatory in the sense that it should be a purchasing requirement.

Lab 4 Shell Scripting

> **Note**
>
> For this lab, you should do your work in Client1, as it contains some required software components. Also you need to install a few more packages using **sudo apt install libtext-csv-perl libnet-patricia-perl**

Before you proceed to go on with this lab, please make sure you have completed the pre-lab materials in the first section that covered the use of the Vim editor. Also, make sure you have read the handouts that were provided for the accompanying lecture. If you have questions as to how they work, please ask a demonstrator, in order to clear up any confusion.

This lab will be fairly simple, but there will be some essential commands that we shall need to cover. After that has been completed, you will work through the examples given in the lecture to appreciate how a pipeline is built up, then we'll let you develop your own simple script.

You will likely find this lab very foundational, but also enormously empowering, once you begin to appreciate everything that scripting could do for you.

4.1. Some Useful Commands

There are many commands that you might find useful for scripting, but if we were to give them to you all at once, you would be overwhelmed. So we'll give you some commands that are used very commonly, along with a brief description. You can refer to their manual pages to understand them better. Many of these commands are demonstrated in the lecture handouts.

cat
: Concatenate files and print on the standard output. Commonly this is just used to output a (single) file to stdout, and thus introduce it into a pipeline, although that is commonly unnecessary.

echo
: Display a line of text. Rather like a simple 'print' command.

printf
: Format and print data. Much like C's printf(3) function.

head
: Output only the first n lines of the input.

tail
: Output only the last n lines of the input.

tr
: Transliterate (eg. change to upper-case) or delete characters.

cut
: Remove sections from each line of input. Not very sophisticated; **sed** offers more power.

sort
: Sort lines of text, possibly as numbers.

uniq
Remove duplicate lines from sorted input.

grep
Print lines from input that match a pattern.

wc
Print the number of bytes, words, and lines in input.

tee
Read from standard input and write to standard output and files. If you think of the plumbing analogy, this provides a 'T' junction. It can be very useful when inspecting what is going through a particular part of a pipeline.

ps
Report process status. Using **ps -eo pid,command** can be useful in scripts, as using a custom output format can be easier to parse. Supports a huge number of other options as well, most of which are not useful for scripting purposes.

kill
Send a signal to a process. Signals are not necessarily fatal, but are a primitive form of inter-process communication.

xargs
Build and execute command lines from standard input. See the last slide of the lecture notes for an example. Very useful in conjunction with **find**.

find
Search for files (or directories, etc.) in a directory hierarchy.

find needs plenty of examples to show how best to use it. The manual page for find(1) should contain a section showing various examples. We shall revisit **find** later in this section.

mktemp
Make temporary file name (unique).

du
Estimate file space usage. Note that this program can be particularly annoying at times.

Here is an example you can put into a file called `big5`, that shows the largest five entries in the current directory. It's useful for figuring out where all the disk space is being used up.

```
$ du -k -d1 | grep -v '^[0-9]*[[:space:]]*\.$' | sort -rn | head -5
Your output will look different, you might not have any.
42080   ./Lectures
7020    ./Labbook
…
```

We should mention that there are GUI tools that are much more enlightening about where disk-space is being used, such as the Disk Usage Analyzer application in Ubuntu.

basename
Strip directory, and optionally a named suffix, from filenames.

```
$ basename "/path/to/foo.txt"
foo.txt
```

dirname
Strip non-directory suffix from file name.

```
$ dirname "/path/to/foo.txt"
/path/to
```

date
Print or set the system date and time.

```
$ date "+Today: %y/%m/%d"
Today: 07/04/18
```

sleep
Delay for a specified amount of time, in seconds. Some systems may have **usleep** available, for sub-second intervals.

getopt
Parse command options (enhanced).

An example was provided with the lecture notes.

4.1.1. find, -prune and -print

find can become a very confusing command to understand if you don't specify `-print` as the operation to perform upon match. Note that if you do not specify an operation, such as `-print` or `-exec` then you will get something that appears to be the same as `-print`, *but it is not*. You will notice a (very confusing) difference if you, for example, use the `-prune` option to omit directories from your search. Here is an example, which could take quite a while to figure out, of how to use `-prune` correctly. The reason it took a long time figuring out was because one could be previously under the misapprehension that **find** behaves as if `-print` was the default operation.

Let's assume that our files may have spaces between them, which could prove problematic as spaces separate arguments in the Unix command-line. **find** and **xargs** have the ability to separate arguments using an ASCII NUL character, so we shall use that as well.

In this example, we want to remove certain files from a Subversion working directory, but we *never* want to change anything inside any directory called `.svn` because that is private to Subversion. This is perhaps one of the most common use-cases for wanting to use `-prune`, and it will be useful to you later in your later studies, which is why I'm showing you this now. The way to remove a file under a Subversion repository is with the command **svn remove `filename ...`**.

The particular files we wish to remove all start with `._`, which were put there by my Mac and we accidentally imported them. We could do a similar thing with, for example, LaTeX temporary files.

```
This is an example, we don't expect you to run this command.
$ find . -name .svn -prune -or -name ._\* -print0 \
>  | xargs -0 svn remove --
```

We've also added `--` to the end of the **svn remove** command, as this should cause **svn remove** to not treat as an option any filename which might otherwise be seen as an option. Note that this is unnecessary in this case as we know all results are going to start with `._` and never with `-` or `--`, but in principle this is a very good thing to do.

4.2. Understand the Examples from the Lecture

Now that you've acquainted yourself briefly with the commands listed above, go through the examples from the lecture slides, and understand how they work. It will be useful for you to build up each stage of the pipeline at a time, to see how the output of each command is transformed by the next.

Make sure you ask about those things you do not understand. It may be useful to talk with your peers first. Ensure you understand the example that looks through the web-server logs. You'll be doing a similar thing in this lab.

4.3. Self-assessment

There is a video available in the "resources" folder demonstrating how to get started for the assessment. It shows such techniques such as: creating a place to put your scripts; modifying your `PATH` variable so you can easily run your script; as well as copying and modifying the resources for this self-assessment.

> **Note**
>
> You should watch the video before creating the following script. You will know how to create the script already after watching the video. All the required resources for creating the script are in the "resources" folder, which should be shared with Client1 as below.
>
> Click on "cosc301-client1 [Running]", at the bottom of the Settings window, you will see Shared folders. Click it.
>
> Click on the folder icon with a green plus icon; this will add a new entry. In the Folder Path drop-down box, select Other… and navigate to your desktop. Click on resources and then click on Open and OK.
>
> Now follow the commands below.

```
$ sudo mount -t vboxsf resources /mnt
$ mkdir scripting
$ cp /mnt/scripting/* scripting
$ cd scripting
```

1. You will create a script that is passed Apache server logs as input, and creates a summary showing each unique client's IP address, and whether it is a local, domestic, or international client. By itself, it's not all that useful, but it can be useful if we were to include other data such as the amount transferred; this could be useful if you are charged different rates depending on where the traffic is coming from or going to. This is a basic application of *geolocation*: trying to find out which country a particular request is coming from. Since we're only interested in determining the scope of a client (local, national, internation), we might call this *geoscoping* instead). This is not a particularly accurate business and it's important to keep any geo-location data up-to-date; weekly updates are suggested.

   To help you, we have created two auxiliary Perl scripts in the above directory. The first, called **geolocation-country-prefixes**, converts the IP ranges from the provided

Comma-Separated-Values (CSV) formatted geolocation database into a list of CIDR prefixes for a particular country. The geolocation database we're using comes from IP-to-Country.com [http://ip-to-country.webhosting.info/]. You will only need to use this script once in order to provide the reference data which we shall use over and over again. You need to use the script as below to create `domestics.txt` from `ip-to-country.csv`.

```
$ ./geolocation-country-prefixes country-code < ip-to-country.csv > domestics.txt
```

Tip

The `country-code` is the two of three-letter ISO country code or name for your country. For example: NZ, NZL and "NEW ZEALAND" is New Zealand while OM, OMN or "OMAN" is Oman.

The generated `domestics.txt` will be used by the the second script, called **ip_classify**, which is used to answer the question of whether a particular IP address (or rather, a list of them) are local, domestic or international. The rest of the task therefore is to extract the data we want out of the web-server log files, presenting it in some suitable format, removing duplicates, and presenting the output to **ip_classify**.

ip_classify is given a list of addresses on stdin, and writes out a list of <address,classification> pairs. This script is very fast, because it uses a data-structure called a Patricia Trie[1], which is the same kind of data-structure used for routing table lookups.

If this component of the system were to be done using just shell commands, you would find a vast performance drop. This highlights one useful thing about shell scripting: we can use a selection of different tools in order to fulfill different criteria for different sub-tasks. Because the auxiliary script uses some extra Perl modules (libraries), you will need to do this assessment on Client1 which has the particular modules already available.

Edit your copy of **ip_classify** in order to tell it where to find its resources. Create `domestics.txt` using **geolocation-country-prefixes**.

We should give a proper name for the script. This task is left up to you (indeed, it is part of the assessment). **You must choose a suitable name!** How do we choose a suitable name? We shall offer you some advice: first, use the principle of Huffman encoding: commands that are referred to very commonly should have short names (eg. **ls**), while commands used infrequently should have longer, more descriptive and preferably structured names. The structuring helps when you make use of Tab completion: put the most significant theme at the beginning. For example, suppose we have a number of scripts that do things related to the running of a web server; we might have scripts that generate various reports and we might have scripts that manage the various sites etc. Therefore, it would be useful to have such scripts begin with **web-report-** or **web-site-**, for example. By typing **web-** and then using Tab completion we get a little menu of the suitable commands.

To help get you started, think of noun and verb phrases that describe what the script does, what it operates on, and what it produces. In our case, we might start with a list such as: web logs, geoscope, classify, report, source address, and client. Also think

[1] "Trie" is pronounced "try" and is short for "retrieval"

about what the script *is not*, so as to avoid confusion: its input is web logs, *not* IP addresses.

Warning

Let the following be a lesson in what *not* to do: there are two commands for adding a user to a system, one is interactive and the other is meant for use in scripts. One (we *always* forget which) is called **adduser**, the other **useradd**. If you accidentally use the non-interactive one, you end up creating a user, which you then have to remove and recreate using the other command.

Create your script file and mark it executable using **chmod**.

You will find a copy of some logs in `access_log`; have a look at this file using **less** or a similar tool to become familiar with the format. Remember, all we need to produce from this file is a list of unique IP addresses.

In the shell, which is a good place to develop parts of your script, develop a pipeline that a) outputs only the IP address (hint: **cut** everything into space-delimited fields and output only the first field), b) **sort** the input of IP addresses into a sorted list of unique IP addresses, and c) classifies each line according to **ip_classify**.

Take care of the following:

- Since a call to **cut** will be first in your script, give it whatever arguments you have been given `"$@"` as these could be filenames, or expand to nothing, in which case **cut** will take its input from stdin. This is similar to **cat** in terms of its input: multiple files can be specified *at the end* of the command, or if none is specified input will come from stdin. Many of the Unix toolbox commands (**cut** and **sort** among them) work in this way. See the “Special Parameters” section in bash(1).
- You must name your script suitably.
- You must have a she-bang (`#!`) line.
- Do not assume that you are running in a particular directory. Put required resource files in well-known places. Hard-coding the locations of resources such as configuration files and auxiliary resources can be useful, but hard-code nothing that is particular to a single invocation of the program, such as the location of the input files, or even which log entries you are interested in.

Here are some examples of how your script should be able to be used:

```
One argument
$ your-script ./access.log
…
Multiple arguments
$ your-script ./access.log ./access.log.2
…
Input redirected from file into stdin
$ your-script < ./access.log
…
Input filtered from another command into stdin
$ grep -w 404 ./access.log | your-script
…
```

The output should look like the following.

```
Your numbers will differ.
127.0.0.1 local
139.80.123.34 domestic
139.80.123.36 domestic
139.80.32.2 domestic
```

2. **[Optional challenge]** Add the number of hits from each client, and sort (on the second column/key) the output in descending number of hits. Add column headers and align the output using **column -t**. Output should look something like the following (Hints: You need to use **uniq** and **awk**):

```
IP           HITS   GEOSCOPE
1.2.3.4      123    international
12.3.4.5     113    national
192.168.1.2  12     local
```

4.4. Last Words

Scripting is an incredibly useful skill, and you end up using it in many places. When building software using **make**, you are using shell commands in the `Makefile`. Shell scripts are frequently used to define complex firewall rule-sets for Unix systems, drive nightly backup and maintenance systems, start up and shutdown Unix systems and services and save a lot of repetition in common workflows.

Shell scripting, however, is not a particularly fast way of automating tasks, although it can be much faster in some cases, but they can save a lot of developer time (remember, CPU time is cheap compared to developer/user time). If you find your shell script is too slow or awkward, you would either introduce more complex processing using a tool such as **awk**, or re-write the script in a higher level scripting language, such as Perl, TCL or Python. Perl is a mainstay tool for many Unix system administrators, although Python has been steadily increasing in popularity for many years, particularly on Linux systems.

Lab 5 Filesystems

This lab should be reasonably relaxed, assuming you read the lab before you come to class. So you should have started your first assignment by now when you have time. We will have a stroll around the standard Unix file hierarchy on your virtual machine Client1, and study some of the things we find there. For people familiar with the Unix file system, you may find it rather easy.

We will talk of file-system permissions and how they can be managed, and finally we shall learn about creating archives and backups.

The Design of File-Systems

As a small aside into the design of file-systems (and operating systems in general), it is useful to look at other possible solutions. History is a good teacher here. There are two books you should read when you have some time spare. The first is The Art of Unix Programming [http://www.catb.org/esr/writings/taoup/html/], and the other is Unix Haters Handbook [http://www.simson.net/ref/ugh.pdf], both of which are good reading. The latter book gives some good historical insight into other systems, but beware that much of its criticisms are at the historical Unix, some of the problems have been remedied in modern systems.

5.1. Tour the Virtual File System (VFS)

5.1.1. Items Found in the Filesystem

Unlike some other Operating Systems you may have heard of, the Unix filesystem is a tree structure. It has only one root (/). It doesn't use any concept of drive letters. Every filesystem you use is *mounted* (attached, or made available) onto a directory (*mount-point*) of the filesystem.

There are various types of items you find in the file-system.

Regular files and directories
: Nothing unusual about these. In **ls -F** output (**ls** is short for list), directories have / appended to them. In **ls -l** output, the first character on the line is a - for a regular file, or d for a directory.

Hard link
: Recall that Unix filesystems are *inode* based. The inode is a number that points to the data. Directory entries point to an inode, not to the data directly. A hard link, as created using the **ln** command (short for link), creates another directory entry that points to the same inode.

  A hard link can only point to a file on the same filesystem, and are invisible to the naked eye in standard **ls -l** output. You can tell by using **ls -li** command to display the inode column, and looking for files with the same inode..

***Symbolic links, soft link* or *symlinks*.**
: These are similar to shortcuts under Windows, or aliases under MacOS or Mac OS X, but are more transparent. When a program opens a symbolic link, it opens the file it points to. Symbolic links can also point to directories. Actually, symbolic links can contain any text. If the target does not exist, the link is said to be a *broken link* or *dangling link*.

Symbolic links are created with the **ln -s** command. In **ls -l** output the first character on the line is a `l`, the link target is also printed after the filename.

The only way to remember in what order the arguments of **ln** go is to think of the **ln** command a bit like **cp**, where the source goes first, and the destination (or *new thing*) comes last.

Device node

One of the design philosophies of Unix is "Everything is a file". While this isn't entirely true in Linux (network interfaces, for example) all other devices are represented by a device node in `/dev`, which might or might not be a virtual filesystem.

For example, the first serial port on a machine would be presented by `/dev/ttyS0` (equivalent to COM1 in DOS). An application that interacts with the serial port, such as **minicom**, a serial terminal emulator, will open `/dev/ttyS0`.

This allows access-control to be placed on such devices, in the same way as you would any other file-system object.

Device nodes are defined with a *major number* and *minor number*, and whether or not the device is a *block special file* or a *character special file*.

ls -l output that starts with a `b` is a block special file, a `c` indicates a character special file. A device such as a hard disk is a block device, most others are character devices.

The major number indicates the type of device, such a hard disk. The minor number indicates the particular device, such as a particular partition on the a particular hard disk. It is this `major:minor`, not the name, that tells the operating system kernel which device the user is opening, and thus which driver to pass the request to.

Socket

Unix systems have a form of network connection known as a *Unix domain socket*[1], which is parallel to an *IP socket*, but is entirely local to the machine. Unix domain sockets can do some things that can't be done over other socket types, such as passing open files and credentials, and because they use the filesystem, additional access control can be applied.

A server process would create a *socket file* as its socket address, and client processes on the same machine can connect to the server by using the socket file as the destination address (i.e. the server address).

We can know which file is a socket file with the following commands. In **ls -F** output, a = is printed after a socket file. In **ls -l** output, a `s` at the first character means a socket file.

There is no command to create a socket file. It is created automatically when a program binds a socket to the file with the system call bind(2) .

Named pipe* or *FIFO

Pipes are very useful in Unix. *Anonymous pipes* in particular form of the most important constructs for the Unix command-line environment. They allow the output of one process to be fed into the input of another, such as when we use the command-line construct such as `ls -l | sort -n +4`, which sorts the output of **ls -l**. Unlike sockets, pipes are unidirectional.

However, sometimes you want to be able to create something more flexible which would be impractical to express using the shell's pipe (`|`) operator. To do this, you can create

[1] Also known as *local sockets*

a *named pipe* somewhere in the filesystem, using **mkfifo**. In **ls -l** output, `p` at the start indicates a FIFO.

Named pipes are somewhat deprecated in favour of sockets. However, because they are more useful in shell scripts compared to sockets, they are still occassionally used, and don't appear to be in any danger of disappearing soon.

5.1.2. Hierarchy

The file hierarchy in Unix systems, including Linux, can be a confusing beast at times. This section will give you some familiarity with the purpose of various directories. Afterwards, you can have a brief read of hier(7), which should describe the hierarchy of the Unix filesystem on such a system.

`/`
The root of the entire filesystem.

`/bin`
Programs (binary files) that don't require administrative rights, and need to be available at boot time. Commands such as **ls** and **mkdir** can be found here.

`/sbin`
Supervisor programs (binary files) that do require administrative rights to make full use of them, and need to be available at boot time. Commands such as **mount** and network configuration commands such as **ifconfig** can be found here. Note that normal users can use these commands, but for querying only, they won't be able to use the command to make changes to the system.

`/lib`
Libraries that are needed when the system is booting (before `/usr` is mounted). It also includes kernel modules (device drivers and such).

`/usr`
This is for static (non-changing) data. Most of the software is installed here. A system should be able to run well if this directory (which is often on its own filesystem or mounted from across the network) is mounted read-only.

Historically, users home directories were stored in here, which may explain the peculiar name.

This contains `bin`, `sbin` and `lib` directories, in addition to the following:

`include`
Header files (`*.h`) that are included into programs used for compiling programs against libraries.

`local`
Has a structure much like `/usr`, but is for software that the system administrator has compiled and installed. This directory is outside the scope of the package management system.

`share`
Used for files that are shared amongst different system/processor architectures, such as documentation, pictures etc.

`doc`
Documentation for all the installed software. This does not include manual pages.

man
This is where the manual pages can be found, although you generally don't go there yourself, but use **man**.

src
Used to store software source code such as Linux kernel source code, though there is no requirement that source code by kept there.

/var
This is for variable (changing) data, such as databases, mail queues, logs, lock files and other things.

log
Log data produced by the system and its services.

pid
Process ID files so startup/shutdown scripts (*init scripts*) know which process to kill. PID files are generally written by *daemons* (background services) when they start.

mail
Location of e-mail queues and mailboxes.

/etc
This is where configuration files are stored. Historically, its where anything else in the system was stored, which explains the name.

/dev
This is where device special files are stored. It may be a virtual filesystem, meaning the contents might not exist on disk, but are determined by the operating system device drivers and hardware management facilities (eg. USB insertions).

/tmp
A temporary directory that anyone can write to. The contents may be purged occasionally, or on system boot. If you want a more permanent location, use `/var/tmp`.

/proc, **/sys**
These are virtual directories that give you information about processes and the system.

/root
On Linux systems, this is the home directory of the root user.

/home
On Linux systems, this is where the home directories for the users can usually be found. It is often network mounted and possibly a symbolic link to elsewhere in the system.

5.1.3. Self-assessment

1. Read the ls(1) to familiarise yourself with the available options (don't try to remember them all, except `-l` (that's lowercase L, not I), `-R`, `-a` and `-h`).

Write down the **ls** command you would use to do the following; most of them will require the `-l` option and generally some others:

1. List the contents of your home directory recursively. You can use either ~ or `$HOME` to refer to your home directory from the shell.

Note

There will be a lot of output generated. You can pipe it to **less**, which will allow you to page through the output using the **Space** key and the **Page Up** and **Page Down** keys. Type **q** to exit **less**.

```
command | less
```

less is a *pager* which supercedes an older program called **more**.

2. List all files, including hidden files and directories, in your home directory, non-recursively. Hidden files or directories are those that begin with a dot, and are not usually shown in **ls** output.

3. List the contents of `/lib` using **ls -l /lib**. What is meant by the `foo -> bar` notation?

4. List the files in your home directory, with file sizes shown with human-friendly units (ie. bytes, kilobytes, megabytes, gigabytes).

5. List the info about a directory, *and only that directory*, without **ls** recursing into that directory. In other words, the permissions etc. of the directory itself, not the contents of the directory.

   The "size" of a directory in the **ls** listing does not tell you how much disk space the contents of the directory consume. Use **du -s `directory`** to measure that.

5.2. Filesystem Permissions

Classical Unix filesystem permissions are one of the things that are often criticised as being too coarse and inflexible, and rightly so. Compared to the access-control model of other systems, such as Novell Netware™ and Microsoft Windows™, they are very coarse, and often you need something more fine-grained.

Thankfully, Linux, like any modern Unix-like operating system, generally comes with the ability to attach an *Access Control List* to files and directories. There are also other access control models available for Linux, although they are not ubiquitous, such as SELinux. We won't cover either of those in this paper because they are still not commonly used.

If, after reading this lab, you are still unsure, you might try looking at Linux File Permission Confusion [http://www.hackinglinuxexposed.com/articles/20030417.html] and Linux File Permission Confusion part 2 [http://www.hackinglinuxexposed.com/articles/20030424.html], which are both excellent articles by Brian Hatch, the author of *Hacking Linux Exposed*.

5.2.1. Basic Unix Permissions

In the common permissions model, there are three permissions we are concerned with; the three permissions are read, write and execute. The meaning of each differs depending on whether we are talking about a file or a directory.

If you have the read permission, you can open a file for reading, or get a directory listing. The write permission means we can open a file for writing, or change directory entries (create a new file or subdirectory; rename a file or subdirectory; or delete a file or subdirectory). The execute permission *on a file* means you can run the program; on script files, which are text

files, this causes the script to be made runnable just like a regular program. On a directory, execute means you can traverse it (such as when you **cd** into it or using **find**), but it doesn't say if you may read the directory contents; additionally, write access to a directory generally requires execute also.

Each set of permissions (read, write and execute) can be applied to three things: the user a file or directory belongs to; the Group a file or directory belongs to; and finally to everyone else (others).

Let's look at some **ls -l** output to refer to.

```
$ ls -l ~/.bashrc
-rw-r-----  1 mal mal 516 2006-03-24 13:53 /home/mal/.bashrc
```

The very first character in the permission set (the leftmost column), says what kind of filesystem object it is, which we talked about earlier in the lab.

The next three characters (`rw-` for the `~/.bashrc`) are the permissions which affect the User (the person that owns the file), which in this example is the user `mal` (the 3rd field). So the user `mal` can read and write to that file.

The three characters after that (`r--`) are for people that are in the same group as the file (the group name is displayed in the 4th field of the above **ls -l** output). These people may only read the file. It is common practice on Linux systems for users to have their own group for local accounts. For your accounts, it will likely differ if they are not local accounts. In this example, the group is `mal`

The last three affect everyone else (Other people).

The order of checking access permissions is first User, then Group, and finally Other. Only one set of permission bits is queried once matched. For example, if you are matched by the Group, but its permission bits deny the group access, the Other bits would *not* be checked, even though they may allow the access.

Look at the below **ls** output[2]. What permissions will the user `jill` be granted on this directory? Assume that `jill` is not in the group `staff`.

```
$ ls -ld /home/cosc301/teaching
drwxrwx---  3 bob staff 4096 2007-01-30 16:23 /home/cosc301/teaching
```

Because the user `jill` is not the user `bob`, who owns this directory, nor is she a member of the `staff` group, so she only gets the permissions granted to the others, which have no permissions.

Now assume that the user `alice`, who *is* a member of `staff`, tries to list the contents of the directory, which requires a read priviledge. `alice` is not the user `bob`, so is not matched against the User permissions; but `alice` is a member of `staff`, so gets the permissions `rwx`, which grants read, write and execute. This means that listing the directory will work.

Changing Ownership and Permissions

The two commands for changing the owner and group of a file or directory are **chown** and **chgrp**. **chown** can be used to change the group at the same time. Both can be used to apply the changes you specify in a recursive manner.

[2]This has been slightly modified from the true permission set on this particular directory.

chown may only be used by the root user. The user invoking **chgrp** must belong to the specified group and be the owner of the file, or be the super-user (root).

Have a quick look at the manual pages for **chown** and **chgrp**. Take notice of how you can specify both user and group with **chown** by using the `user:group` syntax.

When we come to changing the permission (or *mode*) of a file or directory, there are two syntaxes you can use, *symbolic notation* and *octal notation*.

Symbolic notation is useful when you only need to change some of the permissions. Recall that we can apply permissions to the User, Group, and Others. Recall also that we have three permissions: read (`r`), write (`w`) and execute (`x`). We can grant these permissions in an absolute or relative manner, or we can revoke these permissions. We seperate the User, Group and Other sections with commas. The syntax is most easily learned with examples as below.

Note

In the following examples, `u` represents User, `g` for Group, and `o` for Other. Don't get confused and think that `o` stands for owner!

```
This command will create an empty file if it doesn't exist.
$ touch myfile
$ ls -l myfile
-rw-r--r--  1 mal mal 0 2007-02-17 20:44 myfile

Turn on the execute bit for the owning user.
$ chmod u+x myfile
$ ls -l myfile
-rwxr--r--  1 mal mal 0 2007-02-17 20:44 myfile

Remove read bit for all.
$ chmod -r myfile
$ ls -l myfile
--wx------  1 mal mal 0 2007-02-17 20:44 myfile

Grant read to User and Group.
$ chmod ug+r myfile
$ ls -l myfile
-rwxr-----  1 mal mal 0 2007-02-17 20:44 myfile

Set multiple parts.
$ chmod u=rw,g=rw,o=r myfile
$ ls -l myfile
-rw-rw-r--  1 mal mal 0 2007-02-17 20:44 myfile
```

On the other hand, if the octal notation is used, you cannot grant or revoke individual permissions. The notation is octal because each permission set for a User, Group or Other takes three bits, and so has a digit between 0 and 7 inclusive. Therefore, we need three octal digits to specify User, Group and Other.

For each digit, read is worth 4, write is worth 2 and execute is worth 1. For example, if we want to specify read and write, but not execute, that is 4+2=6.

```
$ ls -l myfile
-rw-rw-r--  1 mal mal 0 2007-02-17 20:44 myfile
The above file has octal permissions 664.

$ chmod 755 myfile
$ ls -l myfile
-rwxr-xr-x  1 mal mal 0 2007-02-17 20:44 myfile
```

The install command

If you want a command that combines the features of **cp**, **chown**, **chgrp**, **chmod** and **mkdir**, then you might like to look at the **install** command. You may find more details about **install** using **man 1 install**.

Initial Permissions (umask)

When a file is created, the permissions that it gets are 666 (for files), or 777 (for directories), minus[3] the **umask** value. This **umask** value turns off various bits in the permission set in order to make the default permissions for new files safe. **umask** settings are inherited in the process hierarchy. The value is generally octal 022, but can be changed in the shell using the **umask** command. With the **umask** value 022, files created would have permissions `rw-r--r--` (octal 644) by default, and directories would be `rwxr-xr-x` (octal 755).

5.2.2. Special Permissions

In addition to the standard permissions, there are the extended permissions. These are special bits that you as an administrator need to be aware of.

The special bits take up a forth octal digit (the most significant digit), so when you see an octal permission such as 2775, the special bits are contained in the digit 2.

Set User ID bit (SetUID)

On an executable *binary file*, this lets the program run as the person who *owns* the file, often root. In contrast, in a normal situation where the SetUID bit is not set, if *you* run **ls** that is owned by root, it would run with *your* permissions.

Note that using the SetUID bit can be dangerous, and must be carefully managed, since it is often used on root-owned system programs. Also the SetUID bit won't work for scripts on many Unix-like systems for security reasons.

If the SetUID bit is used on a root-owned system program, the program has the root privilege even if it is run by a normal user. Therefore, the program should drop its root privilege as early as it can, as long as the access that requires root privilege has been done. One such program is **ping**. It uses ICMP which requires root access. Another example is **passwd**, which needs to be able to modify files normal users cannot write to. Once those files are opened, the root privilege should be dropped in the program. This is an example where the *Least Privilege Principle* is applicable.

On a directory, SetUID has no effect. The SetUID bit has a value of 4. The bit can be turned on using **chmod u+s**. In **ls -l** output, a file with the SetUID bit will appear as follows:

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x  1 root root 37140 2010-01-27 06:09 /usr/bin/passwd
```

Here, we can tell that SetUID is on because of the `s` in the User execute position. Because the `s` is lower-case, we know that there is a `x` underneath it also, so the User execute bit is also set. However, if the upper-case `S` is shown instead, it means the User execute bit is not set.

[3] Actually, the Boolean bitwise operation AND and NOT are used, as in the formula mode = 666 AND NOT(umask) for files.

Set Group ID bit (SetGID)

Much like SetUID on files, SetGID files run with the priviledge of the *group* that owns the file.

This is often used for games, that need protected access to a shared scoreboard, without giving the user write access to the scoreboard.

On the other hand, if an entry is added to a directory with the SetGID bit set, a file becomes owned by the group that owns the directory when the file is created. In contrast, in a normal situation where the SetGID bit is not set on the directory, when a new file is created, the group of the file will be the *primary group* of the user who has put it there.

SetGID, along with **umask**, is useful for groupwork, or when you need to share responsibility between multiple people (such as a staff group). You can use **umask** to allow group read/write of files, while making all files created under the shared directory automatically belong to the group of the directory by setting the SetGID bit. In this way, all files created under the shared directory are readable and writable by anyone belonging to the group of the directory.

The SetGID bit has a value of 2. It can be set using **chmod g+s**. In **ls -l** output, it appears as follows:

```
$ ls -ld /var/mail
drwxrwsr-x 2 root mail 4096 2009-10-29 09:55 /var/mail
```

This time the `s` is in the Group execute position; so SetGID is set. The `s` is lower-case, so the Group execute bit is also set. Likewise, the upper-case `S` means the Group execute bit is not set.

Sticky bit

On files this has no effect under Linux[4]. But when set on a directory, it is very special. It means that only the user who created an item inside a sticky directory may delete it. This is the behaviour of the `/tmp` directory.

The `/tmp` directory has an octal mode of 1777. This means that anyone can add or delete entries into the `/tmp` directory. If the sticky bit was not set (ie, the overall mode would be 0777), then the user `bob` could delete the user `alice`'s files in `/tmp`. But with the sticky bit, this could not happen.

The sticky bit has a value of 1. It can be set using **chmod +t**. In **ls -l** output, it appears as follows:

```
$ ls -ld /tmp
drwxrwxrwt  5 root root 1160 2007-02-18 00:36 /tmp
```

The `t` is shown in the Other execute position. The `t` is lower-case, so the Other execute bit is also set. Likewise, the upper-case `T` means the Other execute bit is not set.

5.2.3. Self-assessment

1. Fill in the blanks to convert between octal and symbolic notation; all but the last two you would be likely to find on a real system.

[4]Originally on UNIX systems, it told the operating system to keep a file loaded in memory. Today it is not needed due to improvements in operating system design.

Octal	Symbolic
644	
	rwxr-xr-x
1777	
	rwsr-xr-x
3070	
	---rwS---

2. If you look around the typical Unix filesystem long enough, you find some unusual permissions. Take for example the permissions that are classically set on the *files* in `/usr/games/`, and the permissions on some of the *files* in `/var/games/`.

```
$ ls -l /usr/games/
total 2772
…
-rwxr-xr-x 1 root root    2461 2009-09-22 21:23 glchess   SetGID is not set
-r-xr-sr-x 1 root games 142716 2009-09-22 21:25 glines    SetGID is set
…
$ ls -l /var/games/
total 0
            glchess has no shared scoreboard files
-rw-rw-r-- 1 root games 0 2009-10-29 10:02 glines.Large.scores
-rw-rw-r-- 1 root games 0 2009-10-29 10:02 glines.Medium.scores
-rw-rw-r-- 1 root games 0 2009-10-29 10:02 glines.Small.scores
…
```

The files in `/var/games/` are scoreboard files, which need to be updateable by the games when run as different users. However, we don't want users to be able to edit them by themselves, only by the games that the user is running. Looking at these permissions, describe how this works. Hint: users are not meant to be in the 'games' group in order to play games; indeed that would be rather bad.

3. Assume you see a directory which **ls -l** describes as `drwxrwsr-x`. Write a paragraph to completely describe the effect of this permission set.

You should consult ls(1) for some of the subtleties of the printed permissions with regard to special bits and execute bits, if you haven't already.

4. Give a typical default umask value and describe its effect.

5.3. Archival and Backup

Archival and Backup are two distinct activities, though the terms are used somewhat interchangably. Archival is long-term, often indefinite, while backups are often rotated, only keeping material for a certain window of time, while archiving information is not intended to ever be deleted. Backups are generally done according to a schedule, such as a daily, while archival is often done on an as-needed basis, such as when creating a software release (eg. `foo-1.2.3.tar.gz` to upload to a website) or when you want to move your precious digital photo collection from your hard-disk onto DVD.

The media and method you use will depend on whether you are archiving or backing up. Backups should ideally be able to be done unattended to a remote location, so backing up over the network to a cheap but reliable hard disk is a useful way of handling this.

When archiving, removable media is generally the best way to go, as it scales better. However, any media has aging issues, and this can be prevalent for a number of optical media, such as recordable CDs and DVDs, so you need to occasionally check the integrity of your archives to ensure the data is still intact, and pay attention to things like the brand and type of disc, and how they are stored.

Another option is to rely on a cloud provider. There are various options depending on the scale of the backup/archival needs. Services such as Google Drive, Apple iCloud, Microsoft OneDrive are useful for an individual's documents. Some offerings, Google Suite (their Docs, Sheets, Drive, Gmail etc. for business) allow you to manage a small business and provide backup/archival solutions. Others (Google Cloud Storage, Amazon Glacier) allow you to store vast amounts of data very cheaply but cost more to retrieve[5].

USB Flash drives, DVD-ROM, USB HDDs or the cloud?

It's hard to have long-term confidence in DVDs, particularly those bought in big spindles, given the move towards removing of optical media drives from computers. USB flash drives are interesting devices; they do have a significant lifetime issue though, but only in terms of how often you write to them.

USB flash drives (unlike SD cards you find in most digital cameras) don't have a physical write-protection switch, which creates an issue regarding accidental deletion and viruses.

USB hard drives are an option, and behave like a flash drive but have the benefits and drawbacks of traditional spinning hard drives.

Recently developments in online infrastructure allow you to put a (relatively) large amount of data online for free. Services such as Google Drive, Dropbox and iCloud all offer storage of personal files, Amazon Glacier is more catered towards storage of large files (and a slow retrieval time). However, there are jurisdiction, security, and privacy issues with this approach, especially when medical records are concerned.

It may be better to "hedge your bets" and put your precious things in as many places as possible; then you can store at least one copy off-site (perhaps mailing a flash drive to a family member for safe keeping--and is often the only practical way to deal with large data sets).

Archiving and backup are otherwise technically very similar, and many tools can be used for both activities. To illustrate that point, any single archive should be usable by itself, whereas a backup *might* record changes only from a previous backup on which it becomes dependent.

Note

For the rest of the section, the terms "archive" and "backup" can be treated as synonymous. Just beware that there is a distinction, but that it only really matters when considering issues such as media, storage, etc.

In the Unix world, backup is often done using the **tar** command to create the archive/backup file, which is then burned to disc or stored elsewhere. **tar** was originally meant for use with

[5] Under an old pricing scheme, an Amazon Glacier user was surprised by the bill [https://medium.com/@karppinen/how-i-ended-up-paying-150-for-a-single-60gb-download-from-amazon-glacier-6cb77b288c3e]! Do note the pricing scheme has been changed since. Moral of the story is to plan ahead!

magnetic tapes as a destination medium. Tapes are still used today, but are quite expensive compared to hard disks.

5.3.1. Using tar

tar is one of those really old Unix programs, and is still quite useful for its task. The purpose of **tar** is to take a directory or a list of files, and package it as a single file. It is then usually passed to a compressor, commonly **gzip** or **bzip2**. The process is reversed to get the files out. If you have used other compression programs, such as **zip** or **jar**, please realise that those tools perform *both* the archiving and compression.

tar has three basic modes: create, list, and extract. **tar** archives, sometimes called *tarballs*[6], often have a `.tar.gz` or `.tgz` extension, which means they've also been compressed with **gzip**. There are also **bzip2** compressed files, which are slightly smaller, but take longer to process. **bzip2** compressed files have a **.tar.bz2**, or **.tbz2** extension.

To create an archive, we use the `c` option. You can specify an output file using the `f` option. `v` is for verbose output[7]. `z` passes the output (or input) through **gzip** (`j` for **bzip2** instead) to (de)compress the data.

```
This is generally a bad way of doing it, we'll see why shortly
# tar -zcvf /tmp/logs-bad.tgz /var/log
tar: Removing leading `/' from member names
/var/log/
/var/log/syslog
/var/log/apt/
/var/log/apt/history.log
/var/log/apt/term.log
/var/log/Xorg.0.log
…

Instead, cd to where you want to go first
then operate on the current directory (.)
$ cd /var/log
# tar -zcvf /tmp/logs-good.tgz .
./
./syslog
./apt/
./apt/history.log
./apt/term.log
./Xorg.0.log
…
```

We can list the contents of an tar archive using the `t` option. Note that the `v` option when listing will give output similar to **ls -l**, whereas without `v`, the output will be like plain **ls**. Additionally the command **head -5** only shows the first five lines of the output.

```
This one is listing the archive we made using the "bad" method.
Notice the path var/log contained inside the archive.
$ tar -ztf /tmp/logs-bad.tgz | head -5
var/log/
var/log/syslog
var/log/apt/
var/log/apt/history.log
var/log/apt/term.log
```

[6] To the disgust of many people.
[7] Not verify, as some mistakenly believe.

```
In the “good” method, we don’t have that problem.
We could unpack it anywhere and not get var/log
$ tar -ztf /tmp/logs-good.tgz | head -5
./
./syslog
./apt/
./apt/history.log
./apt/term.log

This shows the verbose output.
$ tar -ztvf /tmp/logs-good.tgz | head -5
drwxr-xr-x root/root         0 2010-11-19 09:31 ./
-rw-r----- syslog/adm     4139 2010-11-23 13:30 ./syslog
drwxr-xr-x root/root         0 2010-11-11 13:16 ./apt/
-rw-r--r-- root/root     55950 2010-11-11 16:21 ./apt/history.log
-rw------- root/root     94518 2010-11-11 16:21 ./apt/term.log
```

Finally, we can extract the contents using the x option. Note that if you unpack the archive as root, the Owner and Group will be set to those in the archive; however, if you unpack the archive as a normal user, they would end up with all files and directories belonging to you and your group. If you have to unpack source as root, you might like to use **chown -R root:root** to reset the User and Group on all files to the root user; otherwise, you might end up giving write permissions of some important files to some other user unintentionally, which could be disastrous. Another thing to be wary of is the permissions you give to the archive. You don’t want to allow normal users to read a backup of everyone’s home directories.

```
Make a place to extract to…
$ mkdir /tmp/restore
$ cd /tmp/restore

Time to illustrate why the “bad” method is annoying…
$ tar -zxvf /tmp/logs-bad.tgz
var/log/
var/log/syslog
var/log/apt/
var/log/apt/history.log
var/log/apt/term.log
var/log/Xorg.0.log
…

…what did we get?
$ ls -R .
.:
var

./var:
log

./var/log:
apparmor      dmesg           kern.log.1         syslog
apt           dmesg.0         lastlog            syslog.1
aptitude      dmesg.1.gz      lpr.log            udev

…bother! Everything is in var/log/FOO
and I wanted just ./FOO. Let’s start again…

$ cd ..
$ rm -rf restore
$ mkdir restore
$ cd restore

…this time using using the “good” archive.
$ tar -zxvf /tmp/logs-good.tgz
./
./syslog
```

```
./apt/
./apt/history.log
./apt/term.log
./Xorg.0.log
…
$ ls -R .
.:
apparmor      dmesg           kern.log.1         syslog
apt           dmesg.0         lastlog            syslog.1
aptitude      dmesg.1.gz      lpr.log            udev
auth.log      dmesg.2.gz      lpr.log.1          ufw.log

Great, that's more like it!
```

If we only want to extract certain members of an archive, such as a partial restore, you can list those members (*precisely* as they appear in the listing output), at the end of the **tar** command. Here's an example extracting just the `./apt/` directory:

```
$ cd /tmp
$ rm -rf restore
$ mkdir restore
$ cd restore
$ tar -zxvf /tmp/logs-good.tgz ./apt/
./apt/
./apt/history.log
./apt/term.log
```

5.3.2. Investigating Backups

Let us first consider some of the things we would like to have in a modern backup system.

- Remote, to protect against localised disaster, such as flooding. It should at least be in a different building. The amount of data would suggest somewhere else on the LAN. Off-site is better.

- Cheap to implement. Largest consideration here is media, drives, software, and network performance and possibly charges. Hard disks are cheap, large, and fairly reliable. Good for backups, and any host with sufficient space and able to sustain a high amount of traffic at off-peak times should suffice.

- Plentiful snapshots. You want to be able to restore at any point in time, and be able to do so with minimal loss and fuss. Storing snapshots can be done in a variety of ways, some of which are quite cheap. It can even be cheap enough to provide a snapshot every 30 minutes.

- Easy to restore. For easy, you can also read *fast*. This can also mean that it is easy to do a partial restore, often just a single file. Ideally, the user might be able to perform this operation themselves.

- Secure. The transmission must be secure, as well as how it is stored.

- Maintain file meta-data, such as access control lists. On Mac OS X for example, you can set a particular colour for a file or directory, and enter comments. It would be good if we don't have to backup to the same type of operating system or file system as the one we are backing from.

- Selection of items to include/exclude. There are a *lot* of things that don't really need backed up. But on the other hand, it can be feasible to include everything up in case you forget something important.

- Suitable for very large files, such as virtual machine disk images: you wouldn't want a small change in one area of the file to cause the entire very large file to be backed up again.

A system administrator, perhaps as part of a wider site-policy, might often use a commercial piece of software for peace of mind.

They might instead write a fairly complex shell-script which determines a list of files that need backed up, which is then fed into **tar**, as **tar** is more appropriate as a backup engine, not a "differencing" engine.

rsync, a file synchronisation tool, has some very useful functionality for doing backups, but it also has some issues that make it incomplete as a backup tool. It tries to transmit only what has changed, but it isn't really designed for backups. Other tools can use **rsync**, or the underlying mechanisms, as part of a backup solution. One such product is called **rdiff-backup**.

We won't be using **rdiff-backup** today, but you will be finding out about its feature set. You need to install it with **sudo apt install rdiff-backup**, if you want to try it with the examples below.

5.3.3. Self-assessment

1. Create a backup of Mal's home directory (`/home/mal`), storing it in `/tmp/mal-backup.tar.gz`. Set the permissions such that only Mal can read it. Ensure that it does not have `home/mal/…` at the start of each entry in the archive, as done in the previous "bad" method.

2. The above command likely has a security problem, in that the archive does not have a suitable mode when it is being created. This creates a window of opportunity for someone to open the file and read the contents before **tar** has finished and you have fixed the permissions.

How can you fix this problem? (Hint: use **umask**).

3. Part of a diet of the system administrator is to look at various products and evaluate their usefulness to solve particular situations. Have a look at the homepage of rdiff-backup [http://www.nongnu.org/rdiff-backup/]. Compare the features of rdiff-backup to the features we desire, listed above, marking each with a tick, cross or question mark, for "meets requirement", "does not meet requirement", or "unsure" respectively. Do you think rdiff-backup meets our stated needs?

5.3.4. Rdiff-backup Example

rdiff-backup is quite useful for most purposes (not for all though, for example it doesn't handle sparse files well), which generally includes sending them to a remote system over SSH every night. So here is one example that shows a number of the features of **rdiff-backup**. You WILL want to edit this to suit other systems.

```
#!/bin/bash
#
# Backup using rdiff-backup.
#

rdiff-backup \
```

```
  --exclude-if-present NOT_BACKED_UP \
  --include /home \
  --include /etc \
  --include /var/mail \
  --include /var/www \
  --include /var/log \
  --include /usr/local \
  --include /usr/lib/cgi-bin \
  --include /var/lib/dpkg \
  --exclude '**' \
  --remote-schema \
     '/usr/bin/ssh -o BatchMode=yes -C -i/path/to/.ssh/backup %s' \
  / \
  remote_user@remote_host::remote_path
```

Let us walk through this script briefly. First, to help make it clear to users, and to enable users to say what *doesn't* get backed up, we ignore any directory that has a file called `NOT_BACKED_UP` inside it. This also enables greater maintainability of this script (We could, for example, use it a template for other systems).

In this particular example, because all the software is stock Debian packages, we only want to include those parts that can't easily be reinstalled, plus any configuration files, plus any other data that might need to be backed up (We sure hope we haven't missed anything; it's generally much easier just to backup everything). Because this is a Debian system, we have backed up the directory that contains information about all the installed packages (`/var/lib/dpkg/`) which will enable us to reinstall without too much fuss.

By default, we exclude anything else that doesn't match our previous include or exclude directives; see the manual for what ** means.

Because we are backing up over SSH (which we shall learn about much later in this course), we have changed the template (schema) that **rdiff-backup** uses to run the **ssh** command. We have enabled batch operation (don't bother every trying to ask for a password), enabled compression, and pointed to a particular private key we want to use just for backing up.

We then say where we want to start our backup from (in this case, the root directory `/`, and where we want the backup to be stored.

There is a little more to this, to do with SSH public-key authentication and saying which command gets run on the server. This is configured in the remote user's `~/.ssh/authorized_keys` and looks something like the following. Don't worry if you don't understand it, as you'll learn about SSH in a later lab, and we won't be doing any assessment in this section.

```
command="/path/to/rdiff-backup-1.1.14 --server",from="client-host.domain" public key
```

Lab 6 Catchup Lab

This lab is set aside for help people catch up; priority will be given to students doing previous labs and assignments.

Lab 7 System Installation and Basic Administration

Ubuntu

Ubuntu (pronouced “oo-BOON-too [https://www.ubuntu.com/about/about-ubuntu]”, not “oo-BUN-too”) is a South African ethical ideology focusing on people’s allegiances and relations with each other.

Many of you might have installed an operating system before, probably Windows, possibly Mac OS X or Linux, or possibly even dabbled with other operating systems. The desktop versions of Windows and Mac OS X, which are aimed at the mass market, is designed to be very easy for the user, with minimal choice. Linux systems have been moving in that direction for a long time now, and are now almost as easy, but because Linux caters for a more technical audience, there is still plenty of options to choose from during installation, all the way from “easy and quite painless” up to “frustrating and error-prone”, depending on the distribution you wish to install. Ubuntu and Redhat are both at the “easy and quite painless” end of the spectrum.

Installation of these “easy and painless” systems is comparatively fairly boring, but none-the-less important, and so we have included reading material in today's lab aimed at getting you to think about operating system installation differently (eg. how might you install and maintain a computer laboratory?), as well as trying to get you thinking about how one installation might vary from another (eg. workstation versus database server versus web server).

In brief, today we shall be installing our server, which will be used for the rest of the paper. Here is an overview of what you will need to accomplish today:

1. You will need to create suitable documentation for what you are doing today, such that another person, with the same training as yourself, can follow your instructions and arrive at the same result. **This is the prime self-assessment for this lab.**
2. Create a virtual machine in VirtualBox.
3. We shall install “Ubuntu 18.04 LTS Server Edition” into our machine. This will be a command-line only environment, for reasons discussed later. We will aim at a suitable “first” system, where we don’t have much of an idea of exactly how we should approach some tasks, e.g. how to partition disk storage.
4. We shall ensure all system updates come from an appropriate source and are applied to our server.
5. We shall install the VirtualBox Guest Additions, and make use of the “Shared Folders” feature.
6. We shall remove some of our ignorance that we had when installing our system for the first time, and figure out just how large different parts of the filesystem are.
7. As time allows, you will have a look at some of the extra reading material which covers important background material concerning storage, security issues during installation and different options for installing systems.

   To cater for students at different levels of experience, and to help you manage your time and workload more effectively, some material is marked as optional.

8. In the next lab, which deals with post-installation, we will apply further configurations to introduce it into its new network setting and start performing just some of the many things we would generally do after having just installed a server.

9. From this lab onwards, you should start your second assignment based on what you learnt from the labs.

7.1. Thinking about your Documentation

It is recommended you read this section before coming to do the lab.

The largest assessable, though not the largest deliverable, in this laboratory will be your documentation. You need to create sufficient documentation such that one of your class-mates could follow your instructions and arrive at the same result. It is also very important for you to repeat them while doing your assignments.

First, a few ideas about documentation:

- You don't need to include default options, except perhaps where they are significant. In real-life, some defaults change between operating system releases, and so detailing important default options can be important.

- Documentation becomes faulty if it is not maintained, and thus a liability. Therefore, documentation needs to be kept up-to-date with a list of revisions. A list of dated revisions is important for quickly determining what change might have caused a problem. In larger environments, "Configuration Management" and "Change Management" procedures are formal procedures for ensuring that you can easily back out of a change if it causes problems, minimising downtime. Missing documentation is even more of a liability, particularly in complex systems: it makes it hard to determine what services another service might be dependent on, and thus the effect of a service failing.

- What does this system do? What has it done in the past? When reading someone elses documentation, it is very useful to appreciate the task that the system was installed for. A lot of servers have had a long life and may have accumulated a lot of cruft over years of service. Knowing what a server does, why a service is needed, how important it is, and the key configuration of it is important information for a team-member to put their hands on.

- Of course, putting your hands on such documentation indicates that such documentation should be easy to find. However, in many environments you find very disorganised standards regarding where documentation should be stored, how it should be edited, etc. One person might use a Wiki, one might use a paper exercise book, one might use Google Docs and another might use a Microsoft Word document. This is not helpful if you are a backup systems engineer.

  Given the amount of work that is done on servers remotely, a paper exercise book can be too easy to forget to update. Similarly, don't store your documentation for a server on the same server, lest you can't get to it when you need it. Collaborative editing can occassionally be useful, particularly as it results in fewer revisions of the document floating around at one time. Being able to access the document is obviously very important, and so if your documentation is kept "in the cloud" (such as on Google Docs), then it would also be useful to occassionally keep the most recent version printed, with its attendent synchronisation problems.

So, what should documentation contain? To illustrate, here is a (slightly edited) table of contents of some documentation we have maintained for one of our servers. Note that each

server would be different, so take this simply as a guide. You will not yet understand all that presented here; that's okay, we've put a † next to those items that you aim to complete for this lab.

```
Management of the server 'NAME'
  Administrator †          Who is the administrator, and their contact details.
  Release Information †    What OS and version is this machine currently running?
  Recent Changes †         On every change, this gets updated manually
    5 November 2017
    4 November 2017
    …
  History                   The systems history and artefacts
  Hardware †                CPU, Memory, Location, Physical Access
  General Management
    Filesystems †           How are the disks partitioned?
    Network Interfaces †    How is the system connected?
      inside                IP address…, Connected to…
      outside               If this host has two interfaces
    Administration Rights † How to get them
    Authentication          How is user-authentication managed?
      Password Requirements These are additional to the system default behaviour
      Password Expiration   These are additional to the system default behaviour
    Software Upgrades †     Where does the software come from. How are updates made?
    Time Synchronisation    Is the time synchronised with a time server?
    Cron                    What periodic jobs are run on this host?
      backup                Every host should have something similar
      report-users-old      These are particular to this host…
      …
    Firewall                Where is it defined, brief description
    TCPWrappers             Useful for limiting access to network services
    Log maintenance and monitoring
    Backup (client)         This is IMPORTANT
      Bare-Metal Restoration Preparation
                            DOCUMENT how to restore onto an empty disk
      Restoration Procedure [Last tested: 12 October 2014]
                            And TEST that it works, periodically
  Role Management           What services does this server house?
  Web Server
    How important is it?
    Who should have access?
    What is its "normal" behaviour?
    What are its major configuration changes?
    Any particular policies that need to be catered to?
  Local E-mail Server
    Another fairly standard service…
    …but perhaps you have some added monitoring which
    you should document.
  SSH Server
    Another very standard service…
    …but perhaps with some added service-specific notes.
  GIT Repositories for Research Students (via SSH)
    A lot of servers do fairly unique things, so be sure to document such things well,
    particularly with regard to any management tools that might be developed.
```

You may create your documentation anywhere you wish, so long as it is *not* stored on the server itself. You do not have to follow the outline above, but you do need to at least include the material marked with a †.

7.2. Selecting an Operating System

In the next section, we're going to begin preparing the virtual machine into which we shall install our operating system. Before we go on, we should first consider the differences

between different releases of operating systems, such as server-class or desktop-class. What are the different priorities of each?

Here are some thoughts to get you thinking. Many operating systems, particularly those containing open-source desktop software, have a developmental branch that changes fairly quickly, and also a more stable, slower-changing branch. If we take Ubuntu as an example we see that Ubuntu offers Long Term Support [http://www.ubuntu.com/products/whatisubuntu/serveredition/benefits/lifecycle] (LTS) releases, which are supported for three years for the desktop version, and five years for the server product. Non-LTS releases are only supported for six months. Considering that core server daemons don't change much over time (compared to desktop software), and servers don't need, and often don't run, desktop environments, how might this affect your choice?

Microsoft Windows and Mac OS X have a similar thing; whereby there are distinct versions released for servers and clients, which are optimised differently, have a different feature set, licencing, support cycle, etc. For client and server devices the common wisdom is to avoid a new major release until at least the first service pack (for Windows) or the second update (ie. 10.x.2) release for Mac OS X. Paying attention to the experiences of other community users, as well as having a testing laborary to try for yourself, are valuable steps to smoothly integrating a new release into your production environment.

It is useful to appreciate that there are differences between client operating systems and server operating systems. For example, you wouldn't use Windows 7 Starter/Home Basic/Home Premium as a server, because it has a number of limitations, such as how many connections it will accept, and what services it may provide.[1] Table 7.1, "Client and Server Operating Systems" lists some differences between client and servers. *You should complete what you can of this table* while you work on the rest of this lab. You will have plenty of waiting time once the installation process begins, and as you install an operating system, you may get inspired about some of these differences.

Table 7.1. Comparison of Client and Server Operating Systems

Area	Client	Server
Process Scheduling	Optimised for interactivity and foreground processes. Often real-time requirements for multimedia. Timeslices might be small and often pre-emptive to allow real-time tasks to get CPU time immediately as they need it.	Optimised for background processes, I/O performance for storage and network. Timeslices are often larger, sacrificing a small amount of interactivity, which is negated by network latency, for better memory/cache performance; being able to do more processing in one timeslice[a].
Hardware Class	Generally aimed at devices spanning Workstation class machines to Desktops to Notebooks and now even Netbooks (although Netbooks are generally quite limited and may need a cut-down version of an operating	Generally aimed at server-class devices, which may include Workstations all the way up to very powerful dedicated servers, but now also includes virtualised environments. Graphics may be minimal, if present

[1] Wikipedia has a nice feature comparison chart for Windows editions.

Area	Client	Server
	system or special installation mechanisms).	at all. May have more capabilities with regard to how storage and connectivity are managed. Generally quite “hands-off”, an operating system may even be installed remotely on some servers, which is very good for remote management.
Connectivity	Often equipped with enhanced connectivity using mobile technologies such as Wifi and Bluetooth.	Focus on scalable, redundant and typically wired connectivity to a fixed network.
CPU and Memory		Will generally have greater multiprocessing capability compared to client systems (although workstation-class machines can be very powerful too). Servers and Workstations often benefit from 64-bit processors. Memory speed and capacity is often critical to scalability.
Access		System either accessed remotely over network, or locally. Local access is often reserved for when access via network is inconvenient or not available. System will often be housed in secure storage, and in many cases, may be stored in a remote data-centre so as to be closer to where its data is needed. This has implications for management functionality when the system is otherwise unreachable. Eg. can you reboot the server remotely, or access the “local” console remotely? Generally this is a feature more associated with the server host rather than the server operating system.
Feature Set	Often graphically rich, optimised for user experience and productivity. With regard to network services, will generally	Minimal graphics, if any. Aims to support many different services to clients, though only those that are needed should be installed.

Area	Client	Server
	only have client-software installed.	
Licences		For Open-Source operating systems, this is generally very simple. For commercial offerings, it can vary quite a bit. May depend on some notion of number of users, number of CPUs, number of servers. Eg. Apple, starting with Mac OS X 10.5 Leopard Server, changed their licencing to allow it to be installed into a virtual machine (but only the server edition, the client must be installed on Apple hardware).
Management		In an enterprise environment, which often have many servers, they might often be managed using technologies such as Microsoft's Active Directory (if using Windows), or any other technology that allows for a "hands-off" approach (one example for Linux: cfengine). Smaller servers are often managed "hands-on" over a network, either using SSH or a screen-sharing solution such as RDP or VNC.
Security	Aims to provide for security without inconveniencing the user too much.	

[a]To give you an idea of a timeslice, a server might switch between tasks 100 times per second, while a desktop might be 250 or even 1000 Hz.

7.3. Adding Server1 to VirtualBox

In this section, we shall create a new machine in VirtualBox.

For the rest of this section, please pay close attention, as it can be easy to skip along and miss out an important task.

Procedure 7.1. Creating the Virtual Machine "cosc301-server1"

1. In VirtualBox, click on New to start creating our new virtual machine.

   You should now see the "Create New Virtual Machine" wizard. Click Continue

2. Give the virtual machine the name “cosc301-server1”, which is what VirtualBox will call it. Giving our virtual machines a common prefix can be useful when sorting them (consider the case if you have other Virtual Machines for other papers).

   Specify the operating system as Linux, with the version being “Ubuntu (64-bit)”. This sets some default values appropriately for the rest of the wizard, such as the amount of memory and an appropriate network card.

   Click on Continue

3. Choose the default value 1024MB as the amount of memory. Remember, defaults values are only suggestions. It's not uncommon to come across machines with a decent amount of memory. The machines in the lab have 8GB. Plenty of space to give our virtual machine 1024MB. Keep in mind though, there may be minimum requirements for the operating system and any applications you may be running. We are only going to be using a server (and so no GUI) and no applications that require large amounts of memory. So we'll go with the default and just click Continue.

4. When configuring the virtual "Hard Disk”, make sure Create a virtual hard disk now is selected, as we do want to create a new hard disk for the virtual machine. These should be the defaults, but at the moment we just want to be very careful. Click Create, and then click Continue with the VDI disk.

5. When you get to Storage on physical hard disk, select Dynamically allocated, which should be the default anyway, and click Continue, and then click Create.

6. When you get to “File location and size”, just click "Create" and you have now created a VM for the server.

We have now created and configured the virtual hardware for our server. But we still need to check some of the settings for the new virtual machine. In the VirtualBox window, click on the machine called “cosc301-server1”, and then click on Settings to access the settings dialog.

In order to install the operating system, we need to put the “virtual” CD-ROM (which is represented as an “ISO” disc image), into the virtual CD-ROM drive. Click on the Storage icon of the Settings window. Click on the “Empty” slot under the “Controller: IDE”. In the CD/DVD icon, you *would normally* find out the ISO images that you have used. But since you havn’t used it yet, so VirtualBox doesn’t have that information available for you. Instead, click on Choose Virtual Optical Disk File... from the drop-down menu of the CD/DVD icon. In the following window, navigate to the “resources” folder, to the folder `ISOs/Ubuntu`, and to the file `ubuntu-18.04-server-amd64.iso`. Click on Open. Then click on OK to install the disk of the new ISO file.

You are now back in the Settings window for cosc301-server1. Click on Audio and *un-tick* the Enable Audio. This is because a server doesn’t need audio, and it introduces a bit of complexity that could otherwise cause a problem, such as the virtual machine crashing due to some audio-related bug.

Click on Display and choose VBoxVGA as the Graphics Controller. Ignore the “Invalid settings detected” message. This setting can mute an error message when the virtual machine starts.

There are some other configurations we will do, but we shall leave those until the relavent sections, to better explain them. Do have a brief look around all the other parts of the Settings, but don’t make any other changes, lest you cause it to behave differently from what we expect. Close the Settings dialog by clicking OK. You are now ready to install the operating system.

Disk or disc?

Just in-case this causes you confusion, *disc* refers to the round, flat CDs and DVDs, while *disk* refer to things like magnetic hard-disks and floppy-disks. Interestingly enough, disks contain discs, but don't resemble a geometrical disc on the outside. It can be all terribly confusing, and the Usage Note at dictionary.com's entry for "compact disk" [http://dictionary.reference.com/browse/compact%20disk#sharethis] gives some explanation, but you'll probably find it also depends on whether you use British or American English.

7.4. Installing Server1 with Ubuntu 18.04 LTS Server

We are installing a Linux server, and like any Unix-like server system, it does not require a graphical environment. Indeed, there are good reasons for not *wanting* a graphical environment. Chief among these is complexity. Complexity is the often-times enemy of stability and security (ie. things are more likely to fail in ways that could easily disrupt the rest of the system). A graphical interface also consumes rather a lot of system resources, and we would typically want background processes to have preferential treatment, so it would also be rather sluggish.

Graphical user interfaces are also not very scalable in terms of management operations *on the system itself*. Graphical user interfaces can however be wonderful when *coordinating* management operations across a large number of machines.

Microsoft Windows Server Core

It took a while, but Microsoft eventually came out with a minimal version of its Windows Server 2008 product that wasn't tied to a graphical interface. This came largely because Microsoft Windows operating systems can be managed so well using Microsoft Active Directory; and many systems are aggregated using virtual machines, causing a desire for a smaller footprint; there is a smaller "Attack surface", meaning there are fewer things running for an attacker to target.

Server Core can also be managed using a command-line environment, which Microsoft calls Windows PowerShell. Unlike Unix-like systems, Server Core must be installed into an existing production network.

More information can be found in the Microsoft Developer Network (MSDN) documentation for Server Core [http://msdn.microsoft.com/en-us/library/ms723891(v=VS.85).aspx].

Because this is likely the first time for you (although for a number of you, you might have been down this road a number of times already), we shall aim for a "first" system; one where we don't really have any experience on which to base some of our decisions. These decisions include questions relating to how much memory should have been allocated to your system, how should you disk be partitioned into filesystems and what software should you install.

We've already told you how much memory you should allocate to your server, although later on we shall see how much is actually used. Figuring out how best to set up your storage (partitioning your disks and formatting the partitions with a filesystem) can be somewhat onerous, and it will often depend on what you will be doing with a particular machine. There

are plenty of guidelines, but without seeing for ourselves how large different parts of the filesystem should be, it is not particularly useful, so we shall let Ubuntu help us out with some defaults. Finally, with Ubuntu, the choice of which software to installed is made very very simple, to a point where it can be a bit more annoying if you know exactly what you want, so we can basically leave that question aside for now.

So, our “first” system is basically going to be installed practically using all default values, then we shall have a look at what the installed system looks like, and start gaining some experience which we could then use to reinstall the server — although we’re not going to reinstalling the server today because we don’t have time.

“Plan To Throw The First One Away”

This is a quote from a famous software engineer by the name of Fred Brookes, who wrote a series of well-regarded essays on Software Engineering called the *Mythical Man Month* (ISBN: 0201835959). This saying is perhaps more true of installing an unfamiliar operating system than it is of software engineering, as you will inevitably learn more about the system and realise that you could have made wiser decisions when you installed the system.

Important

As you proceed through the installation, be certain to record suitable documentation as instructed in a previous section.

Procedure 7.2. Running the Ubuntu Installation

1. In the VirtualBox main window, click on your new server, which you called “cosc301-server1”, and click on Start to start the virtual machine. It should soon prompt you what language to install. Choose English by pressing the **return** key.

2. The installation should start, and you should be asked a a question regarding keyboard. Select the default by pressing the **return** key.

3. Ubuntu will attempt to configure the network interface using DHCPv4. You can press **return** to accept the default setting.

   Then you will be asked to configure a proxy server, which we don't need. So just press **return**.

   Regarding Ubuntu archive mirror, accept the default by pressing **return**.

   Then you will be prompted with filesystem setup. Just choose the default Use An Entire Disk by pressing **return**. Then choose the only virtual disk as the local disk for installation by pressing **return**.

4. Now comes what could be our first major set of decisions: partititioning. We only have one disk in our virtual system (we could have made others, if we wanted to experiment with different disk management strategies, but that is not what we want at this stage).

   We could manually create a bunch of different filesystems, but at the current stage, you don’t know enough to appreciate it and you may end up making some parts of your system too small and have to repeat the entire the procedure all-over again; remember, we’re creating a “first” system.

So we just accept the default partitions by pressing **return** on Done. The default has two partitions. One partition is for root filesystem /, which has a type of “ext4”, and this is where all of our system will be installed. Another partition is for grub, the booting software.

For the next Confirm destructive action notice, just choose Continue.

5. Now you will be asked a number of important questions. The first question is the name of the first user. It is the administrative user account. This is a normal user who is also given the power to use the **sudo** command. Use the full name “Miss A. Laneous”. The server name should better be “server1”, which is consistent with the name the machine will have on the network.

   For the username, we shall continue our own standard nomenclature by using a username “mal”. The password will be `Quack1nce4^`.

6. Then you will be asked to install SSH or not. Don't install it at the current stage. We will install and configure it in a later lab.

   Next you will be asked about Featured Server Snaps. Don't install any such package as the default installation will be sufficient for us.

7. Now begins the wait of perhaps several minutes, while the “base” system is installed. This installs a minimal bootable system onto the disk.

> **Tip**
>
> While it is installing, go and read some of the other sections of this lab, and return to here when you are ready to continue.

8. Installation complete! After being prompted with this message, reboot the server. Then you will be asked to remove the installation medium. just press **Return**). The system will reboot, and it should reboot into your new system. This is generally referred to as “the moment of truth” because you are seeing if your installation actually worked.

9. You should see the following on screen, after some initial startup messages:

```
Ubuntu 18.04 LTS server1 tty1

server1 login:
```

   This is your login prompt. Use alt+fn+F2 to choose a different window to login as the first presented window is the console which has many log messages to appear. You can now login as the user “mal” with the password `Quack1nce4^`. After a bit of processing, it will greet you with some information regarding the system statistics (load, memory user, number of processes, etc.) as well as tell you how many packages can be updated.

10. Update the apt software packages with the following commands.

```
$ sudo apt update
$ sudo apt upgrade
```

11. Install the ifupdown software package, which allows us to configure network interfaces with **ifup** and **ifdown** commands.

```
$ sudo apt install ifupdown
```

12. Edit the configuration file `/etc/netplan/50-cloud-init.yaml` to disable **netplan**. Initially the network in server1 is managed by **netplan**. In this paper, we are using the ifupdown package to manage the network interfaces. To disable **netplan**, change `/etc/netplan/50-cloud-init.yaml` as below.

```
network: {config: disabled}
#network:
#    ethernets:
#        enp0s3:
#          dhcp4: true
#    version 2
```

    Basically we uncomment the line for disabling the network and comment out the lines that set up the network interface `enp0s3`.

7.5. Disabling the Unaccelerated Framebuffer

If you start playing with the system now, you'll quickly find that it feels very slow, particularly when output is scrolling on screen. That's because Ubuntu is using something called a "framebuffer", which allows it to display output suitable for all the different human languages that Ubuntu supports. In our case, we don't need such enhanced support, and if it were to use the standard old VGA interface, it would be much faster, so we shall turn it off to get our performance back.

Edit, using **sudo**, the file `/etc/default/grub`, and uncomment the following line by removing the #:

```
#GRUB_TERMINAL=console
```

Now, as root (using **sudo**), run the command **update-grub** in order to affect the change. This will prevent GRUB from trying to use a framebuffer but will not prevent Ubuntu from setting one up later. To prevent that, as root, add the following line to the end of the file `/etc/modprobe.d/blacklist-framebuffer.conf`:

```
blacklist vga16fb
```

Note

In this case there is no command you need to run in order to affect the changes. This was not always the case in earlier versions of Ubuntu or Debian. Indeed, figuring out how to disable the framebuffer can be an exercise in frustration, as there have been many ways this could be done in the past, and many don't work today.

Now reboot (**sudo reboot**) and when it comes back up, you should notice the window is slightly different shape, and when you run a command that produces a lot of output (such as **dmesg** to output the kernel logs), it should scroll very very quickly.

7.6. Connecting to the Network

Okay, so at this stage, we should have created the (virtual) hardware for Server1, installed it with Ubuntu and reclaimed some performance. But the network interface `enp0s3` should be down as **netplan** is disabled. Now we're going to change the interface name to a proper name that makes better sense and is easy to remember, and to configure it to connect to the Internet again.

Procedure 7.3. Connecting to the Network

1. First, check the detail of the interface `enp0s3`, which is not "UP" yet.

```
$ /sbin/ifconfig enp0s3
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:e3:4d:42
          no lines saying "inet" or "inet6"
          BROADCAST MULTICAST  MTU:1500  Metric:1
          …
```

2. Because we will occasionally need access to the Internet in order to get software packages, and for other tasks, we shall be keep our current network interface attached to the Internet, and later on we shall add another interface that we shall use for offering services to our internal network. To reduce confusion, we shall rename our current interface from "enp0s3" to "outside".

   Rename the interface by creating an appropriate `/etc/systemd/network/70-outside.link` file, as we practiced in an earlier lab about basic interface management (ie. adjust the "Name" and "MACAddress" parameters).

   You will need to create this file for both of your adapters. You could use the name of the network you are connecting to as the name of the link file to help you keep track of what's going on.

   Warning

   Don't forget to run **sudo update-initramfs -u**.

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

   Reboot when you have completed the edit and ensure the renaming has worked by listing the interfaces with **/sbin/ifconfig -a**. At this stage, it should not yet have an address.

3. Our outer interface is connected to the VirtualBox "NAT" attachment, and so should be configured using DHCP. To affect this, edit the file `/etc/network/interfaces`, *adding* the following lines.

```
auto lo
iface lo inet loopback

auto outside
iface outside inet dhcp
```

   It's as easy as that. Now test that you can get an address using DHCP by bringing up the interface:

```
# ifup outside
```

```
… You will see a bunch of output
The following line can be ignored, as the file will be created…
chown: failed to get attributes of `/et/resolv.conf': No such file or directory
bound to 10.0.2.15 -- renewal in 36648 seconds. Success!
```

4. As a final “moment of truth”, reboot the virtual machine (**sudo reboot**). After you log in again, check the interface details:

```
$ /sbin/ifconfig
outside   Link encap:Ethernet  HWaddr 08:00:27:e3:4d:42
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0    Success!
          inet6 addr: fe80::a00:27ff:fee3:4d42/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2070 (2.0 KB)  TX bytes:1788 (1.7 KB)
```

7.7. Software Updates

Now we should have a freshly installed, updated machine that is connected to the network. This represents some of the very first things we would do to a new system post-installation; we shall do some further work in the following lab.

For software updates, the Ubuntu software is mirrored all over the world, and many people even provide their own mirrors or caches for their own local network. However, we are going to use the default APT proxy-cache configured in `/etc/apt/sources.list`. This allows us to avoid problems like unavailable mirrors/caches or non-active caches (see the warning box below). It may cause more global Internet traffic which is not an issue anymore with today's Internet.

Not all mirrors are equal

There is, in fact, a mirror already available on the local campus, but we have found it to be somewhat unreliable with regard to keeping security updates current. This is something you need to be vigilent with when using a mirror.

In case you want to configure Server1 to access this a local mirror (which from the client-side, looks much like any other mirror), you can write the following (as an example but do *not* do it) into the file `/etc/apt/sources.list`.

```
deb http://mathmirror.otago.ac.nz/mirror/ubuntu/ xenial main restricted universe multiverse
deb http://mathmirror.otago.ac.nz/mirror/ubuntu/ xenial-updates main restricted universe multiverse
deb http://mathmirror.otago.ac.nz/mirror/ubuntu/ xenial-security main restricted universe multiverse
```

We shall explain what this means shortly. Note we have already updated the set of packages that APT can know about using the following command, so you don't need to do this again. The purpose to do this is to see what packages have updates available, and to learn of any software that is available.

```
# apt-get update
Hit:1 http://mathmirror.otago.ac.nz/mirror/ubuntu xenial InRelease
Hit:2 http://mathmirror.otago.ac.nz/mirror/ubuntu xenial-updates InRelease
Hit:3 http://mathmirror.otago.ac.nz/mirror/ubuntu xenial-security InRelease
Reading package lists...
Building dependency tree...
Reading state information...
…
```

The output looks a little bewildering at first, but the important thing is that it doesn't have any Error lines, so everything appears to have gone smoothly. To give you a bit more understanding of what it is doing here, lines that start with `Get` are files that need to be downloaded (if the local system already had a fresh copy, it would say `Hit` instead). Lines that start with `Ign` are ignored, typically these are translation files that are not needed. The `Release.gpg` files are digital signatures, used to ensure that the other files have not been tampered with since publication. The `Release` files contain information relating to what files are available and currently considered to be "currently in the archive" and include a checksum to ensure the files have not been damaged[2]. The `Packages` files are largest, as they describe every single package that is available, as well as the package meta-data, such as version and dependencies.

To apply all of the available updates, use the following command.

```
# apt-get dist-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following NEW packages will be installed:
  … a few
The following packages will be upgraded:
  … many
72 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 73.8MB of archives.
After this operation, 104MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
… a lot of work begins downloading and applying updates
```

We'll cover package installation more in the lab on post-installation, but the key thing to note here is that first we used **apt-get update** to update Server1's knowledge of what packages are currently available, and then we used **apt-get dist-upgrade** to perform a major upgrade of any packages. Typically, we would only use **apt-get upgrade** for day-to-day upgrades, but since this first update after installing could contain more significant[3] changes, it is best to use a command such as **apt-get dist-upgrade**.

apt-get and Friends

There are multiple tools similar to **apt-get**. For example, **aptitude** is a bit smarter, and generally replaces **apt-get**; **synaptic** is like a graphical tool similar to **aptitude**, and there are still others. This is not a course in Debian administration tools, so we shall just stick with **apt-get**.

While that is completing, let's look again at what we put in that `sources.list` file. Here's the general format:

```
deb URI release section …
…
```

deb
: Each line (ignoring comments and blank lines) will have either a `deb` or `deb-src` to describe either a binary or source package respectively. We shall only be dealing with binary packages, source packages are not often used. `.deb` is the standard file extension for a Debian package.

[2] In particular, since the release files are digitally signed, it also ensure the packages have not been tampered with.
[3] Meaning that it would cause other previously-not-installed packages to be installed, or some existing packages to be removed.

`URI`
A URI (or URL if you prefer) specifies where such packages may be found, and will generally specify a method (such as via HTTP, FTP, or locally available on a CD-ROM or elsewhere in the filesystem), and usually a host for network-oriented access methods. In our particular entry, we also specified an optional port number, since the default port number for HTTP is port 80. We also needed to say where on the server the Ubuntu "archive" can be found; typically this will be `/ubuntu`, but that is not a requirement.

`release`
This basically specifies the *version* of Ubuntu we are interested in installing. Because this is typically a moving target, we have three versions which we typically use, although five are available:

`Bionic` is the codename for Ubuntu 18.04 LTS, so the `Bionic` version contains whatever is in the latest version of the official Ubuntu CDs. All Ubuntu hosts will have at least this.

`Bionic-updates` are major software updates, that may add (or remove) features, but are not security updates. They represent the changes between "point" releases, such as "14.04" and "14.04.1". You can choose whether you want to have these installed or not; they may cause unplanned downtime due to changes.

`Bionic-security` are security updates for software that the Ubuntu Security team has put together. All machines should include this release.

`Bionic-backports` represent software that has been packaged for newer versions of Ubuntu, and has also been "back-ported" into the Bionic release. This is useful only if you need something on a Bionic box that is not available in Bionic, such as a particular feature of some software that is available in Ubuntu 14.10 but not 14.04. It is not commonly used.

`Bionic-proposed` is generally for people wanting to test updates (ie. developers and people who really need a bug fixed) to test something about to be put in `Bionic-updates`. It is not intended for general consumption.

Further, different versions of Ubuntu will have different code-names, so for example 14.04 LTS is "Trusty Tahr" (trusty), 13.10 was "Saucy Salamander" (saucy) and 13.04 was "Raring Ringtail" (raring). The codenames are in alphabetical order.

`section …`
The sections always have the names `main`, `restricted`, `universe` and `multiverse`.

These sections have different restrictions with regard to licencing and support. Everything will have at least `main`. A desktop system will commonly have all of them to incorporate software such as Adobe Flash support for popular video sites such as YouTube, for which there is no good open-source product available. Wireless drivers often require the `restricted` section, as they are generally binary-only or have some firmware with a restrictive copyright. Only `main` is looked after by the Ubuntu Security team, so if you don't need something in the other sections, it is good to omit those sections.

`universe` contains a lot of other useful programs that you would often want, so you can generally include that also.

Okay, hopefully by now you will have finished performing the updates, which will have very likely installed a new version of the kernel; after which we should reboot into the new kernel. Note that you should choose the first kernel after reboot.

```
# reboot
```

You can now proceed onto the next section to help integrate your virtual machine with the host.

7.8. Installing Guest Additions

In this section, we are going to install the VirtualBox Guest Additions, which enable the guest (virtual machine) to have some greater integration with the host. Typically, this is much more useful in a desktop environment, but in this lab we shall only be making use of the shared folders feature, which allows us to easily share files between the guest and the host.

On a desktop system, it would also enable clipboard integration for copy-paste, graphics performance enhancements as well as a number of other things. On other virtualisation platforms, such guest additions are also used, and can help with host memory utilisation, and enhanced performance using specialised drivers for virtual network cards etc, so in general guest additions are useful for all classes of guests.

Procedure 7.4. Installing the Guest Additions

1. To install the guest additions, with the VirtualBox window titled “cosc301-server1 [Running]” in the foreground, select Devices → Insert Guest Additions CD image… from the VirtualBox menu at the top of the screen. This inserts a virtual CD (an ISO image) into the virtual CD-ROM drive of the guest.

2. Because Ubuntu Server will not automatically mount the disc with the Guest Additions on it, we shall have to mount it manually. “Mounting” a filesystem means to attach it and make it available somewhere under the filesystem hierarchy.

   When we performed this step, we discovered that Ubuntu Server did not have the file-system table (file `/etc/fstab`) configured with instructions where to mount the CD-ROM device, so we shall add that now. *Add* the following line to `/etc/fstab`:

```
/dev/cdrom /cdrom iso9660 ro
```

   Now you should be able to mount the filesystem which is on the CD-ROM, into the system’s filesystem, making its contents available under the directory `/cdrom`:

```
# mount /cdrom
$ ls /cdrom
32Bit         runasroot.sh                    VBoxWindowsAdditions.exe
64Bit         VBoxLinuxAdditions.run      VBoxWindowsAdditions-x86.exe
AUTORUN.INF   VBoxSolarisAdditions.pkg
autorun.sh    VBoxWindowsAdditions-amd64.exe
```

   Now we need to run the **VBoxLinuxAdditions.run** command, which will build and install the Guest Additions appropriate to our particular Linux kernel and environment.

```
$ cd /cdrom
# ./VBoxLinuxAdditions.run  run as root!
Verifying archive integrity... All good.
…
VirtualBox Guest Additions: Building the VirtualBox Guest Additions kernel
modules. This may take a while.
…
This system is currently not set up to build kernel modules.   Oops!
Please install the gcc make perl packages from your distribution.
```

```
… some more messages
```

According to the instructions above, we need to install a set of building tools as below.

```
# apt-get install gcc make perl
…
Do you want to continue [Y/n]? Y
Get:1 …
…
Selecting …
Preparing …
Unpacking …
…
Processing triggers …
…
```

Okay, so now hopefully all the packages will be installed which will allow the Guest Additions to install correctly. Let's try and reinstall the Guest Additions now:

```
$ cd /cdrom     You're probably still there
# ./VBoxLinuxAdditions.run
Verifying archive integrity... All good.
…
Removing installed version 6.1.2 Guest Additions...........
…
VirtualBox Guest Additions: Building the VirtualBox Guest Additions kernel
modules. This may take a while.
this takes a while, don't panic
…
VirtualBox Guest Additions: Building the modules for kernel 4.15.0…  Success!
…
VirtualBox Guest Additions: Running kernel modules will not be replaced until
the system is restarted
```

3. Now you are finished installing the Guest Additions. Reboot the server machine.

Procedure 7.5. Setting up Shared Folders

1. In this procedure, we shall configure VirtualBox to share a folder on your hosts desktop (we shall call it `VBoxShare`) with the guest. This allows files to be moved into and out of the virtual machine to the host, without the need for any networking.

2. Click on "cosc301-server1 [Running]", at the bottom of the Settings window, you will see Shared folders. Click it.

   Click on the folder icon with a green plus icon; this will add a new entry. In the Folder Path drop-down box, select Other… and navigate to your desktop. Click on New Folder and create a folder called `VBoxShare`. Then click on Open.

   > **Tip**
   >
   > It can be useful to share a single folder among many guests.

   Because we want this to always be available, tick the box labelled Make Permanent, and then click OK. Click OK again to close the Shared Folders dialog.

   We have now configured our host and the VirtualBox virtual machine with the shared folder, but the operating system inside the virtual machine still needs to be configured to do something useful with it.

3. Add the following line to `/etc/fstab`:

```
VBoxShare /media/host vboxsf defaults,uid=mal,gid=mal
```

   The first field (`VBoxShare`) is referring to the Folder Name that was specified in the previous step. We are going to make it available on the directory `/media/host`, which we shall very soon create. Because we want our regular user to have convenient access to it, we shall specify that everything is owned by the user and group called “mal”.

4. Create the directory that shall be used to access the filesystem (this is typically called a “mount-point”).

```
# mkdir /media/host
```

5. Add the item `vboxsf` to `/etc/modules`. This causes the `vboxsf` kernel module to be loaded earlier in the boot process so that it is ready for trying to mount the shared folder when the computer boots.

6. Time for the moment of truth. Reboot and ensure that it still comes up smoothly:

```
# reboot
```

7. Time to test. Log in as the user “mal”. Inspect the currently mounted filesystems and see if /media/host is mounted:

```
$ mount
…
VBoxShare on /media/host type vboxsf (uid=1000,gid=1000,rw) Success!
…
```

   Now create a file inside `/media/host`:

```
$ echo “Hello” > /media/host/hello.txt
There should be no error message produced.
```

   On your host’s desktop, go into the `VBoxShare` folder and check that a file called `hello.txt` is present and that it contents match.

8. Now test in the opposite direction. Copy a file (such as a screenshot) from the host into the `VBoxShare` folder on the host. Inside the virtual machine, use **ls -l** to inspect the permissions. If you see something like this, then it is working well and you can go onto to next section.

```
$ ls -l /media/host
…
-rw-r--r-- 1 mal mal         6 2010-12-09 11:55 hello.txt
-rw-r--r-- 1 mal mal    288593 2010-12-09 11:56 some-picture.png
…
```

Use this for Keeping your Work Handy

Students often want to take the work they have done in these labs and take it home to set up their own home network. You will find this shared folder a reasonably convenient way to export files so you can move them from the VBoxShare folder to removable media etc.

7.9. Fixing some of our Install-Time Ignorance

You may have noticed the time zone is not quite right. Use **sudo dpkg-reconfigure tzdata** to set the right time zone.

Now we shall have a brief look at how large various parts of the filesystem are on our system, and how much memory is currently used. This will give a baseline for a Ubuntu Server, which we can then add-to when considering requirements for further installations.

First, let's just take a peek at memory utilisation:

```
$ free -tm
             total       used       free     shared    buffers     cached
Mem:           244         51        193          0         11         25
-/+ buffers/cache:         14        229
Swap:          232          0        232
Total:         477         51        426
```

Linux operates a "memory full" sort of memory allocation strategy, in which physical memory not in use can be used for caching file-system objects etc, which is a good thing. Unfortunately, when you look at the Used Mem (here, 51MB) it can be a bit misleading. Instead, look at the entry below it (Use Mem -/+ buffers/cache), which is currently 14MB. So all of the running processes on our baseline Ubuntu system use just 14MB of "physical" memory, out of the 244MB of "physical" memory currently installed in the virtual machine. We actually allocated 256MB, but 12MB is given to a display adaptor which has its memory carved out of the main-memory.

Note also that the Used Swap is currently 0. We want this to be very little (you may find a few MB are used, but so long as its not actively accessing swap frequently, that's okay).

So what processes *are* running on a basic Ubuntu Server with VirtualBox Guest Additions installed?

```
$ pstree
init─┬─VBoxService───6*[{VBoxService}]
     ├─atd
     ├─cron
     ├─dhclient3
     ├─6*[getty]
     ├─login───bash───pstree
     ├─rsyslogd───2*[{rsyslogd}]
     ├─udevd───2*[udevd]
     └─upstart-udev-br
```

pstree is a useful little command that shows all of the processes in an easily digested way. It is not as available as the venerable old **ps** command, but it is by far easier to use for casual purposes. Currently, you don't need to understand what everything is for, but it's useful to develop a feel for what is "normal" in a system… documenting what is "normal" behaviour can also be useful.

Okay, so that's memory and processes, let's now have a quick look at network activity and then we'll look at the filesystem. We will practice this plenty of times later on, but here is how we can see what processes are listening for network connections:

```
# lsof -Pni    as root!
COMMAND   PID USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
```

```
dhclient3 538 root   4u  IPv4   3066      0t0  UDP *:68
```

Ubuntu has a policy of not running with any network-reachable processes by default, and that certainly seems to be the case, as the only thing listening on the network is **dhclient3**, which is the DHCP client – that's how we got our network address, so it needs to be running. This sort of thing is useful to know for later on when you begin securing your operating system.

Okay, time to look at the filesystem usage. First, how much is actually used?

```
$ df -h      Mnemonic: disk full
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2             9.8G  4.0G  5.4G  43% /
… The rest can be ignored, they are virtual filesystems
```

Okay, so of the single data partition that Ubuntu Server created for us when we installed, 43% is used. How is that usage distributed through the filesystem?

```
$ sudo du -xm --max-depth 2 / | awk '$1 > 2'
17 /opt/VBoxGuestAdditions-6.1.2
17 /opt
16 /sbin
12 /usr/sbin
183 /usr/share
135 /usr/src
378 /usr/lib
91 /usr/bin
11 /usr/include
808 /usr                              Note
8 /boot/grub
75 /boot                            Note
235 /lib/modules
324 /lib/firmware
9 /lib/systemd
15 /lib/udev
25 /lib/x86_64-linux-gnu
607 /lib
15 /bin
27 /var/log
330 /var/lib
143 /var/cache
499 /var                            Note
6 /etc
4010 /                               Ignore, misleading
```

What that command does is simply to report the "disk usage" (**du**) in megabytes (`-m`) of everything in the root (`/`), without going across onto other (typically virtual) filesystems (`-x`) and without recursing further than two levels. The **awk** command is being used to filter the output, only outputting lines that have a value in the first field (megabytes used) greater than 2.

We have noted those lines that are given for files in the same partition. However, for those files that are not in the same partition, **du -x** does not show. For example, `/usr/local` and `/home` often have their own partitions so they are not shown in the above lines. Therefore, the final entry only shows the total for the current entire partition for the root (`/`) filesystem. But if you wanted to put `/var` on a separate partition, you would need to subtract the usage of `/var` from the usage of `/` in order to get accurate estimation of the usage of of `/`.

It is important to realise that this is only a baseline, and as a system grows, parts of the system will increase (particularly in places such as `/var`, and will differ a lot from one system to another depending on the nature of the services being run, so you would need to repeat this process after you have developed your services and run them for a while.

Right, we've done a lot in this lab, so let's move to self-assessment.

7.10. Self-assessment

There has been plenty of things to do today, but there are two assessments for this lab. The first is the documentation you have prepared, the second is to fill in the blanks in Table 7.1, "Client and Server Operating Systems".

7.11. Appendix: Deploying Many Machines

This is optional material that is made available for those students who find nothing particularly new in installing Ubuntu. There is no work that needs to be done, but provides some valuable further insight into some real-world related tasks.

If you install multiple, or many, machines by hand, you are wasting a lot of valuable time, plus you're much more likely to make a mistake. Thus, we need a way of being able to get a working system onto many machines in a short amount of time. This is what operating system deployment is all about.

7.11.1. Deploying Using Disk Images

Perhaps the most common paradigm for deploying workstations is that of the *disk image*. This is probably due to the predominance of Windows in the IT space, and the experience IT staff have with this paradigm. Norton Ghost is a very well-known piece of software for the creation and deployment of disk images, which has been around for many years and supports nice features such as multicast image distribution and also now supports Linux filesystems. A selection of other disk imaging products can be found at Business.com's Best Server Backup And Imaging Software [https://www.business.com/categories/server-backup-and-imaging-software/], and The Free Country [http://www.thefreecountry.com/utilities/backupandimage.shtml] has good coverage of free tools.

As imaging goes, there are basically two common methods: *sector-based* and *file-based* images. Sector-based images are made by copying each sector on disk, and so don't care about what file-system is used on top of them. File-based imagers need to understand the particular filesystems involved. File-based imagers create smaller images and can offer richer features (such as knowing what files not to copy) but can be more complex and even more timeconsuming.[4]

Imagers work somewhat as follows: first the machine to be *imaged* is prepared, typically removing any temporary files, defragmenting the filesystem and zeroing out any free-space; file-based imagers can often do this for you. This is all done to reduce the size of the resulting, typically compressed, image.

On Linux and similar machines, which use filesystems that automatically defragment as needed, both defragmentation and zeroing of free-space can be done using the following commands as root: **dd if=/dev/zero of=/largefile; rm /largefile**[5] The command works by writing a file containing zeros to the disk, the file will be written until all the space is

[4] So **dd** is a very naïve sector—based imager whereas **dump** is a file-based imager—we shall see them in a later lab about filesystems.

[5] This assumes that there is only one filesystem on the disk. If there are multiple, the command should be run replacing `/largefile` with `/mountpoint/largefile` for each local filesystem.

consumed. Near-filling of the filesystem is one of the events that causes a defragmentation event on filesystems such as ext3. As a side effect, the parts of the filesystem that were free-space are now filled with long strings of zeros (NUL bytes) which compress very nicely. Deleting the file doesn't cause these blocks to be overwritten. As another side-effect, any old data blocks will have been overwritten, but you should not rely on this for security.

Apple have made this easy in Mac OS X. In the Disk Utility program, you can use the Erase Free Space… Unlike the basic naïve **dd** approach, you can securely remove old data this way.

Most imaging programs understand the Windows file-systems sufficiently well enough to ignore those data-blocks that are not currently allocated. Windows machines will need to have a tool called SysPrep run on them to remove their identity, otherwise after deploying the image, multiple instances of the same computer (same identity) will show up on the network, which causes problems. This identity is called the Security ID or SID.

The imaging program is not run from the system you are imaging, rather you boot from another system, either from a bootable CD or floppy-disk, or you can remove the hard-disk to be imaged, and plug it into a different machine as the non-boot device. Apple have made this very easy, as Apple Macintosh machines have a feature called *Target-Disk Mode*[6]. You boot the machine to be imaged, holding down **T**. Now the storage devices on the machine can be accessed as a Firewire storage device, and can be imaged from a neighbouring Mac. A very nice feature, especially for laptops.

When you create the image file, you generally store it on a network, on a machine with plenty of hard disk space, and is compressed as it is created. This is particularly true for Windows deployments, because with Windows, you generally need one image for each hardware configuration.

Deployment can be done individually, typically booting the machine to be imaged either over a network, CD or floppy, or mounted on another machine. The process is then run in reverse, writing the disk image contents onto a selected hard disk.

However, because you often want to deploy many machines at once, it can be much easier to boot each into the imaging software, such as Norton Ghost, and then *multicast* the image from the imaging server (generally on the same subnet as the clients) to all the clients at once, which reduces network traffic. This assumes that your network can handle multicast traffic; many cannot[7].

One advantage of disk images, that Apple takes advantage of, is that you can use them for network booting. Apple calls this *NetBoot*, as opposed to *NetInstall*, which is for installing (er, deploying) an image over the network to the hard disk. With NetBoot, clients boot from the network, and don't require a hard disk, but rather run from the read-only image on the network. This is useful when you have a lab of Macs that need to run a particular environment for a short period of time, or when you want to reset the machine to a known state very easily. It does require a fast network though. Each client will typically have some writable storage on the server (called a "shadow file") so the entire computer still feels writable, and users will have home directories on the network.

Disk images are most commonly used for deploying a minimal base-system. Other technologies are often used in tandem for deploying applications and managing configuration; one particularly well-known, if not well-loved by users, example of this is Novell's ZEN (Zero Effort Networking) product.

[6] An excellent example of how the legacy BIOS impedes PC innovatation, and a sign of things to come with EFI.
[7] You need switches that employ *IGMP Snooping*, otherwise the traffic turns into broadcast traffic, which floods the network and can bring a local network to its knees.

7.11.2. Scripted Installations

Installation can be a tedious process, and deployment can be impractical when you have heterogeneous devices. Therefore, what would be really nice is the ability to perform an unattended or scripted installation. This can commonly be performed either by booting from the network, or providing some install-time parameter that points to a set of answers for the questions asked during installation. This file is commonly retrieved via HTTP, but may also be made available on a floppy etc. If booting from the network, this information may be configured based on the particular machine.

In the Linux space, two tools are commonly used: *Kickstart* for RPM-based systems, such as Fedora; while Debian-based systems use *pre-seed* files, which can answer questions not only at operating-system install-time, but also when individual packages are installed later. For information on using pre-seed files, see Unattended Ubuntu Deployment over Network [http://www.debuntu.org/how-to-unattended-ubuntu-network-install]; for Kickstart, see the Kickstart Installations [http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-kickstart2.html] part of the Redhat Linux manual. Kickstart also has a nifty little program to make the process of creating a kickstart file much easier, and a suitable Kickstart file is made when you install the OS manually, which provides a useful starting point.

Windows also supports scripted installations. Mac OS X notably does not; Apple prefers people to use imaging technologies for a base install. Given Apple's fairly uniform hardware offering and simple installation process, there is less need for scripted installations, so long as the operating system you are installing is at least as modern as the most modern machine you wish to deploy onto. Further client administration is expected to be done using Apple's OpenDirectory and Remote Desktop products.

7.11.3. The "Golden Client" Methodology

One way to deploy, and maintain deployed machines, is to use a golden-client methodology, as made popular by software such as SystemImager [http://wiki.systemimager.org/] and Radmind [http://rsug.itd.umich.edu/software/radmind/].

The basic concept is thus: you install a system (your *Golden Client*) and get it to the state you want to deploy to. You run some software which uploads all the files (excluding certain files which should be unique to the particular machine) to a server. You then run an initial install-client on the machines you want to deploy onto, which downloads the files from the server. Some scripting may be used to provide for tasks such as partitioning the hard disks. A concept of "classes" may be used to provide different files to different machines, so for example you might have a class for machines with ATI graphics cards and another class for NVidea, or you might have a class for particular machines which need some software with a limited number of licences.

When you want to make an update, you make the update to your Golden Client, run the agent on the Golden Client to synchronise the changes up to the server. You then cause all the other clients (perhaps by initiating a command over the network, or by some scheduled task) which runs a *differencing engine* to make the clients' files the same as the client files installed on the server.

This has the advantage of making updates relatively easily, especially when you have a lot of software being installed from different formats. However, there can be a fair bit of work involved in ensuring you don't upload something to the server that should be local to the machine. The granularity is typically limited to a file which means modifications to isolated parts of a file (such as adjusting a configuration file) become rather more difficult.

Configuration engines such as **cfengine**, **puppet**, and **chef** are much more capable in this regard.

7.11.4. Dealing with Heterogeity

Deployment can be much easier when the hardware platform you are deploying to is consistent. There is some room for flex, depending on the operating system in question. For example, Windows can be quite perplexed if it finds itself running with different hardware than the last time it booted, so Microsoft have a tool called SysPrep that removes some of the more unique aspects of the underlying registry, ready for deployment.

Linux systems tend to be rather more forgiving about suddenly finding itself on different hardware. So long as the system is not configured to particular hardware, and there is some hardware detection performed at bootup, then the process can be quite smooth. A lot of problems can be worked around by editing startup-scripts which check to see what devices are used in the system (**lspci**) and, for example, re-pointing a symbolic link to use the most suitable X.org configuration file, depending on what video card is installed. Thankfully, modern Linux systems have done a lot of work to make this less and less necessary.

Mac OS X, with its narrow hardware divergence, has no apparent problems with this either.

When deploying, you want to avoid as much local configuration as possible. For example, avoid unnecessary local user accounts and instead using network-based accounts and home directories. Linux systems have less ability to store their configuration in *directory servers*, which we we shall cover later in this paper. However work is being done to make Linux more Enterprise friendly, especially by Novell (which acquired SuSE Linux, another major player in the European Linux space). Microsoft has done a good job on making many information elements configurable via a directory server. Apple has also done a lot of good work, and is maturing nicely. Linux has a harder time because of its plain-text configuration file heritage of Unix.

7.11.5. Maintenance

Inevitably, you will need to maintain these machines, and this will include some degree of maintenance to a) possibly numerous disk images, b) installation scripts, or c) a golden client.

Disk images are easily the most amount of work to update, especially if you have to update multiple images, because you need to restore the image, update the system, recreate the image, and repeat for all images.

Installation scripts probably won’t need a lot of work, often just adding or removing a directive to install a particular package, although this might be managed in some other way, such as **cfengine** [http://www.cfengine.com].

A Golden Client will need updated, synced to the server, and then synced to all the clients. This is typically fairly fast, and it is an effective way maintaining a lab of similar machines, especially where the files have been customised post-installation.

Lab 8 Post Installation

In the previous lab, we installed our server virtual machine, Server1, and applied a bunch of updates. In this lab, we're going to continue from there, and actually make it part of the same network as our client, set up some IPv4 and IPv6 addressing and run a few tests. We shall also spend some time looking at one way of creating network services, as well as one common way to restrict access to network services.

> **Important**
>
> Except where explicitly noted (typically in command prompts, but also in the text), all work is to be performed on the server you installed previously. You will also need Client1, but nothing else.

8.1. Adding the "inside" Interface

Currently, the "outside" interface of Server1 is configured to attach to VirtualBox's "NAT", allowing it to access the outer (campus) network without the server needing an address on the outer network. But with this attachment comes a restriction; we can't talk to other virtual machines. We want our virtual machines to all be connected to each other in a dedicated LAN.

We could re-attach the "outside" interface of Server1 to the internal network, which would allow us to connect with Client1, but that means we wouldn't be able to connect to the outer network if we needed packages. That's really annoying.

To solve this problem, we shall add another adaptor to VirtualBox, and configure the new interface as the "inside" interface. This will allow us to create a "dual-homed" configuration whereby we can talk to both the outside world and the internal network.

Shutdown the server cleanly. It needs to show its status as "Powered off". If it is showing as "Saved" you will need to start it, then shut it down cleanly using **shutdown -h now**.

In the VirtualBox network setting for the server, leave Adaptor 1 as it is (it should be attached to NAT), and go into Adaptor 2. Enable the interface and attach it to the Internal Network "COSC301 Internal Network 1".

Start the server. To prevent confusion, use the same method we used in earlier labs to add then change the interface name from "enp0s8" to "inside" (hint: `/etc/systemd/network/70-intnet.link`, and **update-initramfs -u**).

Change the network interface configuration inside the server by editing the file `/etc/network/interfaces`.

```
auto lo
iface lo inet loopback

auto outside
iface outside inet dhcp

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

Affect the changes by rebooting.

```
# shutdown -r now
```

Exercise

As an exercise check that everything is now as expected. Are the interfaces configured as we would expect?

```
$ ifconfig inside
…IP address should be 192.168.1.1
$ ifconfig outside
…IP address should be 10.0.2.15
```

Okay, now check that the routing table is as we expect, with a default route going out the "outside" interface:

```
$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         _gateway        0.0.0.0         UG    0      0        0 outside
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 outside
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 inside
```

To find out the IP address of _gateway:

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 outside
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 outside
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 inside
```

Finally, a "test by doing": can we get to the package repository?

```
# apt-get update
…
Fetched 565kB in 6s (87.56kB/s)     Success!
Reading package lists... Done
```

Testing by doing

How might you test if a web server is working? You could look to see if it running (**pstree** etc.), but that isn't an exhaustive test. It is much better to simply retrieve a page (possibly multiple pages), which tests the entire "stack". Testing by doing is a great way of testing for success conditions, but is not a good way of diagnosing faults.

Okay, so now we know that Server1 can talk to the world. Let's re-introduce Client1 into the network so we can test that we can communicate with hosts in the internal network.

Important

Start Client1, run the command **sudo apt install resolvconf** in a terminal window to install the resolvconf package. Then shutdown Client1

Before you start Client1 again, go into the settings for cosc301-client1 and ensure that the network adaptor Adaptor 1 is connected to the Internal Network "COSC301 Internal Network 1". Then start Client1.

If you recall, in the lab on basic interface management, we only gave Client1 a temporary IP address, and did not make a permanent configuration. Let's create one now. **On Client1**, edit `/etc/network/interfaces` and *add* the following stanza for `iface eth0` or *replace* the stanza if there is one:

```
auto eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
```

Affect the change using **ifup**:

```
client1# ifup eth0
 … should have no output if it works
client1$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.11  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 08:00:27:f8:ef:dc  txqueuelen 1000  (Ethernet)
        RX packets 151  bytes 10752 (10.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1076 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Exercise

As an exercise ensure you get the IPv4 address specified. Make sure that you have successfully given Client1 its address.

We added Client1 to the network to ensure that Server1 could talk to other internal hosts, so let's test that now. Still on Client1, let's check that we can ping Server1:

```
client1$ ping -c2 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=3.39 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.358 ms

--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.358/1.874/3.391/1.517 ms
```

That worked, which means traffic can flow in both directions. To be doubly sure[1], let's try the same thing on Server1:

```
server1$ ping -c2 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=4.88 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.385 ms

--- 192.168.1.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.385/2.636/4.887/2.251 ms
```

Hooray! So we know local deliveries are working on the internal network. Time to go onto the next section.

[1] At present, its not actually needed, but in general, because of devices such as firewalls and NAT, it pays to test both directions if needed.

8.2. Configuring Basic NAT

So local deliveries work, but soon we shall want to access the outer network from Client1, so let's test remote deliveries using the same "test by doing" as we did earlier on the server:

```
client1# apt-get update
…
W: Failed to fetch http://…  … Temporary failure resolving …)
E: Some index files failed to download, …
```

What happened there? The message "Temporary failure resolving …" indicates that it failed to convert an Internet name into an Internet address. This is a component called "DNS", which we shall see later.

Because we will be encountering DNS later on, we shall just skim over the necessary configuration. First, let's just ensure that Client1 is configured to access a DNS correctly. Add in the file `/etc/resolvconf/resolv.conf.d/head` on Client1 the following lines:

```
nameserver 139.80.64.1
nameserver 139.80.64.3
```

Once you have these lines, run **$ sudo service resolvconf restart** to make the change effective.

These are the same DNS servers that your lab iMac is configured to access, and will be different on different networks.

So how do we get to those 139.80.64.* addresses, which are not on the local network; do we have a route which would allow Client1 to get there?

```
client1$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0  Ignore
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

There is no route, such as a default route, to allow us to get out of the network. Open up `/etc/network/interfaces` on Client1, and edit the "eth0" stanza to add a default route via a gateway:

```
auto eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    gateway 192.168.1.1
```

Bring the interface down then up again:

```
client1# ifdown eth0
SIOCDELRT: No such process  Trying to delete the gateway, which has not yet been added
client# ifup eth0
client$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    100    0        0 eth0   Success!
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

Okay, so now can we get out to the wider network?

```
client1# apt-get update
0% [Working]  Long hang
^C
```

It took a long time because there is an error. Rather than show you how you could diagnose it (we can save that for a later lab), we'll simply tell you that it's a routing issue: packets aren't being forwarded by Server1, because it's not configured to do so. On Server1, enable IP forwarding by setting the appropriate system control ("sysctl"), which configures kernel behaviour. We can do this persistently by editing the file `/etc/sysctl.conf` on Server1:

```
…
net.ipv4.ip_forward = 1   Uncomment this line
…
```

```
server1# sysctl -p
net.ipv4.ip_forward = 1   Prints changes it applies
```

However, this is not the end of the problem. Although packets can now be forwarded from Server1's inside to outside, VirtualBox's "NAT" attachment doesn't have a "return route" (it doesn't know that in order to get packets to 192.168.1.0/24 it should go through 10.0.2.15 (Server1's outside interface). Because we can't add a return route (VirtualBox doesn't expose such functionality, we shall have to add another level of NAT — it's ugly, but it will work.

Confused? Don't panic!

At this point, you don't need to really understand what we're doing right now, it's just necessary stuff we need to do in order to get a network that we can further play with. We'll meet it again in more detail later, at which point you should come to understand.

To enable NAT, write the following onto `/etc/network/if-up.d/nat` on Server1:

```
#!/bin/sh

if [ "$MODE" = start -a "$LOGICAL" = outside ]; then
    echo "Enabling NAT on 'outside' interface"

    iptables -t nat -F
    iptables -t nat -X
    iptables -F
    iptables -X

    iptables -t nat -A POSTROUTING -o outside -j SNAT --to-source 10.0.2.15
fi
```

Use **chmod** to make the script executable. This script should now run when we use **ifup**, after the interface has been bought up. To test, first take the interface down, then bring it back up.

```
# ifdown outside
…
# ifup outside
… messages from DHCP client
Enabling NAT on outside   success
```

Does it work yet? Now try again on Client1:

```
Client1# apt-get update
… should work to completion
```

Exercise

Hooray it works! As an exercise take a little break. In the next section, we'll enable IPv6 connectivity on the local link. Don't worry, it'll be pretty easy, we won't even have to do anything on Client1!

8.3. Configuring IPv6 Router Advertisements and a Static Address

At this point, we have Client1 and Server1 reachable via IPv4, and Client1 can even connect to addresses beyond Server1, also over IPv4. In this section, we're going to show you how to set up simple IPv6 connectivity, but only for the local link (because going beyond Server1 using IPv6 is not yet supported on our outer network).

Client devices generally require no configuration. They will by default act on router advertisements. We don't have a router on our internal network that sends out router advertisements, so we shall add that feature to Server1 as well, as it is going to be a router (albeit only for IPv4 at present).

When we've done that, we shall configure Server1's inside interface with a static IPv6 address. This is important because servers generally need static addresses, and because Server1 is acting as a router, it will not participate in the SLAAC (StateLess Address AutoConfiguration) process, which was introduced in the lab of IPv6 Bootcamp.

Note

In the IPv6 Bootcamp lab, we used a virtual appliance called Radv to send out router advertisements. That appliance is now made redundant by Server1 and should not be enabled on your internal network.

The first thing we have to do is to make Server1 perform the functions of SLAAC, in order to send out router advertisements from Server1 (which is our router). The software for doing this on Linux systems is generally the **radvd** package, which we could install using the Debian/Ubuntu package of the same name

```
# apt-get install radvd
…
Setting up radvd  …
 …
 …
```

Good, it fails to start because there is no configuration file. This is good because no default configuration file means we are not going to risk polluting the network with poor router advertisements. We just need to simply create the configuration file `/etc/radvd.conf` as root:

```
interface inside
{
    AdvSendAdvert             on;

    prefix fd6b:4104:35ce::/64
    {
        AdvOnLink             on;
        AdvAutonomous         on;
    };
};
```

```
interface outside
{
    AdvSendAdvert             off;
};
```

Check radvd.conf(5) for more information about this file format. This is a fairly minimal example, and makes use of radvd's default values for most things, which are sensible defaults.

According to the `README.Debian` file (mentioned in the startup attempt of **radvd**), IPv6 forwarding needs to be enabled (even though at present we're not going to be doing any forwarding of IPv6). So uncomment or add the following line to `/etc/sysctl.conf`:

```
net.ipv6.conf.all.forwarding=1
```

Confirm the changes of sysctl, then start **radvd**:

```
# sysctl -p
net.ipv6.conf.all.forwarding = 1  prints changes it makes
# /etc/init.d/radvd start
Starting radvd: radvd.
```

Check that it's up and running:

```
$ ps -eo pid,command | grep radvd
 2057 /usr/sbin/radvd …
 2058 /usr/sbin/radvd …
 2141 grep --color=auto radvd    ignore this line
```

You may find that it's not running, in which case check the status of the service.

```
# service radvd status
● radvd.service - LSB: Router Advertising Daemon
   Loaded: loaded (/etc/init.d/radvd; bad; vendor preset: enabled)
   Active: active (exited) since Fri 2018-04-13 02:14:38 UTC; 2min 41s ago
     Docs: man:systemd-sysv-generator(8)

Apr 13 02:14:38 ubuntu systemd[1]: Starting LSB: Router Advertising Daemon...
Apr 13 02:14:38 ubuntu radvd[5333]: Starting radvd:
Apr 13 02:14:38 ubuntu radvd[5333]: * /etc/radvd.conf does not exist or is empty.
Apr 13 02:14:38 ubuntu radvd[5333]: * See /usr/share/doc/radvd/README.Debian
Apr 13 02:14:38 ubuntu radvd[5333]: * radvd will *not* be started.
Apr 13 02:14:38 ubuntu systemd[1]: Started LSB: Router Advertising Daemon.
```

You'll see that the service is active (exited). This means that it tried to startup, but failed to so and has stopped. It stopped because it couldn't find the /etc/radvd.conf file when it started. You will need to restart the service (as opposed to just starting it). There are two ways to do this, firstly, by stopping then starting it, or simply issueing restart. In addition, enable the *radvd* service so that it can be automatically started after reboot.

```
# service radvd restart
# systemctl enable radvd
```

Double check the status as before.

```
# service radvd status
● radvd.service - LSB: Router Advertising Daemon
   Loaded: loaded (/etc/init.d/radvd; bad; vendor preset: enabled)
   Active: active (running) since Fri 2018-04-13 02:23:26 UTC; 9s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 5729 ExecStop=/etc/init.d/radvd stop (code=exited, status=0/SUCCESS)
  Process: 5737 ExecStart=/etc/init.d/radvd start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/radvd.service
```

```
           ├─5746 /usr/sbin/radvd -u radvd -p /var/run/radvd/radvd.pid
           └─5747 /usr/sbin/radvd -u radvd -p /var/run/radvd/radvd.pid

Apr 13 02:23:26 ubuntu systemd[1]: Starting LSB: Router Advertising Daemon...
Apr 13 02:23:26 ubuntu radvd[5745]: version 2.11 started
Apr 13 02:23:26 ubuntu radvd[5737]: Starting radvd: radvd.
Apr 13 02:23:26 ubuntu systemd[1]: Started LSB: Router Advertising Daemon.
```

That looks healthier, shall we have a look at what it's doing on the network?

```
# lsof -Pni
nothing related to radvd!
```

Apparently it doesn't use IPv6 sockets at all: can't believe that, let's have a closer look:

```
# lsof | grep radvd
…lot's of lines, including a couple like this one:
radvd … raw6 …
```

Ah, so **radvd** does its work by using "raw" IPv6 sockets. It does this so it can specify exactly what to put in the header fields, even if the operating system doesn't have library support for newer IPv6 header options.

Anyway, let's get back to our testing. Start up Client1 connected to the same internal network. When its interface comes up, it should send out a Router Solicitation, which should cause **radvd** to send out a Router Advertisement. Let's see what addresses Client1 has generated for itself:

```
Client1$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:99:c2:7d
          inet addr:192.168.1.11  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fd6b:4104:35ce:0:a00:27ff:fe99:c27d/64 Scope:Global
          inet6 addr: fe80::a00:27ff:fe99:c27d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:624 errors:0 dropped:0 overruns:0 frame:0
          TX packets:250 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:820367 (820.3 KB)  TX bytes:29767 (29.7 KB)
```

Exercise

As an exercise make sure the interface has a Unique-Local Address. Did you get a default route? Use the following command to make sure Client1 has got an address and a useful default route. Mind you that we haven't had to do any configuration on Client1 to get that.

```
Client1$ ip -6 route
fd6b::4104:35ce/64 dev eth0 …
fe80::/64 dev eth0 …
default via fe80::a00:27ff:fe69:e1ae dev eth0 …  success
```

You might perhaps be thinking it odd that we did in fact get a default route. After all, you would think that you should have to manually configure Client1 to get a default route, but this is not the case. With IPv6, it makes it easier to have multiple routers that a host can use for a default route, and they each advertise a "default router priority", such as low, medium or high, which allows for graceful failover.

So now that Client1 has a fd6b:… address (a "Unique-Local Address", or ULA for short), we should be able to make connections using it. The only other thing in our network in the moment is Server1. Does it have a ULA?

```
$ ifconfig inside
inside    Link encap:Ethernet  HWaddr 08:00:27:50:e0:93
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe50:e093/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:297 errors:0 dropped:0 overruns:0 frame:0
          TX packets:701 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31067 (31.0 KB)  TX bytes:828807 (828.8 KB)
```

No, it doesn't have a ULA, only a LLA (Link-Local Address). Why is that? It is because Server1 is configured as a router, no longer a host. As such, it does not generate SLAAC addresses. So we need to configure a static address, which is really very easy. On Server1, edit `/etc/network/interfaces`, and make the alterations as indicated:

```
…
auto inside
iface inside inet static
    address 192.168.1.1
    netmask 255.255.255.0
iface inside inet6 static
    address fd6b:4104:35ce::1
    netmask 64
```

Affect the change using **ifup**:

```
# ifdown inside
…
# ifup inside
…
$ ifconfig inside
inside    Link encap:Ethernet  HWaddr 08:00:27:50:e0:93
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fd6b:4104:35ce::1/64 Scope:Global      Success
          inet6 addr: fe80::a00:27ff:fe50:e093/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:297 errors:0 dropped:0 overruns:0 frame:0
          TX packets:709 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31067 (31.0 KB)  TX bytes:829623 (829.6 KB)
```

Great, so now both Client1 and Server1 have a ULA. Let's just check that we can communicate using them with **ping6**; because this is a ULA, and not a LLA, we do not need to supply a scope identifier (eg. `%inside` or `%eth0`, as we have done previously).

```
Client1$ ping6 -c1 fd6b:4104:35ce::1
PING fd6b:4104:35ce::1(fd6b:4104:35ce::1) 56 data bytes
64 bytes from fd6b:4104:35ce::1: icmp_seq=1 ttl=64 time=0.669 ms

--- fd6b:4104:35ce::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.669/0.669/0.669/0.000 ms
```

Exercise

So now we have Client1 and our server with both IPv4 and IPv6 addresses. As an exercise let's just do some brief testing.

```
Client1$ ping6 -c1 fdb6:4104:35ce::1
… Destination unreachable: No route
Blast! Typing mistake.
```

```
Client1$ ping6 -c1 fd6b:4104:35ce::1
… 64 bytes from fd6b:4104:35ce::1 … success!
Client1$ ping -c1 192.168.1.1
64 bytes from 192.168.1.1 … success!
```

These IPv6 addresses are rather more cumbersome to type and recognise (but thankfully after a while, the shorter static addresses, are reasonably easy) . Let's make it easier on ourselves: **on both Client1 and Server1**, *add* the following into `/etc/hosts`, remembering to use the correct address for Client1, which will be different from that shown below:

```
fd6b:4104:35ce::1 ip6-server1
fd6b:4104:35ce::a00:27ff:fe99:c27d ip6-client1
```

What this does is to make a mapping between an address and a name. It is important to realise that these changes are only visible to these two machines: if you were to have more machines they would not see the change. We shall improve on this situation when we encounter DNS.

```
Client1$ ping6 ip6-server1
64 bytes from ip6-server1 … much easier!
```

```
Server1$ ping6 ip6-client1
64 bytes from ip6-client1 …
```

What we've done is to create a simple little shorthand. You may be wondering why we bothered to put the (informal) prefix of "ip6-". That is simply for ease of debugging, making it easier to recognise/specify the use of IPv6 or IPv4. At this stage, you don't need to do anything similar for IPv4. This is just a little convenience for ourselves to make it easier to deal with IPv6.

8.4. Pruning Services

Most operating systems seem to come loaded with services enabled out of the box; most of which you don't need and therefore ought not to be running. Ubuntu, on the other hand, has a good policy of not shipping with any network services reachable from outside the machine by default, though naturally if you ask for a particular network service to be installed, it will be reachable.

In order to give us something interesting to look at in this section, you will first need to run the following "mystery commands" on Server1, which will install and configure some services for us to look at:

```
# apt-get install openbsd-inetd netcat-openbsd openssh-server
# sed -i -e 's/^#daytime/daytime/' /etc/inetd.conf
# /etc/init.d/openbsd-inetd restart
```

What that does, in short, is to install some software, which includes an "Internet Super-Server"[2], which we shall be using shortly; alter its configuration file a little to enable a particular service by uncommenting a line; and restart it, affecting the change. We also install the OpenBSD version of the useful netcat utility (the "TCP Swiss Army Knife"), as it again has much better IPv6 support than other versions of netcat[3]. Finally, we also installed the OpenSSH SSH service. We can now run **lsof** and see what is listening on the network:

```
# lsof -i
```

[2]This particular version comes from OpenBSD, and supports IPv6 much better than the other available version: Inetutils-inetd.
[3]As an example, the netcat6 package doesn't allow you to specify an IPv6 address to connect to, only a hostname.

```
COMMAND    PID USER   FD TYPE … NODE NAME
dhclient  294 root   4u IPv4 … UDP *:68
sshd       902 root   3u IPv4 … TCP *:ssh (LISTEN)
sshd       902 root   4u IPv6 … TCP *:ssh (LISTEN)
inetd     1275 root   4u IPv6 … TCP *:daytime (LISTEN)
```

Note

If you don't see any lines output, check that you have run **lsof** with root privilege. Otherwise, it will only report on your own processes, of which you will most likely have none that are using network sockets.

`dhclient` is the DHCP client for the NAT network attachment (we'll learn more about DHCP in a later lab). Likewise, we'll look at that `sshd` entry a little later, but right now we want to have a look at the `daytime` entry, and find out more about it.

You'll notice that **inetd** is handling the connections for the `daytime` service. The configuration file for **inetd** is `/etc/inetd.conf`. Before we disable anything in `/etc/inetd.conf`, we should know that a service is represented as a pair of **`protocol`/`port`**, where `protocol` is either `tcp` or `udp`, and `port` is the port number. This is a common way of specifying port numbers of services in documentation. To find the protocol and port number of a service, you can use the `-Pni` options to **lsof**. You can google **`protocol`/`port`** to find out information about the service.

It is useful also to look at what interfaces services are listening on, for that tells us much about where a service is being offered. For the `daytime` entry above, a * indicates it is listening on all IPv6 interfaces (furthermore, in a common dual-stack feature, because there is no IPv4 service on the same port number, IPv6 will also likely get IPv4 requests as well). If you see a particular hostname (or IP address if you are using the `-n` option to **lsof**), then it is only accepting connections on that interface. Commonly, services that need to be running, but don't need to be remotely accessible by default will listen on 127.0.0.1 or ::1. Such an address is commonly shown as `localhost`, `ip6-localhost` or the *canonical hostname* of the machine, depending on the contents of `/etc/hosts`, which differs on different distributions.

inetd is configured via the file `/etc/inetd.conf`. An entry in that file is disabled either by commenting it out or removing it entirely. To make it easier for package installation scripts to be run, various configuration files can be managed using system-provided scripts, such as **update-inetd**[4]. This is easier than having to have each inetd-related package having to edit the file itself, separating policy and implementation. In this lab, we disable the "daytime" service manually by editing `inetd.conf`.

Once you have modified `inetd.conf`, you need to tell **inetd** that it has changed. This is generally done by the services startup script, such as by using the command **/etc/init.d/openbsd-inetd reload**.

Note

In summary, to disable or enable a service from **inetd**, you generally want to use a pattern of commands such as the following:

1. **vim**
   Use **vim** or **nano** to edit `inetd.conf` and comment or uncomment the corresponding line of the service (judging by **`protocol`/`port`** of the service).

[4]This would replace the call to **sed** we used earlier, and would be expected to be more robust.

2. **pstree**
   Use this to verify that the right number of daemon processes is running. In the case of **inetd**, there should only be one **inetd** process in a quiescent system (meaning the system is not serving any requests.)

3. **sudo /etc/init.d/openbsd-inetd stop**
   Use this to shut down the service.

4. **sudo killall inetd**
   You can use this to kill off any remaining **inetd** processes so that processes get a second chance to terminate gracefully. Give a little time for them to be shut down first though.

5. **pstree**
   Has it gone yet? If not, use **sudo killall -KILL inetd** to kill it forcefully. Check that it has really died this time with another **pstree**.

6. **sudo /etc/init.d/openbsd-inetd start**
   This will start the service again. Note that if you have commented out all services in `inetd.conf`, **inetd** will fail to start, which is ok.

7. **pstree** and **lsof -ni**, to verify that the process is running, and listening as expected.

8. Check your system logs, typically in `/var/log/syslog`. You can use the **tail** command to view the end of this, but beware that viewing logs using **tail** exposes a security weakness[5]

Following this general procedure throughout the rest of the course will greatly help you in diagnosing problems, and save yourself a lot of time in the future.

Now that we've disabled the "daytime" service, let's just ensure that it is no longer available. Here, We're using a particular invocation of **lsof** that just tells us about things on the "daytime" port:

```
# lsof -Pni:daytime
```

Because there is no output reported (and yes, we are running this with root privilege), we can infer that nothing is listening on the "daytime" port.

What about that SSH service?

Here is what is currently listening:

```
# lsof -Pni
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient …
sshd      9723 root    3u  IPv4  23591      0t0  TCP *:22 (LISTEN)
sshd      9723 root    4u  IPv6  23593      0t0  TCP *:22 (LISTEN)
```

[5]Discussed in Hacking Linux Exposed [http://www.hackinglinuxexposed.com/]. You should instead create an alias in your `~/.bashrc`, such as `alias vlog='sudo less --follow-name +G +F'`.

Let's assume that we don't need the SSH service; how best to disable it? Unlike the "daytime" service, which was run by **inetd**, the SSH service is run by the **sshd** process. We have a number of options available to disable the service, listed in no particular order of suitability:

1. Remove the links from under `/etc/rc*.d/Snnname` that start cause the associated script in `/etc/init.d/name` to be run.

   This is generally the best way to disable a background service either permanently or have it not running by default. You can stop a service with **/etc/init.d/name stop**. It can still be run by hand by calling **/etc/init.d/name start** etc.

   There is generally a tool to help you manage the contents of `/etc/rc*.d/`. On Debian-based systems, you can use the **update-rc.d** script[6].

2. Uninstall the package, retaining its configuration files. This can be done using **apt-get autoremove packagename**. This is useful if you think you may want to keep the configuration files and data files for a later time. It also means the program cannot be (accidentally or otherwise) run. Note that we have used `autoremove` instead of `remove`, which removes any other packages that were installed to satisfy a dependency that are no-longer needed.

   This is generally the best way to remove a service if it doesn't need to be run anymore. One particular risk is that the package could accidentally be installed at a later date if it gets installed due to installation of other packages via package dependency.

3. Uninstall the package, purging its configuration files. This can be done using **apt-get autoremove --purge packagename**. This is useful when you want to permanently remove a package, or you don't want stale (possibly broken) configuration files hanging around, which might confuse issues later if you decide to re-install the service. Your original configuration files should be in backup and ideally in version control before the uninstallation.

In our case, we don't want the SSH service installed at all, so we shall practice removing the package (in a later lab, we shall install it when we need it).

To make things a little more real for a beginning administrator, let's assume that we don't know what package contains this `sshd` process. Where is this **sshd** process anyway?

```
$ ps -eo command | grep sshd
/usr/sbin/sshd
…
```

Okay, so it seems that the `sshd` mentioned in the **lsof** output above is actually `/usr/sbin/sshd`. What package contains that file?

```
$ dpkg -S /usr/sbin/sshd
openssh-server: /usr/sbin/sshd
```

Now we know the package name, we should be able to remove it:

```
# apt-get autoremove openssh-server
…
```

Verifying that it is no-longer running:

[6] On Redhat systems, the equivalent is **chkconfig**.

```
# lsof -Pni:22
No output, therefore nothing listening on port 22 (ssh)
```

8.5. The Internet Super Server

In this section we shall create a new **inetd** service, which illustrates how easily creating an service can be when using an Internet Super Server. In the next section, we shall go on to restrict access to it using TCP-Wrappers.

Procedure 8.1. Tiny File Server

1. As root (ie. using **sudo**), create the file `/usr/local/sbin/tinyfs` and mark it executable. This file is a simple Perl script; the contents are listed below (To save the editing time, you could skip the comments):

```
#!/usr/bin/perl -w
#
# Naive file server. Gets a filename in the first line from stdin,
# removes any CRLF line-ending, and sends out the file that has been
# requested. Provides NO authentication, NO authorisation, NO extra
# access control, and NO accounting; this should be run as the 'nobody'
# user and guarded with at least TCP Wrappers. Best not to use this
# in production, it is here to illustrate the basic principles of inetd.
#
# Lines are terminated by CRLF, the internet standard line-ending.

use strict;

$/ = '\r\n';            # read records (lines) terminated by CRLF
my $filename = <>;      # read a record (line) from stdin
chomp $filename;        # remove line ending

# Open the file, or die trying. Any error would be sent to stderr, which
# is connected to the client also, so typically you don't want to send
# anything to stderr!
#
# Seeing as this would have to implement some sort of application-layer
# network protocol, we should have some sort of result code. We'll just
# either say "OK\r\n" or "ERROR: reason\r\n" on the first line of the
# result.

if (open FILE, '<', $filename) {
    print "OK\r\n";
} else {
    print "ERROR: $!\r\n";
    exit
}

# Note that in this case, the files are output verbatim, no line-ending
# translation is performed (like FTP 'BINARY', not like FTP 'TEXT')

$/ = 4096;              # read 4kB of data at a time
while(<FILE>) {         # while we can read a record...
    print;              # ...print it
}
close FILE;
```

   This will offer a very naïve and rather insecure file transfer service. You will notice that it reads from stdin, and writes to stdout. This is the basic principle of any server that uses the Internet Super Server. The stdin, stdout and stderr(!) get connected to the TCP (or UDP) socket, and so communicates with the remote peer.

Before we attempt to use it over the network, let's just demonstrate how it works, showing that it doesn't have to know anything about networking.

```
$ echo '/etc/hostname\r\n' | tinyfs
OK         our result code
server1    /etc/hostname is very short, only one line
```

2. Add the following entry to `/etc/inetd.conf`.

```
tinyfs stream tcp4 nowait nobody /usr/local/sbin/tinyfs tinyfs
tinyfs stream tcp6 nowait nobody /usr/local/sbin/tinyfs tinyfs
```

What does this mean? The `tinyfs` will be its service name: it will tell **inetd** which port to listen on. The `stream` specifies that it is a stream protocol (and not a datagram protocol); the `tcp4` and `tcp6` says we are using TCP (which is always a stream protocol) over IPv4 or IPv6 respectively. `nowait` tells **inetd** it may process multiple connections at once, rather than having to wait for one to finish before dealing with another. `nobody` is the user the service should run as. If this were `root`, then sensitive files, such as `/etc/shadow`, which are normally only readable by the root user, would be world readable, because **tinyfs** does not perform any authentication or authorisation. Running as `nobody` should protect against this, as the `nobody` user should have no privileges what-so-ever (meaning it should only end up using the Others permission bits).

The final two parts is the location of the program to be run, and the arguments (the first argument here, `tinyfs`) gives the 0th argument, called `argv[0]` in the C programming language (which is the native system-level programming language on Unix-like systems), which gives the name of the program.

3. Add the following to `/etc/services` on both server and client. It must be on at least the server (as we refer to it in `/etc/inet.conf`); the client has it only for our convenience. This will let each side refer to port 900 as `tinyfs`. Note that ports are given both UDP and TCP allocations, even though they probably don't use both.

```
tinyfs   900/tcp   # Tiny File Service
tinyfs   900/udp   # Tiny File Service
```

4. Reload **inetd**.

```
# /etc/init.d/openbsd-inetd restart
```

5. Test that **lsof -Pni** shows the listening socket. Remember to use root privileges.

```
# lsof -Pni
COMMAND     PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient  …
inetd     10422 root    4u  IPv4  25808      0t0  TCP *:900 (LISTEN)
inetd     10422 root    5u  IPv6  25811      0t0  TCP *:900 (LISTEN)
```

6. Now to test it. Aim to test all cases (e.g. local and global IPv4 and IPv6 addresses) in the server.

Exercise

As an exercise try connecting using the **nc** "netcat" program, which is for IPv4 and IPv6. Recall that we installed the OpenBSD version, which supports IPv6 quite nicely.

Make sure you can successfully retrieve `/etc/hostname`. Also you may try and fail to access a file such as `/etc/shadow`.

```
Loopbacks
$ echo '/etc/hostname\r\n' | nc -q-1 127.0.0.1 tinyfs
…
$ echo '/etc/hostname\r\n' | nc -q-1 ip6-localhost tinyfs
…
$ echo '/etc/hostname\r\n' | nc -q-1 ::1 900
…

Our Unique-Local Addresses…
$ echo '/etc/hostname\r\n' | nc -q-1 ip6-server1 900
…

Don't forget about your link-locals…
$ echo '/etc/hostname\r\n' | nc -q-1 fe80::a00:27ff:fee3:4d42%outside tinyfs
…
```

We've not shown all addresses that our server has. There will be at least two others (remember, Server1 has at least two interfaces). It is instructive to point out that we generally don't need to support connections coming in via the Link-Local addresses. It is important not to forget about all the various addresses that we have, in relation to restricting access to a service.

8.6. Access Control using TCP Wrappers

As it currently stands, `tinyfs` is currently open to the world (which is to say, it has no access control with regard to which machines can connect to the service). We shall now use TCP-Wrappers to protect `tinyfs`, and then look at how TCP-Wrappers can be used to protect other services that use `libwrap`, such as **sshd**.

tcpd is a program that is used as an access-control wrapper to protect services started from **inetd**. It grants access based on on two files: `/etc/hosts.allow` and `/etc/hosts.deny`.

If you were to look at the manual page for **tcpd**, you might find the following description.

> There are two possible modes of operation: execution of **tcpd** before a service started by **inetd**, or linking a daemon with the `libwrap` shared library as documented in the hosts_access(3) manual page. Operation when started by **inetd** is as follows: whenever a request for service arrives, the **inetd** daemon is tricked into running the **tcpd** program instead of the desired server. **tcpd** logs the request and does some additional checks. When all is well, **tcpd** runs the appropriate server program and goes away.
>
> —tcpd(8)

TCP-Wrappers will test `/etc/hosts.allow` first, and if a match is found, it will allow the connection. Otherwise, it will test `/etc/hosts.deny`, and if a match is found, it will drop the connection. Otherwise, it will allow the connection, so if you want to use TCP-Wrappers, you generally always want to ensure that a suitable deny-by-default rule is in place in `hosts.deny`.

Procedure 8.2. Using TCPd to Protect TinyFS

1. Start by putting the deny-by-default entry in `/etc/hosts.deny`:

```
…
```

```
ALL:ALL
```

2. Put in place a suitable rule, or set of rules, to allow the traffic you want. These rules go into `/etc/hosts.allow`:

```
…
tinyfs: 127.0.0.1 [::1] 192.168.1.0/24 [fd6b:4104:35ce::]/64
```

   The policy we have put in place here is that the server can access itself (via loopback), and our regular client ranges can access tinyfs also. Note that we haven't included the Link-Local Addresses, as these should not generally be used for applications.

   Note it is useful to pause here, and write down exactly what has effectively been *denied* entry, based on your testing cases in the previous section.

3. At this stage, `hosts.allow` and `hosts.deny` will not be consulted, because **inetd** has not yet been instructed to use **tcpd**. It is **tcpd** that checks these files, and if it allows the connection, it will pass execution (via the `exec` system call) to **tinyfs**. To do this, we change the lines regarding **tinyfs** in `inetd.conf` to the following:

```
tinyfs stream tcp4 nowait nobody /usr/sbin/tcpd /usr/local/sbin/tinyfs
tinyfs stream tcp6 nowait nobody /usr/sbin/tcpd /usr/local/sbin/tinyfs
```

4. Reload **inetd** to affect the changes you made in `inetd.conf`. Check the system logs to ensure there were no problems, and that you can see the service with an appropriate invocation of **lsof**.

5. **Exercise**

   As an exercise use **nc** (or **telnet**, which is also useful for this sort of thing) as we have done previously to connect to the service. Test all cases that should be allowed. Also test other cases that should be denied (such as the Link-Local Addresses). You should use **nc** on Client1 to try the case of `ip6-server1`. Check the system logs (in `/var/log/syslog`) reporting the actions.

Services can also be protected using `libwrap`, but we'll cover those issues when we come across the particular services, such as SSH.

Tinyfs is not for production use

Do not deceive yourself into thinking that tinyfs is ready for production use. There are many features that are still lacking, such as accounting, filtering (eg. sharing only part of the filesystem), authentication (ie. who), access control (ie. who can do what) and privacy (encrypting traffic).

8.7. Self-assessment

1. Ensure you have done the following successfully:

- the interfaces with appropriate interface names and IPv4 addresses;
- the right IPv4 routing table, including the default route;
- you can access the APT repository over the network;

- the IPv4 address you gave Client1 is correct;
- the **apt-get update** command working on Client1, which shows that the NAT we briefly configured is working;
- the correct IPv6 interface details for Client1 after we set up router advertisements;
- the correct IPv6 routing table for Client1 after we set up router advertisements, which should include a default route;
- the successful IPv6 ping using the Unique-Local Addresses of Server1 and Client1;
- successful testing access to **tinyfs** using **nc**, *before* protecting it with **tcpd**;
- the correct contents of the three files `hosts.allow`, `hosts.deny` and `inetd.conf`, after implementing protection using **tcpd**;
- and finally, find the log entries showing that some connections have been accepted, and others have been rejected.

8.8. [Appendix] Building a Replacement radvd Package

*When we were running this lab with an older version of the **radvd** package it had some problems. Fortunately the package in the repositories for Ubuntu 18.04 has been updated such that the problem doesn't exist anymore. We feel that we should keep the following in the lab manual for information purposes.*
There is nothing in this section you need to perform, it is simply here to demonstrate one way in which you can build a Debian package. Other systems, such as Redhat-based systems, will have a similar technique, though the tools will be different. You are not required to read this, unless you are interested.

There were multiple ways we could have built a newer version of **radvd**. To give you some idea, here is what we tried:

1. Running the standard Ubuntu (10.04) version of **radvd** by hand, with debugging turned on, we saw that it was crashing with a Segmentation fault when it received the first Router Solicitation packet. We adjusted our `radvd.conf` to make a very simple configuration, much like the simplest configuration example supplied with the software, but it still crashed.

2. We tried using an unofficial APT repository that was built to fix some problems. We found this while following Ubuntu and Debian bug reports about the program. It didn’t solve the problem.

3. We downloaded the latest release of the source code and build it from source. It didn’t solve the problem either, though we did figure out where it was crashing, which gave us more information.

4. Using the information we learned in the previous failed attempt, we looked through the developers’ mailing list for the software, and found that there were some changes made to CVS around the area where the crash was occuring, and some other messages about different errors which lead to a crash in the same place. So we checked out the latest CVS (fingers crossed it has fewer bugs and not more). We built it from source, and found that the problem appears have to been fixed.

5. Because we didn't want you to have do a lot of things associated with installing this software from source (in this case, it would involve configuring the software at build-time to install things into the correct places, writing startup scripts and creating a special user), we downloaded the set of patches that Ubuntu and Debian use to apply to the "upstream" source-code. These patches largely are files that say how to build it into a Debian-style package.

   We could have gone further and put the package into a repository, which would allow you to install it using APT, as well as to install any dependencies, and allow for easier updates, but we decided not to at this point in time. If we were maintaining this on multiple machines, or were going to be rebuilding this package occassionally, that would be useful.

In brief, here are the commands we used to checkout the latest source from CVS and build it into Debian package:

```
Make a working directory, anywhere will do
$ mkdir -p ~/tmp/radvd-cvs
$ cd $_     $_ expands to the last argument of the previous command

Satisfy build dependencies etc. Some of this you learn through trial and error…
# apt-get install build-essential cvs autoconf libtool debhelper devscripts cdbs \
> flex bison
If asked to configure Postfix, just respond with "Local only" and accept defaults

Download the latest version from CVS
$ cvs -d :pserver:anonymous@cvs.litech.org:/work/cvsroot login
… Use an empty password when prompted
$ cvs -d :pserver:anonymous@cvs.litech.org:/work/cvsroot checkout radvd
…
$ cd radvd

Download the standard Ubuntu/Debian alterations that roughly match the version of radvd.
Link learned from http://packages.ubuntu.com/maverick/radvd
$ wget http://archive.ubuntu.com/ubuntu/pool/main/r/radvd/radvd_1.6-1.diff.gz
…

Apply the patch
$ zcat radvd_1.6-1.diff.gz | patch -p1
…

Generate some initial build scripts (./configure and Makefile)
This is because the file (generated) file ./configure is not tracked in CVS
$ autoreconf -vif

Set the 'local' version number, so we know we're dealing with ouwn version, not from Ubuntu
$ dch -l cvs20101217 'Using CVS head as of 2010-12-17'

Sit back and let the package build
$ dpkg-buildpackage -us -uc
…

Now you can install the package
# dpkg -i ../radvd_1:1.6-1cvs201012171_i386.deb
```

Lab 9 Catchup Lab

This lab is set aside for help people catch up; priority will be given to students doing previous labs and assignments.

Lab 10 Scheduled Tasks and Log Management

Today we learn about how to schedule programs to run in the background at a certain time. We also learn about logging on a UNIX-like system, how to manage the system logs, and cover a couple of very useful logging related tools for rotating and reporting on the various log entries. We'll also learn about simple regular expressions, which gives us a powerful tool for matching text against a pattern.

> **Note**
>
> In this lab, we'll just be doing all our work on your Server1 unless specified otherwise.

10.1. Cron

Being able to run jobs at scheduled time, and in the background, is a great assistance to a system administrator (and to the user as well). Cron is able to assist you to run jobs at scheduled times. Here is just a sample of the types of things you might use cron to do:

- Fetch any mail from your ISP every five minutes. You could then automatically have it sorted and processed.
- Download package updates at night, when traffic costs may be lower, or the network less busy.
- Check the system periodically, and page the administrator when something is wrong.
- Initiate system maintainance tasks, such as rotating logs or performing backups.
- Run commands once only, at a certain time (using **at**).
- Run commands when the system is not heavily loaded (using **batch**)

To use Cron, you need to know how to use **crontab**, which is the command for editing your own Cron Table (crontab). You also need to know the format of a crontab file. See crontab(1) for information about the command, and crontab(5) for more information on the format of a crontab.

For most system administration tasks, we don't use personal crontabs. Instead, we use some standard directories into which we can place jobs. These directories are `/etc/cron.d/`, `cron.hourly/`, `cron.daily/`, `cron.weekly/` and `cron.monthly/`. With the exception of `cron.d/`, these contain executable scripts, which are run every day, week and month. `cron.d/` stores normal crontab entries, including a time and specification. This modularity helps immensely with package management, due to its drop-in nature. You do not use **crontab** to edit these files, just a regular text-editor.

There is also the file `/etc/crontab` which is the crontab for the system (ie. the root user). The main items it contains are entries to start all the daily, weekly and monthly jobs. You do not generally need to edit `/etc/crontab`; use the directories instead. Shown below is what you would see in `/etc/crontab`. Note that it differs from a normal users crontab, in that the `user` field does not appear in a users crontab.

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user command
Note, we have edited this for clarity.
17 * * * * root run-parts --report /etc/cron.hourly
25 6 * * * root run-parts --report /etc/cron.daily
47 6 * * 7 root run-parts --report /etc/cron.weekly
52 6 1 * * root run-parts --report /etc/cron.monthly
```

run-parts is a helper program that runs all the executable scripts in a directory, such as `cron.daily/`.

The scripts in `cron.when/` etc. are run with the privileges of the root user, and run one after another (sequentially, based on its filename). The files in `cron.d/`, which can be seen of as fragments of `/etc/crontab`, have a user field in them, commonly root, but can easily be changed.

Each user can have a crontab file under `/var/spool/cron/crontabs/`[1]. This directory will most likely be empty on your system, because users don't most often will not have created any crontab entries. A user modifies or creates a crontab using the **crontab -e** command. This will start the editor specified by the `VISUAL` or `EDITOR` environment variable, or the powerful-but-not-very-newbie-friendly **vi**[2]. The format for the file is fairly simple, but easy to forget, so you might like to put a comment at the start of the file that reminds you of the format.

Here is an example of the crontab you would be editing using **crontab -e**. Remember, as this is a users crontab, it does not allow you to specify a user field. Consider carefully the difference between each time specification. Note that you are not required to make any modifications at this stage.

```
* 8 * * *     command  Will run every minute from 8:00am to 8:59am
0 8 * * *     command  Will run at 8:00am
0 */8 * * *   command  Will run every 8th hour
* */8 * * *   command  Will run every minute of every 8th hour
```

All entries can be thought of as being checked every minute. The * is a wildcard which always matches. `*/2` means every 2 hours (when in the hourly column). Likewise, `9,17,21` matches 9am, 5pm, and 9pm in 24-hour format. If you want something to run once in a particular hour(s), then the minute field must be absolutely specified. Consider the difference beween the first two entries above.

10.1.1. Self-assessment

1. When will the following crontab entry run?

```
* * 13 jan * command
```

[1] This can be disabled or allowed only to particular users.
[2] **vi** is a common Unix default, but other systems can differ. Debian-based systems in particular, use its "alternatives" subsystem to run whatever is configured as **sensible-editor**, which is **nano** by default.

2. When will the following crontab entry run? Notice that both day of week and day of month are specified; check crontab(5) for how this edge-case is treated on your system.

```
30 4 1,15 * 5 command
```

3. Imagine you have a virtualisation host with a lot of identically configured virtual machines (guests). You notice that you get massive load spikes on a regular basis, and upon investigation during one of these load spikes, you determine that they are caused by cron jobs all starting at the same time (and thus every machine simultaneously wants to wake up and run something, causing the host to become very busy, even though each virtual machine has very little to do). How could you improve this situation? (You do not need to implement anything for this question, just brainstorm some solutions.)

10.2. Syslog

The venerable BSD Syslog remote logging protocol is implemented in Ubuntu using the **rsyslogd** program. Other systems use similar pieces of software, but conceptually they are quite similar. The remote aspects of Syslog have a number of limitations and problems related to reliability and security, and so there have been improvements made in a series of RFC documents to come up with the Reliability Enhanced Logging Protocol (RELP), which is what sets **rsyslogd** apart from the rest. However, few devices support that yet, and most of the time all we need is the usual Syslog protocol. Many Linux distributions today use the newer **rsyslogd** over its pre-decessor **sysklogd**[3]; Ubuntu is no exception here.

rsyslogd sits in the background, waiting for any program to give it a log entry. When it does, that program will specify a facility and a priority.

Examples of facilities are `auth`, `cron`, `daemon`, `kern`, and `mail`. You can find more detail in the syslog.conf(5) manual page. In addition to the standard keywords, there are also `local0` through `local7`, which you can use for any local purpose you desire. For example, you could configure all your active network elements (routers, switches, access-points etc.) to send to `local0`, phone systems to log to `local1`, building management services (security, air conditioning etc.) to `local2`, a SQL database might be configured to send to `local3` and your own software you've developed might use `local4`… you can see that it wouldn't be too hard to run out of facilities if you really embrace remote logging, but most of the time you only accept local log messages, and if you're using that many log sources, you should probably have multiple log servers.

A priority says how important the log entry is. Priorities range from `debug` (the lowest), through `warning` and `err`, to `emerg` (system about to crash). There are 8 different priorities you can assign, though most software should not get more severe than `err`.

rsyslogd is configured with the rsyslog.conf(5) configuration file. However, unlike other versions of **syslogd**, **rsyslogd** uses this file for overall settings, but the actual message routing elements are split out into indivitual files dropped into the directory `/etc/rsyslog.d/` ; have a look at the files that are there.

Whenever you change something in here, or manage any files in `/var/log/`, you should reload **syslogd** (**reload rsyslog** or **/etc/init.d/rsyslog reload**).

The format of each line in the file is `facility.priority target`. The `facility.priority` part is often called the SELECTOR, and the `target` is often called the ACTION. You will see these terms used in the documentation for **rsyslogd**.

[3] **sysklogd** is comprised of two parts, a portable userland Syslog server called **syslogd** and a Linux-specific kernel logging proxy called **klogd**.

You can specify multiple facilities in a line by separating them with a comma. For priority, you can (these are for reference, don't actually make any edits) say things like the following:

warn
: warn priority and up.

=warn
: Only warn priority.

!=warn
: Anything but warn priority.

!warn
: Only priorities above warn (not including warn).

none
: Used for excluding facilities.

For the target, you can specify either a log file, a remote host, a usercode(s) to who the log entry will be written on their terminal, or a terminal device, such as the system console.

/var/log/messages
: A fully qualified path to a log file. If a - prefixes the log file, the log will be written synchronously, meaning the changes will be written to disk immediately[4].

@valhalla
: Send the log message to a machine called valhalla. **rsyslogd** can receive log messages over the network, but like any Syslog implementation, needs to be configured to allow this first.

theauthor,root
: Write the message to these users terminal. You can use * to mean everyone logged in (you can use the **wall** command to send messages like this in an ad-hoc manner).

/dev/console
: A terminal device. The message will be written to the appropriate terminal.

^command
: Starts a program and sends the log message to it. Because the program is started each time, the program should be very fast to run and should not run too frequently.

|fifo
: Sends the log message to a program that is already running. The message is sent using a form of Inter-Process Communication (IPC) called a "named fifo" (First-In First-Out) which can be found in the filesystem. Because the program is already running, it is not bad if a lot of messages are sent this way, or if the program takes a while to run or start up.

Read the first few paragraphs of the "SELECTORS" and "ACTIONS" sections of the rsyslog.conf(5) manual page. We won't be altering any of the other rules today, but just looking at the defaults should give you a reasonable idea of how it works. The man page has an examples section, should you be interested.

Logs are generally kept in `/var/log/`. As a brief guide, the important ones are as follows, the most important to keep an eye on when debugging is `syslog`. How the logging is organised is particular to the distribution in use.

[4]Actually, guaranteeing this is very difficult on modern hardware. Additionally, a lot of synchronous writes can slow the system, so use it only when something awful might be about to happen.

messages
By default, everything, if not put anywhere else, will fall into here. Typically uninteresting.

syslog
System messages, such as from kernel and daemons. Generally the most interesting log file.

secure
Security related messages regarding authentication go in here.

utmp, wtmp, faillog
Binary database of current logins, login history and login failures. Used with programs such as **who**, **last**, **lastlog** and **faillog**.

Some services often have their own logging, such as web servers and proxy caches, as the volume of data they log is typically quite high, and more flexibility may be desirable.

10.2.1. Self-assessment

1. In this question, we are going to route a copy of all of our non-sensitive logs to a spare virtual console, which can make it easy to keep an eye on what the system is doing.

We are going to use the virtual console tty2[5].

A virtual console is a type of terminal, much like Gnome Terminal or **xterm** is a type of terminal. Terminals are often referred to as "TTY" devices[6]. Linux systems often have a number of virtual consoles configured, and you can switch between them using **Alt+F1** through **Alt+F6**, depending on how many have been configured (note that on a Mac keyboard you need to use **Alt+fn+F6**). You can also use **Alt+←** and **Alt+→** to go to the next and previous virtual console respectively.

Comment out the `PrivDrop` lines in `/etc/rsyslog.conf`,[7] then add the following to a new file `/etc/rsyslog.d/99-tty2.conf`:

```
*.notice;auth,authpriv.none    /dev/tty2
```

Exercise

As an exercise restart Rsyslogd using **sudo service rsyslog restart**. Use the **logger** program to submit a log message, and have a look on the virtual console tty2, where we send a copy of our log messages. Make sure you see the log message displayed on virtual console tty2.

You can use the **logger** command to help your understanding of how different facilities and priorities are routed by the Syslog server. To do this, simply submit a unique log message, and then use **grep** to see which file(s) the log message appears in:

```
$ logger -p daemon.warning "syslog test N"
# grep -r "syslog test N" /var/log/
```

[5]You may find that you open the volumn controls instead of switching to the virtual terminal. There are two ways to fix this, either use the **Fn** button (beside the **home** button above the arrow keys), or change the keyboard settings so that you "Use F1, F2, etc. keys as standard function keys".
[6]The term "TTY" comes from TeleTYpe terminal; a historical artifact.
[7]This leaves us open to security isues, but we shant worry about that now.

2. **[Optional challenge]** How might you send a log message to a cell-phone as an SMS message? You don't need to set this up for this lab. Assume you can send a SMS message using the Gnokii software, which interfaces well with Nokia phones: even very cheap phones can send text messages this way, so long as you have the right cable.

```
$ echo "Hello, World!" | gnokii --sendsms phone-number
```

   Assuming you had **gnokii** configured to use a particular phone, you *could* hook this into **rsyslog** using an entry such as:

```
SELECTOR  ^gnokii --sendsms phone-number
```

   And the command will be run *every* time a log message matches the selector. However, while this will work, there is a possibly very expensive and very annoying problem with this. What is this problem? How might you deal with this problem?

   As an alternative mechanism that doesn't require any physical setup, sms gateways can be accessible using an API over the internet. Why might this mechanism be indesirable?

10.3. Rotating Logs

It's a fact of life, logs grow. Just like your lawn, the logs need processed every so often to keep them manageable, and to help you find any snakes in the grass. What you do with your old logs will reflect on your local policies, and possibly even laws eg. logs may need to be kept for several years, or on the other hand, logs perhaps may not be kept for longer than a few years due to privacy laws; you should seek advice from your lawyer or industry.

In Linux distributions, logs are usually rotated using a tool called **logrotate**. Have a look at the default Ubuntu configuration of **logrotate** by looking in the file `/etc/logrotate.conf`. Note that it includes all the files in `/etc/logrotate.d/`, to enable package-maintainers to easily install rules for having **logrotate** rotate its logs, simply by dropping a file into the `logrotate.d/` directory. Here is what the `/etc/logrotate.conf` file looks like. The file has been edited to save space.

```
Specifying global defaults
weekly               Rotate logs weekly
rotate 4             Keep 4 weeks worth of backlogs
create               Create new (empty) log files after rotating old ones
#compress            Uncomment to compress rotated files
A directory so packages can install log rotation policies.
include /etc/logrotate.d/
/var/log/wtmp {      Stores login history.
    missingok        Don't complain if wtmp file is missing.
    monthly
    create 0664 root utmp   perm user group
    rotate 1
}
```

The above is a simple example; we could do a lot more. We've selected the condition to rotate based on time (`weekly` and `monthly`), but we could also choose to rotate based on file size. You can also specify commands to be run before and after rotating the log file. Here is a fictitious entry which shows some of the other useful entries. This could be a file in `/etc/logrotate.d/` Remember that the following is an *example*, so I don't expect you to input it.

```
You can specify multiple files
/var/log/foo/access /var/log/foo/errors {
  size=100k           Rotate when it reaches a certain size
```

```
  sharedscripts       Run post and prerotate only once for all files in this set
  postrotate          You can run a list of commands after rotation
    killall -HUP food food would be the daemon for the foo service
  endscript           End the list using endscript
}
```

logrotate gets run by **cron**, using `/etc/cron.daily/logrotate`. The scripts in `cron.daily` get run early each morning, typically about 6am. You can alter this by editing `/etc/crontab`.

You can force **logrotate** to rotate (ie. ignoring the selection tags such as `weekly` or `size`) the files by using the **-f** argument to **logrotate**. Have a look, using **ls -l**, in the `/var/log` directory, and then run the command **logrotate -f /etc/logrotate.conf**. Have a look to see what changed. Repeat a few times, and describe what happens.

Although Ubuntu doesn't compress them by default, **logrotate** is able to compress logs using **gzip**, and thus get an extension of `.gz`. Compressed logs can be viewed using the **zless** or **zcat** program, and grepped using **zgrep** (we'll cover **grep** later in the lab.

10.3.1. Self-assessment

1. **On paper**, write logrotate entry that rotates the file `/var/log/auth.log` on a weekly basis. This file is where you would find any log messages relating to failed login attempts etc. Any further rotation options are up to you. Rationalise why you chose particular options, particularly the number of logs to keep.

2. This question is designed to help you understand the concept of log rotation. Before we continue, we must first disable (temporarily comment-out) the `notifempty` directive in `/etc/logrotate.d/rsyslog`. This directive tells **rsyslogd** not to bother rotating a file if that file is empty. Normally, that is useful, but it does get in the way of understanding the concept we are trying to illustrate.

After disabling `notifempty` and restarting **rsyslogd**, **cd** into `/var/log`. From there, run the following sequence of commands three or so times, until you see a pattern emerge. If you don't see a pattern after doing this, go on with the next question, and start democall to ask a demonstrator to explain.

```
$ ls auth.log* | while read file; do
> echo -en "$file\t"
> (zcat "$file" 2>/dev/null || cat "$file"; echo) \
>   | head -1 | cut -b-60
> done
# logrotate -f /etc/logrotate.conf
```

This somewhat lengthy command, besides being another example of how useful scripting can be, will print out all the files matching `auth.log*`, with their name, and the first 60 characters of the first line. Pay attention to the association between the file name and the contents. The output from a single run will look something like this:

```
auth.log        May 11 12:25:36 client1 sudo: mal : TTY=unknown ; PWD=/
auth.log.1      Mar 30 08:09:01 client1 CRON[3513]: pam_unix(cron:session):
auth.log.2.gz   Mar 26 08:09:02 client1 CRON[13480]: pam_unix(cron:session):
auth.log.3.gz   Mar 15 22:00:39 client1 gnome-keyring-daemon[1504]: removing
auth.log.4.gz   Jan 12 15:49:48 client1 gdm-session-worker[1341]: pam_unix(g
```

Run the lengthy command three or so times (you may like to put the commands into a script) and describe what happens to the files in `/var/log/` when they are rotated, based on the pattern you find from the outputs.

3. What would be a good thing to do with archived logs? What factors might you need to take into consideration?

10.4. Filtering Logs

Now that we have some self-maintenance in our logging, we can now work on keeping the administrator informed on what is going on by bringing attention to new and possibly undesirable events. This generally requires the administrator to check their e-mail regularly. Serious messages might be delivered via a Pager, SMS, audio alarm etc.[8]

There are two fundamental methods of log scanning, the first is looking for various patterns that may constitute something interesting. The second method involves filtering out the mundane entries from the logs, so we're left with only interesting or new types of log information. The latter method is better in the general case, as most unexpected events would not be configured to be picked up in the first case (that's why they're unexpected). It does mean though, that the admin needs to tailor the configuration to filter out the uninteresting messages they would normally get, which can involve a lot of work on a new system, but it does mean the administrator gets a better feel for the day-to-day behaviour of the system.

We're going to use the latter method, using a tool called Logcheck. Install **logcheck** using the following procedure.

Procedure 10.1. Install Logcheck and Perform Initial Testing

1. Ensure you have a fresh record of what packages are available in the configured APT repositories.

```
# apt-get update
…
Reading package lists... Done
```

2. Install and configure an email server package called exim4, which will be used by logcheck.

```
# apt install exim4
…
# dpkg-reconfigure exim4-config
```

   When configuring exim4, choose `Internet site`, use `server1.localdomain` as the mail name, leave empty the field of `IP-aadresses to listen`, use `server1.localdomain` as the only destination for which mail is accepted, leave `Domains to relay` and `Machines to relay` empty, and more importantly, choose *mal* as the destination for mails sent to *root* and *postmaster*. We will reconfigure exim4 in more detail in the lab for email server.

   After the configuration, make the changes effective and start the email server.

```
# update-exim4.conf
# systemctl start exim4.service
$ pstree
…
   -exim4
…
```

   You should see exim4 in the output of **pstree** if everything went well.

[8] See the **swatch** program if you want to run commands in response to certain log entries as they happen.

3. Install mail utilities.

```
# apt install mailutils
```

4. Install the Logcheck package and its dependencies.

```
# apt-get install logcheck
```

5. Find the cron entry that was installed when you installed logcheck. This file will be in one of the `/etc/cron*` files or directories (**ls /etc/cron***). Write down this entry; there is a question in the assessment relating to this.

6. Test whether logcheck works by running **logcheck** by hand. In Ubuntu, when we installed LogCheck, an e-mail server (Exim) was installed as well. Exim, per Ubuntu's default policy for that package, does not accept mail from users other than 'root' and 'postmaster' (a special mailbox name for the e-mail administrator). In addition, **logcheck** should be run as the user "logcheck", so run it using **sudo** as follows:

```
$ sudo -u logcheck logcheck
This could take up to a minute, but usually much less
You have new mail in /var/mail/mal
```

   A word of explanation about the final line regarding new mail: this line is output by your shell. After a command is completed, your shell will check your mail spool file to see if the file timestamp has changed, and if it has, it will output the message you see above.

   One of the more primitive tools we can use for checking our mail on a Unix-like machine is the **mail** command, which is sufficient for looking at the messages in your mail spool file. There are, of course, much nicer programs, including **pine** and **mutt**, but we shall have a look at those when we take a closer look at e-mail in a later lab.

```
$ mail
```

   To work in the primitive **mail** application, type a number of a message to view the message (**q** to exit the viewer). Type **q** to exit the mail prompt (&) prompt and return to your shell.

   Your first message will be very large, as it contains all the logs it hasn't already seen (and most of them will not be filtered as they are bootup messages that typically contain a lot of hardware-specific kernel messages). Subsequent runs will be much smaller, and if there is no output–which is very common–no mail will be sent.

   You may be wondering why the e-mail got sent to `mal`, and not `root`. It's because when you installed the operating-system for you, the installer automatically adds the user created during install to the file `/etc/aliases`, so any mail to root gets sent instead to that user. Have a quick look at it now, it's very small. We'll revisit it when we come to look closer at e-mail servers.

   If you didn't receive the message about the new mail, then the problem is in the `/etc/aliases`, add `root: mal` to the end of the file, then run **newaliases**. You will need to regenerate the log message.

Procedure 10.2. Instructing Logcheck to Ignore Particular Log Entries

One of the tasks that you will commonly be doing, quite repetively, on a new server is adjusting the logcheck rules, ignoring messages are not interesting.

1. So now let's try telling logcheck to ignore a certain entry. Let's assume you are regularly seeing a particular log message that you don't care about, and the log entry contains `-- HELLO WORLD --`.

   Create the file `/etc/logcheck/ignore.d.server/local`, and insert a single line with `-- HELLO WORLD --`. Spaces matter, and there must be no blank lines in the file.

2. Run **sudo -u logcheck logcheck**, just so that we can easily see (or rather, not see) our log message that we're about to submit.

   Check your mail, just so you can tell the old messages from the one that might appear on the next run of **logcheck**.

3. Use **logger -- '-- HELLO WORLD --'** to log the entry we want to ignore.

4. **Exercise**
   As an exercise run Logcheck, then check your mail again. You shouldn't get a message, but if do, check that it doesn't contain the log message we are wanting to ignore. If you didn't get a message, that's excellent: all new log messages (including the one we just logged) were ignored, and so a report wasn't sent. Make sure your logcheck configuration file (where you added the entry) is correct.

10.4.1. Self-assessment

1. At what times is **logcheck** run by **crond**? Explain how **logcheck** is being run; you need to explain the use of **nice** as well as `-R`.

2. **[Optional]** Why should **logcheck** be run as the "logcheck" user, and not "root" or "mal"? What system administration principle does this illustrate?

3. Make sure you have done all the exercises.

10.5. Regular Expressions

In order to meaningfully filter log files, we need to learn how to write good regular expressions. A regular expression (*regexp* for short), is a pattern that matches text. It's similar to, but more advanced than wildcards[9], such as `*.pdf`. Wildcards are generally only used for matching filenames etc, whereas regular expressions are used for most other matching tasks, and are much more powerful.

Regular expressions consist of various meta-characters which have special meaning. There are a handful of regexp flavours, and so there are some differences between them. Since learning regular expressions can be a bit of work, we'll stick to reasonably easy expressions, enough to get you through Logcheck effectively.

Regular expressions can be immensely powerful, being represented to it's highest degree in Perl Compatible Regular Expressions, which other languages often have some support for as well. It is largely the power of regular expressions that draws many administrators to Perl.

I'll show you the POSIX Extended regular expression syntax, since that's what **logcheck** uses, and it should be preferred compared to the older Basic syntax. **egrep** (or **grep -E**) is a command that uses the POSIX Extended syntax, whereas **grep** uses POSIX Basic syntax. Logcheck uses **egrep -i** internally, the `-i` makes the matching case-insensitive.

[9] Referred to as *glob* patterns.

The following table explains the meta-characters available for use with POSIX Extended regular expression syntax.

In the following examples, a custom tool called **regex_test** has been used. It works much the same as **egrep**, but its output is more useful for learning how regular expressions work. In particular, it will only show the *first* match. This tool should be available in the `Lab Resources/Regular Expressions` folder. On your workstation, copy the file `regex_test.c` from the Lab Resources into your virtual machines shared folder, and from inside your virtual machine compile it:

```
$ cd /media/host/
$ make regex_test
cc regex_test.c -o regex_test
$ sudo install -o root -g root -m 0755 regex_test /usr/local/bin/
```

Table 10.1. Regular Expression Cheatsheet

Syntax	Description
`.`	Matches any single character, but not a new-line character.
	`$ echo "hello" | regex_test '.'` `hello`
*	Matches *zero* or more occurences of the previous item.
	`$ echo "hello" | regex_test '.*'` `hello` `$ echo "abba" | regex_test 'a*'` `abba` `$ echo "ac" | regex_test 'ab*'` `ac` `$ echo "abba" | regex_test 'b*'` `abba` `  (successfully matched 0 characters from index 0 to 0)` `BEWARE: Leftmost longest match, especially with *` `$ echo "abba" | regex_test 'ab*'` `abba`
+	Matches *one* or more occurences of the previous item. `a+` is much like `aa*`, but much more convenient when you have larger things to repeat, as we see in the grouping operator below.
	`$ echo "ac" | regex_test 'ab+c'` `<NO MATCH>` `$ echo "abc" | regex_test 'ab+c'` `abc` `$ echo "abbc" | regex_test 'ab+c'` `abc`
?	Matches zero or one of the previous item. In other words, the previous item is optional.
	`$ echo "ac" | regex_test 'ab?c'` `ac` `$ echo "abc" | regex_test 'ab?c'` `abc` `$ echo "abbc" | regex_test 'ab?c'` `<NO MATCH>`
^	*Anchors* the match to begin at the start of the line.
	`$ echo "spline" | regex_test 'line'`

Syntax	Description
	spline $ **echo "spline" | regex_test '^line'** <NO MATCH> $ **echo "line #1" | regex_test '^line'** line #1
$	Anchors the match to finish matching at the start of the line.
	$ **echo "linear" | regex_test 'line'** linear $ **echo "linear" | regex_test 'line$'** <NO MATCH> $ **echo "spline" | regex_test 'line$'** spline $ **echo -e "... in Section 5, where ...** *two lines of input* **> Section 5" | regex_test '^Section 5$'** <NO MATCH> Section 5
[…]	Matches a *single* input character, which must be one of the characters listed between the square brackets. Most characters inside the square brackets lose any special significance they usually have, though some gain special significance, to allow things like ranges and negation.
	$ **echo "square" | regex_test '[aeiou]+'** square $ **echo "–123.45" | regex_test '[0-9]+'** *- introduces a range* -123.45 $ **echo "-123.45" | regex_test '[0-9.-]+'** *A literal - must be last. . is no longer a meta-character.* -123.45 $ **echo "0x0800C4DF" | regex_test '0[xX][0-9a-fA-F]+'** 0x0800C4DF $ **echo 'some "quoted" text' | regex_test '"[^"]*"'** *^ at the start negates the match: any but a "* some "quoted" text $ **echo 'but "there are \"limits\" to regexps" so watch out' ** **> | regex_test '"[^"]*"'** but "there are \"limits\" to regexps"
{n} and {m,n}	Bounded repitition. Matches the previous item m times exactly, or between m and n times. The upper or lower bound may be omitted, but leave the comma if you want an the range unbounded high or low.
\	Removes (escapes) special meaning from a meta-character.
	$ **echo '123 1.2 123' | regex_test '[0-9].[0-9]'** 123 1.2 123 $ **echo '123 1.2 123' | regex_test '[0-9]\.[0-9]'** 123 1.2 123
(foo)	Grouping construct. You could use it with the repetition qualifiers above.
	$ **echo '192.168.1.2' | regex_test '([0-9]+\.){3}[0-9]+** 192.168.1.2 $ **echo '1.2 1.2.3.4.5 192.168.1.2' | regex_test '([0-9]+\.){3}[0-9]+** 1.2 1.2.3.4.5 192.168.1.2 *Trap!: beware what comes after*
(foo|bar)	Alternation inside a group. For example, the pattern (foo|bar) would match all of foo *or* bar.
	$ **echo '... Figure 1.2 ...**

Syntax	Description
	`> ...` `> ... Table 1.1 ...' | regex_test -i '(figure|table) [0-9.]+'` `... Figure 1.2 ...` `<NO MATCH>` `... Table 1.1 ...`
`[[:alpha:]]`	Named character classes, can be used to specify things such as a "alphabetic" character, a "lowercase" character, a "digit", etc. See re_format(7) for further details. Constructions such as `[a-zA-Z]` are not sufficient in a non-English locale. For example, in Spanish you would also consider 'ñ' as a letter. What is matched depends on the *locale*: in an English locale, 'ñ' would not be expected to matched, but unfortunately, it probably would be[a]. **You are not expected to try this.** Nor are you expected to know how to type in Spanish or Chinese, but it is important to be aware of the differences. *Match some alphabetic characters* `$ echo '¡Español!' | LANG=es_MX.UTF-8 regex_test '[[:alpha:]]+'` `¡Español!` *Use LANG=C to ensure 7-bit ASCII* `$ echo '¡Español!' | LANG=C regex_test '[[:alpha:]]+'` `¡Español!` *But matching is entirely too inclusive* `$ echo '###' | LANG=es_MX.utf8 regex_test -i '[[:alpha:]]+'` `###  Chinese characters are not valid Spanish alphabetic characters.`

[a]Any Unicode character with a "Letter" property is actually what is matched. While explainable, is is likely unexpected in the context of a locale.

Here is a simple procedure to show you how to write a simple regular expression to match log file entries.

1. Use the following log entry as an example for the rest of procedure.

```
Feb 17 13:17:44 belgarath snmpd[2978]: Connection from 10.18.1.1
```

2. Identify the parts that will change, and the parts that will stay the same. Syslog entries have the date and time stamp, which will definately change. `belgarath` in this example is the name of the host that submitted the log. That will stay the same, and is worth including for a point of reference[10]. `snmpd` is the process name the submitting agent gave. That is a key part to identifying the log message. The number isn't important, it's the PID and will change. The string `Connection from` is important. Together with the `snmpd` part it practically identifies the whole line.

   We have an IP address in the line. If the service is used by many different IP addresses in the same subnet, you may elect not to match only part of it, say `10.18.`, which can be used for matching everything in the 10.18.0.0/16 subnet.

3. Your system's logcheck will likely start with a consistent header to match the timestamp and hostname part that Syslog prepends to the message. Thus, you can replace the timestamp and hostname with whatever other logcheck entries start with:

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ snmpd[2978]: Connection from 10.18.1.1
```

[10]Syslog can be made to accept remote log messages.

The \w is not documented in regex(7), but is mentioned in the GNU "Info" page. It is equivalent to [[:alnum:]] and is a shorthand notation borrowed from Perl regular expressions. Consider it a GNU extension.

4. Escape every meta-character in the input you wish to match, by prefixing it with a \ in the regular expression. Note that not all punctuation characters are meta-characters. Ignore the header part which we've already fixed up.

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ snmpd\[2978\]: Connection from 10\.18\.1\.1
```

5. Replace varying numerical sequences with [0-9]+ This is useful for the process identifier.

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ snmpd\[[0-9]+\]: ↩
Connection from 10\.18\.[0-9]+\.[0-9]+
```

6. Your completed regular expression should look like this.

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ snmpd\[[0-9]+\]: ↩
Connection from 10\.18\.[0-9]+\.[0-9]+
```

7. Test the regexp using **egrep -i**. We suggest that you put the input text into a file, to save typing.

```
$ echo 'Feb 17 13:17:44 belgarath snmpd[2978]: Connection from 10.18.1.1' \
> > log_message.txt
$ egrep -i 'belgarath snmpd.[0-9]*.: Connection from 10.18.1.1' log_message.txt
Feb 17 13:17:44 belgarath snmpd[2978]: Connection from 10.18.1.1
The line was printed, so it matches.
```

10.5.1. Self-assessment

1. Transform the following two log messages into two suitable **egrep** (POSIX Extended) style regular expressions for use with Logcheck.

```
Feb 10 18:32:22 belgarath sshd[2947]: Accepted publickey for theauthor from ↩
10.18.2.11 port 34061 ssh2
```

```
Feb 17 06:25:02 belgarath su[20870]: + ??? root:nobody
```

I have indicated those parts of each message that would change each time *or* will appear differently in each. Some of it you may wish to keep. For example, you may wish to only match part of the IP address, if you routinely have people logging in from 10.18.2.X; or perhaps you have many users and don't want to match each user explicitly.

In the second example, the + indicates the beginning of a session. The ??? is for the terminal (typically something like /dev/tty1,) but in this case there is no controlling terminal associated with this command. The ? is special to **egrep** (meaning the preceding is optional,) so you will need to escape it, turning each literal ? into \?. root:nobody indicates that the root user transitioned to the nobody user. Since this is effectively dropping privileges, it's generally okay to ignore this event.

10.6. Final Words

There is, as always, a *lot* more we could have taught you in this lab, but limited time in a paper does not allow us to go that deeply.

One thing that you may want to look at is remote logging. Then you just need to point your devices at your syslog server... this includes such devices as routers, wireless access point, and network-enabled printers, as well as servers and even workstations.

But there are some security issues that you should at least consider when doing this: you generally want to at least limit who can submit logs, and implement some rate-limiting and file-system management so that your log server doesn't run out of disk space. Rsyslog, as part of the RELP logging improvements to Syslog, has added features for reliability and security, although most senders won't support them at present.

The value of logs coming from remote devices is generally *very* useful for diagnosing faults quickly, as it removes a lot of guesswork, and gives you a concrete message with which to start your diagnosis.

Lab 11 DNS using BIND 9

Today we will learn the basics of setting up and maintaining a Domain Name Server using the popular BIND 9 package. You should work on `server1` unless specified otherwise.

DNS can be fraught with difficulty for the unsuspecting newbie. To help keep this lab manageable, it will be a very basic introduction to setting up a DNS server, for a small LAN for IPv4 and IPv6. You will be given a template to work from.

Because DNS is such a crucial part of a network, and pretty much every service makes use of it, you will use the knowledge gained in this lab throughout the rest of this paper. You will be required to make changes to your DNS server in some of the labs that follow.

> **Important**
>
> Please ensure that you read the entire lab before coming to class, otherwise it's very likely you will not complete it during the lab.
>
> Make sure to install the resolvconf package on server1 using **sudo apt install resolvconf**.

11.1. Using Dig

dig is the Domain Information Groper, and is the successor to a tool called **nslookup**. There is also a simplified version of **dig** called **host**, but we won't cover that. So without further ado, here are some ways you can use **dig**. You should be able to try all these on your server or client.

You will get more information by not using the `+short` option. The answer will be found in the Answer section. You'll also get some supplementary information, as well as result codes. Using the `+short` option is a good way of getting DNS information in shell scripts.

- What is the IP address of www.isc.org? This is querying for an A record. The domain is that of the Internet Software Consortium, the organisation behind software such as BIND.

```
$ dig isc.org
; <<>> DiG 9.8.1-P1 <<>> isc.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17448
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 8

;; QUESTION SECTION:
;isc.org.   IN A

;; ANSWER SECTION:
isc.org.  60 IN A 149.20.64.69

;; AUTHORITY SECTION:
isc.org.  5098 IN NS ams.sns-pb.isc.org.
isc.org.  5098 IN NS ns.isc.afilias-nst.info.
isc.org.  5098 IN NS sfba.sns-pb.isc.org.
isc.org.  5098 IN NS ord.sns-pb.isc.org.

;; ADDITIONAL SECTION:
```

```
ns.isc.afilias-nst.info. 3355 IN A 199.254.63.254
ns.isc.afilias-nst.info. 3355 IN AAAA 2001:500:2c::254
ams.sns-pb.isc.org. 6488 IN A 199.6.1.30
ams.sns-pb.isc.org. 6488 IN AAAA 2001:500:60::30
ord.sns-pb.isc.org. 6199 IN A 199.6.0.30
ord.sns-pb.isc.org. 6199 IN AAAA 2001:500:71::30
sfba.sns-pb.isc.org. 6488 IN A 149.20.64.3
sfba.sns-pb.isc.org. 6488 IN AAAA 2001:4f8:0:2::19

;; Query time: 155 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Mon Feb  6 15:24:00 2017
;; MSG SIZE  rcvd: 316
```

- The same query, but this time in short format.

```
$ dig +short www.isc.org
149.20.64.69   Don't panic if yours is different
```

- Same as above, `-t A` is the default behaviour. `-t A` asks for resource records of type `A`, which maps a hostname to an IPv4 address.

```
$ dig +short -t A www.isc.org
149.20.64.69
```

- You can use **dig** without specifying a fully qualified hostname. Note that **dig** will not use the default search path (in `/etc/resolv.conf`) by default; we can change this behaviour and use the search path using `+search`

```
$ dig vertex

; <<>> DiG 9.7.0-P1 <<>> vertex
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 38969            FAILED!
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;vertex.         IN      A            Because it wasn't searching in our domain.
…
$ dig +search +short vertex
139.80.32.49
```

- What's the (canonical) hostname for 139.80.32.49?

```
$ dig +short -x 139.80.32.49
vertex.otago.ac.nz.
```

- Note that the above inverse query is equivalent to this:

```
$ dig +short -t PTR 49.32.80.139.in-addr.arpa
vertex.otago.ac.nz.
```

Exercise

For the following exercise you will use **dig** to explore your local network enviroment a little. Make sure they work as expected.

- Who are the name servers for `otago.ac.nz`?

```
$ dig +short -t NS otago.ac.nz
```

- Which of the nameservers (each domain should have at least two) is the master server? You can find this out by looking at the `SOA` for the domain.

```
$ dig -t SOA otago.ac.nz
```

- You can query a particular name server using the `@` character before the nameserver's DNS name or IP address. Select one of the nameservers you discovered in the previous step.

```
$ dig @nameserver +short www.otago.ac.nz
```

- Who are the mail exchangers for `otago.ac.nz`?

```
$ dig +short -t MX otago.ac.nz
```

> **Note**
>
> Like web servers, which are also high-traffic, e-mail services can be designed in various ways for high availability of for load balancing. So if you get a single result, it may be that there are multiple servers behind one address and a device called a *load balancer* is acting as the entry point to a cluster of e-mail servers.

- While it is possible to use **dig** to perform a *zone transfer* --- to get a list of all the data in a particular zone (domain) --- it is considered rude or hostile.

> **Warning**
>
> **Do not** request this from a machine you do not administer, as it will be considered rude or hostile. You will have a chance to do the following later in the lab to your own network.
>
> ```
> $ dig @127.0.0.1 -t AXFR domain
> ```

- Once you have your DNS server up and running, you can try to find out which version of BIND is running on the server. Servers are often configured to give a different result instead, to make it harder for an attacker to know what software version you are using. Don't use `+search`.

> **Note**
>
> You can do this in your virtual machine later.

```
$ dig @localhost -t TXT -c chaos +short version.bind
"9.7.0-P1"
```

The `chaos` "class" (like the "inet" class) refers to a historical networking system called Chaosnet that you don't need to know anything about. Just know that this it was around when the Internet was still referred to as the ARPAnet, and that the only reason we use it

today is as a way to determine the version of the software running on a DNS server running BIND.

11.2. Basic Configuration

Unless otherwise specified, do all your work from this part onwards in your server.

First, you will need to install the following packages: **bind9**, **bind9-host** and **ipv6calc**:

```
# apt-get install bind9{,-host} ipv6calc
…
```

> **Note**
>
> If your shell (bash) sees a pattern like foo{bar,baz,bax}, it will expand to foobar foobaz foobax. Therefore, bind9{,-host} will expand to bind9 bind9-host.

You are about to edit some files which could easily lead to locking yourself out of **sudo**. You should have a root shell available (**sudo -s**) in another virtual console, or set a password for the root user (**sudo passwd root**) temporarily. You can lock the root account again later by using **sudo passwd -l root**.

The file `/etc/hosts` contains static mappings of hostnames to IP addresses. If you're going to set up a DNS server (and you are), then this file should contain a mapping between localhost and 127.0.0.1, and the name of the machine, similar to what is shown below; you probably won't need to modify anything. Note that here, we assume that you called your server `server1`; adjust to suit your environment.

> **Tip**
>
> DNS is case insensitive, and everything is generally input in lower-case.

```
127.0.0.1       localhost
127.0.0.1       server1
192.168.1.1     server1.localdomain
```

Before you set up a DNS server, you should configure the system resolver library with the address of your DNS server(s). You would typically put this into `/etc/resolv.conf`; however, this file is prone to being overwritten by tools like DHCP clients, such as **dhclient**, which we are using to configure our outside interface, so let's tell our DHCP client to put something different in there instead. Another alternative would be to disable **dhclient** from writing that file at all.[1]. Edit the file `/etc/dhcp/dhclient.conf`, and add these lines before the `request` stanza. This causes the DHCP *client* on Server1 to ignore some of the settings it was given, and replace (supersede) them with settings we manually specify.

> **Tip**
>
> You wouldn't normally expect to this on a DNS server, as DNS servers wouldn't normally be on a machine configured by DHCP. But our server is a gateway machine, and is both client and server at the same time.

[1] We could, if we really wanted, even have the DHCP client run a script which updates the forwarders we will configure later.

```
supersede host-name "server1";
supersede domain-name localdomain;
supersede domain-name-servers 127.0.0.1;
supersede domain-search "localdomain";
```

Now disable *systemd-resolved*, which would interfere our changes of `/etc/resolv.conf` below. Note: do the same on Client1.

```
# systemctl disable systemd-resolved
# systemctl stop systemd-resolved
```

Next down/up the interface and check that your `resolv.conf` is as shown below:

```
# ifdown outside
…
# ifup outside
…
$ cat /etc/resolv.conf
nameserver 127.0.0.1
search localdomain
```

Okay, that looks like we would write it by hand.

Now we need to configure Client1 to use the DNS server on Server1. *For Client1*, it will use DHCP to configure its DNS subsystem correctly, but for now you can just manually tell Client1 to use 192.168.1.1 as the nameserver, since we don't have DHCP server set up yet. Remember that you can't modify `/etc/resolv.conf` directly. Instead you should change `/etc/resolvconf/resolv.conf.d/head` as you did before. Make sure `/etc/resolvconf/resolv.conf.d/head` on Client1 has only the following content.

```
search localdomain
nameserver 192.168.1.1
```

Then use the following command to make the change effective.

```
$ sudo resolvconf -u
```

In future, if we use DHCP, we will need to change `/etc/network/interfaces`, where the "static" method should be replaced by "dhcp". No matter which approach we use, the `/etc/resolv.conf` file should end up like something above.

The `search` keyword means that if we were to use an unqualified host-name, such as `server1`, then the local resolver would instead try to find `server1.localdomain` instead (assuming that `server1` was not found in `/etc/hosts`.

So, for example, if you *were* to type **ping server1** (not that you could, currently, as we haven't configured our DNS server yet), then the local resolver would try to find an address for `server1.localdomain`. Note that you can have multiple search domains, by separating them with spaces on the same line. The nameserver keyword specifies a DNS server to ask. If we have more than one nameserver line, then if the first fails, we ask the second, and so on. To specify multiple nameservers, enter each nameserver in its own entry, like below.

```
This is an example only, don't enter it today
search localdomain
nameserver 192.168.1.2
nameserver 192.168.2.4
```

To configure the order in which various name server databases are looked up, you need to configure two files, `/etc/nsswitch.conf` and `/etc/host.conf`, on both the server and the client. `host.conf` is the older version of `nsswitch.conf`, you should check both of these to keep both (very) old and new programs happy.

Generally you don't need to change them when setting up for DNS, but do have a look at them, especially `nsswitch.conf`. The line that says `hosts` is the relevant one for DNS, it specifies in what order to consult the file `/etc/hosts` and the DNS resolver (or server). We shall revisit the `nsswitch.conf` file later in the paper when we look at authentication.

Here is what is currently configured in `nsswitch.conf`:

```
…
hosts:          files dns
…
```

The full details are in nsswitch.conf(5). But let us go briefly over it. We shall assume that we are looking up 'foo.localdomain':

hosts
: This is the database that we wish to consult. This particular database determines mappings between hostnames and IP addresses.

  Other databases are 'passwd', for storing information about user accounts, and others exist too.

files
: Order matters here. *First*, the file `/etc/hosts` will be consulted to see if there is a 'foo.localdomain' can be found in there. If it is found, the lookup process ends and the result is returned.

mdns4_minimal
: Here's where things get a bit different. In environments where there is no DNS server locally available to answer questions about machines in the local network, such as at home or in on an ad-hoc wireless network, then we use something called Multicast DNS, or mDNS for short. Basically, we multicast a DNS query onto the local network, and if any machine sees that someone is asking for their own information, it will respond with the answer.

  Multicast DNS is defined for the domain "local.", which is why you should never call your home or private network domain "local.", as this will cause lookup problems.

  Note that a mDNS lookup is only initiated for requests about domains in "local.", which "foo.localdomain." is not, so no lookup is attempted: *no answer, positive or negative, is generated*.

[NOTFOUND=return]
: In the previous step, we *may* have executed a mDNS lookup. *If we did*, and if we didn't get a result (or in other words, "X.local." didn't exist), then we would return a lookup failure *and not continue*.

  This is why you should not call your home or private domain "local.", because by doing so Ubuntu (and others) will not proceed to lookup (standard unicast) DNS, even if the DNS server has the information.

dns
: If we get here, we do our regular DNS lookup to the nameservers listed in `/etc/resolv.conf`.

The current entry for *hosts* is fine. Keep it unchanged:

```
hosts: files dns
```

In our network, let's assume that Multicast DNS is not desired, because all hosts should be listed in DNS.

Finally, here is `host.conf`. Because this is only consulted by very old programs, you should never need to change it. **ping** is perhaps one program that might still use it, which may explain why it acts a bit different sometimes when diagnosing DNS problems on some systems.

```
order hosts, bind
multi on
```

11.3. The Master Bind Configuration File

Distributions based on Debian, such as Ubuntu, manage the layout of BIND's configuration files differently from the standard BIND software. This is to allow for greater modularity. The default configuration conforms well to best practices to DNS operation, such as being authoritative for the correct zones, including special cases such as broadcast and private addresses.

Firstly, have a good look at the files under `/etc/bind/`. Ensure you can understand the structure of the `named.conf*` files, starting with `named.conf`.

To aid your understanding, here is a functionally similar `named.conf`, to show you how it works. *It is not the same as what you see in your virtual machines, but we have added annotations to specify which files the various blocks should be added to.*

```
The following lines will go in named.conf.options

acl "clients" {
  192.168.1.0/24;
  fd6b:4104:35ce::/64;
  127.0.0.1;
  ::1;
};

options {
  directory "/var/cache/bind";
  allow-query { "clients"; };
  allow-transfer { 127.0.0.1; ::1; };
  allow-recursion { "clients"; };
  listen-on { any; };
  listen-on-v6 { any; };
  forwarders { 139.80.64.1; 139.80.64.3; };
  auth-nxdomain yes;
};

The following (and others) will already be in named.conf.default-zones
which is already included in named.conf.
There is no need to edit named.conf or named.conf.default-zones

zone "." {
  type hint;
  file "/etc/bind/db.root";
};

zone "localhost" {
```

```
  type master;
  file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
  type master;
  file "/etc/bind/db.127";
};

The rest goes into named.conf.local

zone "localdomain" {
  type master;
  notify no;
  file "/etc/bind/db.localdomain";
};

zone "1.168.192.in-addr.arpa" {
  type master;
  notify no;
  file "/etc/bind/db.192.168.1";
};

// convert using "ipv6calc --in ipv6 --out revnibbles.arpa fd6b:4104:35ce::/64"
zone "0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa" {
  type master;
  notify no;
  file "/etc/bind/db.fd6b-4104-35ce-0000--64";
};
```

`acl "clients" { … };`
: This provides a way to reference a set of clients using IP addresses or DNS names. It is the list part of an access control list (ACL). We will use it to control access to various types of requests made to the server.

`options { … };`
: Here we set various global options for the rest of the config file.

  `directory` is where the zone files can be found if not given a fully-specified pathname. Because Debian puts the BIND configuration files and the master zones under `/etc/bind/`, but the `directory` still points to `/var/cache/bind`. This means that for the zones stored in `/etc/bind`, we must specify their location absolutely. Slave zones (which the server would update during runtime) get cached under `/var/cache/bind/`.

  `allow-query` specifies which clients may query the server. This can be an IP address or range, domain, or access control list (in double-quotes), as defined previously. It's important to limit your exposure, since DNS is a very important service, and if compromised, can make further compromise easier. This is why most networks use different DNS servers for the public zones than for the internal zones.

  `allow-transfer` says which clients may perform a zone transfer. A zone transfer is output which tells the receiver everything about a zone. You should only allow slave servers, and possibly localhost to receive a zone transfer. As we shall see later on, a zone-transfer is a great way to debug your zone information.

  Note that for all of these `allow-*` statements, you can specify the keywords `none` and `all`. Check the Bind9 Administrators Handbook for more information. This document is cited at the end of this labsheet.

  `allow-recursion` allows the DNS server to go and get the complete answer to the clients question, rather than just the next step. This makes more work for the server. Generally

it's good to allow recursion to your known clients only, certainly never to the public internet. Recursion is fundamental principle in caching name servers.

`forwarders` is a very useful option for most smaller networks. It allows you to specify any upstream DNS servers, such as those of your ISP, which would probably be recursive and have a larger set of cached results, in order to improve performance, and keep unnecessary traffic off the root servers.

`listen-on` and `listen-on-v6` say where the server should accept packets from. We just say `any` here because later on we will further limit who can do what using ACLs.

Finally, `auth-nxdomain` enables the server to give authoritative answers in the case of a lookup failing because it doesn't exist (in DNS terms: no such domain). This allows these negative responses to be cached, which helps to reduce the amount of traffic going to the root nameservers, and reducing latency in the DNS system. It is off by default, but DNS experience has shown the industry that it is worth doing, and is now considered an operational requirement.

`zone "." { … };`

Here we have the entry for the root servers. This is of type `hint`, because we store these addresses in a file. The address list doesn't change very often, and was last updated on the 20th October, 2016. You should make it part of your periodic maintainance to check for updates, although it should be reasonably up-to-date anyway if you have installed Bind from a supported package. The file can be found at InterNIC [ftp://ftp.rs.internic.net/domain/named.root], though for the purposes of this lab, you needn't bother.

The `.` zone is called the root. All fully-qualified hostnames, such as `www.otago.ac.nz` have an implied dot at the end, as in `www.otago.ac.nz.`. Sometimes it is useful to specify the dot, especially when your domain only has one domain component, such as `localdomain.`.

Since we have configured our server to forward queries to other servers, we won't typically be using the root zone, unless our upstream server (which we send our forwarded queries to) fails to respond.

`zone "localhost" { … };`, **`zone "127.in-addr.arpa" { … };`**

Here are two very basic zones, for mapping between localhost and 127.0.0.1 (one for each direction). Not terribly interesting in themselves, they're just like the ones we'll see later. You will see other basic zones as well, in addition to those shown here; do not remove them.

Notice the unusual looking name for the second zone. This is how reverse lookups for IPv4 addresses are performed. The dotted decimal components are reversed[2] (the digits of each component are not, however) and "`.in-addr.arpa`" is appended. The following two commands are identical. You can try these on your Mac host, as you don't have DNS working yet.

```
$ dig -t PTR +short 49.32.80.139.in-addr.arpa
vertex.otago.ac.nz.
$ dig +short -x 139.80.32.49
vertex.otago.ac.nz.
```

And an IPv6 example is given below as well. Mind you that the reverse mapping may not work if the PTR record is not provided by the `www.kame.net` domain. Also note how

[2] Making it into least-significant compoment first, just like a standard DNS name.

horrible the reversed nibble is to work with; it is simple (very good), but explosively verbose, which is annoying and a source of many mistakes. We'll look at that issue later on.

```
$ dig -t AAAA +short www.kame.net
2001:200:0:8002:203:47ff:fea5:3085
$ ipv6calc --in ipv6 --out revnibbles.arpa 2001:200:0:8002:203:47ff:fea5:3085
5.8.0.3.5.a.e.f.f.f.7.4.3.0.2.0.2.0.0.8.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.
$ dig -t PTR +short \
> 5.8.0.3.5.a.e.f.f.f.7.4.3.0.2.0.2.0.0.8.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.
orange.kame.net.
$ dig +short -x 2001:200:0:8002:203:47ff:fea5:3085
orange.kame.net.
```

`zone "localdomain" { … };`
: The forward zone for "localdomain". Server1 has the authoritative data for this domain, so we specify the type as `master`. Since we don't have any slave servers, we don't need to `notify` them of any changes.

`zone "1.168.192.in-addr.arpa" { … };`, **`zone "0.0…d.f.ip6.arpa" { … };`**
: The reverse zones for "localdomain". We have two reverse zones, one for IPv4 and our IPv6 Unique-local addresses respectively (we never add link-local addresses into DNS).

  Just as with the forward zone, Server1 has the authoritative data for this domain, so we specify the type as `master`. Since we don't have any slave servers, we don't need to `notify` them of any changes.

Integrate the configuration above with your current configuration. You should only need to change `named.conf.options` and `named.conf.local`.

When you have entered the information, use the **named-checkconf** command to check the syntax of the file and make sure there are no errors. On a well-formed file, it should not print anything. Note that it will not check for the existence of files specified in the zone configuration statements.

```
$ named-checkconf
If it outputs nothing, it has nothing to complain about.
```

Exercise

As an exercise make sure there is no output from **named-checkconf**, and the contents of the three `named.conf*` files are as expected.

11.4. Forward Zones

Forward zones are for specifying the mappings from hostnames to IP addresses. we suggest you start with a template (such as the one below) and work from that. Put this into the `/etc/bind/db.localdomain` file. You will need to create it. Comments in zone files start with a `;` and continue to the end of the line.

```
$TTL 3D

@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2020030601         ; serial
        8H                 ; refresh
        2H                 ; retry
        4W                 ; expire
```

```
        1D                ; minimum
        )

        NS  ns1.localdomain.
        ; MX  0  mailhub1.localdomain.
        ; MX  5  mailhub2.localdomain.
        ; A  192.168.1.3 ; Would allow http://localdomain/

localhost     A        127.0.0.1
              AAAA     ::1

server1       A        192.168.1.1
              AAAA     fd6b:4104:35ce::1
server1.ipv4  A        192.168.1.1
server1.ipv6  AAAA     fd6b:4104:35ce::1
ns1           A        192.168.1.1        An alias
              AAAA     fd6b:4104:35ce::1

client1       A        192.168.1.11
              AAAA     fd6b:4104:35ce::look-this-up
```

Line 1

The TTL statement *must* be the first non-comment line in the file. This specifies the default time to live (3 days in this case) until cached entries should be invalidated. I strongly recommend that when you put your first Internet-serving DNS server on the internet, you use a short TTL until you are happy that all the data is correct, otherwise other servers will cache the data and won't see any changes until the TTL expires.

If you're anticipating that data might change soon (perhaps you know that next week you will be moving to a different IP address), decrease the TTL to a shorter and shorter value, so that on the day of the move no server will be caching an answer for longer than a few minutes.

Lines 3 to 9

The contents of line 3 must not be broken onto multiple lines, else things will break. `@` is shorthand for the origin (`.localdomain.` in this case). It's already declared in `named.conf.local`, so we needn't write it again[3]. `IN` says that this zone is an Internet addressing system. `SOA` (Start Of Authority) describes this zone: the server which is authoritative for it, who is responsible for it, and various timeouts.

`ns1.localdomain.` is the nameserver which is authoritative for this domain. Note: You need to include the dot at the end, otherwise, you'll end up with `ns1.localdomain.localdomain`. This is why it can be useful it allow zone transfers to localhost, as it's easiest to spot these sorts of errors by looking at the output of a zone transfer.

`hostmaster.localdomain` is actually the email address hostmaster@localdomain of the person responsible for the zone. It's customary to have an email address `hostmaster` for this purpose, which is an alias to a real person.[4]

The SOA resource record (RR) contains various timeouts, they are as follows:

Serial Number

Slave servers use this as a version number to find out if they need to update. You should update it whenever you make a change to the zone file. Generally you use a form `yyyymmdd` + today's version.

[3] Though you can by using the `$ORIGIN` directive.
[4] If the email user has a dot in it, you must escape the dot to make it `\.`.

Refresh
How often slave servers should check for an updated version of the zone file. The H suffix means hours, and you can also specify days (D), minutes (M) or seconds (no suffix) etc. The suffix is case-insensitive.

Retry
If a refresh failed, the retry interval says how long to wait before trying again.

Expire
After this interval, if the data hasn't been updated, remove it.

Negative or Minimum
Specifies how long a negative result should be cached for. A negative result is, for example, when the record being looked up does not exist. These values do just fine for normal environments. The maximum allowable value is 3 hours (3h).

Lines 11 to 14
Networks, as well as hosts, have resource records attached to them. We've already seen the SOA RR, here are a few more.

Important

Make certain to include some amount of white space at the start of these lines, otherwise **named** will think you're defining hosts. You'll end up trying (and failing) to define a host called NS with a RR called `ns1.localdomain.`, which is not a supported RR (well, of course not!).

NS
Gives a nameserver for this domain. There must be at least two NS records for each public domain. They could be master or slaves, the master is listed in the SOA record for the zone.

MX
Gives the various mail exchangers that accept mail for this domain. Each MX entry has a priority. Mail is send to the one with the highest priority (which, ironically, is the one with the lowest value), and if it fails, tries the others, in order of priority.

Since we haven't covered email servers at this time, these are just for example and are commented out.

Lines 16 to 27
Now we start to declare the hosts that are in our network. The form is: hostname, class, RR type, then RR value(s). The class is generally `IN` and can be omitted, as it is inherited from the SOA record. Note that the first two lines makes the hostname localhost.localdomain point to 127.0.0.1 and ::1, which isn't important for users, but can be of assistance when clients don't resolve localhost to themselves first (through `/etc/hosts` or similar).

Line 19 to 22
Here we've specified that server1 has an A record as well as an AAAA record, so if request for any records attached to server1, it will likely give you results for both IPv4 and IPv6.

In case you want to prefer IPv4 *or* IPv6 (for example, when rolling out IPv6 support for your own clients in your own domain), then we've also added records that return just

those results. This should only really be useful for troubleshooting, most of the time you should be using `server1.localdomain` instead of `server.ipv4.localdomain`.

You might wonder why we chose `foo`.ipv6 instead of ipv6.`foo`, or `foo_ipv6` or similar: one (minor) reason is that you can adjust who you roll-out IPv6 to by adjusting your clients DNS search path (perhaps via DHCP), but this is only a very minor difference really. RFC 4472 contains some guidance here. To make the difference clear, imagine you have `ipv6.localdomain` and `localdomain` in the search path of machines you wish to roll-out IPv6 to. Then, if a client is configured to access the unqualified hostname `foo`, it will first try `foo.ipv6.localdomain`, then `foo.localdomain`. In the same way, you could have other clients prefer IPv4. But this is only effective for clients in your own domain which are looking up domains that you control.

Line 23 to 24

DNS servers (nameservers) generally work in pairs. A public domain on the internet needs to have at least two nameservers, and are often refered to as ns1.`domain` and ns2.`domain` etc. In our case however, we only have one nameserver.

Here we are creating an alias called ns1.`domain`. There are two common ways of creating an alias; either using duplical address records or by using the CNAME RR, but CNAMEs are becoming less common, and it is generally preferred to use a duplicate address records. It also gives us better control over what IPv4 and IPv6 records get returned.

For example, `server1` currently has an A record for a particular address. A *duplicate address record* could be made for `www` that is also an A record *to the same address* (ie. it is the address that is duplicated, not the name).

Note

Another way (less preferred but still very common) of creating aliases is with the CNAME record.

If we were to see a line `ns1 CNAME server1` in the “localdomain” zone, that means that we are creating a label (record) called “ns1.localdomain”, which is an alias. The canonical (real) name of the machine the alias points to is “server1.localdomain”.

In this case we are using duplical address records: the only thing that makes it an alias (rather than a canonical name) is that in the reverse zone, 192.168.1.1 should only return a PTR record for `server1.localdomain.` (it’s *canonical* hostname).

It is common practice to call servers one name (its canonical name), then give them aliases that refer to their services, such as www or ftp. This makes it easier to reassign a particular service to another machine, because all the clients don’t need to be re-configured: a single point of change.

Once done, you can use the **named-checkzone** on the zone file to check for errors.

```
$ named-checkzone localdomain /etc/bind/db.localdomain
zone localdomain/IN: loaded serial serial-number
OK
```

Exercise

As an exercise make sure the above **named-checkzone** reports no error.

11.5. Reverse Zones

Reverse zones are for specifying the mappings from IP addresses to DNS names. It's often used on servers for logging purposes, and access control. If you notice that it takes about 6 seconds to use a service, it may be that your reverse zone is not well configured, and the lookup timed out. As with the forward zone, you should use a template, such as the following.

Put this into the `/etc/bind/db.192.168.1` file. You will need to create it. There's nothing terribly new at the start of the file, what's different is when it comes to the PTR (pointer) records.

```
$TTL 3D

@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2020030601 8H 2H 4W 1D)

       NS        ns1.localdomain.

The current $ORIGIN is 1.168.192.in-addr.arpa.
so this unqualifies entries like 192.168.2.1
as the first three bytes of the address have to be 192.168.1.
1      PTR       server1.localdomain.
11     PTR       client1.localdomain.
```

Exercise

As an exercise check the well-formedness of the file using **named-checkzone** as below.

```
$ named-checkzone 1.168.192.in-addr.arpa. /etc/bind/db.192.168.1
zone 1.168.192.in-addr.arpa/IN: loaded serial serial-number
OK
```

But wait, that's only the IPv4 reverse zone, we still have to create the IPv6 version. Create a new file `/etc/bind/db.fd6b-4104-35ce-0000--64` (there's nothing special about the name, its just a name mangling scheme we came up with to describe the zone).

```
$TTL 3D
@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2020030601 8H 2H 4W 1D)

     NS ns1.localdomain.

This $ORIGIN is as the default anyway, just to remind ourselves…
Remember to include the trailing .
We used the following command to get the right format
ipv6calc --in ipv6 --out revnibbles.arpa fd6b:4104:35ce::/64
$ORIGIN 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa.

Here's another example of ipv6calc to the rescue:
ipv6calc --in ipv6 --out revnibbles.arpa fd6b:4104:35ce::1 | cut -d. -f-16
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0    PTR server1.localdomain.
Your Client1 will have a different address!
e.0.7.3.8.2.e.f.f.f.7.2.0.0.a.0    PTR client1.localdomain.
```

Exercise

Well, that was painful, and it's not immediately obvious what we have specified, which makes it harder to recognise and diagnose errors. We'll have a look at how we can

make this much easier a little later; for now, as an exercise let's get onto testing using **named-checkzone**, showing your progress.

```
$ named-checkzone 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa. \
> /etc/bind/db.fd6b-4104-35ce-0000--64
zone 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa/IN: loaded serial serial-number
OK
```

11.5.1. Self-assessment

1. Make sure you have done the exercises for: `named.conf*`, **named-checkconf**, `db.localdomain`, **named-checkzone** for that zone, `db.192.168.1` and its **named-checkzone**, and similar with the IPv6 reverse zone.

   If submitting remotely, also include the zone files themselves (remember, you can copy them in/out using the VirtualBox shared folder you set up).

11.6. Affecting the Changes

Restart Bind to affect the changes and load the new zones:

```
# /etc/init.d/bind9 restart
```

Check the logs to familiarise yourself with what it says when it starts.

Ensure that the **named** process is running, and listening on UDP (and TCP) port 53 (the domain port), plus a few others. You should notice that there are entries for each interface.

```
# lsof -Pni
COMMAND … TYPE … NODE NAME
…
named   … IPv6 … TCP  *:53 (LISTEN)
named   … IPv4 … TCP  127.0.0.1:53 (LISTEN)
named   … IPv4 … TCP  192.168.1.1:53 (LISTEN)
named   … IPv4 … TCP  10.0.2.15:53 (LISTEN)
named   … IPv4 … TCP  127.0.0.1:953 (LISTEN)
named   … IPv6 … TCP  [::1]:953 (LISTEN)
named   … IPv6 … UDP  *:53
named   … IPv4 … UDP  127.0.0.1:53
named   … IPv4 … UDP  192.168.1.1:53
named   … IPv4 … UDP  10.0.2.15:53
```

Important

If a server is failing to start, check the logs!

You may be wondering what all these entries are for, and if so, then you're in the right mindset for an administrator. The TCP entries relating to port 53 is because if DNS requests don't fit into a UDP response (typically this is 512 bytes for UDP DNS, but can be increased) then it will try again using TCP; this can be a performance hit. Queries such as zone transfers would commonly need to use TCP. The entries relating to port 953 are related to the **rndc** tool, which allows control of the server during runtime (for example, to tell it to reload some zones), or to write out some statistics for performance analysis. The entries relating to UDP port 53 are the standard DNS traffic. Notice that we support both IPv4 and IPv6 traffic, and that both IPv4 and IPv6 queries can be made over each (indeed, currently, the majority of IPv6 queries go over IPv4).

11.6.1. Controlling the Server During Runtime

rndc is a tool that controls **named** during runtime. This means that when we make a configuration change, we can use **rndc** to tell **named** to reload the configuration. This is good, because when **named** is *restarted*, all the cached answers are forgotten.

When you installed the `bind9` package, a key was created for you in `/etc/rndc.key`. Both **named** and **rndc** read this file to retrieve the key they need to use in order to authenticate, so by default, there is no configuration needed to control **named** on the local machine[5].

If you change the contents of any of the zone files or `named.conf`, you should run the command below to tell the server to reload its configuration.

```
# rndc reload
```

It pays to check the logs just to make sure that it reloaded successfully, and that reloading didn't cause it to crash or to fail to load a particular zone.

11.7. Testing

The devil is in the details of most of the core things a Network Administrator does, so testing is important. It's all too easy to leave out a crucial dot or semi-colon, and have things either not start, or worse, start but not work correctly.

The easiest way to test what the server has read in from your zone files is to do a zone transfer on that zone. Remember, we have three zones to check, the forward zone and the two reverse zones for IPv4 and IPv6 respectively.

Exercise

As an exercise retrieve a zone transfer for each zone, and check it looks correct. Resolve (in both directions) names and numbers to make sure that resolution is working correctly.

```
$ dig @localhost -t AXFR localdomain
…
$ dig @localhost -t AXFR 1.168.192.in-addr.arpa
…
$ dig @localhost -t AXFR \
> 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa.
…
```

Let's test some basic forward and reverse entries:

```
$ dig +short server1.localdomain
… should return an A record
$ dig +short -t ANY server1.localdomain
… should return an A and AAAA record
$ dig +short -t AAAA server1.localdomain
… should return a AAAA record
$ dig +short -t AAAA server1.ipv6.localdomain
… should return a AAAA record
$ dig +short server1.ipv4.localdomain
… should return an A record
$ dig +short ns1.localdomain
```

[5] **rndc** can also be used to control other machines, but we won't go into that.

```
… should return an A record (-t argument defaults to A)
$ dig +short -x 192.168.1.1
… should return server's canonical name
$ dig +short -x fd6b:4104:35ce::1
… should return server's canonical name
$ dig +short -x client1's fd6b::… address
…
```

It can be useful just to do a quick ping to check if you've mistyped an address (note though that this doesn't guarantee you've put in the correct address, just one that exists):

```
$ ping6 -c1 client1.localdomain
…
```

Now let's check that our forwarding is working by looking something that would not be satified either by our local zones or in our DNS server's cache:

```
$ host www.google.com
…
```

11.8. Self-assessment

Exercise

Make the following additions. For each of your changes, make sure they are correct by testing. This exercise will enhance your understanding of the DNS configuration.

1. Add in mappings for a (non-existent) host named goliah, which has an address 192.168.1.5 as well as fd6b:4104:35ce::dead:beef.

2. Create an alias (as duplicate A and AAAA records) for goliah called `www`. To test, use the following:

```
$ dig +short -t ANY www.localdomain
… You will get SOA as well as NS records, etc.
192.168.1.5
fd6b:4104:35ce::dead:beef
$ dig +short -x 192.168.1.5
goliah.localdomain.
$ dig +short -x fd6b:4104:35ce::dead:beef
goliah.localdomain.
```

3. Make it so if a user were to type http://localdomain/ into a web browser, it would end up going to goliah (who we are assuming is a web-server). You do not need to have a webserver installed or client installed. To test, use the following.

```
$ dig +short -t ANY localdomain
… You will get SOA and NS records as well
192.168.1.5
fd6b:4104:35ce::dead:beef
```

4. Assume you are providing a file mirroring service for many clients (eg. a public Ubuntu mirror). Assume further that you are providing mirrors for other distributions. What is the benefit of using the hostname `http://ubuntu.example.com/` instead of `http://www.example.com/ubuntu/`? Is there a difference? Why might one be more preferable above the other. Tip: consider what would need to happen if we wanted to change which server offered the Ubuntu repository.

11.9. [Optional] Fixing that Mess with IPv6

Running short on time?

The remainder of this lab is optional. You are not required to complete this for any assessment or assignment, but you will find that doing so can make later alterations a bit more pleasant.

We saw how horrible the IPv6 addresses were to deal with in the reverse zones. A tool such as **ipv6calc** helps a lot in some respects, but only when adding the new records; it's still quite difficult to see everything that is currently there. We can do better by generating the correct (verbose) format using a program with input from a zone file that goes through some filter to do the conversions. We have written just a program for you: **ipv6-dns-revnibbles**. This program works somewhat like the venerable **m4** macro processor, but is highly specialised (and therefore rather useless for other tasks). With this tool, your input starts looking like this:

```
$TTL 3D

@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2010042801 8H 2H 4W 1D)

       NS        ns1.localdomain.

%RN-PREFIX(fd6b:4104:35ce::/64)

%RN(::1)                  PTR   server1.localdomain.
%RN(::a00:27ff:fe28:370e) PTR   client1.localdomain.
```

You would store in a file such as `db.foo.rn` and then, using a simple Makefile, create `db.foo` from that.

If you haven't already, rename the reverse zone file you want to manage so it has a `.rn` extension.

```
# mv /etc/bind/db.fd6b-4104-35ce-0000--64{,.rn}
```

You'll need to build the software, which uses the Flex tool and the C compiler; you should already have the C compiler installed, but you will need to install the **flex** package:

```
# apt-get install flex
…
```

Now, to build the software. First copy the `ipv6-dns-revnibbles.tgz` file from the `Lab Resources/DNS` folder into your virtual machine's shared folder (you set this up during the System Installation lab, remember?). Now from inside your server, unpack it and build it:

```
$ mkdir -p ~/src/ipv6-dns-revnibbles
$ cd ~/src/ipv6-dns-revnibbles
$ tar -zxf /media/host/ipv6-dns-revnibbles.tgz
$ less db.foo.rn
$ make
# install --owner root --group root --mode 0755 \
> ipv6-dns-revnibbles /usr/local/bin/
$ make -f Makefile.etc-bind
Updating db.foo from db.foo.rn
```

```
…
# install --owner root --group root --mode 0755 \
> Makefile.etc-bind /etc/bind/Makefile
```

Now, using `db.foo.rn` as a guide, update your own IPv6 reverse zone and run **make** inside the `/etc/bind/` directory.

11.10. [Optional] Fun with Hexadecimal

All this verbosity of IPv6 addresses has had some people trying to spell words in hexadecimal, much as you do with personalised number plates. Here is a completely optional, entirely unassessed, although still rather fun activity of generating easy to remember address components using a dictionary and some scripting.

```
iconv -f utf8 \                         Dictionary is in UTF-8
      -t us-ascii//TRANSLIT \           We want ASCII, removing diacriticals
      /usr/share/dict/words \           This is the default dictionary
  | grep -i '^[a-flosg]\+$' \           We want hex letters, plus some others…
  | tr '[[:upper:]]' '[[:lower:]]' \    Make them all lowercase
  | tr 'losg' '1059' \                  'l' can be replaced by '1', 'o' with '0'…
  | egrep -v '([0-9].*){3,}' \          Avoid results with more than three digits
  | egrep -v '^[0-9]' \                 Avoid results that start with a digit…
  | egrep -v '[0-9]$' \                 …or end with a digit
  | egrep '^.{4,8}'                     Want results that are 4 to 8 characters.
```

Playing around with the generated works, you could come up with often humerous yet memorable names. For example, you might take `a1fa1fa` and `deba1e` and come up with an address such as `fd6b:4104:35ce::a1fa:1fa:deba:c1e` "Alfalfa Debacle". Hmm, though perhaps it would be better to stick with blocks of 4 or 8 characters results, otherwise it gets potentially confusing where to put the `:`s, which are significant.

11.11. Last Words

Common DNS Operational and Configuration Errors
RFC1912 [ftp://ftp.rfc-editor.org/in-notes/rfc1912.txt]. Recommended reading, though quite old.

DNS-HOWTO
en.tldp.org [http://en.tldp.org/HOWTO/DNS-HOWTO.html]

BIND 9 Administrators Handbook
You'll find an HTML version in the bind-doc package. After installation, you will find the documents in `/usr/share/doc/bind/html/`. A PDF version is also available from the Vendors website.

Securing an Internet Name Server
PDF [http://www.cert.org/archive/pdf/dns.pdf] from the CERT Coordination Centre

RFC 4472 - Operational Considerations and Issues with IPv6 DNS
A useful introduction to the real-world deployment issues, observed behaviour and problems in the world of DNS.

Lab 12 Dynamic Host Configuration Protocol (DHCP)

> **Note**
>
> In this lab, we do not consider IPv6 at all, DHCP is related only to IPv4. We could have a look at DHCPv6, but that is a more advanced issue that is not currently well supported at present in default configurations.

The DHCP server that ships with most Linux distributions is the standard ISC (Internet Software Consortium) DHCPd software. You can get more information from www.isc.org [http://www.isc.org].

In order to set up DHCP on our server today, you will need to **apt-get install isc-dhcp-server**. After you install it, log messages will tell you that it failed to start. This is expected, as there is no useful default configuration that can be supplied, so the server fails to start because of the missing information. However, it has been installed correctly on Server1 after using the above **apt-get** command, so don't worry.

> **Note**
>
> In this lab, and most others, we have been loading all our services onto your Server1. However, in general there is no operational requirement for the DHCP service to be hosted on the same server as the DNS service, and on larger network it would be preferable to have them housed on different servers.

12.1. DNS Alterations

During this lab, we shall be offering a pool of addresses. These addresses should be listed in DNS. There may be other things you need to check too. You need to make the following maintenance to your DNS server:

1. There is no well-known-alias for DHCP services, so you don't need to worry about adding anything like `dhcp.domain` to our DNS.

2. Ensure you have forward and reverse DNS mappings for Client1, with the address 192.168.1.11. You do not need to concern yourself with IPv6 for this lab. Client1 will be given a static IPv4 address from the DHCP server.

3. Add in mappings for 32 hosts, starting from 192.168.1.32 (this range is exactly the same as the CIDR prefix 192.168.1.32/27). We will use these for our pool of dynamic addresses. You should employ some suitable editor wizardry, as outlined below. You may like to refer to the video about this task on the website.

   When you have a lot of workstations, you generally have some sort of numerical scheme, and so can easily write a small script to generate lots of A or PTR records. Here's an example which you would write into the DNS server configuration file. We normally store it in the file commented out, then copy, uncomment and execute to generate the output.

```
for (( i=32; i<32+32; i++ ))
do
  This version is suitable for a forward zone.
  printf "ip%03d\tA\t192.168.1.%d\n" $i $i
  This version is suitable for a reverse zone.
  # printf "%d\tPTR\tip%03d.localdomain.\n" $i $i
done
```

In **vim**, you can select this text (using the `V` command), then use `!bash` followed by **RETURN** to execute and replace the text using the **bash** interpreter. You will end up with the set of entries below.

```
ip032   A       192.168.1.32
ip033   A       192.168.1.33
ip034   A       192.168.1.34
…
```

Hit **u** to undo the changes, make a copy of the script text, and comment out one copy, so you can reuse it later. Execute the uncommented version so you get the output you want. You will also need to follow a similar procedure for the reverse zone also. Don't ignore the reverse zone, its correctness is relied on for many servers and services.

Note

If you want to do a similar thing in Emacs, use `C-u M-|` on a selection (region). Without the `C-u`, the output will be shown in a seperate buffer, rather than replacing the region in the current buffer.

BIND $GENERATE statement (for reference only, no action required)

Actually, generated data is a common requirement, so the BIND implementation has a shorthand for this in the master zone file. That would allow us to use one line in each zone file to replace each script that we used.

```
$GENERATE 32-63 ip${3,d}  A  192.168.1.$       Forward entries
$GENERATE 32-63 $  PTR  ip${3,d}.localdomain.  Reverse entries
```

We're not going to use it here, but if you want to know more, consult the BIND 9 Administrator Reference Manual [http://www.bind9.net/arm97.pdf] for more information.

4. Make the changes take effect by using **rndc reload**, and then check the logs to make sure that it loaded without complaint.

5. Test that you can resolve forwards and backwards the mappings you just modified or added. For the dynamic addresses, you need only test the border cases.

12.2. DHCP Server Configuration

You have already installed the DHCP server package earlier in this lab, so all that is left to do is to configure it.

Now we shall revise the configuration file `/etc/dhcp/dhcpd.conf`, a template of which is shown below. You will need to edit this as indicated.

```
ddns-update-style none;
option domain-name "localdomain";
option domain-name-servers ns1.localdomain;
default-lease-time 3600; 1 hour
max-lease-time 3600;

authoritative;          Send DHCPNAK when nodes request an incorrect address.

This is for the "outside" interface; we don't do anything with it.
subnet 10.0.2.0 netmask 255.255.255.0 {}

subnet 192.168.1.0 netmask 255.255.255.0 {
   option routers 192.168.1.1;
   range 192.168.1.start 192.168.1.end;     Change to suit pool addresses
}

This is an example of a static mapping.
host client1 {
   hardware ethernet 00:05:1c:05:38:ae;     Change
   fixed-address client1.localdomain;
}
```

ddns-update-style ...
When we issue a lease, we can update the data in the effected DNS zones by telling the DNS server the new values. This is called Dynamic DNS and is used mostly by Windows in a domain environment. We won't deal with it here, although it could be more useful perhaps with DHCPv6 for IPv6 addresses.

option domain-name ..., **option domain-name-servers ...**
These are the DNS-related configuration details we give to the DHCP client machines. They do not affect how Server1 interacts with the DNS system (although commonly the same DNS server will be used).

The DNS details are often common to a network, rather than a subnet, and so we have specified the DNS domain name and nameservers as globals. If needed, they can be overridden in the *subnet* stanza. Note that we have specified them by name, not address; the DHCP server resolves these names at the startup of the DHCP server.

Important

If you specify anything using DNS names, rather than an IP address, if the lookup fails, the DHCP server will *silently* ignore the entry! If this bothers you, specify IP addresses instead.

On the other hand, specifying names instead of addresses could be easier to maintain.

default-lease-time ..., **max-lease-time ...**
A lease time is how long a client can have an address before they have to renew the lease. A shorter lease time allows for updates to be made faster, at the cost of more traffic. Clients can ask for a particular lease time, or we give them a default; the requested lease time can be restricted to a particular maximum and minimum value. Suggested defaults vary widely; it depends on your network. You can change these settings for different subnets etc. Clients configured with a static address might have a longer lease time compared to those configured via a dynamic pool.

authoritative
The single most common misconfiguration is to *not* specify the “authoritative” statement. This means that your server contains the knowledge to tell clients when they are requesting something that is invalid. This is important in portable devices. Consider a laptop that requests a lease from a network at home, then later on requests to use the same address on the network at work. By instructing the DHCP server that it has the authoritative configuration for the network, it will respond with a DHCPNAK, which tells the client it must reject its current lease and request a new one.

subnet 10.0.2.0 netmask 255.255.255.0 {}
When the server starts, it will look at the available interfaces; if there is not a `subnet` clause for each subnet the host is connected to, it will refuse to start. So we have to tell the server to not do anything on our “outside” subnet.

Protecting ourselves from being a DHCP server on a network in which we are not meant to be is a very important (particularly on a multi-homed host, such as our server which is in-between two networks). We can also edit the command-line arguments of the DHCP server to specify which interfaces the DHCP server should use. To do this, edit `/etc/default/isc-dhcp-server`, changing the `INTERFACES` statement to be `inside`.

subnet 192.168.1.0 netmask 255.255.255.0 { … }
This is the “inside” subnet, the “localdomain” zone, which has a network address of 192.168.1.0/255.255.255.0 (commonly expressed as 192.168.1.0/24).

It must correspond to the configuration details of one of the host’s network interfaces.

option routers …
You can specify various options that that take effect inside the subnet. Notice that the globals options also are in effect. There are *many* different options that you can support, although they all depend on client support. Have a look at dhcp-options(5) for further ideas.

range …
Any requests for a DHCP lease that is not fulfilled by a matching `host` statement are given a dynamic address from this range. This is where client2 will get its address from. You will need to adjust this to make it agree with the 32 entries you added to DNS.

host client1 { … }
This starts a scope where we provide a static mapping between MAC address and IP address (which can also be provided as a DNS name, which is then looked up before giving it to the client).

The `fixed-address` argument could be an IP address, but we prefer to use DNS names, because that way all the mappings between ethernet MAC addresses and DNS names is contained in the DHCP configuration, and all the mappings between DNS names and IP addresses are in the DNS. It just makes things easier to deal with, and less error prone. But you do need to take care that the address maps to a valid address when the server is started, because it fails silently. You would then see the client getting an address provided by the dynamic pool configured with the `range` statement.

The argument to the host statement, which in this example is `client1` is the Client Identifier. If you don’t use a `hardware` statement, the client would have to provide a DHCP Client Identifier in order to be given the address specified by the `fixed-address` statement. Some ISPs, most notably some cable internet providers overseas, require their users to provide such a Client Identifier.

To find the MAC address of a machine, you can use the **ifconfig eth0** command on the client, or by **ping**ing the machine then checking your **arp** table. Most physical ethernet cards have it written on the card as well. Laptops often have it written on a sticker on the bottom of the unit.

The server program **dhcpd** will be started at boot-time, because you have installed the server and by doing so its startup script was activated at installation. If you want to change the options that the *startup script* uses to start the DHCP server, just edit `/etc/default/isc-dhcp-server`. For example, if you have multiple interfaces, it's useful to specify the interfaces on which it should listen in `/etc/default/isc-dhcp-server`.

Now restart the service (**sudo service isc-dhcp-server restart**) to affect the changes you have made.

Exercise
As an exercise check your logs (they will appear in `/var/log/syslog`) and make sure there is no error in the startup messages logged by the server.

12.3. Client Configuration

Note

In this section, you will need to make changes to *both* Client1 and Client2. Ensure both clients are connected to the Internal Network.

We shall assume that Client1 is like a workstation, and doesn't move around like a laptop might. Therefore, we shall *not* use Gnome Network Manager, and instead we shall manage Client1's network configuration in `/etc/network/interfaces`.

Change Client1's eth0 stanza in the `interfaces` file to the following, leaving any other interface (ie. the loopback interface) unmodified:

```
auto eth0
iface eth0 inet dhcp
Delete the address, netmask and gateway lines
```

Restart Client1. There are other things we could have done instead of restarting, but restarting can be faster in this case, and gets Network Manager out of the way. It should not be visible after it has started (because there are no interfaces available for it to manage).

There is no really special configuration for the clients, and generally no extra software to install, as a DHCP client is generally installed by default on most systems. Configuration can be as simple a single checkbox.

Exercise
As an exercise check that Client1 has an IP address of 192.168.1.11. Restart Client2, and check it has an address in the range 192.168.1.32-63.

12.4. Self-assessment

1. DNS must be well-configured.

2. The DHCP server process must start correctly.

3. Both clients must get their correct addresses: client1 gets bound to its MAC address, and the IP address 192.168.1.11; client2 gets a dynamic address.

4. What are the risks to a network when using DHCP for configuration of network details? What can be done about those risks while keeping the benefits of DHCP.

Hopefully, this will have been a short lab. You can use whatever time is left for catching up on previous labs. You should spend some time reading about layer-2 Ethernet security; you should find the *Cisco SAFE Layer 2 Security In-depth [http://www.cisco.com/warp/public/cc/so/cuso/epso/sqfr/sfblu_wp.pdf]* whitepaper informative reading.

Lab 13 Remote Terminal Services

> **Note**
>
> Note that you don't go into a virtual session until later in this lab; the first part is done on your local Mac OS X workstation.

The Secure SHell (SSH) is a very powerful tool for administering both servers and clients. The server side is generally found on Unix like machines or other devices with a dominant Command-Line Experience (eg. enterprise-grade routers), though there are clients for most operating systems in use today.

As SSH is such a useful and powerful tool for admins and users alike, half of today's lab will be on using **ssh**, and will be done on your local machine. The other half will be done on your virtual machines, and will be about administering SSH. A significant part of the labsheet is for your reference, should you want to use this outside of the lab.

The most basic use of **ssh** is as a secure replacement for **telnet**. Do these on your local machine. We will use a machine called vertex.otago.ac.nz in this lab, on which you should be able to login with your Computer Science user code and your student ID as password.

```
workstation$ ssh vertex.otago.ac.nz
The authenticity of host 'vertex.otago.ac.nz (139.80.32.49)' can't be established.
ECDSA key fingerprint is SHA256:z9M2TMCOyl0hCrcsvMcxLevUs7xEs0Pw/bsA7Fg94GU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vertex.otago.ac.nz,139.80.32.49' (ECDSA) to the list of known hosts.
Password: This is not echoed.
The following text is the message of the day /etc/motd
Last login: Thu Feb 21 15:55:02 2019 from somemachine.otago.ac.nz
This system is managed by CFEngine 3
And now we get our interactive shell.
theauthor@vertex:~ logout
Connection to vertex.otago.ac.nz closed.
Returned to where we started.
workstation$
```

13.1. Logging in Without a Password

In the lecture on SSH you learned about public-key authentication so let's try that now. Do this on your local machine.

> **Note**
>
> Note that if you've done this before, you would know your old keys will be overwritten, so you may want to move them out of the way for the duration of this lab. You can use this command to move the `~/.ssh` directory to `~/.ssh.off`. Make sure to move the old keys back if you want to remove the effect of this lab.

```
workstation$ mv ~/.ssh{,.off}
```

1. Create yourself a keypair. We'll create a RSA key with 2048 bits. Be sure to use a strong passphrase. Press **Enter** to accept the default location when asked where to put the key. Passphrases, unlike passwords, can contain spaces, hence their name. A good passphrase will have plenty of letters, mixed case, numbers, and symbols.

```
workstation$ ssh-keygen -b 2048 -t rsa
Generating public/private rsa key pair.
Just press enter for this next one to accept the default.
Enter file in which to save the key (/Users/theauthor/.ssh/id_rsa):
Use a strong passphrase! These won't be echoed.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/theauthor/.ssh/id_rsa.
Your public key has been saved in /Users/theauthor/.ssh/id_rsa.pub.
The key fingerprint is: Yours will be different…
1a:91:52:bf:41:78:7b:bc:19:a7:c1:ea:1a:cd:1f:2d Comment
The key's randomart image is:
+---[RSA 2048]----+
|o.o o.o . o.     |
| = o = . o       |
|. X + B o        |
| = * O *         |
|. . o O S        |
|.. . + o o       |
|=.. . . o .      |
|E+ .   . .       |
|O=.     .        |
+----[SHA256]-----+
```

   If you mistype the passphrase, you may need to do it all over again, and again, until you get it right. It's not uncommon for us to have to have three goes before we get it right. Some passphrase is very long; longer even than the length of the passphrase entry dialog box.

   If you find that it is taking a long time to create the key, the operating system entropy pool (which feeds the random-number generator) may have run dry. You can "stir in" some randomness by doing something like moving the mouse a lot, or causing a lot of system activity. The kernel uses the timings of such things as a source for entropy.

> **Important**
>
> Remember the private key `~/.ssh/id_rsa` must be kept secret, so it should be encrypted with a suitable passphrase.

2. Now we are ready to install the public key of the key-pair that we generated on our Mac OS X workstation into your account on Vertex. Still on your local workstation, output the public key, and copy it to the clipboard:

```
workstation$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3Nza…  copy this long line to the clipboard
```

   Now SSH into Vertex, and edit the file `~/.ssh/authorized_keys`, adding the public key to this file, which adds your public key to the list of keys allowed to access your account. Note that you will likely need to create the directory, and beware that the filename uses American English.

```
vertex$ mkdir -m 0700 ~/.ssh
```

Don’t break the lines!

The lines in this file are very long, and would wrap onto multiple lines in your terminal. Ensure that your editor does not insert newlines to break the lines, otherwise the entire file will be unusable.

This is particularly important for users of the **nano** editor. If you really want to use **nano**, use **nano -w**, which turns off wrapping.

```
$ vim ~/.ssh/authorized_keys   American English!
Paste in the public key into a single line.
ssh-rsa AAAAB4Nza…
```

3. Now log out of your SSH session from Vertex. In the next section, we will login without using a password.

13.1.1. Using the Mac OS X Keychain for SSH keys

When Apple released Mac OS X 10.5 “Leopard” they integrated the **SSH** agent with the local Keychain subsystem, which is the part of the system responsible for remembering passwords since then.

The system Keychain is generally (rarely is it otherwise) encrypted with the users login password, and here we shall take pains to try and get to you understand something. Which is likely to be stronger: a typical users login password or a typical users **SSH** passphrase? Actually, this is something of a moot point due to the *typical* qualifier: a typical user will likely have as poor of a password and passphrase as they can.

A security-conscious user on the other hand will have a fairly good password and hopefully an even better passphrase. When a key is added to the users Keychain, it is first decrypted by asking the user for their passphrase, and then adding the decrypted private key to the user’s Keychain. Remembering that the user’s Keychain is encrypted on disk with the user’s login password, and that a well-chosen passphrase should be stronger than a well-chosen password, you should be able to see that we have likely reduced the strength of the protection for the private key by using the login password to encrypt the Keychain.

Below are the steps to add your SSH key into the Keychain of MacOS and to enable the **ssh-agent** to handle the key for you automatically. Once successful, you can login to vertex without either password or passphrase.

1. First, store the key in the Keychain.

```
workstation$ ssh-add -K ~/.ssh/id_rsa
Enter passphrase for /home/cshome/User/.ssh/id_rsa:
Enter your key passphrase and you won't be asked again!
Identity added: /home/cshome/User/.ssh/id_rsa (User@workstation.otago.ac.nz)
```

2. Create a `~/.ssh/config` file and add the following lines.

```
Host *
  UseKeychain yes
  AddKeysToAgent yes
```

```
IdentityFile ~/.ssh/id_rsa
```

where it tells SSH to use the Keychain and to add the following keys to the SSH agent. Note that you can add more keys with additional lines of `IdentityFile`.

13.1.2. [Optional] Logging in Without a Password From Windows

This section is optional; but it is useful to know in order to access a server from off-campus using a Windows machine. Public-key authentication can be really useful, and sometimes it can even be required (to prevent dictionary attacks on passwords); we shall see more of this in the section on Section 13.7, "[Optional] Preventing Dictionary Attacks".

Windows, in the past, did not supply SSH client software, so third party software must be used. With the recent Windows 10 update there is a built-in SSH client, however, we have not tested it. PuTTY is a very common application for this task, which does a good job and is freely available (another option is SecureCRT, however this costs money). PuTTY is also available on other platforms such as Linux; its position in that market can be seen as a GUI SSH client.

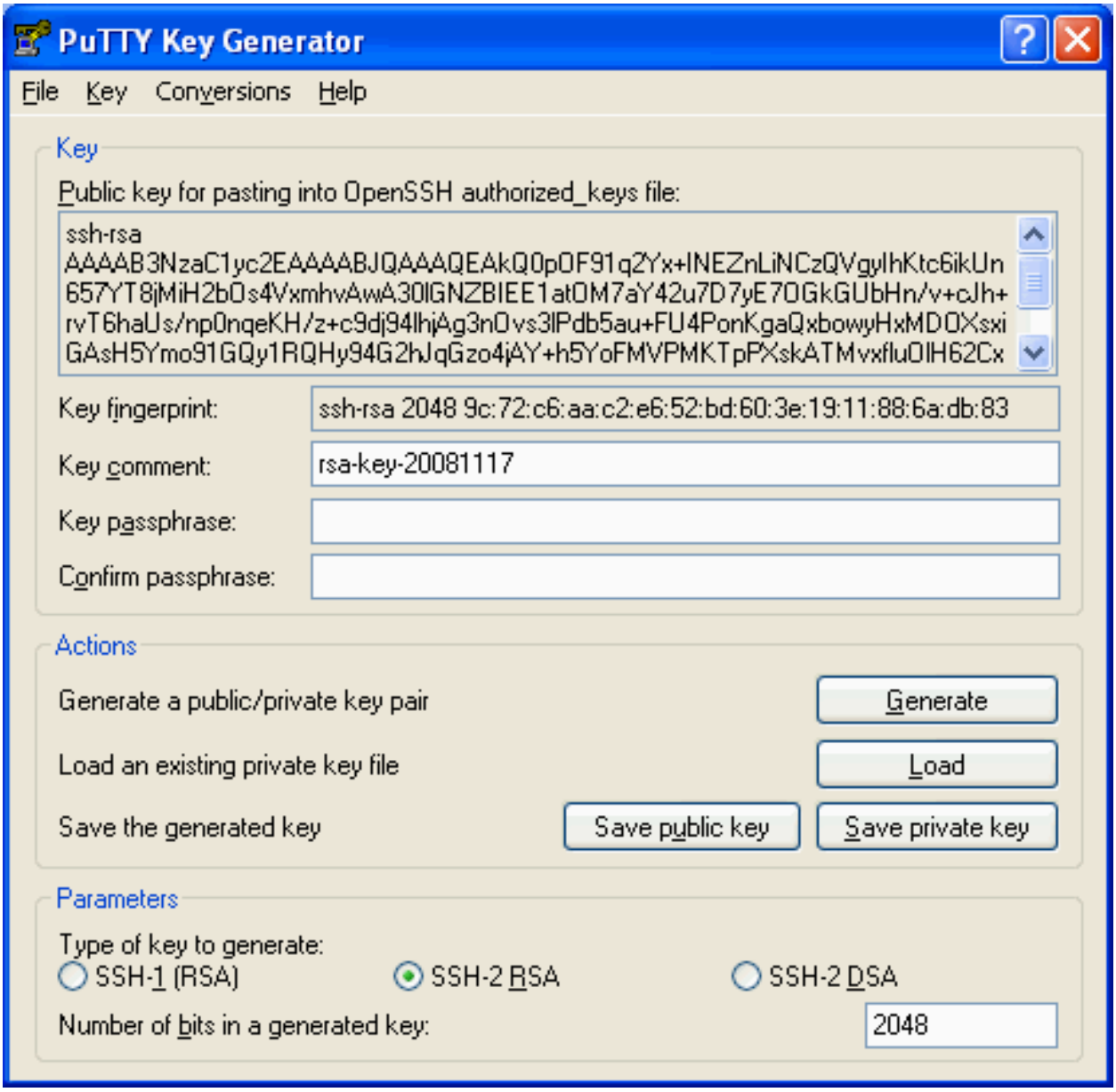
In this section we shall look at PuTTYgen for generating our key-pair; PuTTY for our SSH client program; Pageant for our SSH Key agent and WinSCP for transferring files using **SFTP**. All software mentioned here is also Open Source software.

You could reuse (import) an existing key-pair, but for this section we shall assume that we will create a new key-pair for ourselves. To do this we shall use PuTTYgen. Launch PuTTYgen from within the Start menu (you'll find it under the PuTTY folder). Select the SSH-2 RSA option (it will likely already be the default), and make the number of bits in the key 2048 instead of 1024, which is a little short for today.

Click Generate; you will notice the helpful progress meter. You will need to move the mouse as instructed… it appears that this is the only source PuTTYgen gets it entropy from. When this has completed, you should see a window similar to the following:

Tip

Despite the screen-shots being from Windows XP the proces (and interface) has remained unchanged over the years.



What you should see after initial creation of a key-pair using PuTTYgen. The default comment is not useful and should be changed, and the passphrase needs to be set.

Change the comment to something more meaningful, and set a good passphrase. When done, you need to click Save public key and also Save private key. There appears to be no standard location for this, such as `~/.ssh` on Unix hosts, so I suggest you create a folder `My Documents\SSH` and store them in there. Give the public and private key the same base name, but distinguish them by putting `public` and `private` at the end of the name (not as the extension).

Before you can use the key to log into an OpenSSH server, you must first convert the public key to the format expected by OpenSSH. Fortunately, PuTTYgen recognises this as a very common requirement and in its main window has the public key which in the correct format. Select and copy the public key.

Now login to your server using PuTTY. You will want to make some changes and set up a session for the particular server. This includes data such as: what host you wish to connect to; what username (if any) you want to log in as by default; which private key you want to use to authenticate and whether agent forwarding should be enabled. Save the session using the Save button in the Session pane. The first time you log into a machine from the client you will be prompted regarding the server's fingerprint, as the client will not have seen the server's certificate before and by default has no way of knowing if it is legitimate. Click Yes, noting that it is right to be suspicious if you don't expect this dialog.

It is best to check the fingerprint of the key to ensure that you are connecting to who you think you are connecting to.

You will see that PuTTY outputs `Server refused our key` on the terminal. This is because we have yet to install our public key in the target account's `~/.ssh/authorized_keys` file. Open this file in an editor and paste your key into it ensuring it is stored as a single line.

```
Server refused our key
Using keyboard-interactive authentication.
Password:
```

The server refuses our key because our public key has not been installed into the account.

Tip

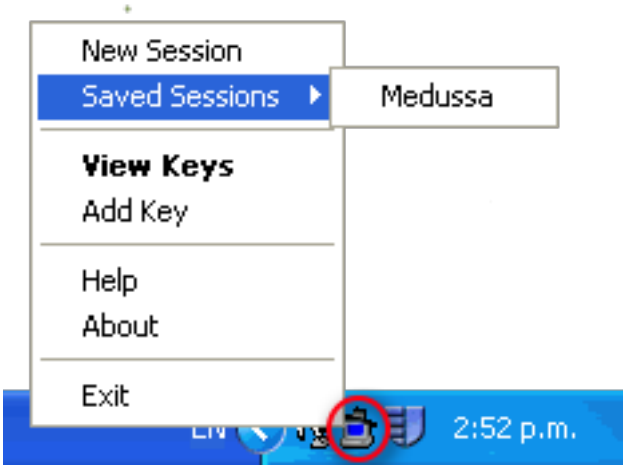
To paste in PuTTY simply right-click. This is counter-intuitive (note that middle-click doesn't work, but a Windows user wouldn't expect that to work), and Ctrl-V doesn't work because that is passed through the SSH channel.

Log out when you are done. Now when you log into the server again you should be asked `Passphrase for key "key comment"` rather than for the password. We will be asked this each time because we're not using a key agent (Pageant) at this time; let's remedy this now.

```
Authenticating with public key "Cameron CKERR-XP"
Passphrase for key "Cameron CKERR-XP":
```

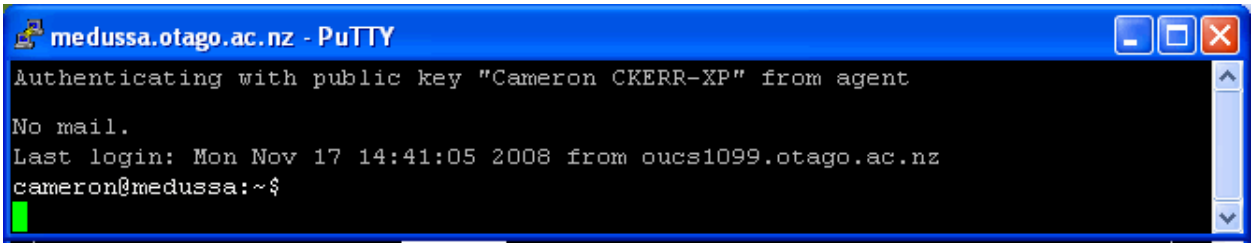
We are asked for our passphrase each time because there is key agent currently running.

Start Pageant from within the PuTTY menu of the Start menu. A new icon will appear in your system tray. If you right-click on this icon you will get a menu from which you can interact with Pageant or start any saved PuTTY session. Use the View Keys item to add the key to your key list; you will be asked to unlock your private key using by entering your passphrase.



Pageant is software that caches unlocked private keys; it plays the role of the key agent.

Now, when you use PuTTY to connect Pageant will be consulted for the unlocked private key instead of prompting the user for the passphrase.



Now that our private key has been unlocked and loaded into the key agent our SSH client doesn't need to ask the user.

Now let's try WinSCP, which is made by different people. WinSCP also includes Pageant and PuTTYgen as part of its installation, so it should work.

Launch WinSCP. Fill in the entries for hostname, username and private key (*not* password). We would also suggest setting the SSH protocol version to 2 only and only selecting Enable compression if you are going over an Internet link or WAN. Click Save… to save your session details. Then click Login to log into the server.

You will notice that a different server key cache is used, so you will need to tell WinSCP to trust the server's key since this is the first time you will have connected to the given server using WinSCP on this machine. WinSCP is not a bad program, but it does have one major annoyance: it doesn't have good support for Unicode filenames… the WinSCP program was not written as a Unicode application, and it would apparently take a lot of work to make it so.

13.2. Using ssh

Now that we've got fast, secure, and convenient access, the world (or at least, our network) is our oyster. Lets try using **ssh** in various ways. We've already seen how we can get a simple terminal, how about something more exciting? Try these commands.

> **Note**
> Some commands may not work on MacOS if the X11 server is not available.

- Run a command on a remote host and receive its output.

```
workstation$ ssh vertex.otago.ac.nz uptime
Banner removed.
 11:29:06 up 41 days, 33 min, 10 users, load average: 0.00, 0.00, 0.00
```

- Interactive programs often require a terminal (or rather, a psuedo-terminal)

```
workstation$ ssh -t vertex.otago.ac.nz top
Banner romoved.
The 'top' program should run. Type q to quit.
```

- What about an X11 program? We can do that too. This requires LAN speeds to be really effective, although with compression turned on can also be used over a broadband internet link. Servers have this turned off by default.

```
workstation$ ssh -X vertex.otago.ac.nz xclock
Banner romoved.
A clock should pop up on your display.
It may not work on MacOS if the X11 server is not available.
```

- Banners are written to stderr, so we can redirect that if we really don't want to see banners (note though that this might also cause other things to be ignored, which could be bad). In this case we're counting the number of lines in the output (which as it happens, isn't very interesting at all.)

```
workstation$ ssh vertex.otago.ac.nz uptime 2>/dev/null \
> | wc -l
      1
```

- **ssh** is just another process, so you can still easily pipe the output. In the command **local_cmds | ssh ... remote_commands**, the output of local_commands is passed to the local command **ssh**. **ssh** logs into the remote end. On the remote end, the remote SSH (**sshd**) starts remote_commands. The local and remote side send any data between them, essentially gluing the output of local_commands to remote_commands.

  This example shows how you can use **ssh** to send a very simple backup (the output of **tar** in this case) to a remote machine. The command **cat > backup.tgz** will be run on the remote end. We need to escape special characters in the command to prevent the shell from interpreting them on the local side.

```
workstation$ tar -zcv some_directory | ssh vertex.otago.ac.nz \
> cat \> /tmp/$USER-backup.tgz
$USER isn't escaped, so it will be expanded locally.
```

- What if the command we want to run remotely contains shell meta characters? In the first command below, the * is expanded on the local side, expanding to a set of files particular to the local machine. In the second command, the * is escaped, making it be expanded on the remote side, which is what we want.

  You can get some really complex commands going; for complex commands, you should put them into a script, if you value your sanity, as the amount of escaping that is required grows wildly.

```
Your output for these commands will look different.
Here the * is expanded on the local side.
workstation$ ssh vertex.otago.ac.nz du -sh /tmp/*
Your output will differ, but it will still likely fail.
du: cannot access `/tmp/501': No such file or directory
…
Here the * is expanded on the remote side.
workstation$ ssh vertex.otago.ac.nz du -sh /tmp/\*
0       /tmp/file-on-vertex
4.0K    /tmp/another-on-vertex
…
```

- What about running **ssh** in the background? It may need to ask you for authentication (either a password, of a pass-phrase if it can't get it from an SSH Agent), so we can't use & to background it normally, so we inste-ad use the option `-f`, which tells **ssh** to go into the background only after it has completed the authentication process.

```
workstation$ ssh -f -X vertex.otago.ac.nz xload
```

- When scripting, you generally don't want **ssh** asking for any interaction from the user, otherwise it would hang indefinately waiting for input. We can use a particular option to tell **ssh** not to do anything that requires interaction. If it needed to, it would fail immediately.

```
workstation$ ssh -o 'BatchMode=yes' vertex.otago.ac.nz uptime
 11:29:06 up 41 days, 33 min, 10 users, load average: 0.00, 0.00, 0.00
```

- Things not working as expected? The first thing to try would be turning on some debugging messages. If this or the system logs on the server don't reveal the problem, then you can start the server with debugging turned on. Consult sshd(8) for more information. In a production system where others are expecting to use the SSH service, you will probably want to start the debugging server on a different port (but check firewall configuration) so other users can continue to log in normally… assuming of course SSH is currently usable.

  On the client-side, you can enable more verbose output using `-v`, although to be honest in our experience they don't tend to be very useful for solving most authentication-related problems.

```
workstation$ ssh -v vertex.otago.ac.nz
…
```

- Sometimes, you need to abort a connection. This is most common when you've lost your network connection (eg. you've forgotten to log-out before disconnecting from the network on your laptop). You can't use **Ctrl+C**, because that gets passed through to the remote side.

  Instead, type the sequence **Return ~ .**. That's three keys, one at a time, and remembering that **~** also needs **Shift**. If you make a mistake, you must start over, as a backspace will break the sequence. This key sequence will abort the session and return you to your local prompt. The **~** is the escape character, and is only recognised at the beginning of a line (hence the **Return** first). The dot after the escape character says to abort the connection. Another **~** just sends a ~. You can see a list using **~ ?**

13.3. scp & sftp

scp is a service that is provided by the SSH protocol, and its all about copying files (and directories). Its not as flexible as **ssh** however, and slower compared to unencrypted traffic, but still useful for the task it was designed for. You can copy remote to local, local to remote,

local to local, and even remote to remote, although that's inefficient compared to direct copying between the two machines.

- Local to remote. It will end up in the home directory (the directory you end up in just after you login) on the remote system. Try this between your local machine and vertex. Note that the colon (`:`) is important as it signifies a remote file.

```
workstation$ scp myfile.txt vertex.otago.ac.nz:
…
```

- Remote to local. Here we show how you can specify a different usercode on the remote system. Remember, `.` is the current directory.

```
workstation$ scp user@vertex.otago.ac.nz:myfile.txt .
…
```

- You can copy a directory using the `-r` (recursive) option.
  Use a small directory for trying this command.

```
workstation$ scp -r source-dir vertex.otago.ac.nz:target-dir
…
```

FTP is not a protocol that can have cryptography easily wrapped around it. We can use **sftp** to provide the feel of a fairly basic ftp client, though it is, in no way, the FTP protocol. Actualy, it uses the scp mechanism.

```
workstation$ sftp vertex.otago.ac.nz
Connecting to vertex.otago.ac.nz...
sftp> Type `?' if you don't know how to use ftp.
```

13.4. Server (sshd) and Client Configuration in Linux

In this section we shall move into our virtual environment, so start up Server1 and Client1.

Install and configure sshd

Install the `openssh-server` package on Server1 only.

The configuration files for the SSH server are stored in `/etc/ssh/`. The main configuration file is `sshd_config`, which controls the daemon. There is also `ssh_config`, which controls the default behaviour of the SSH client program. Finally, there are a number of keys, which are the public and private keys for the server. The only thing we need to edit is `sshd_config`.

You can get information on what the various keywords mean by consulting the manual page for sshd_config(5) and/or sshd(8). You'll need to harden the configuration and tailor the settings of the daemon to suit our requirements. Note that not all the options may have an effect, as not all possible features are compiled into the binary at compile-time.

Make the following settings in `sshd_config` on your SSH server. You should consult the appropriate manpage.

- Turn off anything to do with rhosts, or host based authentication. (Should be off by default anyway).

- Do you want to allow root to login? On a workstation this can be useful for management tools. For example, on a management machine you could imagine a manager wanting to run an administrative command on each machine, perhaps in a loop. For example:

```
example$ for wsn in $(cat workstations-lab-A.txt)
example> do
example>   ssh -o BatchMode=yes root@${wsn} apt-get install …
example> done
```

  At other times this is inadvisable. You should never enable this option on servers. On Ubuntu it is, by default, pointless. This is because root's account is locked by default.

- Do you want X-Forwarding available? You'll find it will default to different values on different systems, so it pays to check and make any configuration explicit. This is useful if you want to use GUI programs on the server, but have them display locally. Most servers ought not to have GUI environments, but many do. It's more useful on workstations.

- In some countries, in order to get protection from the law, you have to show a banner when someone tries to login. Make a simple banner `/etc/issue.net` that says something containing the hostname (hint: look at the **banner** program available in the 'sysvbanner' package), and the string AUTHORISED USERS ONLY, and tell **sshd** to display it.

> ## Tip
>
> Choose your words carefully. Avoid using words like "welcome", which could allow an attacker to defend their actions.

Restart the server (**/etc/init.d/ssh restart**) and check that it's running.

Currently, you will find that you won't be able to access the SSH service, because most versions of OpenSSH are configured to use TCP Wrappers, which we have previously configured to default-deny.

You should normally decide on an appropriate access policy, whether people can access any machine from the internet (typically this is bad, due to password dictionary attacks), or whether they must first log into another machine.

You could use one of the following lines in `hosts.allow` to allow the appropriate level of access to SSH:

```
sshd: ALL
OR
sshd: 127.0.0.1 [::1] 192.168.1.0/24 [fd6b:4104:35ce::]/64
```

> ## Breaking an entry into multiple lines
>
> If the above square brackets around the ::1 address or any IPv6 addresses are forgotten, it will cause the whole line to be ignored by **tcpd**. This will prevent any access to SSH. If instead we had put in at least two lines (one for IPv4, one for IPv6), it would have only been a partial break if we missed the square brackets for IPv6, instead of a complete break. At least IPv4 would work for SSH in that situation. Therefore, it is better to have multiple lines in `hosts.allow`, one for each IP address or each class of IP addresses.

Double-check that `hosts.deny` has `ALL:ALL`, or at least `sshd:ALL` to block unwanted SSH or other connections.

Check that you can login to Server1 from Client1 as the user “mal”. Before doing so, make sure the server and the client are connected via an internal network in VirtualBox.

```
client$ ssh mal@server1.localdomain
```

Set up ssh-agent for password-less login

Once you can login to Server1 from Client1, logout Server1 and follow the steps below to configure **ssh** in Client1 so that you can login to Server1 without password.

1. Generate a RSA key in Client1 with **ssh-keygen** as done previously.

2. Copy the public key in `~/.ssh/id_rsa.pub` in Client1 to the file `~/.ssh/authorized_keys` in Server1 under the home of `mal`, similar to what you have done before. This will tell **sshd** in Server1 to use the key instead of the password for identity check of `mal` at login time. Note you need to create the directory `.ssh` if it does not exist.

3. Start **ssh-agent**:

```
mal@client1:~$ eval $(ssh-agent -s)
```

   The **eval** will execute **ssh-agent** and export the environment variables used by **ssh-agent** to the current login session.

4. Configure **ssh** for **ssh-agent** to use the generated RSA key. Create a file `~/.ssh/config` with the following content in Client1.

```
Host *
  AddKeysToAgent yes
  IdentityFile ~/.ssh/id_rsa
```

   You can add more keys as you wish with more lines of `IdentityFile`. Alternatively, you can use **ssh-add** to add keys manually but you will have to use the command again after logout as the keys are not permanently added by **ssh-add**. To permanently add a key for **ssh-agent**, edit the above `~/.ssh/config` file.

5. Now try login to Server1 from Client1:

```
mal@client1:~$ ssh mal@server1.localdomain
```

   You need to type the passphrase for the first time. Then for future logins, you need to type neither passphrase nor password. Try logout from Server1 and login again. Do this for a couple of times.

Though we have used **ssh-agent** successfully, the previous work was only temporary. If the system reboots, you will have to repeat most of the above procedure. To avoid that, you need to edit some files as below.

1. First, kill the **ssh-agent** process:

```
$ ssh-agent -k
```

2. Add in `~/.profile` the following command for starting **ssh-agent** automatically at login time.

```
eval $(ssh-agent -s)
```

You can just add it at the end of the file.

3. Add in `~/.bashrc` the following line to kill the **ssh-agent** process after logout.

```
trap '/usr/bin/ssh-agent -k' EXIT
```

Again you can just add it at the end of the file.

Now reboot Client1 and see if everything works as expected. You can try every **ssh** command from the previous sections. If you want to try X11 related applications like **xclock**, you should install them on Server1 as below.

```
$ sudo apt install x11-apps
```

Then you should see the following command works.

```
$ ssh -X mal@server1.localdomain xclock
```

13.5. Local Port Forwarding

Port forwarding is an advanced topic, and not something I require you to know how to do, so don't worry about it for this lab. It is such a powerful tool though, it would be disappointing if the more adventurous of you didn't try it.

Here is an example of how to use Local Port Forwarding, which is the most common type. Note that just because you *can* do something, it doesn't mean you should. Seek permission from any administrators before potentially violating policy.

In the lecture on SSH, you were given an example of how you could use local port forwarding to tunnel a clear-text protocol inside an encrypted SSH session. Earlier in the course, we implemented a simple, clear-text file-transfer service called "tinyfs". Let's enarmour tinyfs so we can use it over the network. We can use a tool such as **tcpdump** or **wireshark** to verify that nothing about tinyfs is being sent in the clear over the network.

```
Start the tunnel
client$ ssh -fNL 9000:localhost:900 server1.localdomain
Banner removed.

See what's listening
client$ lsof -Pni    Don't need root privs here.
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
An established connection to the SSH service on Server1
ssh     2481  mal    3u  IPv6  16213      0t0  TCP ↵
    [fd6b:4104:35ce:0:a00:27ff:fe99:c27d]:36218->[fd6b:4104:35ce::1]:22 (ESTABLISHED)
And IPv4 and IPv6 listening ports on loopback TCP/9000
ssh     2481  mal    4u  IPv6  16285      0t0  TCP [::1]:9000 (LISTEN)
ssh     2481  mal    5u  IPv4  16286      0t0  TCP 127.0.0.1:9000 (LISTEN)

Start up wireshark, starting a capture immediately
Don't forget to dismiss the warning window about running as root.
client$ sudo -b wireshark -i eth0 -k

Connect to our forwarded port
client$ echo '/etc/hostname\r\n' | nc -q-1 ::1 9000
OK     Success, output from the server
server1

When you're done, tear down the tunnel
```

```
client$ ps -eo pid,command | grep ssh
  825 /usr/sbin/sshd
 1256 /usr/bin/ssh-agent /usr/bin/dbus-launch --exit-with-session gnome-session
 2481 ssh -fNL 9000:localhost:900 server1.localdomain    Note
 2601 grep --color=auto ssh
client$ kill 2481
```

Have a look in Wireshark. Did you see any TCP/900 traffic? What if you right-click on one of the SSH packets and select Follow TCP Stream? You should see that nothing is sent in the clear over the Ethernet, and the SSH traffic is unintelligable because it is encrypted.

Repeat the experiment, but this time start the capture on the “lo” loopback interface. What do you find now? Should you be worried?

You may be wondering what the significance of the port numbers 9000 and 900 is. There is nothing really significant here, except that port 900 is the TCP port that our tinyfs service uses, and that port 9000 is above 1024 and so doesn’t need root privileges to accept connections. Otherwise, there is no limitation, so long as the port is not already used. The two port numbers could even be the same, but usually we end up using a high-numbered port for the local side.

13.6. User Imposed Restrictions on Public Keys

We’ve talked a lot about using keys to gain access. But how can you restrict what a key may be used for? Edit, creating the directory (mode 0700) and/or file if they don’t exist, `~/.ssh/authorized_keys` on the machine(s) you wish the key to be used to log into. We can restrict what a key can do by inserting some rules at the start of each line (note the lines are very long and will wrap).

The various restrictions you can put on a key are outlined in the man page for sshd, under the section titled “AUTHORIZED_KEYS FILE FORMAT”. There are some examples further down the page. There aren’t many, so you can read them all.

1. Add a restriction such that you can no-longer request X11 forwarding. Test it. You may remove the restriction after you have completed the test.

13.7. [Optional] Preventing Dictionary Attacks

> **Note**
>
> This section is purely for your information; you are not expected to implement these protection measures, but it will be useful to you later on if you ever operate internet-facing SSH servers. These measures are also applicable to other services, not just SSH.

As you will rapidly come to realise when you have an internet facing (meaning able to accept connections from the internet) SSH server, there are many attempts of people trying to break into your system using known/harvested username/password pairs, or just trying to find weak passwords by brute force. Most of the time it just creates a lot of noise in the system, but sometimes they succeed, and then they have a vector into the rest of the network, and possibly

an easier method to get root access to the system (if that was their goal, they may just want to enlist your server in their illicit activities.

Split access policy

For this reason, it can be quite useful to disable password authentication, and prefer to use authentication systems such as public-key authentication or other authentication systems such as one-time pads, where the password is different each time or calculated via some other parameter. However, to use public-keys, you need to able to put the key into the account. It also becomes more important to carry your encrypted private key (and usually also your public key) around with you when you travel, typically on a USB flash drive. In order to install your public key into your account, you either need to a) access the system locally and copy the key over, b) access another system that has the same home directory mounted, c) ask a system admininistrator to install it for you, or more likely d) authenticate via a password and install the key so you can subsequently authenticate using your public-key.

(d) above would be nice, but it does require that we are able to configure **sshd** to have a policy that allows password authentication to known client systems, and disables it for systems on the internet. Fortunately, recent versions of OpenSSH give us a way to do just that, using the `Match` keyword.

Let's say for now that we only want to allow password authentication to IPv4 clients in 192.168.1.0/24, and public-key authentication can be used from anywhere. Phrased another way, be default we want password authentication to be disabled, and allow it explicitly.

Edit `sshd_config`, find, uncomment and disable the PasswordAuthentication option. Then, *at the end of the file*, add the following:

```
PasswordAuthentication no         Deny by default
…
at end of file
Match Address 192.168.1.0/24      Effective until next Match block or End-of-File
     PasswordAuthentication yes
```

To test, we shall need to first ensure that the client won't authenticate using public-keys (temporarily). We can do this by renaming `~/.ssh/authorized_keys` on Server1 to something else, such as `~/.ssh/authorized_keys.OFF`. That way, it will fall-back to trying a password.

Thus, according to our policy, only login via IPv4 should work. This is because public key authentication is the only allowable method for IPv6, and we've essentially broken that temporarily for our testing.

The easiest way in our particular case would be to force the use of IPv4 or IPv6 using the `-4` or `-6` option to **ssh**. Alternatively, we could specify the host as either `server1.ipv4.localdomain` or `server1.ipv6.localdomain`. Remember that typically, `server1.localdomain` will use the IPv6 address first.

When using such a policy, it becomes important to educate your users on the new policy, and provide resources for them to understand how to use the new authentication mechanism, and why this policy is in place to begin with.

Dynamic Banning

Another way to protect yourself is to scan your logs periodically (perhaps every 10 minutes), and dynamically ban — using `hosts.deny` or a firewall rule — any IPs where repeated

authentication failures are happening, especially if the username that is being authenticated does not exist in the system. There are a number of products out there to assist with this. DenyHosts is one that scans the logs and edits `hosts.deny`, depending on particular rules that are set by the administrator. However, because an attacker might get lucky and successfully authenticate before being banned, DenyHosts supports a *synchronised* mode, whereby many participating servers tell a central repository which machines they have banned, so other systems can ban them also.

13.8. Self-assessment

1. Create an SSH keypair on Client1, and demonstrate that you can use it to login to Server1 without a password and use the **ssh** command in various ways.

2. Your server and client configurations must be suitably well-configured. You should understand the relevance of some of it to the behavior of SSH.

3. Give three useful policies or practices that users should be encouraged or even required to do with regard to the secure use of SSH public keys, particularly with regard to Mac OS X 10.5 “Leopard”.

Lab 14 Electronic Mail

E-mail is a large and rather complex topic, but is very important. Being such an important part of everyday life, e-mail has a lot of things to consider. We could talk about many aspects of e-mail administration, but we'll stick to the basics.

By the end of this lab, you should have a good idea of what is involved when setting up a very basic e-mail server, how mail is delivered and received, and how such things as mailing lists can be set up. We don't have time today to cover other important e-mail issues such as SPAM and Virus prevention.

14.1. Preliminary Configuration

1. In our case, we're going to have all our e-mail services run on our Server1, but this needn't be the case.

2. Add an MX record with priority 0, for localdomain, that points to our servers IPv4 and IPv6 address. The second argument of an MX record must not be a name with a CNAME record, but rather something that has an A or AAAA record[1].

3. Add an alias smtp.localdomain which points to the e-mail server, for both IPv4 and IPv6. Remember, this is an alias, so don't add an entry to the reverse zone.

4. Add another alias called pop3.localdomain, in the same way as the previous alias.

5. Make the changes take effect, and test to make sure the mappings are correct and working.

```
$ dig +short -t MX localdomain
0 smtp.localdomain.
$ dig +short -t ANY server1.localdomain
192.168.1.1
fd6b:4104:35ce::1
$ dig +short -t ANY smtp.localdomain
192.168.1.1
fd6b:4104:35ce::1
$ dig +short -t ANY pop3.localdomain
192.168.1.1
fd6b:4104:35ce::1
```

6. Add a usercode for a fictitous user called bob on Server1. Use **adduser bob** for this. We shall be using this user later to test our delivery.

14.2. Install and Configure Exim

> **Note**
>
> This section will be done entirely on Server1, which in our case is the e-mail server.

Exim is a MTA (Mail Transfer Agent) that is largely Sendmail compatible. Sendmail is a very old (but still actively maintained) MTA that is very powerful. Unfortunately, Sendmail has a

[1] See RFC1912.

spotty security history, and its configuration takes a lot of knowledge to do well. Indeed, its been said to require “black magic” to know how to do well, and is a valuable skill for a Unix systems administrator to add to their CV, as Sendmail is one of the most flexible MTAs on the planet.

Exim has a large user base, as it is the default MTA used in the Debian GNU/Linux distribution. Having a large user base means that its relatively stable, and that there is a large community you can ask for help. Exim is also easy to learn, while still being reasonably flexible.

You should have already installed Exim when **logcheck** was installed. However, if you haven't installed it yet, use **apt-get install exim4** to install.

Reconfigure the service using **sudo dpkg-reconfigure exim4-config**. Respond to its questions with the following answers:

Configuration type
Respond with “internet site; mail is sent and received directly using SMTP”.

Note that networks such as our small domain, that has an invalid domain name (“localdomain”), would typically be better off using “mail sent by smarthost; received via SMTP or fetchmail”. This is a useful configuration for a gateway machine such as ours which doesn’t have a public domain. Any message that gets forwarded will need to have its addresses rewritten, so the invalid domain “localdomain” is not seen. It would be changed to something more appropriate, such as our publically addressable e-mail account.

System mail name — used to qualify addresses without a domain.
In our case, we want to consider all our mail as `user`@`domain`, and our domain is `localdomain`, so respond with `localdomain`. Normally, it would be something like `example.com`, but we only have one DNS label.

IP addresses to listen on
We would like to listen on the inside interface and the loopback interface, both for IPv4 and IPv6. In our case, where we don’t actually accept mail coming in from the internet, only be able to send e-mail, we don’t need to listen on the outside interface.

We are asked to respond with a semi-colon seperated list of addresses. Respond with the following:

```
127.0.0.1 ; ::1 ; 192.168.1.1 ; fd6b:4104:35ce::1
```

Other destinations for which mail is accepted
We don’t operate as a final destination for any other domains, so we don’t generally need to add anything here. We already accept e-mail for `mailbox`@localdomain, but perhaps we might also allow `mailbox`@server1.localdomain. However, possibly because we only have one DNS label in our domain, we also need to add `localdomain`, so respond with `localdomain;server1.localdomain`.

Domains to relay mail for
Leave this empty, as this list is for *other* networks which we forward (relay) mail for. We might do this if we were an e-mail gateway or acting as a backup MX (mail exchanger) for someone else’s domain. In other words, what goes here is the list of networks that we may accept mail for, but are not the final destination.

Machines to relay mail for
Generally, this would be set to your internal network, because client machines generally shouldn't be able to send e-mail without going through the mail-server[2], so respond with the following:

```
192.168.1.0/24 ; fd6b:4104:35ce::/64
```

Keep number of DNS-queries minimal?
No

Delivery method for local mail:
Accept the default, as that is what a lot of local tools expect. There are good reasons for using the alternative however.

Split files?
Yes

Before we go on and have a look at what we've just done, let's setup the system aliases so common e-mail addresses, such as postmaster and hostmaster exist. Also, it gives us a way of redirecting system accounts, such as root, to a real person, who should regularly check their e-mail. The right hand side of an alias can either be a qualified or unqualified e-mail address, such as `person`, or `person@example.com`. You could even have multiple right hand sides, useful if there are a few administrators, of if you want to make a small, manually managed mailing list.

Make all the system mail (ie. all the mail for `root` and any system accounts) eventually reach your account (ie. `mal`) on the mailserver. Open `/etc/aliases` and add the following line (it may be already there):

```
root: mal
```

In some versions of Ubuntu (and other Linux versions) it may already be there by default

Note the mailboxes ("users") called postmaster, hostmaster, webmaster, abuse, security etc. These are well-known aliases that are associated with e-mail, DNS, WWW, and security administrators. There are expected to be monitored e-mail accounts so any problems can be properly reported. You will recall that we have used the hostmaster address when defining our DNS zone files.

If these entries don't exist, then create them now. Eventually the mail is sent to `mal`, the administrator.

Okay, now that we've dealt with our aliases for the moment, let's check what happened when we answered all those questions previously. Debian manages the contents of the Exim configuration file for us, to enable as much management to be done via package management drop-ins, and the Debian Configuration (Debconf) system. It was Debconf that was asking us those questions via the menu system earlier. **Read from section 2 to the end of section 2.1 of `/usr/share/doc/exim4-base/README.Debian.gz`; use zless to view it.** You need to understand how the configuration files can be managed as there are a number of choices with different tradeoffs.

Write down the three ways that the Exim configuration can be managed under Debian (hint: one of the ways is to not have Debconf manage the configuration at all).

[2]This enables egress scanning for viruses, etc. To implement it, a firewall would be required to ensure only the e-mail server can create outgoing e-mail connections.

Have a look at the autogenerated file that Exim is given as its configuration file: `/var/lib/exim4/config.autogenerated`. You will notice the file makes heavy use of conditional configuration macros, which makes the file harder to read. Unfortunately, these conditions[3] are interpreted by Exim itself, and not some preprocessor, so it is harder for us to see the file after the all of the macros have been evaluated. For the sake of comparison, have a look at the example configuration file (that doesn't use macros) in `/usr/share/doc/exim4-base/examples/example.conf.gz`, to get a feel for what the end-result looks like.

Thankfully, we can also ask Exim to dump its configuration elements to stdout using the `-bP` command-line option. Run **exim4 -bP | less** and have a look.

Like most packages that contain system services, the Exim package contains a startup script that will be run at boot-time. But now we need to restart Exim to affect our changes. However, before we do this, it is a good practice to check our configuration file (if such a facility is provided). Check the file, and then restart Exim using the following commands. You will notice that the startup script executes **update-exim4.conf** to (re)-generate the configuration file and store it in the correct place.

```
# /etc/init.d/exim4 stop
$ pstree
Ensure that there are no exim4 processes, if there are…
# killall exim4

# /etc/init.d/exim4 start
$ pstree
Check there is ONE exim4 process

# lsof -Pni | grep exim4
exim4 … IPv4 … TCP 127.0.0.1:25 (LISTEN)
exim4 … IPv6 … TCP [::1]:25 (LISTEN)
exim4 … IPv4 … TCP 192.168.1.1:25 (LISTEN)
exim4 … IPv6 … TCP [fd6b:4104:35ce::1]:25 (LISTEN)
```

Exercise

As an exercise make sure the output of **lsof** is as expected.

Being a high-volume service, e-mail servers often do their own logging, rather than using syslog. Exim will log to files in `/var/log/exim4/`. The `mainlog` file is the standard file you will want to look at. There may be a `paniclog` file; it is generated if the server fails to start or crashes for some reason.

For the remainder of this lab, you may like to run **tail -f /var/log/exim4/mainlog** in a spare virtual console to keep an eye on your logs. The `-f` option causes **tail** to not exit, and keep outputting any new entries in the file.

Exercise

As an exercise check your logs to ensure the server started cleanly. This will show that you are able to find logs, which are a valuable resource.

[3] The macros such as `.ifndef`, which read as "if not defined"

14.3. Testing Submission by Hand Using SMTP

In this section, we're going to send some test messages to `bob@localdomain`, from the client machine, which should appear in his mailspool file `/var/spool/mail/bob` on the server. This file is in *mbox* format, which is the common form on Unix systems. Bob (and you) will read your e-mail on the server using the MUA called **mutt**, which you will first need to install on the server:

```
# apt-get install mutt
```

Use **telnet** from Client1 to connect to port 25 (the SMTP port), on the mailserver for localdomain. Here is a transcript of what you can type, this is speaking the SMTP protocol. This is what a MTA would do when sending an e-mail. As you do this, keep an eye on the logs being generated on the server.

```
mal@client1:~$ dig +short -t MX localdomain
0 smtp.localdomain.
mal@client1:~$ telnet smtp.localdomain 25
Trying fd6b:4104:35ce::1...
Connected to smtp.localdomain.
Escape character is '^]'.
220 server1.localdomain ESMTP Exim …
EHLO client1.localdomain
250-server1.localdomain Hello client1.localdomain [fd6b:4104:35ce::a00:27ff:fe28:370e]
250-SIZE 52428800
250-PIPELINING
250 HELP
MAIL FROM:mal@localdomain
250 OK
RCPT TO:bob@localdomain
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
From: mal@localdomain
To: bob@localdomain
Subject: Hi Bob
Blank line seperates header from body.
How are you today?
.
250 OK id=195bKY-0000f3-Uq
QUIT
221 server1.localdomain closing connection
Connection closed by foreign host.
```

If you look at your server logs, you should now see that the delivery succeeded; the log enty will look similar to the following:

```
timestamp 1OEATZ-00003WL-3M <= mal@localdomain H=client1.localdomain ↵
   [fd6b:4104:35ce:0:a00:27ff:fe28:370e] P=esmpt S=size
timestamp 1OEATZ-00003WL-3M => bob <bob@localdomain> R=local_user ↵
   T=mail_spool
timestamp 1OEATZ-00003WL-3M Completed
```

Now log into Server1 as Bob, and have a look at the users spool file. This file is in mbox format, and stores all the messages in the users Inbox.

```
bob@server1:~$ cd /var/mail/
```

```
bob@server1:/var/mail$ ls
bob   mal
bob@server1:/var/mail$ less bob
From mal@localdomain Tue Jan 06 16:40:33 2015
Return-path: <mal@localdomain>
Envelope-to: bob@localdomain
Delivery-date: Tue, 06 Jan 2015 16:40:33 +1200
Received: from client1.localdomain ([192.168.1.11])
        by smtp.localdomain with esmtp (Exim 4.44)
        id 195bKY-0000f3-Uq
        for bob@localdomain; Tue, 06 Jan 2015 16:40:33 +1200
From: mal@localdomain
To: bob@localdomain
Subject: Hi Bob
Message-Id: <E195bKY-0000f3-Uq@smtp.localdomain>
Date: Tue, 06 Jan 2015 16:40:33 +1200

How are you today?
Use q to exit less
```

Exercise

As an exercise make sure at least you can see one message as above (if you did it correctly the first time there will be only one message). Notice the line that starts with `From`, but doesn't have a colon after it. That is part of the mbox format, and is there to serve as a message seperator.

If it didn't work, consult the Section 14.4, "Troubleshooting" for some troubleshooting advice.

On Unix systems, once the MUA has closed the spool file, it removes any seen messages into the user's mail folder in their home directory, such as `~/mail/` or `~/Mail/`. However, a lot of systems are migrating away from the mbox format to a more robust Maildir format, so we don't go over it much here. Most users won't read their e-mail this way anyway.

Exercise

Note the headers that were added to the message by the MTA. Now, as bob on the mailserver, start **mutt** (you installed it previously) as an exercise and navigate your way to your inbox. View the message (**Enter**), and exit (**q**, **q**, **y** to fully quit). Look at the raw file again, you should notice that some new headers have been added. What do you think the Status header is used for?

14.4. Troubleshooting

This section is useful if you are having trouble getting Exim to deliver messages correctly. It addresses inspecting the mail queue, viewing the messages in the mail queue, and what to do when you think you may have fixed the problem.

Beware restarting

When reconfiguring Exim, experience has shown that it pays to explicitly stop the Exim process, reconfigure it, stop it again, check that all Exim processes have stopped, then start it. We have found in the past a simple restart can leave the old process in place, as well as launching new processes, which means your old (possibly broken) configuration might still be in force. The new instance might hang around in the background complaining that it can't bind to the SMTP port.

Managing the mail queue

Normally, the mail queue will be empty, so if you use **sudo mailq** to inspect the queue, you would normally expect the command to produce no output, in which case the queue is empty.

On the other hand, the queue could have data in it. The queue is simply the messages that have yet to be delivered. Having some messages in the queue on a production mail-server is to be expected. Some mail-servers might be too busy to process incoming messages, so they have to be held a while on the sending mail-server. These messages will be retried later. Here is some sample output showing two messages in the queue, including how long they have been in the queue, and the *message ID*. The **mailq** command would be run on Server1.

```
No output means the queue is empty.
# mailq
27m   450 1Au2JH-0000ex-Pw <root@localdomain>
          staff@localdomain

25m   443 1Au2Kz-0000fD-Kb <root@localdomain>
          staff@localdomain
```

We can force Exim to attempt a delivery attempt immediately, instead of waiting according to its normal retry timers. We do this using the message IDs.

```
# exim4 -M 1Au2JH-0000ex-Pw 1Au2Kz-0000fD-Kb
# mailq
No ouput, queue is now empty, everything worked.
```

Other messages, however, might end up as *** `frozen` ***, in which case the mail server can neither send the message, and neither can it send a bounce. This is often because of a misconfiguration. Frozen messages will remain in the queue awaiting manual intervention.

Diagnosing what is wrong with a message can be done by inspecting parts of the message. We could inspect the message headers, the message body (which we generally wouldn't want to do out of respect for the privacy of the parties involved), or the logs that are associated with the message delivery.

```
Display message logs
# exim4 -Mvl message-id
…
Message headers
# exim4 -Mvh message-id
…
Message body (last resort, consider privacy)
# exim4 -Mvb message-id
…
```

We can use the command **exiqgrep -zi** to output the message IDs of all frozen messages in the mail queue. We can feed this into various **exim4 -Mxx** commands, which operate on messages given a list of message IDs as arguments. Here is a list of examples:

```
Thaw all frozen messages and attempt delivery on them
# exiqgrep -zi | xargs exim4 -M
If they are still frozen, the problem likely isn't solved yet.

Remove all frozen messages (assuming you know they are all pointless)
# exiqgrep -zi | xargs exim4 -Mrm
```

14.5. Self-assessment

1. You must be able to send a message using **telnet** as a client, and read the message.

2. Make sure that you can inspect the message log.

14.6. Inspecting Headers

Ever wondered why an e-mail addressed to Joe Smith ended up in your Inbox? The envelope header isn't necessarily the same as the message header.

Consider the following message being sent from the e-mail address zeek@zebedee.com, to the members of a mailing list nohomers@hodum.com. The user bigtim99@ezisp.net is a member of this list. What is shown is a diagram of the communication between the SMTP client (in this case, the server responsible for the mailing list, which is postie.hodum.com), and the SMTP server (in this case, the server for the recipients mail-box, mailhub1.ezisp.net).

```
postie.hodum.com connects to mailhub1.ezisp.net
S→C 220 mailhub1.ezisp.net ESMTP Exim 4.44 Mon, ↵
    14 Mar 2005 16:43:12 +1200
C→S EHLO postie.hodum.com
S→C 250-mailhub1.ezisp.net Hello postie.hodum.com ↵
    [123.123.123.123]
...
C→S MAIL FROM:zeek@zebedee.com   Envelope header
S→C OK
C→S RCPT TO:bigtim99@ezisp.net   Envelope header
S→C OK
C→S DATA
S→C 354 Enter message, ending with "." on a line by itself
C→S From: zeek@zebedee.com       Message header
C→S To: nohomers@hodum.com       Message header
C→S Subject: Minutes of the last meeting...
...
```

The `To:` header in the message might say something different compared to the envelope (`RCPT TO:`). However, this can be normal and legitimate. In the case of mailing-list the `To:` header would be the mailing-list address, not the individual recipient this copy of message is being sent to.

When the message to a mailing-list (or in general, any alias) gets received by the mail server, the server will *explode* the message, so one message will go to each member of the list, with the envelope header set to that invidual's e-mail address, but the message header (`To:`) set to the mailing-list address.

It is very difficult to trust any headers in an e-mail message. As an administrator, it is imperative that you understand the concept of forging headers, as it is one of the common things that spammers and fraudsters do. Envelope (`MAIL FROM:`) and message headers (both `From:` and `To:`) are commonly forged to try and hide their tracks. Received headers are also routinely manipulated to make it harder for people to trace the spam back to where it came from. For this reason, be polite if complaining to the who you think might be responsible for the sending network, as it's not uncommon to be mistaken.

Using **telnet**, try to forge a message into Bob's account that says it's from someone else. Try to make it look like it really did come from that person. As bob on the mailserver (Server1), read the message, inspect the headers, and see if some of your peers can spot if it is genuine

or fake. Sites should have a monitored e-mail address abuse@domain, that e-mail messages about abuses such as spam or intrusion activity should be sent to.

To view all headers in Mutt, use **h** when viewing a message. All mailers have (or at least should have) the ability to view all headers.

Lets say that Bob received a message that was supposedly from his boss telling him to let in a technician into the computer room. Lets assume that Bob doesn't question this. The technician is really an intruder, using *social engineering* to try to get Bob to let him in, so he can steal business secrets, which he sells to their competition.

How can incidents like this, and others, be prevented? Come up with two suitable answers: one technical, and one social.

14.7. POP3 Server

Install the software used in this section on the mail server. **apt-get install dovecot-pop3d**

Setting up a POP3 server is very easy. We only need to install a suitable package, such as Dovecot. There are other alternatives, but we will not cover the topic much in this lab. Dovecot POP3d will run as a standalone service, although POP3 servers on small networks could run from inetd because they don't take long to start up. There are other things we could configure on the POP3 server, such as enabling the use of encryption for sending data over the network; but for now, we shall keep it very simple.

Ensure that `/etc/dovecot/dovecot.conf` contains only pop3 (we aren't going to setup POPS --- the SSL-wrapped POP3 as it adds more complexity that we need at the moment). Save and exit, then restart the dovecot service, using the techniques you should by now be familiar with.

Verify that something is listening for POP3 connections:

```
# lsof -i :pop3
COMMAND    PID    USER   FD   TYPE DEVICE SIZE NODE NAME
dovecot   5200    root    5u  IPv4  13788       TCP *:pop3 (LISTEN)
```

Hmmm, seems its only IPv4. Edit the configuration file again and search for IPv6. You should see a descriptive comment documenting the `listen` configurable. Set it as suggested to enable IPv4 and IPv6 access:

```
listen = *, ::
```

Exercise

Save and exit the configuration file, restart Dovecot (you should be able to figure out how to do this by now). As an exercise check, using **lsof**, that Dovecot's processes are IPv6 enabled. Find out what is listening on the POP3 port (TCP port 110) using the command below.

```
# lsof -Pni TCP:110
…
```

Beware that POP3 (as well as SMTP) is generally a clear text protocol, which means passwords can be captured fairly easily. Because this is a Bad Thing, the protocol has

been extended to use other authentication protocols instead of just plaintext username and password. A discussion of this is beyond this humble lab though.

The Dovecot server is able to get e-mail from many different places, and so we need to tell it where it can find the messages for each user (it can do auto-detection, but this can fail for some users). Search the Dovecot configuration files (in `conf.d`), and ensure the `mail_location` variable is set to:

```
mail_location = mbox:~/mail:INBOX=/var/mail/%u
```

Restart Dovecot.

POP3 servers often have a DNS alias that makes pop3 a valid hostname in the DNS. We've already done this earlier in this lab, so pop3.localdomain should resolve to be Server1's addresses:

```
$ host pop3.localdomain
pop3.localdomain has address 192.168.1.1
pop3.localdomain has IPv6 address fd6b:4104:35ce::1
```

Now we better check to see if our POP service works. Since POP is used for pulling mail off the mail server, send a couple of simple messages to your mailbox on the server (using either **mail**[4] or **mutt**), so you have something interesting to look at. Now lets speak POP to the server. Do this from Client1.

```
$ telnet pop3.localdomain 110
Trying fd6b:4104:35ce::1...
Connected to pop3.localdomain.
Escape character is '^]'.
+OK Dovecot ready.
USER bob
-ERR Plaintext authentication disallowed on non-secure (SSL/TLS) connections.
QUIT
+OK Logging out
Connection closed by foreign host.
```

Ah, Dovecot has disabled plain-text authentication by default, which is a good policy (secure by default: if we want to have to have a less secure system, we have to make it that way deliberately).

For our purposes though, we want to have plain-text authentication enabled, to show how it all works. Search the Dovecot configuration files again and uncomment the `disable_plaintext_auth` line and set it to `no` explicitly.

Now restart Dovecot, and try connecting again from the client.

```
$ telnet pop3.localdomain pop3
Trying fd6b:4104:35ce::1...
Connected to pop3.localdomain.
Escape character is '^]'.
+OK Dovecot ready.
USER bob
+OK
PASS bob's password
+OK Logged in.
LIST
+OK 3 messages:
```

[4] use **apt-get install mailx** if not available.

```
1 486
2 486
3 673
.
RETR 3
+OK 673 octets
Return-path: <mal@localdomain>
Envelope-to: bob@localdomain
Delivery-date: Thu, 06 Jan 2015 23:15:24 +1200
Received: from mal by localdomain with local (Exim 4.60)
        (envelope-from <mal@localdomain>)
        id 1HkIEW-0001N5-Pq
        for bob@localdomain; Sat, 05 May 2007 23:15:24 +1200
Date: Thu, 6 Jan 2015 23:15:24 +1200
To: bob@localdomain
Subject: Test from mutt
Message-ID: <20070505111524.GA5271@localdomain>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
User-Agent: Mutt/1.5.11
From: "Miss A. Laneous" <mal@localdomain>

Hi Bob, its me, your old friend muttley.

Goodbye.
>From mal

.
RETR 2
…
.
DELE 2
+OK Marked to be deleted.
DELE 3
+OK Marked to be deleted.
LIST
+OK 1 messages:
1 486
.
QUIT
+OK Logging out, messages deleted.
Connection closed by foreign host.
```

The command structure should be obvious, you should be able to guess what `USER` `PASS` and `LIST` do. `RETR` retrieves a message (using the number output by `LIST`), and `DELE` deletes a message from the server.

As with all clear-text protocols containing private data, it would be wise to use some sort of secure tunnel, such as SSL. Some people use **ssh** to set up a secure tunnel to the server. We'll cover the use of **ssh** for port forwarding in a later lab.

14.7.1. Self-assessment

1. Make sure your terminal session above showing that you have successfully spoken POP3.

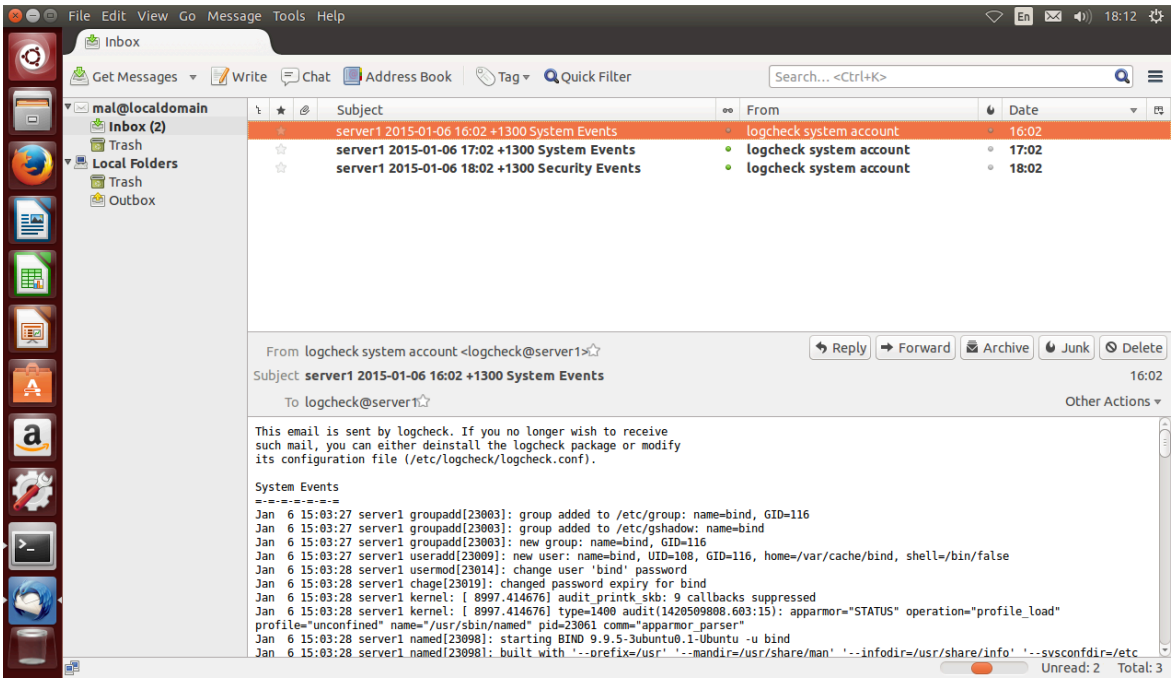
2. Do some research into IMAP and POP3, and create a short list of the biggest differences between them.

3. Configure a typical GUI e-mail program, such as Thunderbird (you can access it from the menu system), on Client1 to access the POP3 and SMTP services on Server1 (use the

DNS aliases, don't say server1.localdomain in the configuration settings). Test that you can send and receive messages. It should end up looking something like Figure 14.1, "Using Thunderbird to access Email via POP3 and SMTP":

Considering the options you saw when setting up the account, what further inprovements could be made?

Figure 14.1. Using Thunderbird to access Email via POP3 and SMTP



An account has been setup using Thunderbird to access the server using SMTP and POP3; a test message has been sent to mal@localdomain from the same account.

14.8. [Optional] A Simple Open Mailing List

> ## Running short on time?
>
> This section is optional, you needn't do it if you are running low on time, but by doing it you will get a much better appreciation of how e-mail is processed and routed.

Most mail servers host a handful of mailing lists as well, such as one for all the staff, or a large organisation might have lists set up for, say, all the system administrators. Naturally, a list can be anything you decide it to be. With Exim, setting up a mailing list is really quite easy, and involves just a few main steps. We shall be setting up a staff@localdomain mailing list.

Although most lists that you can easily subscribe to use a management engine such as Majordomo, or GNU Mailman, we shall not go into the use of such tools, rather we shall

have statically configured mailing lists, which means a human will need to update the list of subscribers.

The administrator of a list will often be reachable as `listname`-owner, with request for subscriptions often going to `listname`-request. Both of these should be aliases.

On the mail server, create a new file in `/etc/exim4/conf.d/router/950_local_mailing_lists`. Into the file, put the following text. You should think about why I gave it the number 950 at the beginning of the filename. To find out what some of these lines mean, look them up in the Exim Specification (consult the index).

```
lists:
  driver = redirect
  file = /etc/lists/$local_part
  forbid_pipe
  forbid_file
  errors_to = $local_part-owner@localdomain
  no_more
```

As you might have guessed, if mail comes in addressed to staff@localdomain, it will (assuming no user or alias called staff exists), redirect it to all the e-mail addresses listed in `/etc/lists/staff`. This is true for all the files in `/etc/lists/`.

You will need to add users to the list, so create the `/etc/lists/` directory, and add a file called `staff`, which should be a simple list (one local user or address per line), of the addresses that are part of this mailing list (just `mal` and `bob`).

You should also create aliases for staff-owner and staff-request in `/etc/aliases` if you haven't already. On many Sendmail-like systems, you need to use the **newaliases** command after editing the **aliases** file, although you don't need it for Exim. Restart **exim** and check it loads correctly.

On the client, log in as mal (they don't have to be part of the group however), and send a mail (using your program you set up in the last section) to the mailing list.

Make sure everyone in the list can receive the message to staff@localdomain, and that the aliases staff-owner@localdomain and staff-request@localdomain work as expected.

List-Id Header

A lot of power users tend to be subscribed to a lot of mailing lists. Some could subscribe to at least 20. You can imagine how chaotic one's life were if all this ended up in their inbox. To sort mail, people generally use some mail-sorting functionality in their e-mail client[5]. You can either look for the mailing list address (which may not always be the same) in the To or Cc headers. A lot of mailing lists[6] add headers to assist you, a fairly typical mailing list header is `List-Id` (there are a lot of other `List-*` headers as well).

Let's add a `List-Id` header to our mailing lists so our users can more easily sort their e-mail. If you're using mailing list software to manage your list, this will probably already be done for you. To do this, we need to add a simple entry in the e-mail router we added before. The entry is as follows. Note that another common header is `X-List-Id`. You should remove any headers of the same type, unless there are supposed to be multiple entries (such as `Received` headers). Here is the what the entry should now contain. The bold lines are new.

[5] I don't want to say MUA here, because most clients are more than just a MUA these days, as they also send mail (speak SMTP), provide advanced message filtering and may also deal with groupware functionality.
[6] The GNU Mailman software is well-known for this.

```
lists:
  driver = redirect
  file = /etc/lists/$local_part
  forbid_pipe
  forbid_file
  errors_to = $local_part-owner@localdomain
  headers_remove = List-Id
  headers_add = List-Id: $local_part@localdomain
  no_more
```

Exercise

As an exercise restart Exim, and send a message to the list. Make sure you get the header added. You can then view the full headers of the message.

Reply-To Munging

Let's say that you are subscribed to a rather large mailing list. Let's say you post a question (which could be a reply to someone elses message). You are listed in the From header for that message. People could respond in one of two ways: Reply, or Reply All.

In the case of Reply, the message would go only to yourself, not the mailing list. This can annoy a lot of people, because it means that others can participate in the discussion (and locks you into the discussion).

On the other had, if Reply All was used, the reply would go to yourself, and the mailing list, and you would end up with two copies of the message. This also annoys a lot of people.

The `Reply-To` header is a header that an e-mail client application may use to decide who to send a response to when replying (or using reply-all), instead of using whichever address is listed in the `From` header.

Many mailing lists have an option to automatically set the Reply-To on all messages, such that all messages go back to the mailing list. This is a controversial thing to do, with some people saying its a good idea [http://www.metasystema.org/essays/reply-to-useful.mhtml] and others insisting that it's a bad idea [http://www.unicom.com/pw/reply-to-harmful.html]. It has its place in some lists, depending largely on the user population.

We can implement `Reply-To` munging the same way we did the `List-Id` header, with the `header-add` parameter in the message router. This example builds on the `List-Id` example above.

```
lists:
  driver = redirect
  file = /etc/lists/$local_part
  forbid_pipe
  forbid_file
  errors_to = $local_part-owner@localdomain
  headers_remove = List-Id:Reply-To
  headers_add = List-Id: $local_part@localdomain\n\
        Reply-To: $local_part@localdomain
  no_more
```

Restart Exim, and send another test message. Check that `Reply-To` has been set. Reply to the message: you should either be asked if you want to reply to staff@localdomain (ie. honour Reply-To) or not. Read the webpages listed in the footnotes below about the sensibilities of `Reply-To` munging.

14.9. Last Words

There is so much more we could go into on the subject of e-mail. For instance, there are the rather significant topics of spam and viruses, cryptographic issues, and virtual hosting. All of this would be good to teach you, but we just don’t have the time in one semester. However, you should know that the practical aspects of these topics is similar to the processes we have already used. Here are some topics and tools the eager student may want to read.

RFC1855: Netiquette Guidelines [http://ftp.rfc-editor.org/in-notes/rfc1855.txt]
How to maintain suitable behaviour on a network.

Procmail
A very powerful mail filtering tool.

Spam Assassin
A very useful spam filter.

The case of the 500-mile email [http://www.ibiblio.org/harris/500milemail.html]
A funny anecdote, worth reading.

Lab 15 Catchup Lab

This lab is set aside for help people catch up; priority will be given to students doing previous labs and assignments.

Lab 16 World Wide Web

Today we will configure the most common web server on the planet: Apache. This lab will in no way give you enough experience to manage an Internet-facing webserver with dynamic content, and we don't have enough time to cover related issues such as web development. Indeed, there are multiple papers dedicated to precisely that. This is the first step down a very long road; an introduction to one aspect of a potent career path.

16.1. DNS Alterations

Web servers are commonly addressed using any of two common notations, either http://www.domain/ or http://domain/. We've configured this in the DNS lab already, but then our web server (which didn't actually exist) was known to DNS as goliah.

Change the DNS so that both `www.localdomain.` and `localdomain.` resolve to our server's addresses for both A and AAAA records[1]. You should *not* add an entry to the reverse zone, as we are defining aliases, and only canonical names go into the reverse zone. Remove the old entry for the non-existant Goliah. Use the following to test what you have done.

The `localdomain.` case is just a little bit tricky; remember that `@` is shorthand for the `$ORIGIN`, which in this particular file (as defined in `named.conf*`), is `localdomain.`, so either use a `@` or leading whitespace, just as we did when specifying the nameservers (`NS` record) for our zone.

```
Move to Server
# rndc reload
Check your logs to ensure it worked, then begin testing
$ dig -t A +short www.localdomain
192.168.1.1
$ dig -t AAAA +short www.localdomain
fd6b:4104:35ce::1
$ dig +short -x 192.168.1.1
server1.localdomain.
$ dig +short -x fd6b:4104:35ce::1
server1.localdomain.
$ dig -t A +short localdomain
192.168.1.1
$ dig -t AAAA +short localdomain
fd6b:4104:35ce::1
```

16.2. Install and Configure Apache

On the server, you will need to install the packages for Apache and PHP for this lab:

```
# apt-get install apache2 php links curl libapache2-mod-php
…
```

Configuring Apache includes giving it a few details on how it should run, and enabling the use of PHP. To make it easier to add drop-in functionality (via packages) to the web-server without requiring an administrator to manually edit configuration files, Debian based systems modularise the Apache configuration directory to a greater extent than other distributions.

[1] Alternatively, we could have made it so www has a CNAME of server1, but having duplicate A and AAAA records means we have greater flexibility, either to add other servers in round-robin DNS or to disable IPv6 by removing the AAAA record when accessing the web service.

Read all of `/usr/share/doc/apache2/README.Debian.gz` before you proceed, so you learn about how Apache's configuration is managed under a Debian-based distribution. Use **zless** to read the file.

Now, here are the configuration elements we want you to insert into the appropriate places ("appropriate" locations as per the `README.Debian.gz` file). In the instructions below, we tell you where to insert the elements. However, in case you can't find them, there is a general approach using **grep** to find them (see detail below).

1. Insert `ServerName server1.localdomain` into `/etc/apache2/apache2.conf` under the section of *global configurations*.

   This tells the server the canonical name of the host. It will be seen in error messages such as `404 File Not Found` responses when a requested web-page does not exist. It will try to automatically determine the value of this by looking up DNS, but you may find startup more reliable to specify it here, in the event that DNS is temporarily unavailable.

2. A web-server can often have multiple "virtual" sites, each site having a different name. The web-server offers different pages depending on which name it was called by. The default site is the site that is used if the name is not recognised as the name of a virtual site. The site configuration for the "default" site can be found in the file `/etc/apache2/sites-available/000-default`.

   Change `ServerAdmin` to `webmaster@localdomain` for the default site. This is a standard address that your email server should accept and should be redirected to a real person; we'll do that in a later lab.

3. It can be challenging to find where particular configuration elements are set because of the split-out management of the various files. The splitting of configuration elements helps the package managament system to manage Apache's configuration — facilitating drop-in configuration — but does make it harder to browse the configuration. To help with this, we can use **grep -r** to search recursively under `/etc/apache2/` which will tell us where we can find information.

   What is the value of `DirectoryIndex`?

```
$ grep -r DirectoryIndex /etc/apache2/
```

   Note that files under `/etc/apache2/mods-enabled/` are actually symbolic links to the real files under `/etc/apache2/mods-available/`. The same goes for the `sites-enabled` and `sites-available`.

   Assuming the value of `DirectoryIndex` were `index.php index.html`, when the server is asked for a directory (eg. `http://www.cs.otago.ac.nz/cosc301/`), this would cause the server to *first* look for `index.php` *then*, if `index.php` were not found, try `index.html` when it has been asked for a directory without a filename. There are often multiple types of files that might be tried, only the first found file is used. Thus, order matters.

4. Using the appropriate tool (ie. mentioned in `README.Debian.gz`), enable the `php7.0` module if it is not enabled already. You should not use a tool such as **ln** to do this manually, but use the tool designed for the job. You will find an example of using the tool in the section of *Virtual Hosts*.

5. Using the appropriate tool, disable the `cgi` module if it is not already disabled. What does this module do? Where can you find authoritative documentation about this module?

6. Using the appropriate tool, enable the `auth_digest` module, which we shall use later for authorisation. What command did you use?

Now check the syntax, restart the server and ensure that its listening.

```
# apache2ctl configtest
Syntax OK
# apache2ctl graceful
```

Check your logs in `/var/log/apache2/error.log` to make sure it started without complaint.

We better check out what it is listening for on the network, as we also want to have IPv6 connectivity as well.

```
# lsof -Pni
COMMAND … USER     … TYPE … NODE NAME
…
apache2 … root     … IPv6 … TCP  *:80 (LISTEN)   one of these
apache2 … www-data … IPv6 … TCP  *:80 (LISTEN)   and five of these
…
```

What do you notice about this? There are a few things we want you to pay particular attention to here. The first is that the server generally doesn't run as root, except at the beginning (the single entry is for starting other helper processes, it needs root permissions to bind to port 80). The helper processes, that actually serve the content, run not as the user "root", but instead run as the user "www-data". Why is that? (You'll need to write this down for the assessment).

The second major thing to notice is that all of **apache2**'s sockets are IPv6 sockets, there are no IPv4 sockets. You may be wondering why that is; surely we want to be able to reach our server over IPv4. This is a common feature of "dual-stack" servers ("dual-stack" means that a server runs both IPv4 and IPv6). The basic rule is this: if something goes to a port (say, port 80) on IPv4, but there is no IPv4 socket on that port, but there is an IPv6 socket on that port, then the connection goes passed to the IPv6 socket. It is important to realise that it is still IPv4 across the network.

You may think that's a bit odd. What's an IPv6 server going to do with an IPv4 connection? Well, the kernel translates it. The IPv6 server sees an IPv6 connection from the address `::ffff:192.168.1.113`. This special type of address notation, with both colons and dots, is called an "IPv4-mapped IPv6 address". The IPv4 address, in decimal is embedded in the lowest 32-bits of the otherwise hexadecimal address. This is done to make this sort of address instantly recognisable; this example would be otherwise written as `::ffff:c0a8:171`.

Some systems turn off IPv4-mapped IPv6 addresses by default

> Applications can explicitly enable or disable this feature, and the default is generally to have it enabled, although some systems will default to having this turned off. Debian, for example, leaves the Linux kernel default alone (ie. enabled), while Redhat turns it off via a sysctl.

This dual-stack transitioning support makes it easier to create network services because you only have to care about IPv6, and not necessarily IPv4 (although at some point you generally *do* need to care, but mostly when dealing with the addresses themselves).

On the client, try to connect to the web server using a web browser.

Note

We've sort of shot ourselves in the foot by only having one component in the domain name (`localdomain`). This is because, if we want to go to http://localdomain/, the browser sees that there is only one domain component, and so believes that it must surely be a non-fully-qualified domain name, and so the resolver on the client adds `localdomain` to the name. Thus, it looks up `localdomain.localdomain`, which fails. In the case of many web-browsers, it may then assume you really meant something like `www.localdomain.com` and then tries that.

To work around this, we can ask it to lookup `localdomain.` (note the trailing dot), which says the name is arleady fully qualified. Not all tools will accept that though.

You should see *Apache2 Ubuntu Default Page* shown in the browser. If so, that indicates that you can reach your web server. Let's do a few more tests, but this time, we'll watch the access logs in real-time as we do that. On the server, **tail -f /var/log/apache2/access.log**, and connect to the following with your browser on the client:

- http://www.localdomain/
- http://192.168.1.1/
- http://192.168.1.1:80/
- http://[fd6b:4104:35ce::1]:80/
- http://[fe80::a00:27ff:fe50:e093]/

  As expected, this may not work with Firefox (or any browser), as it would require a scope identifier, which is not allowed in URI syntax. Remember that Link-local addresses should not generally be used by applications, but as an administrator, you should not ignore them.

- http://server1.localdomain/
- http://server1.ipv4.localdomain/
- http://server1.ipv6.localdomain/

Okay, so we've verified basic operation and reachability. Let's go onto making some content and test the PHP component of our server.

16.3. Write Some Content

We won't assume that you can write HTML, or know any PHP, so copy the following example, just to show that PHP pages are indeed being processed.

Write this into `index.php` (note the `.php` suffix) in your document root (`/var/www/html`). Remove or rename the existing `index.html`.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Bob's Construction Company</title>
```

```
</head>

<body>

<h1>Can we fix it?</h1>

<p><?php echo "Yes we can!"; ?></p>

</body>
</html>
```

Exercise

As an exercise view the site again. You need to click the reload button to force the refresh of the page. You should get a page titled “Bob’s Construction Company”. If you get a paragraph that says “Yes we can!”, then that means PHP is working. You may like to head on ever to www.php.net [http://www.php.net/] and follow a tutorial later.

16.4. Virtual Hosts

Virtual hosting is a common practice amongst web servers. In essence, it allows one server to host many different web-sites. The server must know which site the client is visiting. There are two different mechanisms for this: the first requires each site to have a different IP address (IP-based virtual hosts), and the server will be assigned many different IP addresses. This is a waste of valuable address space, and there is little reason for using this in todays environments, unless you are using HTTPS.

It is too early to see whether it will make much of a comeback with IPv6. In IPv6, each virtual site could easily have its own address (or even a whole bunch of them).

The way we do virtual hosting these days is to use name-based virtual hosts. This means the client sends along a `Host` header, which specifies the name of the server it believes it is talking to (it gets this from the hostname portion of the `URL`.

The biggest disadvantage of name-based virtual hosting is that it isn’t widely available yet for encrypted web-traffic. It’s only now becoming widely spread in modern browsers[2], so it will take a while for all the older devices to be replaced. This is because the SSL certificate is locked to the fully-qualified host-name the client is connecting to. The SSL session is established before HTTP is spoken, thus no `Host` header will have been sent and the server won’t know which certificate it should use.

We’re about to use **telnet** to interact with the web server using HTTP. However, by default, the version of Apache that ships with Ubuntu has a module enabled which imposes a limit on how long it will wait until it receives commands. This can be annoying for us slow humans, so we shall disable it.

```
# a2dismod reqtimeout
…
# /etc/init.d/apache2 restart
…
```

The bare minimum for a HTTP/1.1 request looks like the following. Try this using `telnet` to your web-server. Here we are talking the HTTP version 1.1 protocol. The 1.0 protocol is pretty much the same, but the `Host` header is not required.

[2] It is an extension to TLS called Server Name Identification. The Wikipedia has a good writeup about client and server coverage.

```
$ telnet localdomain. 80
Trying fd6b:4104:35ce::1...
Connected to localdomain.
Escape character is '^]'.
Request type, request path & arguments, HTTP version.
GET / HTTP/1.1
This let's the server know which virtual host to use.
Host: localdomain
Blank line to signify end-of-headers.

HTTP/1.1 200 OK
Date: …
Server: …
…
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Bob's Construction Company</title>
</head>
…
The server will keep the connection open briefly…
… this is to allow the client to make another request.
Connection closed by foreign host.
```

Let's review what's going on here.

`telnet localdomain. 80`
: We connect to port 80 on the web server.

`GET / HTTP/1.1`
: We use the `GET` command to retrieve the document corresponding to `/`, which will be `index.php` in our case. We specify a version of the HTTP protocol, specifying 1.1.

`Host: localdomain`
: We send a `Host` header, which is required for HTTP/1.1. This is what the server can use to determine the *site* being visited, as the client puts the name of the server it thinks it is trying to connect to here.

Blank line
: In HTTP a blank line signifies that we are done sending headers, and it is the servers turn to send data.

`HTTP/1.1 200 OK` … blank line
: The server sends us back a set of headers, followed by a document. The `200 OK` part tells the client software the request was processed without any problems. One other well-known return code is 404: page not found.

Timeout at end
: This server is doing pipelining, which means we can make multiple requests per connection, making the transfer much faster. The server will keep the connection open to allow the client to send through another request, preventing another lengthy TCP connection setup and teardown[3]. After a while with no input, it will close the connection.

Let's create a virtual host for the new domain `boogaloo.nz`. For a public internet, you would need to purchase the domain name first, and have somewhere to house the server, or buy some web-hosting services. We won't be doing any of that in our private networks.

[3] Not to mention the TCP Slowstart algorithm for flow control.

1. Create a new domain called `boogaloo.nz` by adding a forward zone to `named.conf.local` on the DNS server, and copying the `db.localdomain` file to `db.boogaloo`, modifying it to suit the following requirements:

   - We want `http://boogaloo.nz` and `http://www.boogaloo.nz` to be valid points of access to the web server, which should point to our server's IPv4 and IPv6 addresses.

   - We don't need to touch the reverse zones. They will remain pointing to entries in localdomain. This is because all the data in `boogaloo.nz` is alias data, and as such doesn't need to go into the reverse zone[4].

2. Test the new DNS entries.

```
# /etc/init.d/bind9 restart
Starting BIND:  /usr/sbin/named
Check logs, ensure it is still running...
$ dig -t A +short boogaloo.nz
192.168.1.1
$ dig -t AAAA +short boogaloo.nz
fd6b:4104:35ce::1
$ dig -t A +short www.boogaloo.nz
192.168.1.1
$ dig -t AAAA +short www.boogaloo.nz
fd6b:4104:35ce::1
```

3. Create a file `/etc/apache2/sites-available/boogaloo.conf`. Inside it, write the following.

```
<VirtualHost *:80>
    ServerName boogaloo.nz
    ServerAlias www.boogaloo.nz
    ServerAdmin webmaster@boogaloo.nz
    DocumentRoot /var/www/boogaloo
    ErrorLog /var/log/apache2/boogaloo-error.log
    CustomLog  /var/log/apache2/boogaloo-access.log combined
</VirtualHost>
```

4. Create the document root for the virtual host.

```
# mkdir /var/www/boogaloo
```

5. Enable the site in the Apache configuration.

```
# a2ensite boogaloo
```

6. Test the configuration file syntax, and restart the server. If it fails to start, you should check the error log for Apache which can be found in `/var/log/apache2/error.log`.

```
# apache2ctl configtest
Syntax OK
# apache2ctl graceful
# tail /var/log/apache2/error.log
```

7. Create a simple `index.html` file in the docroot for boogaloo.nz.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

[4] While it is technically possible to have multiple PTR records, no software will expect that and you would be likely to get a round-robin result, which is not useful.

```
<html>
<head><title>Boogaloo</title></head>
<body>
  <h1>Funky!</h1>
</body>
</html>
```

8. **Exercise**
   As an exercise point your web-browser to your new site, to both points of access in your new domain, as well as ensuring that your old domain still works as expected.

16.5. Self-assessment

1. Why does the web server run as the user www-data? What rights does that user have to the web page content?

2. In addition, do some research and find some best practices for securing Apache. Write down at least five of them. Make sure you understand what they do.

16.6. Last Words

Web services, and especially applications that run on top of them, are some of the most often attacked services. **Do not** deploy them on the Internet without being very familiar with how they work and the real world issues. Consulting the Apache Documentation [http://httpd.apache.org/docs/] is a great place to start becoming more familiar with the Apache webserver. Start reading some of the material at www.sans.org and all web-developers should at least be familiar with the OWASP Top 10 [http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project] web vulnerabilities.

16.7. [Optional] Authentication and Authorisation

> **Running short on time?**
>
> The remainder of this lab is optional, so if you don't have enough time, you can just stop right here.

Often, you want to limit access to material on your web-server using machanisms such as IP address ranges, or username and password. For this section, we'll have a quick look at username authentication, using the Digest authentication mechanism we enabled earlier.

There are two authentication mechanisms in common use, Basic and Digest. Basic doesn't encrypt the password, it just encodes the username and password values in Base64, which is easily decoded by anyone. Digest authentication makes a MD5 digest (hash) of the username, password and other things that the server decides. The full details are available in RFC 2617[5].

You have to remember however, that the data is still transmitted in the clear. If you need to have strong security, you should be using SSL/TLS. Check the Apache website for more information, particularly the Authentication, Authorisation and Access Control document.

[5] http://ftp.ics.uci.edu/pub/ietf/http/rfc2617.txt

We'll proceed by adding a private directory to the `boogaloo.nz` domain we added in the previous section. We'll then configure Apache to give access only based on user-name and password, using Digest authentication.

1. Create a password file `/etc/apache2/boogaloo-digest` with a username and password for yourself. This can be done using the following command. Notice that the password file is outside the document root of the webserver. You should briefly check the manual page for this command to see what the arguments and options are.

```
# htdigest -c \
> /etc/apache2/boogaloo-digest \
> "Boogaloo Employees Only" theauthor
Adding password for theauthor in realm Boogaloo Employees Only.
New password: Not echoed
Re-type new password:
```

2. Have a look at the generated password file to see what is inside.

3. Create a directory where we will put our protected material.

```
# mkdir /var/www/boogaloo/private/
```

5. In that directory, create a file called `.htaccess`, and place in it the following contents. The `AuthName` must match the realm you used above. The `Require` directive states that a username and correct password is required. Note that we can either specify that we require any valid user, or specify that only certain users may enter.

   On the server we have configured, `.htaccess` files are used by default. These are files that change Apache's configuration for particular directories, often to limit access. What such a file is allowed to reconfigure is listed in the main server configuration.

```
AuthType Digest
AuthName "Boogaloo Employees Only"
AuthUserFile /etc/apache2/boogaloo-digest
Require valid-user
Alternatively, specify a set of users
# Require user asuka sanjay jamilah michael
We could also set up groups of people
```

6. In order to allow use of `.htaccess` files to configure authentication, you need to add some lines to the `VirtualHost` stanza in `sites-available/boogaloo`.

```
<VirtualHost *>
    ServerName boogaloo.nz
    ServerAlias www.boogaloo.nz
    ServerAdmin webmaster@boogaloo.nz
    DocumentRoot /var/www/boogaloo
    ErrorLog /var/log/apache2/boogaloo-error.log
    CustomLog /var/log/apache2/boogaloo-access.log common
    <Directory /var/www/boogaloo>
        AllowOverride AuthConfig Limit
    </Directory>
</VirtualHost>
```

   It is important to note that normally we would prefer to put things under `/etc/apache2/` files rather than in `.htaccess` files. Here, I have used `.htaccess` files to demonstrate the concept. They can be useful in certain situations.

7. Optionally, you can also create groups by specifying a group file to use, which means you can limit access to a group of users. These are not system groups however, but rather

specified in a seperate file, in much the same way the users and their encrypted passwords are.

8. Create a file `index.html` in the private directory, and point your browser to the private area of the Boogaloo website.

   Make sure you get the output above, and not an HTML error page.

Lab 17 Catchup Lab

This lab is set aside for help people catch up; priority will be given to students doing previous labs and assignments.

Part III. Laboratories on Network-based Services

The Network-Network Interface

Lab 18 Internal Routing

> **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 workstations, you should find that the Live CD images, as `*.iso` files, for Vyatta and Ubuntu should be available from the class resource server, 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.

18.1. 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.

18.2. 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.

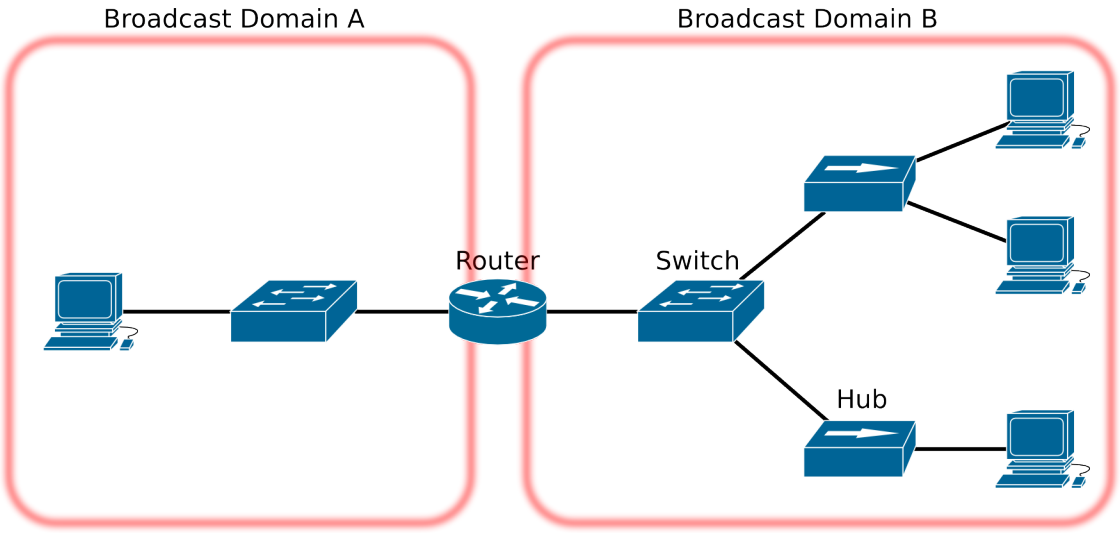
18.2.1. Broadcast Domains

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 18.1. 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.

18.2.2. A Virtual LAN

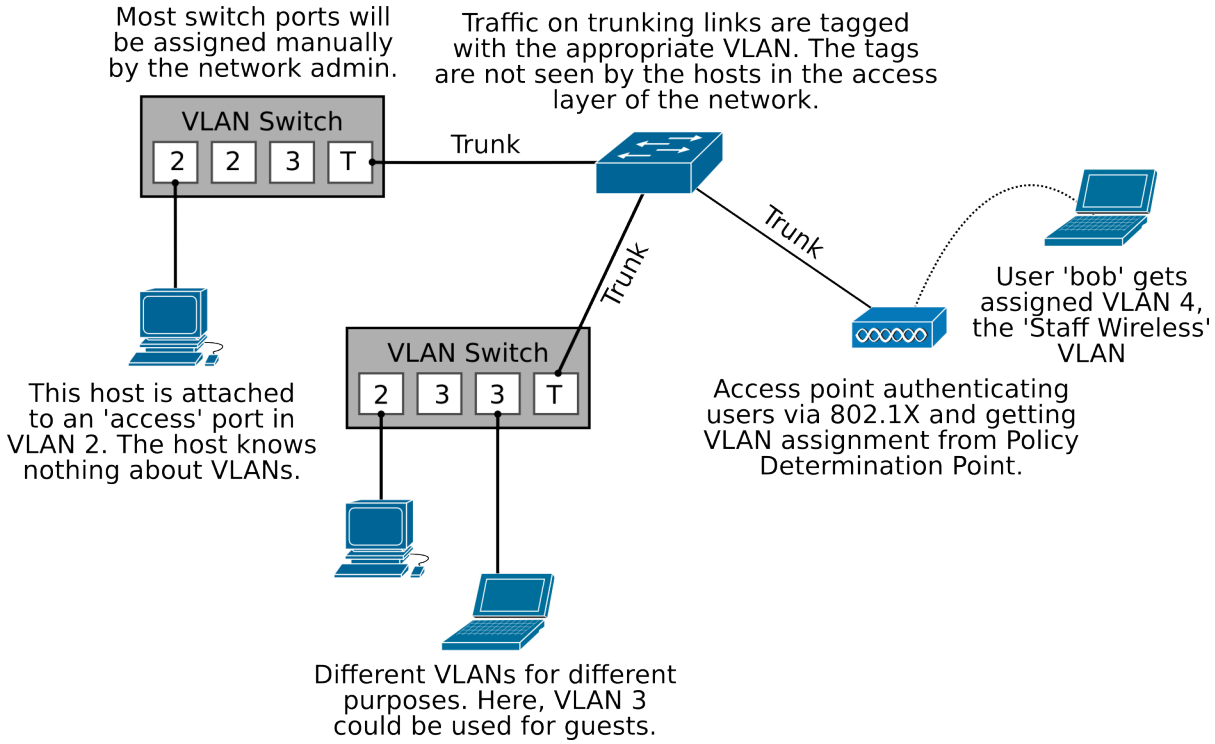
A Virtual LAN (VLAN) is, quite simply, the ability to segregate a switch into seperate broadcast-domains. This means that in order to get between the different VLANs, a router must be used. In the older days, when VLANs were still new, a *one-armed router* was used, which had an interface on both VLANs; 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 18.2, “VLAN Assignment and Trunking” should make this clear.

Figure 18.2. VLAN Assignment and Trunking



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).

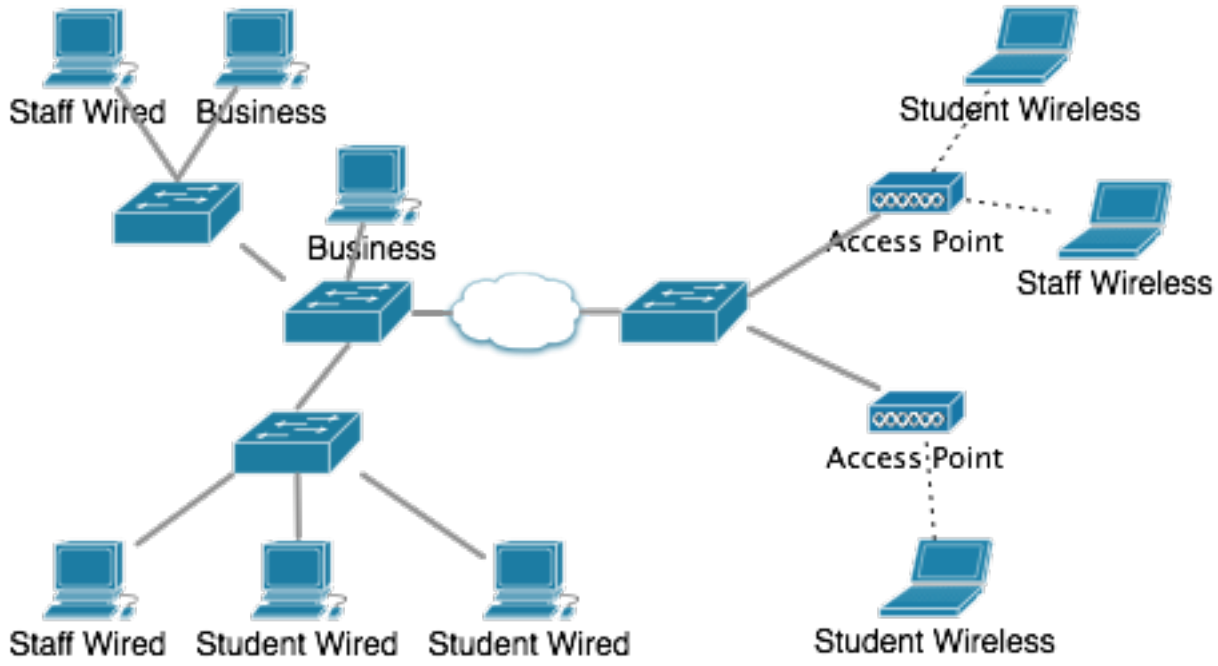
Can you imagine how much more complex the network would have to be if we *didn’t* have VLANs? We would loose a lot of flexibility, and cost would be very much higher. We would at least need many more switches and access-points, routers and cable. Running extra cable would be the most expensive part.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

18.2.3. 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 18.3. Using Virtual LANs to divide a network.



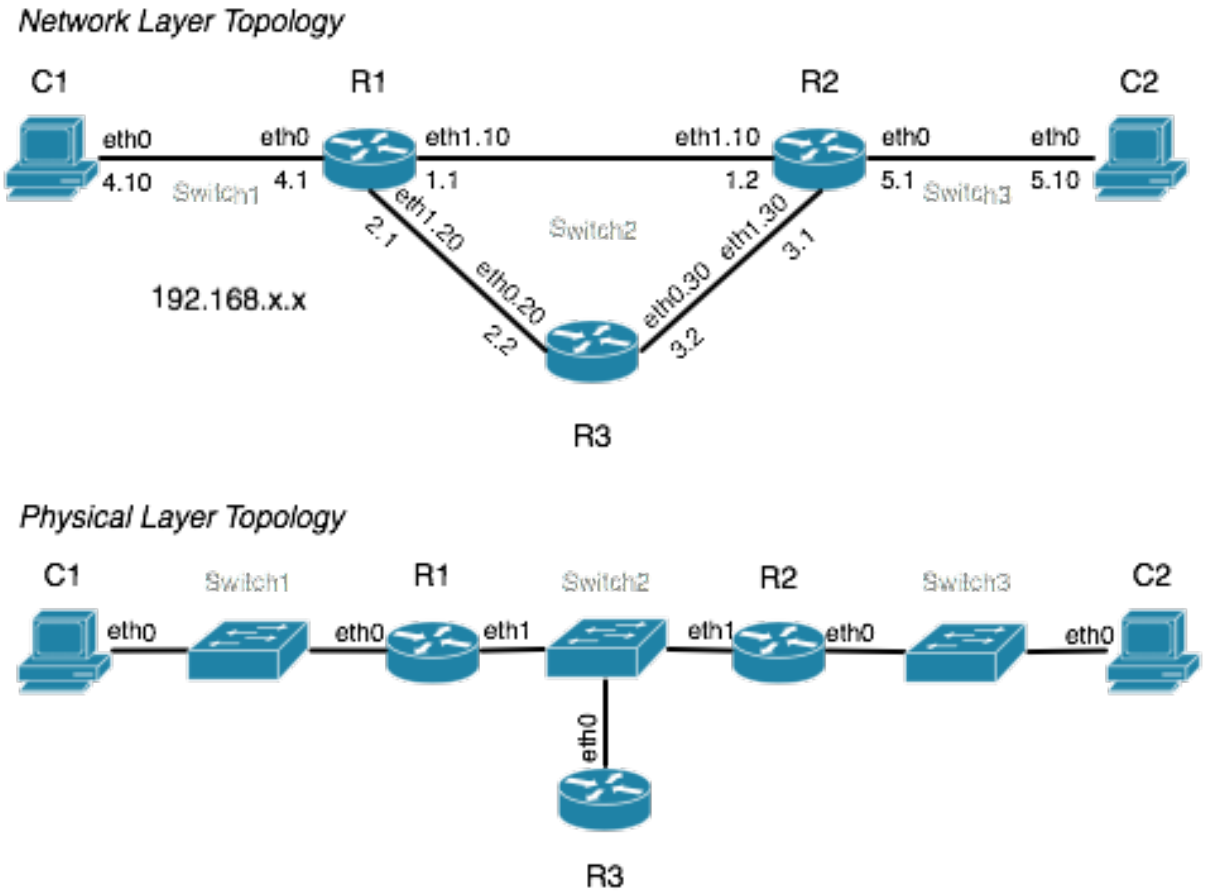
Using Virtual LAN technology gives us much greater flexibility and simplicity in how we design and implement switched networks.

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.

18.3. Configure VirtualBox with the Topology

Figure 18.4. 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 18.4, “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.

Vyatta requires that the virtualised system has a CPU feature called Physical Address Extentions, 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.

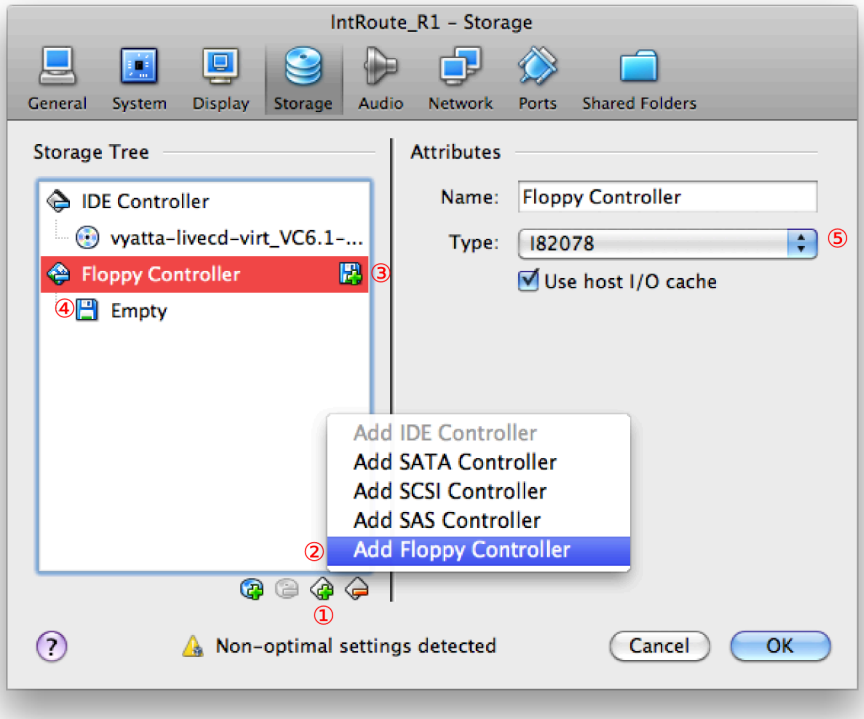
For each router we will need to create a virtual floppy disk (another image file) to store our router configuration. To do this open a terminal window on your Mac computer and type the following command to create a 1.44 MB (actually 2 MB, but close enough) file full of empty space. You could put these inside your `myvms` directory.

```
$ cd ~/Desktop/myvms/
$ 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 (third button from the left under the Storage Tree) and selecting Add Floppy Controller. Then, on the new floppy controller, click the + icon. Click the Empty floppy and add the floppy you made, in much the same way as you did with the ISO file. This sequence is shown in Figure 18.5, “Adding a Floppy Controller and Disk to VirtualBox”. Remember that each machine will have a *different* floppy.

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

Figure 18.5. Adding a Floppy Controller and Disk to VirtualBox



The sequence of steps for adding a floppy controller to a Virtualbox virtual machine, and assigning a floppy disk image to it. Note that you will only see the attributes shown after you click on the Empty floppy disk.

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) and also disable Enable absolute pointing device under the System tab; that option causes VirtualBox to use a virtual 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

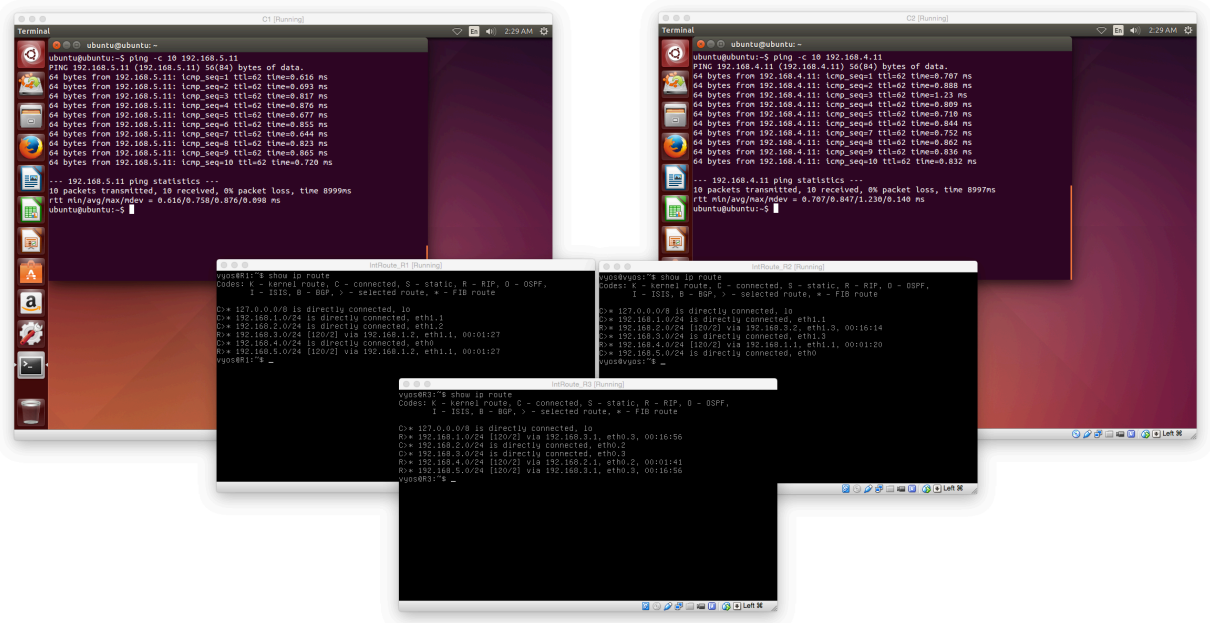
Now go on and configure all of the other routers and hosts. Remember that for the hosts C1 and C2 no floppy is needed and they will be booting from an Ubuntu Desktop Live CD. Also, don't forget to put the CDROM at the top of the boot list, or VirtualBox may look at other existing drives first, which 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.



18.4. 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.

18.5. 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 18.4, “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 eth0 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.

- 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

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

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.

18.6. 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.

18.7. 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?

18.8. [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.

18.9. [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.

18.10. [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].

18.10.1. 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.

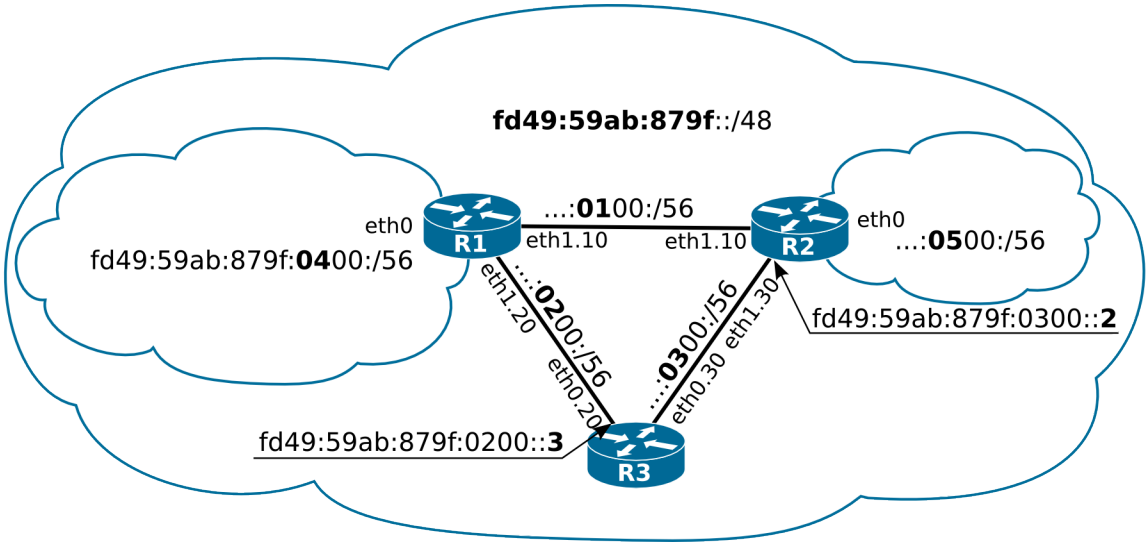
So, looking at the network map, give each of the five subnets a number; because we've already got a nicely organised IPv4 network running in parallel, we shall save some confusion and use the same subnet IDs. Allocate a /56 to each subnet in your network. You might end up with something like the following. This is also shown on the map in Figure 18.6, "IPv6 Subnetting and Addressing".

```
Network allocation:   fd49:59ab:879f::/48
   subnet 1:                        …:01|00::/56
```

[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.

```
subnet 2:                       …:02|00::/56
subnet 3:                       …:03|00::/56
subnet 4:                       …:04|00::/56
subnet 5:                       …:05|00::/56
                                    ↑ /56 boundary
```

Figure 18.6. IPv6 Subnetting and Addressing



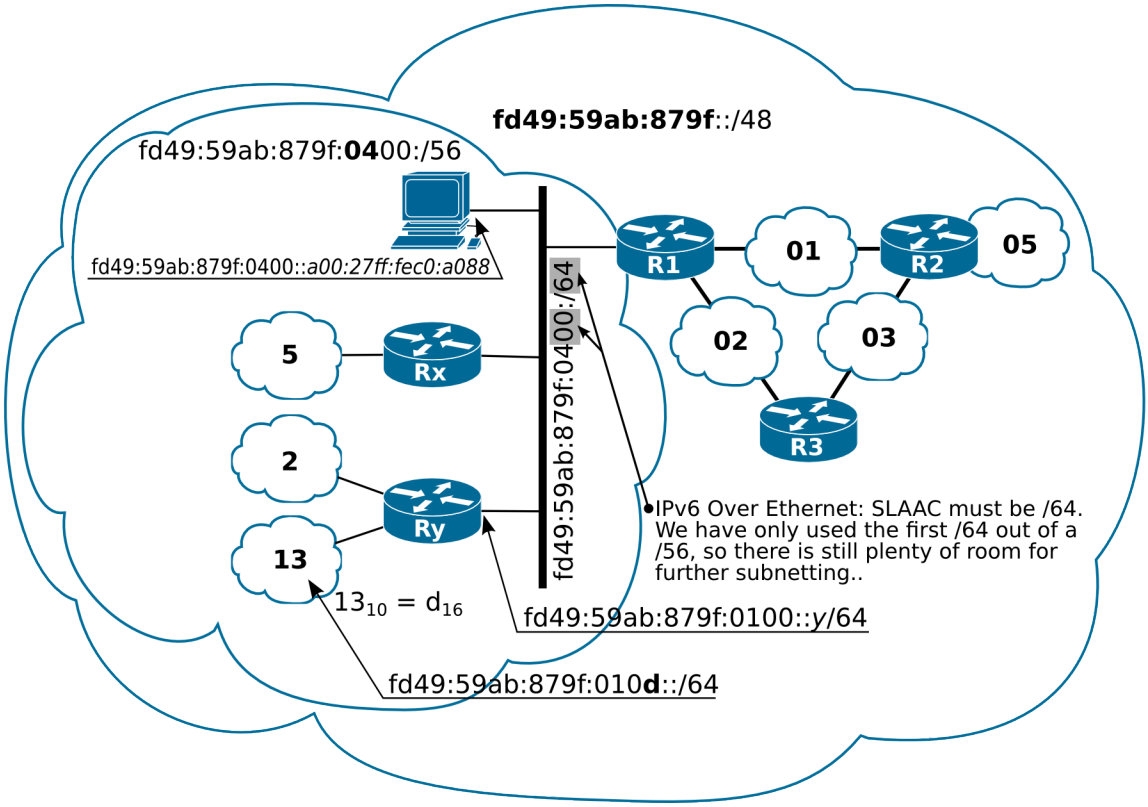
The subnetting and addressing plan for our network.

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 18.7, "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 18.7. IPv6 Subnetting Allowing for Growth



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 18.6, "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
```

18.10.2. 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.

18.10.3. 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**.

Lab 19 Virtual Private Network (VPN)

Over recent years the nature of the internet has changed. In the early days there was an assumption that all the traffic was trusted. This is definitely not the case these days. There is a move towards encrypting traffic, whether it is to/from various services (the 's' variants of the protocols, for example https, smtps, etc.). In this lab we explore configuring a VPN. VPNs are used to grant access to internal resources to a client connecting over an insecure network.

You have seen the term VLAN before now. VLANs are used to segment portions of the network into logical (or broadcast) domains. For example, within a company, there may be a logical network for the engineering department that is separate from the marketing department. Yet they both share the same network infrastructure (i.e. the switches and routers of the organisation).

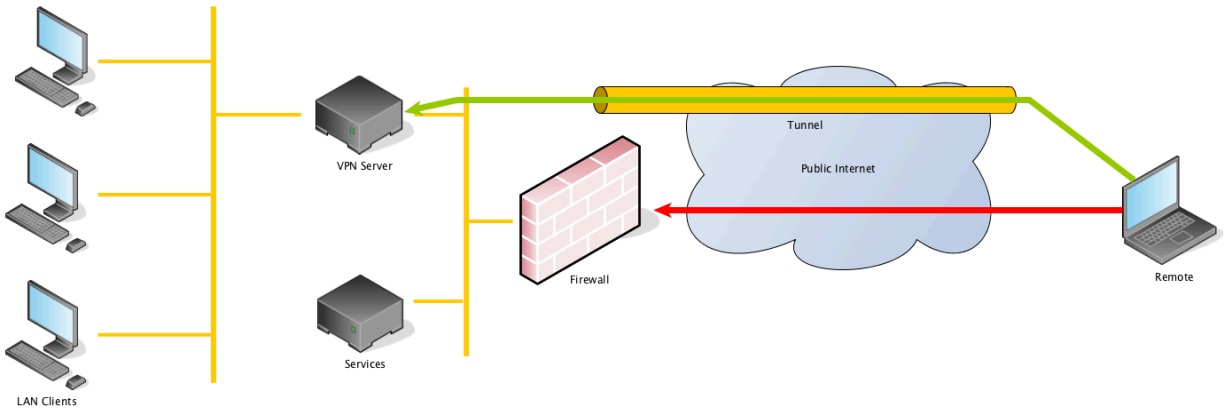
VPNs on the other hand are generally used to link two (or more) networks[1] together over an insecure channel. For example, a remote worker might need access to some internal resource (like a shared file server) from their home over the public internet.

To understand the difference, let's have a brief look at what happens to the packets as they travel across the VPN. Normally, the client addresses the packet to the destination and routers along the way will pass the packet closer to the destination. In a VPN, this packet is packaged inside another that's directed to the VPN server. You can think of it as if you take a letter and envelope, addressed to Bob, and place it inside another one addressed to Carol. You trust Carol to unwrap her envelop and then pass the contained letter to Bob. Note the contained packet (or letter) is encrypted by VPN.

In this lab we will simulate a remote worker using OpenVPN to connect to Server1 (their trusted home server) so that they can ping to Client1 using it's internal address (i.e. 192.168.1.11). OpenVPN uses port number 1194. The firewall would have to open that port in order for Server1 to provide the VPN service. If you would like more information about OpenVPN, have a look at the OpenVPN Home Page [https://openvpn.net], specifically, the Community Pages [https://openvpn.net/index.php/open-source.html][2].

[1] We can connect both whole networks (at the router level) as well as individual client computers.
[2] As is common trying to earn money from open source, they provide a preconfigured services and support for a fee.

Figure 19.1. Remote Client VPN setup

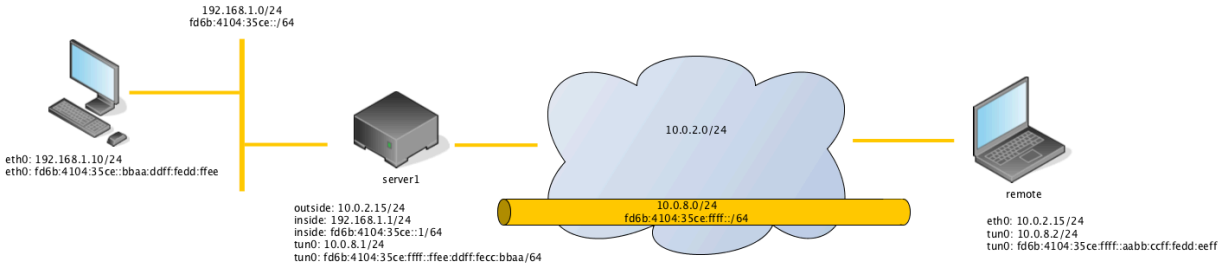


The above figure shows how the VPN works when used across the internet. Access to services have been blocked by firewall rules and the remote client, therefore, cannot access them. By tunneling the network traffic across an encrypted connection the remote client is able to access the services as if they were on one of the LAN Clients.

19.1. Configure VirtualBox with the Topology

> **Warning**
> You should take a snapshot of Server1 in order to undo these changes for future labs.

Figure 19.2. Interior Routing Network Topology



The above figure, Figure 19.2, “Interior Routing Network Topology”, shows the (eventual) network setup we're going to achieve in this lab. At each end of the tunnel (the yellow tube) a virtual device (tap0) is created automatically once the VPN is set up. It shows the topology at layer 3 (the network layer or the IP layer) and also at layer 1 (the physical layer, i.e., the cables), to help you to appreciate how the devices would physically connect to each other. We'll explain more details later in the lab.

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

1. You will create a temporary virtual machine for the remote host (like you did before for Client2), connecting it and server1 appropriately to a NAT Network. This defines the connection at the physical layer (layer 1).

2. The instructions of setting a NAT network for both Server1 and Remote are given in Figure 19.3, “NAT Network settings”.

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

Figure 19.3. NAT Network settings

The above figure, Figure 19.3, “NAT Network settings”, shows the NAT Network settings used for this lab. Make sure both Server1 and Remote use the same name for the NAT network, e.g., `NAT network for VPN`. A NAT network allows guests to connect to each other via the internet. It is as if both the server and the remote client were directly connected to the internet. This means we don't need to worry about setting up NAT ourselves for Internet connection.

Since both Server1 and Remote will need to be connected to a NAT network as opposed to just plain NAT[3], as we did before, we need to reconfigure their adaptors.

Open VirtualBox preferences and navigate to the Network tab. Change Server1's Adapter 1 so that it's attached to a NAT network as shown in Figure 19.3, “NAT Network settings”. Do the same to Remote.

Boot into server1 and you should still be able to access the internet. Test this by installing the openvpn and easy-rsa packages using **sudo apt-get install openvpn easy-rsa**.

While server1 is installing packages, if you haven't done so yet, start Client2 as the Remote, booting from the live CD. This is just going to be a temporary machine that we will use to test that we've configured the VPN correctly and so won't need a hard disk. Make sure that Remote's Adapter 1 is connected to the same NAT Network.

You need to:

1. **Screenshot**
   Make sure Server1's outside interface is connected to the NAT network and show the IP address of the outside interface in a screenshot (we will need this later when connecting to the VPN from Remote).

2. **Screenshot**
   Show the IP address of Remote's Ethernet interface in a screenshot.

3. **Screenshot**
   Finally, a screenshot showing Server1 and Remote can ping each other

19.2. Public-Key Infrastructure (PKI)

Central to the public key infrastructure is the idea of a certificate. Open your browser and go to any SSL enabled website (such as Google [https://www.google.com]) and click on the padlock symbol in the address bar and show the certificate. In safari (and I'm sure other browsers) will show the chain of trust along with the details of the certificate. Expand the details and peruse through and note any fields of interest.

[3] VirtualBox doesn't allow Virtual Machines on NAT to see one another by design.

Some important terms in the process:

Server
The server is where the OpenVPN connections terminate. It has generated a certificate (called the 'issued certificate') that has been signed by a certificate authority.

Client
The client is the OpenVPN program running on the remote machine. It must have access to the certificate of the certificate authority used to sign the server's issued certificate.

User
The user of the machine, knows their username and password.

When connecting to an OpenVPN server, both client and server mutually authenticate. The purpose of this is to ensure that the client it connecting to the correct server and vice versa. It is a similar process when SSH connections are initiated or when you visit SSL enabled websites.

Once the connection has been started the server presents its issued certificate to the client. The client can then verify that the server is the one its claiming to be. If the client determines that the server is lying then the client will terminate the connection. The other way the client will terminate the connection is if the server's certificate has been revoked (the location of a 'certificate revocation list' is included as part of the certificate authority's certificate).

If the client has a set of certificates then it presents them to the server. These certificates are optional and depend on the setup of the client. The server then makes the same decisions about the client's certificates, it checks that: they're signed by the same certificate authority (if not, the connection is terminated); the client certificate is not on the revocation list (otherwise the connection is terminated); the certificate is valid (hasn't expired and is after the issue date).

If the client is configured to send user credentials to the server then it does so now. Once the server has received the credentials the server checks that they're valid. Often this is performed by a separate program or script. In our case we will be using Linux's Pluggable Authentication Modules (PAM) which we will setup later on. If the checks fail, then the connection is terminated.

Once we have authenticated, the configuration is exchanged and the tunnel is brought up.

An important part of any VPN is the authentication of remote clients. There are several methods that can be used to authenticate a client to the server. In order from easiest (and least secure) to more complex (and more secure) they are:

- No authentication
- Username/password
- Client/Server certificates

Self-assessment

1. While it is a small distraction to talk about web certificates, I'd like you think about how the web browsers manage the certificate authorities. How do they end up in your browser? Who makes the decision? How can someone get their certificate authority trusted in the browser? What happens if the authority mis-issues some certificates?

2. Write some brief notes on the advantages and disadvantages of each of the authentication methods described above (and any others you may be able to find). To

focus the notes, think about why the shared secret is bad, and why the client/server certificates are good. Is there a better method?

3. In the next section we are going to setup our own public key infrastructure. Before we start, briefly define the following PKI-related terms:

- Certificate Store
- Certificate Authority (CA)
- Registration Authority
- Central Directory
- Certificate Management System
- Certificate Policy
- x509 Certificate
- Public Key
- Private Key

19.3. Server Certificates

Now that you understand the role PKI plays, we need to setup our own so that we can issue certificates as needed. By the end of this section we will have created our own certificate authority, along with a public/private key pair that clients will use to identify the server.

> **Caution**
>
> At this point it's worth noting that each of the certificates created below will be cryptographically unique. This means that (like the SSH keys you would have created in an earlier lab) there is no way to recover a public key from a private one (and vice versa). Once you start signing certificates with one CA, you can't suddenly switch to another. You will have to start again from this point.

The easy-rsa package, that we installed previously, contains a set of scripts that do a lot of the heavy lifting. Run **make-cadir ~/openvpn-ca** and change into the directory `~/openvpn-ca`. This is our working directory and makes it easier to keep the generated files separated other (personal) files.

You will see various scripts and some configuration files. The `vars` file sets up various variables used during the process of certificate generation. We have included some annotations in the listing below. You will need to edit `vars` as suggested below.

```
…
Take note of this warning!
# WARNING: clean-all will do
# a rm -rf on this directory
# so make sure you define
# it correctly!
export KEY_DIR="$EASY_RSA/keys"

# Issue rm -rf warning
echo NOTE: If you run ./clean-all, it will be doing a rm -rf on $KEY_DIR
```

```
The following is an old warning.
# Increase this to 2048 if you
# are paranoid.  This will slow
# down TLS negotiation performance
# as well as the one-time DH parms
# generation process.
export KEY_SIZE=2048

# In how many days should the root CA key expire?
export CA_EXPIRE=3650 This is 10 years

# In how many days should certificates expire?
export KEY_EXPIRE=3650 For the certificates generated by these scripts -- again 10 years

# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
export KEY_COUNTRY="US" Change this to "NZ"
export KEY_PROVINCE="CA" Change this to "Otago"
export KEY_CITY="SanFrancisco" Change this to "Dunedin"
export KEY_ORG="Fort-Funston" This is the organisation -- like Google, Otago University, etc.
export KEY_EMAIL="me@myhost.mydomain" In the real-world you should use a legitimate address.
export KEY_OU="MyOrganizationalUnit" Something like the Help Desk, or Research and Development.
…
```

Tip

The note about KEY_SIZE in the listing above refers to 'DH' -- Diffie-Hellman. If you're interested in how two people can establish a shared secret (to use for encryption) over an insecure channel (such as the internet), have a look at the Diffie-Hellman key exchange.

Tip

You may be wondering why we need to set parameters for ORG and OU and all that stuff. Well, its related to the ideas of centralised account management as used in large organisations.

Now that we've configured the certificate variables, we need to generate the certificate authority. Run the following commands:

```
mal@server1:~/openvpn-ca$ ln -s openssl-1.0.0.cnf openssl.cnf
mal@server1:~/openvpn-ca$ source vars
NOTE: If you run ./clean-all, it will be doing a rm -rf on /home/mal/openvpn-ca/keys
mal@server1:~/openvpn-ca$ ./clean-all
mal@server1:~/openvpn-ca$ ./build-ca
Generating a 2048 bit RSA private key
...
writing new private key to 'ca.key' This should be kept private and secure.
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called as Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----

These values below are obtained from the vars file we edited before, hit enter to accept the
value in square brackets, or change them here.
```

```
Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Otago]:
Locality Name (eg, city) [Dunedin]:
Organizational Name (eg, company) [...]:
Organizational Unit Name (eg, section [...]:
Common Name (eg, your name or your server's hostname) [...]:
Name [EasyRSA]:
Email Address [...]:
```

Now that we've generated the certificate authority, we need to create a public/private key pair that the clients use to authenticate the server.

Important

The following command is different to the previous one -- take care!

```
mal@server1:~/openvpn-ca$ ./build-key-server server1
Generating a 2048 bit RSA private key
...
writing new private key to 'server1.key' This should be kept private and secure.
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called as Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----

These values below are obtained from the vars file we edited before, hit enter to accept the
value in square brackets, or change them here.

Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Otago]:
Locality Name (eg, city) [Dunedin]:
Organizational Name (eg, company) [...]:
Organizational Unit Name (eg, section [...]:
Common Name (eg, your name or your server's hostname) [server1]:
Name [EasyRSA]:
Email Address [...]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: Leave this blank
An optional company name []: Leave this blank
Using configuration from /home/mal/openvpn-ca/openssl-1.0.0.cnf
Check that the request matche the signature
Signature ok
The Subjects Distinguished Name is as follows:
countryName           :PRINTABLE:'NZ'
stateOrProvinceName   :PRINTABLE:'OTAGO'
localityName          :PRINTABLE:'Dunedin'
organizationalName    :PRINTABLE:'...'
organizationalUnitName:PRINTABLE:'...'
commonName            :PRINTABLE:'server1'
name                  :PRINTABLE:'EasyRSA'
emailAddress          :IA5STRING:'...'
Certificate is to be certified until Jan 16 22:20:16 2028 GMT (3560 days)
Sign the certificate? [y/n]: y

1 out of 1 certificate request certified, commit? [y/n] y
Write out database with 1 new entries
Data Base Updated
```

Now we generate the DH parameters which will be used to generate a shared secret for secure communication between the server and a client after the authentication process is successful.

```
mal@server1:~/openvpn-ca$ ./build-dh
Generating DH parameters, 2048 but long safe print, generator 2
This is going to take a long time
...
```

You should now have the following files which are needed by the OpenVPN daemon.

ca.crt
The certificate authority's certificate, contains the public key of the CA.

server1.key
The private key used by the server.

server1.crt
The certificate used by the server, contains server1's public key and a digital signature from the CA.

dh2048.pem
The Diffie-Hellman exchange parameters.

Now we have all the certificates and other paraphernalia that we need in order to setup the VPN. The final task is to copy the keys to the `/etc/openvpn/` directory. First **cd keys**, then run **sudo cp ca.crt server1.key server1.crt dh2048.pem /etc/openvpn/**.

Finally, generate a secret key for extra security of the communication between Server1 and Remote.

```
mal@server1:~/openvpn-ca$ sudo openvpn --genkey --secret /etc/openvpn/ta.key
```

Over the next few sections, we are going to walk through the following steps to incrementally build the VPN.

19.4. Initial Server Configuration

During this process we are going to use a separate network 10.8.0.0/24 for the remote clients to use as VPN. This is deliberately chosen to be different to the existing LAN network. The main reason is that it helps to keep the different networks logically separate (and it makes it more obvious where the traffic is flowing). When we come to the IPv6 addresses we will, again, use a separate network (we used https://www.ultratools.com/tools/rangeGenerator, and set the Global ID to '6b410435ce' and Subnet ID to 'ffff' -- to make it as visually different from the existing network as possible.).

The networks used by the server *must* be different to the network that the remote client is connecting through. If it's not, then the routing won't work properly. This is often a cause of conflicts and issues.

Use **gunzip -c /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz | sudo tee /etc/openvpn/server.conf** to copy the sample `server.conf` provided by the maintainers of openvpn. Read through the file and edit/add/adjust the file to match the options below leaving the other options at their default value. (We have removed the comments for brevity---you should be sure to understand what the options are doing).

```
port 1194
proto udp

dev tun

ca ca.crt
cert server1.crt
key server1.key
dh dh2048.pem

server 10.8.0.0 255.255.255.0 See the note below.

keepalive 10 120

tls-auth ta.key 0

cipher AES-256-CBC

This is where we're configuring the username/password authentication options.
verify-client-cert none
plugin /usr/lib/x86_64-linux-gnu/openvpn/plugins/openvpn-plugin-auth-pam.so login
```

port ...
The port to listen on, 1194 is the default for OpenVPN.

proto ...
The protocol to use to encapsulate the network traffic. It doesn't matter that the traffic going across the VPN is UDP or TCP.

dev ...
The virtual network device to use for the VPN tunnel. It can be one of tap or tun, which one depends on whether you want to bridge or route the VPN to the LAN respectively. (Or if the VPN needs to handle non-IP traffic -- in which case you need to use tap).

For our use case we are setting up a separate, routed, network with only IP-based traffic, hence the use of tun.

ca, cert, key, dh
The PKI files that we created previously.

server ...
The network addresses of the VPN that the server is going to create. Because we're setting up a routed network, we need to use a different network to any of the existing connected networks (viz. something that's not 192.168.1.0/24 or 10.0.2.0/24). The value here is the default value for this option.

keepalive 10 120
This is a shortcut to specify two options and the behaviour differs a little depending on where OpenVPN is running. For the server, it'll send a ping if there's been no activity for 10 seconds, and if it fails to receive a response in 120 seconds restart the connection[4].

tls-auth ta.key 0
This is an extra security beyond that provided by SSL/TLS. It can help block DoS attack and UDP port flooding. The server and each client must have a copy of this key. This file is secret and read-only by the owner. The second parameter should be 0 on the server and 1 on the client (Remote).

[4] In the documentation, they use the example of keepalive 10 60, and say the restart happens in 120 seconds -- maybe there's a typo.

cipher AES-256-CBC
This is the selected cryptographic cipher. This option must be specified in the **openvpn** command on the client side or in the client config file.

verify-client-cert none
This makes the client certificates optional as we are only authenticating by username/password, and don't want to authenticate the client too. To make use of this, the client's certificates would need to be signed by the same CA (in the same manner as the server certs were).

plugin ...
This tells OpenVPN to use PAM to perform the username/password checking. Briefly, PAM allows other applications to check authentication against various options. In this case, we use the 'login' option with the options supplied to PAM from the OpenVPN client.

Bridging vs Routing

Bridging is where the LAN and VPN clients share the broadcast domain, and would allow the existing DHCP server to provide addresses to the VPN clients. The other consideration on this front is if there are any services that require neighbour discovery (some features of Windows servers need this).

Routing on the other hand allows you to segment the VPN traffic from the LAN and makes management of firewall rules easier. We shall see an example of this in the Firewalling lab.

Tap vs Tun, performance issues

Tap devices: typically behave like a real network adapter (despite it being virtual); can transport any network protocol; work in layer 2 (so ethernet frames are passed over the VPN). However, they cause more broadcast traffic across the VPN; add ethernet frame overhead; suffers from poor scalability; and cannot be used with Android or iOS devices.

Tun devices: have lower overhead, only transports layer 3 traffic (IP). However, broadcast traffic is not transported; older versions of OpenVPN lacked support for IPv6; and cannot be used in bridges.

We're now ready to start the service using `systemd`, **systemctl start openvpn@server.service**. This looks a little different from other **service start** commands you would have seen before. We can have multiple VPN services defined each with their own configuration file. This command uses the token after the '@' to figure out which configuration file to use when starting the service. Check `/var/log/syslog` to make sure that the service started properly.

Screenshot

Take a screenshot showing the log information or `status` of the `openvpn@server.service` VPN service.

Once you've successfully started the service, have a look at the output of **ip addr show** and **route -n**. You should see something resembling the following. Note that a new gateway 10.8.0.2 is generated for the VPN in addition to the server's address 10.8.0.1.

```
mal@server1:~$ ip addr show
1: lo:
    ...
2: outside:
    ...
3: inside:
    ...
5: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast ↩
  state UNKNOWN group default qlen 100
    link/none
    inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
    valid_lft forever preferred_lft forever
mal@server1:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.1        0.0.0.0         UG    0      0        0 outside
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 outside
10.8.0.0        10.8.0.2        255.255.255.0   UG    0      0        0 tun0 This is new...
10.8.0.2        0.0.0.0         255.255.255.255 UH    0      0        0 tun0 ... as is this.
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 inside
```

We now need to connect the remote client. For that, the remote client needs to have a copy of the `ca.crt`. On Remote, use SCP to copy the file across **scp mal@10.0.2.15:/etc/openvpn/ca.crt ~/.**. This copies the certificate of the certificate authority (yes, this sounds strange but correct and precise) at `/etc/openvpn/ca.crt` from Server1 to our home directory (preserving the filename).

We need also copy `ta.key` from Server1 to Remote using **scp**. Since `ta.key` is read-only by root, we will need to work out a way to copy the file to Remote. Once it is copied to Remote, the file should be changed back to read-only by the owner. This problem is left for you to solve by yourself.

We've now got all the information the remote client needs to be able to connect to the vpn. Install the `openvpn` package on Remote and connect the VPN using **sudo openvpn --remote 10.0.2.15 --dev tun --auth-user-pass --ca ca.crt --client --remote-cert-tls server --tls-auth ta.key 1 --cipher AES-256-CBC --auth-nocache** under the home directory where `ca.crt` and `ta.key` are copied. You should see the status of the client's connection in the terminal. Use a different terminal to do the assessment below.

Assessment

1. What address ranges do you expect to ping when the VPN is connected? What subnets can you actually reach? Is this different from what you expected? Why or why not?

Screenshot

Take a screenshot showing which hosts you can reach and which you cannot.

2. Examine the routing table on the client before and after connecting to the VPN. What new routes are added? What do you notice about the metrics? Why do you suppose this is?

3. Run **sudo tcpdump -i enp0s3** on Remote. This examines the traffic on the ethernet interface and emulates someone performing a man-in-the-middle (MITM) attack. Run the following two commands on Server1 to **ping** Remote, and compare and contrast the tcpdump output.

- ping -c 1 10.8.0.6
- ping -c 1 10.0.2.4

What implications does this hold for (free) public VPN servers?

19.5. Routing VPN Traffic (optional)

At the end of the previous step, the remote client can only ping server1's VPN address. In this section we're going to adjust the configuration so that the remote clients traffic is routed properly.

We need to make a couple of small modifications to `/etc/openvpn/server.conf`, these are presented below. As previously, modify the configuration to take these tweaks into account.

```
push "redirect-gateway def1"
script-security 2
learn-address "/etc/openvpn/learn-address"
```

push ...
: This tells the server to give any of the connected clients that parameter. In this case it tells the clients that all their traffic should be sent through the vpn.

script-security ...

learn-address ...
: While the push directive can get the traffic from the VPN onto the LAN, we need to be able to get the LAN traffic onto the VPN. I'll discuss this in more detail below.

This works well to get the VPN traffic to the LAN clients, but the LAN clients don't know how to respond to the traffic from the VPN --- there is no route. The`learn-address ...` script handles this case for us, so that the server acts as a proxy for the vpn clients.

You should be familiar with ARP (or Neighbour Discovery), but for the sake of completeness, I'll describe the process here. When a LAN client wants to send a packet to a vpn client, it performs its neighbour discovery protocol (ARP in IPv4), where it asks "who has 10.8.0.1?". Because server1 is acting as a proxy, server1 will respond with its MAC address, and eventually route the traffic through the vpn to the correct host.

Now that we know what needs to happen with the `learn-address ...` stanza. We have created a simple script to add the proxy-ing. This script should be placed in `/etc/openvpn/learn-address`, owned by root, and executable.

```
#!/bin/sh
action="$1"
addr="$2"

logger "learning: $action $addr" Show something in the syslog

case "$action" in
 add | update)
  ip neigh replace proxy "$addr" dev inside
  ;;
 delete)
  ip neigh del proxy "$addr" dev inside
```

```
  ;;
esac
```

This script is responsible for adding and removing routes for each of the remote devices when they connect and disconnect respectively[5]. The script adds/removes clients individually, we could have set it up so that the whole VPN subnet was routed to server 1, and we wouldn't have needed this script, however we wanted to demonstrate this feature.

Disconnect remote1, restart the service (checking syslog to make sure it started properly) and reconnect remote1. If you cannot connect, check the syslog on server1 to see if there are any problems with the service.

Screenshot

Take a screenshot showing which hosts you can reach and which you cannot.

19.6. IPv6 Additions (optional)

Now that we have a functioning IPv4 VPN, we need to setup the same for IPv6. We need to make a couple of small modifications to `/etc/openvpn/server.conf`, these are presented below. As previously, modify the configuration to take these tweaks into account. Disconnect the remote client, restart the service (checking the logs) and connect the remote client again.

```
tun-ipv6
push tun-ipv6
server-ipv6 fd6b:4104:35ce:ffff::/64
push "route-ipv6 fd6b:4104:35ce:0::/64" We have added the '0' after the 35ce to make it clear that we're routing
```

Screenshot

Take a screenshot showing which hosts you can reach and which you cannot.

1. Run **tcpdump** on remote1's `enp0s3` interface. Do you see any IPv6 traffic when pinging server1's IPv6 address? Is this what you would expect? What's happening?

19.7. Client-side Configuration and DNS (optional)

On remote, we've been using the command line to manage the starting and stopping of the OpenVPN client. It's time we move this to a configuration file.

Configuration File

```
client
dev tun
proto udp

comp-lzo

remote 10.0.2.15 1194 This is the public address of the server

nobind
```

[5] Fortunately it requires no changes to work with IPv6!

```
persist-key
persist-tun
ca ca.crt

auth-user-pass

The following lines are to help sort out DNS---which we'll do next.

resolv-retry infinite
script-security 2
up "/etc/openvpn/update-resolv-conf"
down "/etc/openvpn/update-resolv-conf"
```

Screenshot

Take a screenshot showing which hosts you can reach and which you cannot.

This section is optional. It is here so we can get make sure that the services are setup properly. If you have completed the DNS Lab, we strongly suggest that you complete these steps.

In the DNS lab you created an access list that restricted DNS queries to the local networks. Because we have created two new subnetworks (one for each IPv4 and IPv6), we need to allow queries from these hosts.

Edit the `/etc/bind/named.conf.options`, by creating a new ACL and add it to the `allow-query` and `allow-recursion`stanzas. Restart the `bind9` service in the usual way.

```
acl "clients" {
...
}
acl "vpn" {
 10.8.0.0/24;
 fd6b:4104:35ce:ffff::/64;
};
options {
 ...
 allow-query { "clients"; "vpn"; };
 allow-recursion { "clients"; "vpn"; };
 ...
};
```

Now the final remaining step is to push the dns server to the vpn clients. Edit `/etc/openvpn/server.conf` to include the `push "dhcp-option DNS 192.168.1.1"`. Disconnect remote, restart the service (check the logs), and reconnect the remote.

You should be able to resolve all the local (internal) DNS names we defined previously, as well as check that they're reachable via ping.

Screenshot

Take a screenshot showing that you can resolve the internal DNS names from the remote client. Take another screenshot to show that you can ping (by name) server1.

19.8. Final Words

In this lab we have setup and configured a VPN which allows remote clients to appear as if they were on the local LAN. This is a powerful (and useful) tool that is used to ensure the integrity of private data being transferred across an untrusted network.

We've been using username/password authentication to allow clients access. As an optional exercise, you could extend this by using the PKI infrastructure we setup at the start to generate certificates for the clients.

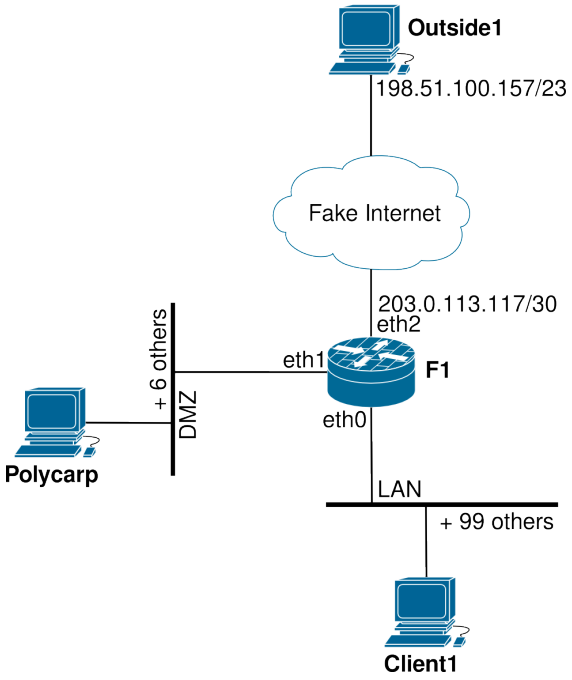
Lab 20 Subnetting

This lab is actually run as a class-tutorial. Thus, when we come to do this tutorial, everyone should do it at the same time. In this tutorial, we shall learn about subnetting, mostly in IPv4, but we shall also cover IPv6, which is easier. IPv4 is a bit more challenging, largely because we have to deal with conversions between decimal and binary a lot, but the concepts are otherwise identical.

As part of this tutorial, you will be given a set of questions. These questions are very similar in style to what you would find in any networking examination, whether it be a professional certification (such as a Cisco CCNA), or for this paper. The content of this tutorial is vendor inspecific: everyone in the networking field needs to know this.

This tutorial and the following firewall lab will use the IPv4 addressing scheme you have developed in this tutorial. The (quite small) network we shall be addressing is shown in Figure 20.1, “Network Map for this Laboratory”.

Figure 20.1. Network Map for this Laboratory



The network that we shall be subnetting. You shall later use the addressing scheme when we implement this network in a the next lab.

The external network will be similar to the Internet, but several orders of magnitude smaller and not as complete. There is a single host on the external network, with a very different public IP address (as you would expect with a distant host on the Internet.) As we are modelling the Internet (and we use the term “model” very loosely here,) the hosts are not using RFC1918 private “non-routeable” addresses. Generally, you should never make up addresses in this way; we just want you to have some experience working with other address ranges and subnet sizes.

Speaking of subnet sizes, you will not be using the /24 addresses we have become accustomed to in this lab. To make it more interesting, the DMZ and internal networks will not use /24

(previously called “Class C” networks). Instead, we shall use CIDR (pronounced “cider”) addressing and use subnets.

F1, which is a router/firewall, will have a single public IP address (203.0.113.117), and the internal and DMZ networks will use two subnets, each appropriately sized, which you will split out from 10.8.2.0/24 yourself, which for the purposes of this exercise you must use sparingly.

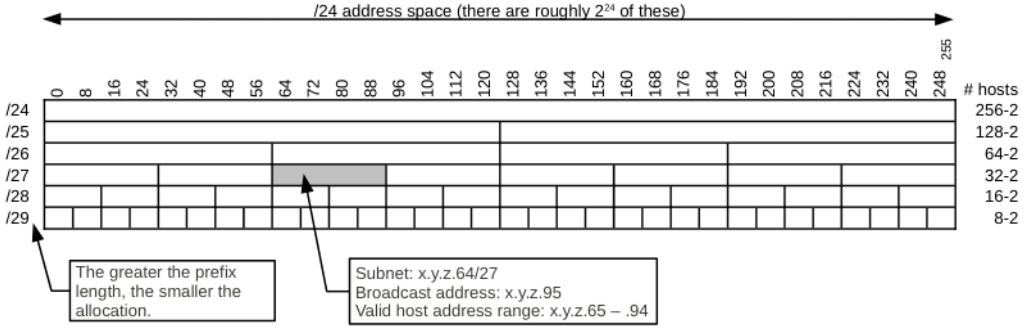
You will only have one machine in your DMZ in this lab, but you should pretend that in the real network that you *currently* have 7 hosts in the DMZ, and you must allow for some growth. **What is the smallest practical subnet size for the DMZ network?** Don’t forget the very first address (subnet address) and very last address (broadcast address) are reserved, and the router will need an address for each subnet it has an interface on. For the internal network, consider your needs to be 100 hosts, which is considered a small business. As a point of comparison, a medium sized network would have about 500 hosts.

For the DMZ network, to make it a little more educational, do not use the first available subnet, but choose the second or third available subnet instead. [In an exam situation, you should be able to handle any subnet.] When you have designed your subnets, write them down, including the following information:

- network address,
- netmask in dotted decimal format, and corresponding prefix-length in CIDR notation,
- gateway address,
- broadcast address,
- first and last usable host address,
- and current number of host addresses unused in each subnet.

For your benefit, Figure 20.2, “Subnetting a /24” shows a chart to help you understand the relationship between prefix-length and address-space. Using this chart will help you check for overlapping allocations, and to visualise “holes” in the address-space. You should not, however, depend on it because it will be unavailable in the examination.

Figure 20.2. Subnetting a /24



This chart shows the relationship between prefix-length and allocation size. You can use it for visualising your address space and checking your subnetting calculations.

In production networks, you generally need to have some tool to assist you in managing your address space. In many networks, one simple solution is to use a spreadsheet. A nicely

designed spreadsheet can show you your address space very nicely. You can drill down to arbitrary levels of detail, even down to individual addresses. This sort of spreadsheet could be useful for adding things like DNS names, MAC addresses, and accounting data. However, for larger networks with more IT staff, this rapidly becomes unmanagable. One sample spreadsheet is available in the `Lab Resources/Subnetting` folder. It spans a few pages, so is not included in this text, but Figure 20.3, “1000-foot View of an Example Address Management Spreadsheet” shows a “1000-foot” view of it.

Figure 20.3. 1000-foot View of an Example Address Management Spreadsheet

1000-foot view of a example address management spreadsheet. The spreadsheet is very long, so it has been split into four pages. Each page covers a /26 exactly.

Note that a visual device has been included to help you see the size of various allocations. Colour is used to delineate different allocations; here there are two subnets shown, one for the LAN, and another for the DMZ. There is still ample empty space. Inside the LAN subnet, a particular allocation has been set aside for a dynamic DHCP range of addresses.

20.1. In-Class Exercises

Most of these questions were sourced from SubnettingQuestions.com [http://subnettingquestions.com/], which is a very useful resource for practicing these skills; recommended for your exam study. The last question is another style of question you could expect in any subnetting examination.

1. Which subnet does host 192.168.77.108/28 belong to?

2. What valid host range is the IP address 172.21.71.219/20 a part of?

3. What is the first valid host on the subnetwork that the node 172.16.139.35/26 belongs to?

4. What is the broadcast address of the network 172.30.134.0/23?

5. How many subnets and hosts can you get from the network 172.20.0.0 if you want to partition it into subnets like 172.20.0.0/255.255.254.0?

Sometimes you need to know the class of an address, for example, to answer a question such as this one. The class of an address, and therefore its network mask, can be determined by the first few bits of the address, as shown below.

```
Class A  ≤127  0nnn nnnn  hhhh hhhh  hhhh hhhh  hhhh hhhh
Class B        10nn nnnn  nnnn nnnn  hhhh hhhh  hhhh hhhh
Class C  >192  110n nnnn  nnnn nnnn  nnnn nnnn  hhhh hhhh
```

6. Given the following address allocations, create two further allocations, a /29 (“Test Network”) and a /27 (“Sensor Network”), such that fragmentation of the address space is minimised. The “Test Network” can be a subnet of the Engineering subnet.

```
172.16.20.0/24      Complete allocation
172.16.20.0/27      DMZ
172.16.20.64/26     Engineering
172.16.20.192/26    Main Office
```

20.2. Subnetting in IPv6

Subnetting in IPv6 is much easier than in IPv4, largely for two reasons. The first is that because the address-space is much larger, we can take a much more structured/readable approach, and also because working with hexadecimal is much easier than decimal, when it comes to subnetting. You will come to appreciate that in this section.

In case students have not already done it, the IPv6 routing for the lab on Interior Routing will be discussed as a class exercise. Even if you haven’t done, or don’t have time to implement the network, all students should understand the basic strategy behind the addressing scheme.

Students should watch the 20-minute IPv6 Subnetting video in the Lab Resources before the tutorial.

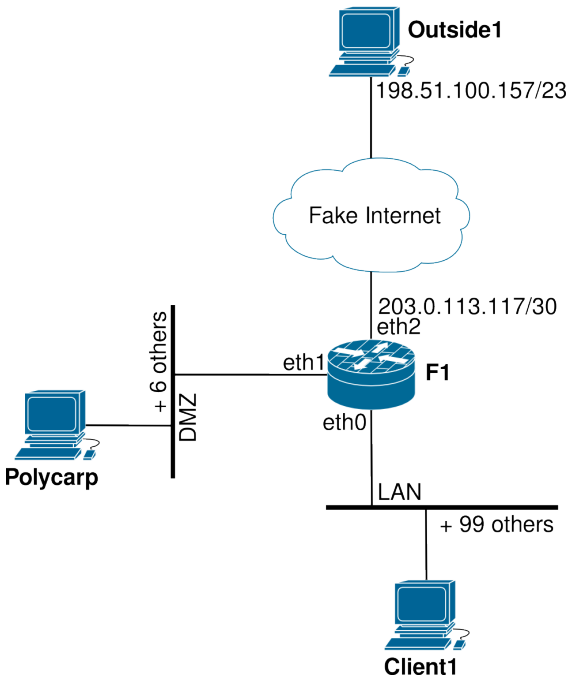
Lab 21 Firewalls

It is recommended that you work in pairs for this lab to make it easier to do testing and troubleshooting. Before doing this lab, you should have completed the Subnetting tutorial for IPv4, as we shall be using the same network design, as shown in Figure 21.1, “Network Map for Firewalling”.

You will find it very useful to have read parts of the Vyatta documentation on Firewalling, which should be available in the `resources` folder of your `Desktop` (`~/Desktop/resources/Lab Resources/Vyatta/Documentation`). Two documents are useful: `Vyatta_Firewall` and `Vyatta_NATRef`.

In this lab, you will make use of the Vyatta firewall to implement several policies suitable for a small network with a few public services. In this network, you will have an external network, an internal LAN and a De-Militarised Zone (DMZ), where your public services are being housed. As already mentioned, this is identical to the network we addressed in the Subnetting tutorial.

Figure 21.1. Network Map for Firewalling



The most peculiar thing about the Fake Internet here is the slight-of-hand we are playing with the default route set on Outside1 and F1. They are in different networks, but there is no routing between them, so we have played a trick whereby the packets are sent directly onto the network, as if they are delivered to a machine on the same subnet, rather than forwarding to a router. This is to make it possible for us to assign Outside1 and F1 IP addresses that are very different but connected without routers. So it looks like the Internet but without the trouble of setting up the routers. It also makes for an interesting thought experiment.

21.1. VirtualBox Configuration

Define the physical topology of your network; you should have learned the skills for this in the *Internal Routing* lab, but here are some reminders.

- All interfaces will be Internal Network adaptors, not NAT or anything else. Remember that F1 will have three interfaces, so call each network/switch "LAN", "DMZ" and "fake_internet" appropriately.
- Create a virtual floppy disk for F1 using **dd** as you did in the *Internal Routing* lab. Remember to use the **init-floppy** command inside Vyatta in order to save. If you don't do this, your changes will only be saved in the temporary filesystem, which is volatile in the LiveCD environment; therefore you will lose your work if you don't prepare the floppy!
- You may have issues with the default network adapter. You should choose either `PCnet-Fast III` or `Paravirtualized Network` adapter.

21.2. Configure and Test Basic Connectivity

Configure the basic system properties and network interfaces on F1 using the skills you learned during the lab on *Internal Routing*. F1 should be created with the Vyatta LiveCD. Do not forget the detail of settings such as the PAE/NX feature of the processor and enable of serial port.

All other machines can be created with a Ubuntu desktop LiveCD.

On the Client1 and Polycarp, just use **ifconfig** and **route** to set the interface address and add a default route to F1. On Outside1, we shall need to do something a bit different to implement our Fake Internet. On Outside1, use **ifconfig** as you would normally, but use the following **route** command:

```
# route add default dev eth0
```

This basically says "if you don't have a better route available, send it *as a local delivery* onto the eth0 link". Normally, we would have expected Outside1 to send it via a router in its own subnet.[1]

We will need a similar rule for F1 as below:

```
set system gateway-address 203.0.113.117
```

This is doing essentially the same thing. Note that the gateway is simply the same IP address of F1's interface, which seems a bit odd really. In effect, it puts it out on the local link attached to the interface with that IP address.

Once you have configured all interfaces and default routes as appropriate to each system, test that each host can ping its directly attached neighbors. You will also find that ping should work between DMZ and LAN too. For example, you should be able to ping Client1 from Polycarp.

[1] Another way of achieving the Fake Internet slight-of-hand is to have the prefix-length of Outside1 and F1s' interfaces on the Fake Internet to be a /0, which puts them in the same subnet.

A ping from Client1 to Outside1 will fail, because Outside1 doesn't know it should send the (return) packets to F1, as it just puts them on the local link (our Fake Internet). Since the destination of the packets is Client1 (10.8.2.2), the ARP for Client1 will fail and so will the delivery of the return packets.

21.3. Implement Source-NAT

If you haven't already, skim-read the relavent sections of the Vyatta documentation on NAT. This will save you a lot of time, and enhance your understanding. Particularly, pay attention to commands **set services nat rule ...** and the configuration examples.

Configure Source NAT for LAN and DMZ to the Fake Internet. We suggest you create two Source-NAT rules: one for traffic coming from the DMZ, and another for traffic coming from the LAN.

After the configuration, use **show service nat rule ...** to double-check if the rules are correctly set.

To test, use **echo "Response from Outside1" | sudo nc -q1 -v -l 80** in one window on Outside1, **sudo tcpdump -n -i eth0** should be running in another window on Outside1, and then launch **echo "Request from Client1" | nc -q1 -v 198.51.100.157 80** in a window on Client1. This is how you can test for a TCP connection on port 80 (or any TCP port, in general). Note you will have to start the fake server each time as it only serves a single connection and then quits. You could instead wrap the dummy-server in a loop:

```
$ while true; do
>   echo "Response from Outside1" | sudo nc -q1 -v -l 80
> done
```

21.4. Implementing Destination-NAT

1. Configure Destination NAT for TCP services to Polycarp and test them by connecting to F1's eth2 IP address. Have **tcpdump** running in a window on Polycarp to see if traffic is getting to it. You should open the TCP ports 25, 53, and 80 of Polycarp to Outside1.

   Ensure you have a dummy-server running on the TCP port on Polycarp that you want to be connecting to.

2. Configure Destination NAT for UDP services (e.g. port 53) to Polycarp and test them.

   We can use **nc** for UDP traffic also, just add a `-u` to the options for both the server and client. Note that, for the server to listen to a udp port, you should drop the `-q1` option of **nc**.

3. Configure Destination NAT for TCP services (e.g. port 22) to Client1 and test them.

The above suggested ports are for DNS, HTTP, SMTP and SSH, which you will need when setting up the firewall in the following sections.

21.5. Firewall Policies

Now that we have set up the required NAT functions and provide services to the outside network, we should set up firewall rules to restrict accesses between machines/networks and to prevent intruders from breaking out of the DMZ into the LAN.

Firewall rules are not something to be made up arbitrarily, but are instead the result of a set of policies. These are the policies that we shall be implementing in the rest of the lab:

- DNS, HTTP and SMTP services are offered by Polycarp in the DMZ, and should be accessable to everyone by default.
  Normally it would be desirable to have these services on different machines to limit the effect of a breakin, but here we are more concerned with lab resources.

- To prevent intruders from breaking out of the DMZ into the LAN, nothing is permitted to be addressed *to* F1 from the DMZ.

  Similarly, nothing in the DMZ may initiate communications with hosts in the LAN.
  In general, you don't want the LAN hosts trusting the DMZ hosts.

- SSH connections, on the standard port, from the Internet are to be forwarded through the NAT to one particular host (Client1) in the LAN. In this case, Client1 is acting as a server.

- Normal hosts in the LAN will not be able to connect to the outside network. They may only make connections to other hosts in the LAN and also into the DMZ, but only for those services offered in the DMZ.
  This sort of policy is common to force users to go through some sort of proxy. It is becoming less common in a lot of networks.

In order to begin implementing these policies, you need to at least know how to match each type of traffic. As a reminder, here is how you match each type of application's traffic. You are suggested to look through `/etc/services` and find out what the *service name* is for each port.

- DNS uses UDP port 53 and also TCP port 53.

- HTTP uses TCP port 80.

- SMTP uses TCP port 25.

- SSH uses TCP port 22.

You should begin by annotating the network diagram with each service's location and where it should be allowed from. Make sure you understand how each type of traffic will cross the router (ie. does it go *through* the router, from one interface to another, or is it addressed *to* the router, or is the traffic coming *originating from* the router.

For each flow of traffic, decide whether each direction of the firewall should have an allow-by-default or drop-by-default policy.

21.6. Starting to Filter

For the purposes of penetration testing, add a route on Outside1 so it sends packets to F1 in order to get to the network 10.8.2.0/24. Hint: use **sudo route add -net 10.8.2.0 gw 203.0.113.117 netmask 255.255.255.0**

Test if Outside1 can address internal hosts (it will be able to, although this is undesirable). You should be able to ping Polycarp and Client1. In this section, we shall aim to prevent this, as well as implement and test the policies defined earlier.

One thing you have to realise about Vyatta firewalls is that as soon as you attach one packet filter to an interface, *all* interface firewalls change their default behaviour from allow to

drop. This means packets will get dropped (silently), which is fine enough on a production firewall, but very frustrating to beginners when you are trying to figure out why packets are getting dropped. To help with this, we shall start by defining a simple packet filter for each firewall that implements the stateful rule (to allow packets associated to/with already allowed sessions) and logs and then drops each other packet that it sees. Then we attach the filters to the firewalls of an interface once and for all. That way, we do not need to attach the filters to the interface again but can add rules to packets filters in a nice piecemeal fashion. The logs will be sufficient for us to know which packet filter we need to adjust, and we simply adjust the packet filters until the policy works.

If you don't quite understand what we are talking about here, don't worry as you will understand after you have done the rest of the lab.

Warning

This is for those with IPTables experience only.

Internally, Vyatta does it firewalling using IPTables. However, for those with IPTables experience, there is one important difference to beware of. **set action accept** does *not* translate to a **iptables ... -j ACCEPT** but rather a **iptables ... -j RETURN**. **set action drop** however, *does* translate to a **iptables ... -j DROP**.

The annoying thing about this is that for traffic that goes through the router (ie. the IPTables FORWARD chain) you need to add rules to allow traffic in the relavent “in” and “out” packet filters. The diagrams in the Vyatta Firewall documentation show clearly how packets traverse the firewall.

Each interface has three firewalls, `in`, `out` and `local`. the `in` firewall is checking the incoming traffic to the interface. `out` is checking the outgoing traffic from the interface. `local` is for the traffic destined to the local machine (i.e. F1).

F1 has three interfaces, so we will eventually need up to nine packet filters. We shall name these nine packet filters in a logical manner, following the template FW_NETWORK_FIREWALL. For example, the packet filter attached to the `in` firewall on the Fake Internet shall be called FW_INTERNET_IN.

There is a lot of tedious typing to create the rules for each filter. For that reason, it is a common technique to use scripts to generate router configurations, which you can then easily paste into a configuration session over a Serial console or SSH (or, if you really want, Telnet). For this reason, you should enable SSH on F1 allowing access from Client1, then SSH into F1 from Client1. Here’s how to do that:

```
# set service ssh protocol-version v2
# commit
```

And now, on Client1 (hey, that's the Client1 in this Firewall lab), connect to F1:

```
client1$ ssh vyatta@10.8.2.1
…
```

Now we shall generate commands to produce our nine initial packet filters by using a shell-script. Each packet filter will start with the stateful rule, and end with a log-and-drop. For testing purposes, we have added the **log enable** directives. After testing, you can manually remove the these directives.

On Client1, put the following into a script and run it. Copy the output of the script, and paste the result into a configuration session on Vyatta.

```
for net in INTERNET DMZ LAN; do
  for fw in IN OUT LOCAL; do
    echo edit firewall name FW_${net}_${fw} rule 10
    echo   set description '"Allow already accepted traffic"'
    echo   set action accept
    echo   set state established enable
    echo   set state related enable
    echo top
    echo edit firewall name FW_${net}_${fw} rule 1024
    echo   set description '"Log and Drop traffic by default"'
    echo   set action drop
    echo   set log enable
    echo top
  done
done
```

After you paste the output of these commands into your SSH session, remember to **commit** and **save**. Don't you just love scripting for automation?

Let's see the effect of our work. **show firewall** and see what has been added. However, the packet filter's aren't actually doing anything yet, because they are not attached to the firewalls of the interfaces. A lot more commands are needed, although this time, less repetitive. Use the following script to generate the commands and then run them in a configuration session of F1.

```
echo -e 'eth2 INTERNET\neth1 DMZ\neth0 LAN' | while read iface net
do
  for fw in in out local
  do
    echo set interfaces ethernet $iface firewall $fw name FW_${net}_${fw^^}
  done
done
```

The Bash substitution ${pattern^^}

The substitution ${fw^^} simply uppercases the value of the fw variable. It is not available in older versions of Bash, such as that which come with Vyatta. It is available on Ubuntu 10.04 LTS at least, which is why you need to do this on your Ubuntu machine, and not on the firewall host.

Don't forget to **show interfaces**, **commit** and then **save**.

However, now you might run into your first stumbling block. If you were connected via SSH, you may very well find that you are no-longer able to interact with the router (this might not be the case, but it did come up during testing). At the least, any new SSH connections will be dropped. If you abort any existing hung SSH connection (using **Return ~ .**) and attempt to reconnect, the connection attempt will hang. If, on the F1 console (not via SSH) you do a **run show log tail 3** on your firewall, you will find an entry such as the following (we've highlighted the more interesting parts):

```
$ run show log tail 3
timestamp vyatta kernel: [ignorable] [FW_LAN_LOCAL-1024-D] ↩
IN=eth0 OUT= MAC=src-mac:dst-mac:ether-proto SRC=10.8.2.2 DST=10.8.2.1 ↩
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=78110 DF ↩
```

```
PROTO=TCP SPT=52099 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0
```

Before we proceed, let's check our understanding of the log message:

FW_LAN_LOCAL-1024-D
: This is formed by the packet filter name, the rule name, and the fact that it was logged in a 'drop' rule. Thus, we know exactly where in our firewall it was dropped, and which packet filter we need to modify to let it through.

IN=eth0 OUT=
: From this we can infer that it is a 'local' chain, because it is going into the router, but not out.

  In IPTables, this would appear on the 'INPUT' primary chain, rather than the 'FORWARD' or 'OUTPUT' chain.

  We can emphatically say that the connection attempt came from the LAN, which is eth0.

SRC=10.8.2.2 DST=10.8.2.1
: The source address tells us the packet *appears* to have come from Client1, and was sent to the the IP address of F1's LAN interface. Note that this could be spoofed, although most firewalls will have options turned on to detect spoofed traffic where possible.

PROTO=TCP
: This is one of the things we need to know in order to match the traffic: that it is TCP.

DPT=22
: Knowing the destination port (in this case it is TCP/22, which is SSH) is also one of the key ways we match different types of traffic.

  For TCP, the *source* port is generally a short-lived, random high-numbered port (we call it an *ephemeral* port) and as such we generally don't concern ourselves with the source port. But occassionally we do also consult the source port for some UDP traffic.

SYN
: For TCP traffic, the SYN bit is set in the header for the first packet, thus this packet is understood to be *initiating* an SSH connection to F1, from the LAN.

So now we should be able to add a rule that allows us to get past this problem:

```
# edit firewall name FW_LAN_LOCAL
# show
rule 10 is the stateful rule…
… and rule 1024 is the default drop …
… so we want our rule to be perhaps rule 20.
# edit rule 20
# set description "Allow SSH from Client1 only"
# set action accept
# set protocol tcp
# set state new enable
# set destination port ssh
# set source address 10.8.2.2
Note that we only want one host in the LAN to access F1 via SSH. For other protocols,
we would typically specify a prefix, such as 10.8.2.0/25
# commit
```

Now check that you can access F1 via SSH from Client1. Make sure you take note of the timestamp in the logs so you can ignore log messages from previous attempts.

Another way of looking for new logs

You may find it a lot more convenient if you log into another virtual terminal on F1, and run the command **tail -f /var/log/kern.log**, which is where the kernel logs are routed to on Vyatta.

The reason that this is more convenient is that you can simple type **Return** a few times in order to visually separate old messages from new messages.

Now you can go on and implement the rest of the policy, using the same techniques.

When you are finished, you may want to adjust your logging. You don't want to log everything, as it could be used as an attack vector (either to fill up hard-disk space, or to erase logs that might be stored in a ring-buffer). But you generally want to keep some record of the sorts of things you are dropping.

Testing-by-doing is the easiest way of checking if the system works, but it is not the only method you should use to audit your firewall. In particular, you should "desk-check" the configuration to see if you are missing anything, particularly to check that you haven't left out a default case. In a real-world situation you would also test the firewall rules with tools such as port-scanners (eg. **nmap**) and packet injection tools, but the latter is rather more advanced that this lab should go and the tools are not available in the default Ubuntu Live CD (although there are Live CDs that do, you might like to look around).

21.7. Assessment

1. Implement and test the policies listed.

2. Use **show configuration** (or **run show configuration** if you are in configuration mode). Unlike **show** inside configuration mode, it will also show default values, which is very useful when you want to see default firewall behaviour.

   This is what you want to submit for marking.

Lab 22 Catchup Lab

This lab is set aside for help people catch up; priority will be given to students doing previous labs and assignments.

Lab 23 Catchup Lab

This lab is set aside for help people catch up; priority will be given to students doing previous labs and assignments.

Part IV. Additional Topics

Lab 24 [Optional] File Transfer and Web Caching

Old Lab

This lab hasn't been run recently, so no guarantees are made regarding the commands and their output.

In this lab, we shall be covering the File Transfer Protocol (FTP) briefly, before going on to the main part of the lab, which is Web Caching.

You will need to make some modifications to DNS. Add an alias for `ftp` that points to Server1 for both IPv4 and IPv6. There is no well-known hostname used for caches, although “cache” and “proxy” would be suitable choices, so add an alias for “cache” to point to Server1 as well; this will help prevent having to touch a lot of clients if we later decide to move the proxy cache.

Screenshot

Remember that names in DNS are generally always lowercase, although DNS is case-insensitive. Reload the DNS zones and test. Take a screenshot of your updated zone files and of your testing.

24.1. File Transfer Protocol (FTP)

Providing FTP services is common, but don’t provide anonymous FTP without being aware of the issues first. As we’ll see later in the course, it may be simple, but it’s not something to be taken lightly.

Depending on the particular FTP server implementation, default access policies will vary: some might allow only anonymous and deny users by default; some might allow users but deny anonymous by default… and I’m pretty sure the software we’re using has switched from one to the other in Ubuntu 10.04 LTS. Beware that FTP is a clear text protocol. There are cryptographic extensions to the FTP protocol (such as ftps)[1], but they are not widespread.

Under Debian-based systems vsftpd (Very Secure FTP Daemon) is often used. Install the package now, on Server1. Verify that the daemon is running, and listening on the FTP port.

Finding packages using apt-cache search

To find packages, you can use **apt-cache search `string`**. For example to search for the package that contains vsftpd, you might search use **apt-cache search vsftp**. In this case, you will find that one of the output lines says `vsftpd - The Very Secure FTP Daemon`.

Edit (as root) the vsFTPd configuration file `/etc/vsftpd.conf`. Make the following policy changes; you’ll have to figure out for yourself which options to change.

[1] Note: sftp is in no way similar to FTP, except as a user interface.

- Allow both users and guests (anonymous) to log in.
- Allow users (not guests) to write (upload) files etc. By default, the ability to write to anything is disabled.
- Set the banner text to “This is the localdomain FTP service.”
- Logs of what has been transferred must be enabled.
- Look at the manual page for vsftpd.conf(5), and find out more about the `dirmessage_enable` option.
- Find out where the anonymous files are to be found on the server. Have a look in `/usr/share/doc/vsftpd/README.Debian`
- When you have made the modifications and answered the questions, restart the FTP service.
- Create the file `/srv/ftp/.message` (note the dot) and put it in some text such as the following.

```
Currently there is nothing here, but if there were, you would
probably find a guide to the layout of the FTP server.
```

  While you are there, create a small file; the name doesn’t matter, you’ll use it for testing the retrieval and display of files.

```
$ sudo sh -c 'echo hello > /srv/ftp/test'
```

Restart vsFTPd, and try connecting to the FTP server from Client1; we’ll first try logging in as the anonymous (guest) user.

```
$ ftp ftp.localdomain
Connected to ftp.localdomain.
220 Welcome to the localdomain FTP service
Name (ftp.localdomain:mal): anonymous
331 Please specify the password.
Password: Your email address
230-Currently there is nothing here, but if there
230-were, you would probably find a guide to the layout.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 0        0             413 May 07 02:02 test
226 Directory send OK.
ftp> get test
local: test remote: test
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for test (6 bytes).
226 File send OK.
413 bytes received in 0.03 secs (12.4 kB/s)
ftp> bye
221 Goodbye.
```

In the above transcript, I was greeted by the FTP service, and I logged in anonomously using my email address as the password. This is just so the administrator can see who has been using the server. Some FTP administrators ban clients that use known-default email addresses, and **vsftpd** can easily support this policy. After logging in, I obtained a directory

listing and retrieved a file. I then disconnected. This is fairly basic stuff, but we haven't tested that local users can log in yet.

```
Make files on client and server for testing
client1$ hostname > ~/file_to_upload.txt
server1$ hostname > ~/file_to_download.txt

client1$ ftp ftp.localdomain
Connected to ftp.localdomain.
220 This is the localdomain FTP service
Name (ftp.localdomain:mal): mal
331 Please specify the password.
Password: mal's password
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 1000     1000            7 May 07 02:33 file_to_download.txt
226 Directory send OK.
ftp> get file_to_download.txt
local: file_to_download.txt remote: file_to_download.txt
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for file_to_download.txt (7 bytes).
226 File send OK.
7 bytes received in 0.02 secs (0.3 kB/s)
ftp> put file_to_upload.txt
local: file_to_upload.txt remote: file_to_upload.txt
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 File receive OK.
8 bytes sent in 0.00 secs (12.5 kB/s)
ftp> bye
221 Goodbye.
```

There are two modes that FTP uses to transfer files. The first is `binary`, the other `ascii`[2]. When it is set to `ascii`, it will translate line-endings sequences, such as `\r\n` to `\n` when transferring from a Windows machine to a Unix system. Binary mode makes no such transformation.

Two other commands of note are `active` and `passive`. Passive (PASV) mode is used when you are behind a firewall, and the incoming data connection from the internet is disallowed. So instead of the server connecting to the client to establish the data connection (the PORT protocol command), the client connects to the server.

24.1.1. Assessment

This lab is optional, but these questions will help you to test your understanding.

1. Describe the default authentication policy that was supplied in the Ubuntu 10.04 LTS version of vsFTPd. Why might a policy that allows only anonymous users (denying authentication of users by default) be preferable from a security point-of-view?

2. You must have a screenshot showing that you have been able to log in as a normal user and download/upload files.

3. Take a screenshot of **wireshark** following the TCP ftp-control stream (or **tcpdump -A ...**) showing at least the authentication process.

[2]There is also ebcdic, which you would only be likely to see on IBM mainframes.

24.1.2. FTP Resources

RFC 2577: FTP Security Considerations
Worth reading to find out how FTP can be abused and what you can do to mitigate it.

24.2. Web Caching with Squid

In this section, we shall look at a tool that can both save bandwidth, account for bandwidth, provide some means of access control to the web and FTP, and if that wasn't enough, it can make the web go faster.

24.2.1. Transparent Proxies

This section on transparent proxies does not constitute lab work, but will be of interest to you to get a better picture of how proxy caches are commonly deployed.

Most caches we use are configured by an administrator, or a user, or autodetected (we'll see how that works later). It is common practice for ISPs and other networks to transparently cache objects. This is more about saving the ISP bandwidth (money), which, one would hope, should make the service more competitive[3]. Since we haven't played with firewalling yet, we won't be touching on transparent proxying in this lab.

One of the things ISPs are starting to do is cache Peer-to-Peer traffic, an early example of which is shown in the paper Deconstructing the Kazaa Network [http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=837393]. There can be some old legislation in various countries that make for interesting legal interactions but these are being addressed. For example, New Zealand explicitly made automated, temporary caching of material legal under the Copyright (New Technologies) Amendment passed in 2008. Details can be found under Section 93E of the Copyright Act [http://legislation.govt.nz/act/public/1994/0143/latest/whole.html], so you can expect this to become increasingly common practice as a mechanism for ISPs to save on traffic costs and provide enhanced service levels.

Transparent proxies work by redirection. Web requests are redirected to the proxy. If the proxy can satisfy that request, it sends it back to the requesting client, with a source IP of the webserver it would have otherwise been coming from. If the request cannot be fulfilled, the proxy gets it on behalf of the client, returning and caching the object as it returns.

The destination server will see the connection request coming from the proxy, so it will consider the proxy as the client. This artefact can be used in a number of ways. As an example of a good purpose, if you're at home, but need to get an academic paper from a journal, for example, but in order to do so you need to be seen as coming from your University, you could use the University's proxy. This is almost identical to the situation with NAT, whereby a number of different machines are seen as one IP address (and presumably, therefore, as one client), so if the server blocks one of those clients, it blocks them all. Proxies can also be used to hide the original client's identity, such as is done when using the Tor proxy system.

You can attempt to find out what proxies are being used (whether transparent or not,) by inspecting the HTTP headers returned from the server. You can use **curl -I** `URI` and look for any `Via` or `X-Cache` headers[4], as these are typically, though not always, added by any intermediaries. Here is an example, using **curl**, which you may need to install. The **-I** option shows only the headers.

[3] Or allow them to take more of our money.
[4] The `Via` header is documented in RFC 2616 [http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.45]

```
$ curl -I http://www.ubuntu.com
HTTP/1.0 200 OK
Date: Mon, 10 Jan 2011 23:43:40 GMT
Server: Apache/2.2.8 (Ubuntu) mod_python/3.3.1 Python/2.5.2 PHP/5.2.4-2ubuntu5.10 …
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Vary: Accept-Encoding
Content-Type: text/html; charset=utf-8
Age: 1
Content-Length: 33890
Connection: close

HTTP/1.0 200 OK
Date: Mon, 07 May 2007 05:44:30 GMT
Server: Apache/2.2.3 (Ubuntu) …
…
Via: 1.1 privet.canonical.com:80 (squid/2.7.STABLE7)  Note: “reverse proxy”
…
X-Cache: HIT from privet.canonical.com
X-Cache-Lookup: HIT from privet.canonical.com:80      Note
…
Connection: close
```

Here we see that even though I have not configured **curl** to use a proxy my connection is being passed through a proxy anyway (by the network, hence the “transparent”). In this particular case, the proxy shown is one belonging to the target site: this is a “reverse proxy”, which is put in front of dynamic web-servers to reduce the amount of work that the web-server scripts need to. We can see that “privet” was able to satisfy this request from its cache, and thus did not have to pass it through to the back-end.

Another common way of encountering transparent proxies is closer to the client: for example you could imagine that all TCP/80 traffic going through Server1 gets automatically forwarded to a transparent proxy. This means clients don’t need to configure a cache, although it does have some limitations (such as no user-authentication).

24.2.2. [Reference Only] Proxy Autodetection

This section on proxy autodetection is for your reference only, because most people would have seen an option for it in their browser, but won’t know much about it. You are not expected to do anything on your machines for this section.

We can autoconfigure web browsers to use a particular proxy using WPAD (Web Proxy Auto-Detection), although support for it in some versions of some browsers can be spotty. WPAD was never a formal standard, its progress seems to never have made it past a draft, but its still used by most browsers, although only to minimal conformance.

Important

WPAD has a number of problems stemming from a poor design. This can be problematic for individual pieces of software that implements WPAD. I suggest having a look at Beau Butler’s talk on Web Proxy Autodetection and your network [http://2008.nznog.org/conferencepapers.html#wpad]. You should be aware of these issues, even if your don’t want to use WPAD.

The idea of WPAD is to find a Configuration URL (CURL) that will result in a HTTP request for a Configuration File (CFILE). The CFILE is the *contents* of the file that is retrieved via the CURL. If the CFILE is invalid, it moves on to the next CURL. If it exhausts the search, it decides to use a direct connection (ie. it doesn’t use a proxy).

Tries DHCP, WINS and DNS

WPAD tries to find a suitable CURL using a variety of means, including DHCP, WINS (used in Windows networking), and DNS lookups for A/CNAME. There are other mechanisms but they are not worth mentioning. The protocols mentioned are commonly used. DHCP should be tried first.

WPAD Configuration via DHCP

The DHCP method involves sending a DHCP INFORM message to a DHCP server (preferably using unicast, if the agent knows the IP address of your DHCP server). This does not involve getting a new IP address. The query is for the DHCP option 252, and should return the CURL, such as `http://server.domain/proxy.pac`.

To configure this behaviour in your DHCP server, you would need the following code in your `dhcpd.conf`. You don't need to do this now, its just a useful reference. It should be noted that although the WPAD Draft standard would prefer DHCP, you're more likely to get more success with the DNS well-known alias method.

```
…
option wpad code 252 = string;

subnet 192.168.1.0 netmask 255.255.255.0 {
    option wpad "http://webserver.example.com/wpad.dat";   CURL to advertise
}
```

DNS Query for Well-Known Alias

The other method that would be tried after DHCP is usually a DNS query[5]. This involves determining the client's fully qualified domain name, and removing the host component. So if the client was bob.engineering.bigco.com, the Target Domain (`target_domain`) would be engineering.bigco.com. It would then do various DNS queries using that `target_domain` as the DNS name to lookup. The most common request would be an A or CNAME lookup for the well-known name, wpad.`target_domain`.

If the search using the `target_domain` fails, it removes the leftmost component, and tries again. It will stop searching when it finds a valid CFILE, or the `target_domain` is one of the official Top Level Domains (TLD) such as com, org, net etc. It should not stop when the `target_domain` is not official, such as localdomain. It would try the following CURLs in this order.

1. http://wpad.engineering.bigco.com

2. http://wpad.bigco.com

3. http://wpad.com, but this would not be looked up, because the `target_domain` is a TLD.

CFILE format

The file that is returned with a particular CURL, should look something like the following. Basically, the file needs to contain the Javascript definition of a single function FindProxyForURL. More information can be found in the Squid documentation, including information on functions you can use to make your autoconfiguration file smarter. Here are

[5]The WPAD specification also uses the Service Location Protocol (SLP), but it isn't widely used.

some examples of the function you might care to define; you could use any *one* of them. You can make a function that is rather more complicated than shown here, that does things like look at the URL or the requesting host to decide which proxy (if any) should be used.

```
EITHER This version would tell the browser not to use a proxy
function FindProxyForURL(url, host) {
    return DIRECT;
}

OR This version would tell the browser to use a cache proxy
function FindProxyForURL(url, host) {
    return "PROXY cache.domain.example:3128";
}

OR The same, but with failover to using no proxy.
function FindProxyForURL(url, host) {
    return "PROXY cache.domain.example:3128; DIRECT";
}

OR Finally, if you have to use SOCKS style proxy
function FindProxyForURL(url, host) {
    return "SOCKS cache.domain.example:3128";
}
```

Client Configuration

To configure a real client, you go into the options, and under Proxies select the option that says samething like “Autodetect Proxy Settings”. You could do this on your Mac using Firefox, as you don’t have permission to change the system default proxy settings which Safari uses. Don’t enter a CURL, although it can be useful, as that defeats the purpose of resource discovery and makes it harder for laptops to stay mobile.

In Firefox, the proxy settings can be found under Edit → Settings Advanced tab, Network sub-tab, Settings… button. However, in environments where it is possible, setting the proxy settings for all programs at once is generally preferable.

24.2.3. Basic Squid Configuration

In this section, you will configure the web proxy with basic settings, and allow everyone to use the proxy. Unless otherwise specified do this section on Server1.

Install the squid package. When it starts for the first time it will create some indexes (on other systems, you may need to initialise these yourself).

The first thing to do is check the configuration file `/etc/squid3/squid.conf`. Make the following changes.

- Ensure that ICP is disabled (`icp_port` should be 0). This disables the Internet Cache Protocol, which is sometimes used when you have a collection of caches. It isn’t useful to us, so we’ll disable it.
- Squid can be monitored using a protocol called SNMP, which we shall see in a later lab. For now, ensure that `snmp_port` is off by setting it to 0, as we have no need for it.
- If you were to create a cache hierarchy, you would insert a `cache_peer` line for every peer you wish to feed off. We don’t need any such lines for our configuration.
- Depending on the amount of disk-space available on the proxy server, you may want to alter the `maximum_object_size` option, although for our purposes, it should be fine. Also, you

should change the `cache_dir` line to set an appropriate cache size. The default is 100MB, which is plenty for our virtual machines, but may be too little for some sites, especially considering disk space is very much cheaper compared to Internet usage.

- Set the `visible_hostname` to `cache.localdomain`.
- Notice the `cache_dir` entry. Squid will keep its cache in `/var/spool/squid3/`[6]. This is something that would be good to keep an eye on to get a feel for how much space your network needs, and to make this a large area when partitioning the disks of a server designed as a proxy.
- The file specified by the `access_log` option is an interesting file to keep an eye on, as this is where accounting information can be sourced. There are many pieces of software available for Squid log analysis, the most popular would probably be Calamaris, but we'll not go into that today. You would, however, need to add some rules for **logrotate**, in order to manage the logs.
- It is good Netiquette to supply a more useful anonymous FTP username than Squid's default. So change `ftp_user` to `proxyuser@localdomain`. Note that in our private environment, this is less useful, but can be more useful on the internet. Some FTP servers will not let you in if you provide a password that is known to be a default password. The email address that is used should normally be aliased to a real person and monitored. Ensure that it also gets filtered for SPAM, just like any other common mail-box name (eg. hostmaster@…, postmaster@… etc.) as these accounts will likely get a higher-than-average amount of spam.
- The email address of the cache administrator will appear in the error messages, and is configured by the `cache_mgr` directive.
- Find the ACL section, and under the line that defines the ACL called `all`, add a line called `clients`, that specifies the IP network address and subnet mask. You can specify multiple networks on the same line. This creates an Access Control List, but does not apply it to anything.

```
…
acl all src 0.0.0.0/0.0.0.0 Matches everything
acl clients src 192.168.1.0/24
…
```

- In the `http_access` area, apply the newly created ACL to allow the clients to access HTTP, by adding in the following line. Squid's default is to otherwise deny to all, so you have to configure it to allow the clients before you can use it.

```
…
#
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
http_access allow clients
…
```

That's all we need to edit for now. Time to restart it:

```
# service squid3 stop
Control returns immediately, but Squid will hang around for 30 seconds more…
# tail -f /var/log/squid/cache.log
# service squid3 start
```

[6] Other systems may keep it under `/var/lib/squid/cache/`

Tip

For simple little reconfigurations, you can use **sudo squid3 -k reconfigure**, which works immediately.

Check your console messages (syslog) to make sure its not complaining. Note that you will get more detailed messages as to what the cache is doing in the log file `/var/log/squid3/cache.log`. Check to see that its listening for connections on TCP/3128.

```
Edited for clarity
# lsof -i | grep squid
COMMAND  USER   TYPE  NODE  NAME
squid3    squid  IPv4  UDP   *:3072            For outgoing DNS
squid3    squid  IPv4  TCP   *:3128  (LISTEN)
```

Now that squid is running, let's test it from the client. Setup Firefox on Client1 to access the proxy, using `cache.localdomain` as the hostname, 3128 as the port, and adding `.localdomain` to the list of domains not to use a proxy for. Visit any website, and look at Squid server logs to ensure that it is accessing the server and working correctly. Try HTTP and HTTPS URIs, and note that for HTTPS services you get a CONNECT request sent to the proxy, not a GET etc.

503 Service Unavailable

If, during the following tests, you get a `503 Service Unavailable` response, it means that the connection from the proxy to the target server, as specified in the URL, is failing. This may be because the web-server software is not running or installed on the web-server, or because you have requested a URL that contains a server name that is not resolvable via DNS, perhaps because of a miss-spelling.

24.2.4. [Optional] User-based Access Control

So far, we've configured minimal access control, allowing only our client IP range access. Squid can do access control using a large number of criteria. Some of the more interesting are as follows.

- Source/Destination IP address
- Source/Destination Domain
- Regular Expression match of requested domain
- Words in the requested URL
- Words in the source or destination domain
- Current day/time
- Protocol (FTP, HTTP, SSL)
- Username (according to the Ident protocol)
- Username/Password pair

- Method (GET, POST, CONNECT, …)

- There are more, see the Squid documentation

We're going to have a closer look at the Username/Password pair, using Basic authentication. Since this could be a large list, we'll put it into a file. Simple Username/Password lookups can be done using a variety of methods. One of the most common is the NCSA lookup, which use a Unix-format passwd file; not neccessarily `/etc/passwd`, but it looks rather similar, though with fewer columns. To configure Squid to use this method of authentication, you need to find the following directives, uncomment and change them to match the following; you should find them all in one block.

```
…
Uncomment the following
auth_param basic program /usr/lib/squid3/basic_ncsa_auth /etc/squid/passwd
auth_param basic children 5 startup=5 idle=1
auth_param basic realm Squid proxy-caching web server
auth_param basic credentialsttl 2 hours
…
```

Create the `passwd` file mentioned in the configuration above, with mode 600, owned by the `proxy` user. We don't need to create local users on the system for the users that will be authenticating to the proxy, we just need to put the usernames and hashed passwords in `/etc/squid/passwd`. Here is how we create and set the permissions appropriately:

```
# touch /etc/squid/passwd
# chmod 600 /etc/squid/passwd
# chown proxy /etc/squid/passwd
```

Squid can use other Username/Password mechanisms, such as local Unix users (`getpwnam`), LDAP, NIS, NTLM (Windows Domain authentication), and PAM; lots of acronyms, but don't worry about that, they just represent different ways that authentication can happen in different environments.

Create an entry for yourself in the file `/etc/squid/passwd`, using the method shown below to create the password hash.

```
htpasswd is actually part of Apache
# htpasswd -d /etc/squid/passwd mal
New password: Not echoed
Re-type new password: Not echoed
Adding password for user mal
```

You may need to install the `apache2-utils` to get the **htpasswd** command.

Edit `squid.conf` so it denies people with a bad username/password pair.

```
…
#Recommended minimum configuration:
acl all src 0.0.0.0/0.0.0.0
acl clients src 192.168.1.0/24
acl password proxy_auth REQUIRED
…
# search for the line below to find where to put the following…
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
#
#http_access allow clients  comment out this line
http_access allow password
…
```

Notice that we commented out the ACL option `http_access` that allows in known clients. Reconfigure Squid:

```
# squid -k reconfigure
```

Checking the logs to check that it isn't complaining about anything.

24.2.5. Assessment

1. Give at least one description of how WPAD could be used as an attack vector. You may need to do some research.

2. **[Optional]** Show that you can get denied when you supply an incorrect password, and that you can also get access when you supply a correct username/password.

3. **[Optional]** What would have been the effect if we didn't comment out the ACL option `http_access` that allows access to known clients? Give a scenario where this may be useful.

24.2.6. Resources

Cacheability Query [http://www.ircache.net/cgi-bin/cacheability.py]
Check how cacheable your website is.

Deconstructing the Kazaa Network [http://people.cs.uchicago.edu/~matei/PAPERS/kazaa.pdf]
Look at how transparent proxies can be used for non-HTTP traffic.

WPAD Draft RFC (Expired) [http://www.wpad.com/draft-ietf-wrec-wpad-01.txt]
The full details on how web proxy autodetection was supposed to be performed (this in an expired RFC, so it has little weight).

Upside-Down-Ternet [http://www.ex-parrot.com/pete/upside-down-ternet.html]
A humourous example of how proxies can be used to modify content. A more practical example would be recompressing large images that are to be sent to mobile devices.

Lab 25 [Optional] Simple Network Management Protocol (SNMP)

> **Old Lab**
>
> This lab hasn't been run recently, so no guarantees are made regarding the commands and their output.

In this lab, we shall use our Server1 and Client1 machines to explore the use of the Simple Network Management Protocol (SNMP) for monitoring network devices such as routers and hosts. We shall explore concepts such as how SNMP works, its conceptual model, and how the data is given meaning using Management Information Bases (MIBs). We shall briefly consider some different tools for processing the data.

We shall also enable SNMP monitoring of a server using the Net-SNMP package. This allows us to monitor a host, rather than just network devices. Finally, we shall extend an extensible SNMP *agent* so we can publish information of our choosing using SNMP.

Please do the first section of the assessment before coming to the lab. This is to ensure you review the conceptual model and architecture that you learned during the corresponding lecture. Once you have done this, please proceed with the rest of the lab.

25.1. Reference only: Configuring SNMP Agent on Cisco and Vyatta

You don't need to do this for this lab, as you don't have a Vyatta router in your network with Server1 and Client1. Therefore, it is for reference only.

Here are the basic commands you would use to configure an SNMP agent on Cisco IOS and Vyatta. These are for reference only; compare them with the following section, but you are not expected to perform anything for this section in the lab. First, here are some basic commands for Cisco devices, just to get you started, which would be done in configuration mode. For production use, you would also want to consider restricting access using ACLs (where queries can come from) and views (what parts of the tree can be seen, depending on the submitted community string).

```
snmp-server community COMMUNITY ro|rw
snmp-server location LOCATION
snmp-server contact CONTACT
```

And here are the commands for Vyatta, also to be done in configuration mode.

```
set service snmp community COMMUNITY authorization ro|rw
set service snmp location LOCATION
set service snmp contact CONTACT
```

25.2. Install the Net-SNMP Agent

We install and configure the Net-SNMP agent software on our Linux-based gateway.

SNMP agents are commonly already available on router products and managed switches, but you can also get them for hosts as well a number of other embedded devices on the network, such as for monitoring the health of battery backup systems, air-conditioning plants etc. One commonly used SNMP agent is Net-SNMP, which is quite comprehensive and extensible. In this section, we install and configure **snmpd** for basic use, including some of the host-resources MIBs. We shall practice querying it in the subsequent sections.

Install the Net-SNMP agent (server) software on Server1, which is available in the package **snmpd** on Debian-based machines. Also, install the **snmp** package, which contains the manager software (clients) on both Server1 and Client1.

Because SNMP can be used to monitor all sorts of devices, we need to map from the OIDs (we'll see these later on) into human readable names. Install the `snmp-mibs-downloader` package, and comment out the `mibs` line in `/etc/snmp/snmp.conf`.

You should now find two other configuration files in `/etc/snmp/`: `snmpd.conf` and `snmptrapd.conf`. Ignore the last one, we won't be making use of any traps for this lab. The first file, `snmpd.conf` is the one we want to be looking at further as it controls the agent (server). On some systems you might also find `snmp.conf`, which lists defaults for the clients; it does not exist by default.

Net-SNMP actually comes with a configuration program called **snmpconf** to get you started, but it doesn't add much value unless you're doing it yourself; the configuration file is fairly simple. As root, open the `snmpd.conf` file in an editor and configure with the following (I have ommited comments to save space):

```
agentaddress udp:161,udp6:161,tcp:161,tcp6:161
rocommunity public
rocommunity6 public
proc  named         1 1
proc  squid
proc  sshd          5 1
disk  /     10%
load  5 3 2
sysLocation  "Rm1.23, FOO Building"
sysContact  "Your e-mail <admin@example.com>"
```

So what is happening in this configuration? First we have configured all the various ways I wish to be able to access this agent, over both IPv4 and IPv6 on all supported addresses, and both UDP (standard) and TCP (Net-SNMP extension). I like having TCP available because it means I can query it easily using SSH tunnelling remotely.

I have configured a read-only string called "public", for both IPv4 and IPv6. I could have configured additional community strings, optionally limiting what each community string can see, but I have opted not to. Note that no read-write access has been configured, and no SNMPv3 access has been configured.

If we just skip down to 'syslocation' and 'syscontact', then we complete the basic SNMP configuration. This leaves us with the 'proc', 'disk' and 'load' lines, which configures support in Net-SNMP for Process Monitoring, Disk Usage Monitoring and Load Monitoring.

'proc' entries are for monitoring various configured processes. Here we see there must be exactly one process of **named** running, at least one **squid** process should be running, and between one and five **sshd** processes should be running. If these limits are exceeded, then the 'prTable', which is where this information will be reported, will show an error state.

'disk' lines are for monitoring file-system utilisation. Here we only have one, but you could have one for every filesystem, particularly for filesystems that are often written to, such as `/var/` and `/tmp/`. We have specified a minimum percentage of the filesystem that must be available, if not an error flag will be seen in the 'diskTable'.

Finally, the 'load' entry provides a way of monitoring system load, which is one of the more useful things to monitor via SNMP for a host.

Change `/etc/defaults/snmpd` to be as the following. The only change should be to the SNMPDOPTS line, which reduces the amount of logging that happens, which is useful when graphing, and also removing the default behaviour of only being accessible from localhost[1]. I have removed comments to save space. Note that the documentation for **snmpd** disagreed with what it actually accepts, which was most annoying.

```
export MIBDIRS=/usr/share/snmp/mibs
SNMPDRUN=yes
SNMPDOPTS='-LS 4d -Lf /dev/null -u snmp -g snmp -I -smux -p /var/run/snmpd.pid'
  DELETE "127.0.0.1" at the end of SNMPDOPTS
TRAPDRUN=no
TRAPDOPTS='-Lsd -p /var/run/snmptrapd.pid'
SNMPDCOMPAT=yes
```

Now you can restart the software.

```
# /etc/init.d/snmpd restart
```

Ensure that it is running and check the logs. I found a couple of messages about getaddrinfo, but otherwise it seems to be okay.

```
$ ps -eo command | grep snmpd
/usr/sbin/snmpd -LS 4d -Lf /dev/null -u snmp -g snmp -I -smux -p /var/run/snmpd.pid
# tail /var/log/syslog
```

Finally, ensure that **snmpd** is listening on the expected interfaces.

```
# lsof -Pni | grep snmpd
snmpd  1821  snmp  7u   IPv4  3641153  UDP  *:161
snmpd  1821  snmp  9u   IPv6  3641158  UDP  *:161
snmpd  1821  snmp  10u  IPv4  3641159  TCP  *:161  (LISTEN)
snmpd  1821  snmp  11u  IPv6  3641161  TCP  *:161  (LISTEN)
```

Great, we should be ready to practice querying it. Before we do, let's just run a very simple query to ensure that it's working.

```
$ snmpget -v2c -c public localhost sysContact.0
Timeout: No Response from localhost.
```

It failed. Looking at the logs, we see that it has reported a "Connection Refused" event, which indicates that TCP Wrappers is likely being used. Edit `/etc/hosts.allow` and create

[1] Configurations on the command-line (which this file is for) override configurations in `snmpd.conf`.

a suitable entry (based on the other entries in that file) for the `snmpd` service. Restart the SNMP service and try the command again:

```
$ snmpget -v2c -c public localhost sysContact.0
SNMPv2-MIB::sysContact.0 = STRING: "Your e-mail <admin@example.com>"
```

If you got that, then you are ready to proceed. Otherwise, check your logs. Timeouts can indicate either that nothing is running on that interface or port, the port is blocked (firewall or other access-control-mechanism on the agent), or the community string is wrong.

25.3. Command-Line Clients from Net-SNMP

In this section, we shall exercise the use of various Net-SNMP command-line tools and briefly investigate how we can configure some defaults to make them easier to use. This section should build your confidence in the use of these tools and make you aware of the sort of programs which are available and when you might want to use them.

All of the Net-SNMP command-line managers share many common arguments, which are documented in snmpcmd(1). Two of the most common options are to specify the SNMP version and community-string, which we briefly saw at the end of the previous section.

`-v 2c`
: Specifies SNMP version 2c (“SNMPv2”)

`-c public`
: Specifies the community string to use.

These are very common, so we can put them in `/etc/snmp/snmp.conf` or `~/.snmp/snmp.conf`. Here is what I have configured in mine, make your version the same.

```
defVersion 2c
defCommunity "public"
```

Now, let’s try the same command we used previously, but this time we don’t need to specify quite as much on the command-line.

```
$ snmpget localhost sysContact.0
SNMPv2-MIB::sysContact.0 = STRING: "Your e-mail <admin@example.com>"
```

25.3.1. snmptranslate

There are numerous ways of specifying the same OID, some more convenient as others. We often wish to input or output these in a particular format, and the **snmptranslate** command allows us to do just that. We shall just cover the basics with some common examples.

All of the options pertaining to the (user) input of OIDs begin with an ‘I’, but the only really useful one is `-IR`: this enables “random access” to the SMI tree, so you don’t have to specify it relative to any fixed point. This is very useful for casual use as it saves a lot of typing. You’ll probably want to use it very often. It’s not on by default because there is potential to be ambiguous.

```
$ snmptranslate ifInOctets
ifInOctets: Unknown Object Identifier (Sub-id not found: (top) -> ifInOctets)
```

```
$ snmptranslate .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets
IF-MIB::ifInOctets
$ snmptranslate -IR ifInOctets
IF-MIB::ifInOctets
```

Here are some alternative ways of specifying the same OID in an unambiguous manner:

IF-MIB::ifInOctets
: Relative to a known MIB module name. This is the most convenient way of specifying on OID while generally not being ambiguous.

.1.3.6.1.2.1.2.2.1.10
: Fully qualified numeric.

.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets
: Fully qualified symbolic (or mixed).

Of slightly less importance is output format, of which the arguments begin with 'O' (Capital 'o' for 'output', not zero). Compare the output from these commands, beginning with the default output:

```
$ snmptranslate -IR ifInOctets
IF-MIB::ifInOctets
$ snmptranslate -IR -Of ifInOctets    output fully qualified format
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets
$ snmptranslate -IR -Os ifInOctets    output short, unqualified format
ifInOctets
$ snmptranslate -IR -OS ifInOctets    output short format with containing MIB
IF-MIB::ifInOctets
$ snmptranslate -IR -On ifInOctets    output numeric, fully-qualified format
.1.3.6.1.2.1.2.2.1.10
```

snmptranslate can be a useful front-end for scripts and tools that require, for example, a numeric OID, but you want to enable the user to provide the OID in a more friendly manner.

There are many more options available, see the snmpcmd(1) for further details.

Finally, there are some arguments that are specific to **snmptranslate**. One argument is useful for printing part of the MIB tree; another is useful for viewing documentation about a particular OID; there are others, but these are the only two demonstrated below.

```
$ snmptranslate -Tp -IR prTable
+--prTable(2)
   |
   +--prEntry(1)
      |  Index: prIndex
      |
      +-- -R-- Integer32 prIndex(1)
      |        Range: 0..65535
      +-- -R-- String    prNames(2)
      |        Textual Convention: DisplayString
      |        Size: 0..255
      +-- -R-- Integer32 prMin(3)
      +-- -R-- Integer32 prMax(4)
      +-- -R-- Integer32 prCount(5)
      +-- -R-- EnumVal   prErrorFlag(100)
      |        Textual Convention: UCDErrorFlag
      |        Values: noError(0), error(1)
      +-- -R-- String    prErrMessage(101)
      |        Textual Convention: DisplayString
      |        Size: 0..255
      +-- -RW- EnumVal   prErrFix(102)
```

```
      |        Textual Convention: UCDErrorFix
      |        Values: noError(0), runFix(1)
      +-- -R-- String    prErrFixCmd(103)
               Textual Convention: DisplayString
               Size: 0..255
$ snmptranslate -Td -IR prTable
UCD-SNMP-MIB::prTable
prTable OBJECT-TYPE
  -- FROM       UCD-SNMP-MIB
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "A table containing information on running
         programs/daemons configured for monitoring in the
         snmpd.conf file of the agent.  Processes violating the
         number of running processes required by the agent's
         configuration file are flagged with numerical and
         textual errors."
::= { iso(1) org(3) dod(6) internet(1) private(4)
      enterprises(1) ucdavis(2021) 2 }
```

25.3.2. snmpget and snmpgetnext

One of the essential skills is knowing how to get data using SNMP. For this, you need to know how to specify the agent. Net-SNMP allows you to connect to the agent using a variety of different mechanisms, such as UDP/IPv4, UDP/IPv6, TCP/IPv4, TCP/IPv6, ATM, IPX or Unix sockets; don't worry if some of these are unfamiliar to you, the important one is UDP/IPv4 and UDP/IPv6, although the TCP ones can also be useful.

The full details can be found in snmpcmd(1), but here are some examples from that manual page to help you quickly get the idea of what you can say. Remember that SNMP is typically UDP port 161, so when using IP (any version), all you need is a hostname, and it will talk to that host on UDP port 161.

`hostname`:161
: Perform query using UDP/IPv4 datagrams to `hostname` on port 161. The `:161` is redundant here since that is the default SNMP port.

udp:`hostname`
: Identical to the previous specification. The `udp:` is redundant here since UDP/IPv4 is the default transport.

TCP:`hostname`:1161
: Connect to `hostname` on port 1161 using TCP/IPv4 and perform the query over that connection.

udp6:`hostname`:10161
: Perform the query using UDP/IPv6 datagrams to port 10161 on `hostname` (which will be looked up as an AAAA record).

UDP6:[fe80::2d0:b7ff:fe21:c6c0]
: Perform the query using UDP/IPv6 datagrams to port 161 at address fe80::2d0:b7ff:fe21:c6c0.

tcpipv6:[::1]:1611
: Connect to port 1611 on the local host (::1 in IPv6 parlance) using TCP/IPv6 and perform the query over that connection.

  Note that `tcpipv6` is the same as `tcp6`.

Screenshot

Practice querying the agent over each of UDP/IPv4, UDP/IPv6, TCP/IPv4 and TCP/IPv6. Take a screenshot to show you have done each step. The command will look something like the following:

```
$ snmpgetnext agent sysContact
```

1. Install the Net-SNMP manager software (the 'snmp' package) onto Client1.

2. Practice querying the agent remotely using at least UDP/IPv6 and UDP/IPv6.

3. Query the agent remotely over an SSH 'local' tunnel using TCP port forwarding.

Later during the assessment, you need to be able to explain the difference between **snmpget** and **snmpgetnext**, and why an OID specified as 'sysContact' won't work for snmpget.

25.3.3. snmpwalk and snmpbulkwalk

Walking a tree in SNMP version 1 is done using a series of GetNext and GetResponse packets, which can take a long to complete for a large tree, even on a fast network. The problem is that we have too many network round-trips to complete. It would be better to ask the agent (server) for the tree under a particular OID, and have it return a bulk set of answers, vastly cutting down the number of round-trips to the server. Version 2c introduced such bulk operations, which has a major impact on even very low-delay communications.

We can measure the time taken to complete a command using the **time** command. Because the cost of writing output to the terminal is very expensive, I have redirected output to `/dev/null` so the cost of writing to the terminal is minimised.

```
$ time snmpbulkwalk localhost . >/dev/null
real    0m1.134s
…
$ time snmpwalk localhost . >/dev/null
real    0m2.428s
…
```

This was testing just on localhost, which is typically a *very* low-delay way of talking to yourself. Imagine how much more time you would save if there was a real network between yourselves, with much greater delay.

snmpwalk and **snmpbulkwalk** are basically equivalent in the arguments you give them:

```
$ snmpwalk localhost ifDesc
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: outside
IF-MIB::ifDescr.3 = STRING: inside
$ snmpbulkwalk localhost ifDesc
… you get the same result
```

25.3.4. snmptable

Many of the interesting items available via SNMP are logically represented as a table. Some examples are the interface table 'ifTable'; the IP address table 'ipAddrTable', routing table 'ipRouteTable', and ARP[2] table 'ipNetToMediaTable'; the TCP connection table 'tcpConnTable' and UDP table 'udpTable'.

[2]Well, network-to-datalink mapping.

The UCD-SNMP MIB (which is where the Net-SNMP-specific additions are found) has other tables which we have previously configured, have a look at the 'dskTable', 'laTable', and 'prTable'.

Other devices may support further tables, such as NAT translations for routers, CAM tables on Ethernet switches (to see which MAC addresses lie on which switch ports).

A number of these tables can get very wide and wrap awkwardly in a terminal, so I suggest you pipe the output to **less -S**, which allows you to scroll side-to-side.

```
$ snmptable localhost ifTable | less -S
… remember to press 'q' to quit …
```

Screenshot
Take a screenshot showing one of the tables.

25.3.5. Other commands…

There are numerous other commands available. We can group them into several groups: those that deal with setting values via SNMP; those that deal with SNMPv3; those that deal with sending asynchronous traps and informs; those that are SNMP versions of well-known commands; and others. You can get a descriptive list of all the snmp* commands using the following command:

```
$ for prog in $(cd /usr/bin; echo snmp*); do
>   whatis $prog
> done
```

25.4. Management Information Bases (MIBs)

In this section, we shall look at how numeric data are translated to and from names, and how the available data are documented. We shall look at how MIBs can be added to the Net-SNMP manager software. This knowledge is important when making use of data under the 'Enterprises' tree.

MIBs are specified in SMI, which is a subset of the ASN.1 standard. Network Management Stations (NMSs) make use of the data in MIBs (once they have been compiled in some way for fast access) to do things such as document the SMI tree and map human-readable OIDs, such as ifInOctets.1 to their numeric equivalent; .1.3.6.1.2.1.2.2.1.10.1 in this case.

Each NMS will often come with a collection of MIBs, which is stored in a software-specific location.

Often, you need to add some vendor-specific MIBs to monitor some of your devices; some agents will support vendor and/or model-specific information that is not in a well-known MIB, so you may need to search the vendor's website for any MIB information. For example, the Apple Extreme range of WiFi access points have an MIB available. How and where you install these MIBs will depend on the particular NMS software you are using, but often they will be contained in a directory called `mibs`, in files with either a `*.txt` or `*.mib` extension.

Under Debian-based distributions, the MIBs should be found in `/usr/share/snmp/mibs/`. Go there and have a look at some of the files. Depending on what software is installed on the

system, you may find other MIBs there as well: for example, if you have the Squid proxy-cache installed, you might find a MIB for Squid, which is useful for monitoring your Squid proxy-cache.

Have a look at `UDP-MIB.txt` — UDP is reasonably simple, though still has plenty of things we could track using SNMP. Here is an extract from that file:

```
-- the UDP group

udp     OBJECT IDENTIFIER ::= { mib-2 7 }

udpInDatagrams OBJECT-TYPE
   SYNTAX     Counter32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
         "The total number of UDP datagrams delivered to UDP
         users.

         Discontinuities in the value of this counter can occur
         at re-initialization of the management system, and at
         other times as indicated by discontinuities in the
         value of sysUpTime."
   ::= { udp 1 }
```

Note that the line beginning with ‘udp’ is defined as (‘::=’) child number 7 under the OID known as ‘mib-2’. The OID ‘mib-2’ is imported earlier in the MIB file and is defined as ‘.1.3.6.1.2.1’ in the MIB file `SNMPv2-SMI.mib`.

The bulk of the lines in the file are for the description, which can be displayed to the user of the NMS.

‘udpInDatagrams’ is being defined as a 32-bit counter that is read-only and is not deprecated; it’s OID will be ‘udp.1’, equivalently ‘mib-2.udp.1’ or ‘.1.3.6.1.2.1.7.1’.

However, if we were to simply ‘GET’ that OID, we would find nothing there, because each object can have a number of instances; for scalar values such as ‘udpInDatagrams’ we need to use the 0th index, so we could look up ‘udpInDatagrams.0’.

Screenshot

Do this: use **snmpget** to get ‘udpInDatagrams’; what do you see? Now try getting ‘udpInDatagrams.0’; what do you see now? What if you use **snmpwalk** to get ‘udpInDatagrams’; what do you see? Take a screenshot showing what you have seen.

A number of the more interesting things available in the SMI tree are tables, these are defined as ‘SEQUENCE OF’ other things; the instance here becomes the row we want to address. Using again the example of UDP, can you find an example of a table? If the values are all zeros, use a different table — the agent doesn’t support all of the MIB.

Do this: use **snmptable** to look at the table you found. When using **snmptable** the output is generally very wide, I suggest piping it to **less -S**. You can compare the results with viewing the table of ‘ifTable’. Repeat using **snmpwalk** to see how the table relates the tree representation.

Now, on Client1, start up the program **tkmib** (if not installed, install it via the package of the same name). **tkmib** is from the same people as Net-SNMP, though can be quite awkward to use (and on smaller screens pratically useless, so make sure you have a nice, large screen).

It is worth noting that most larger NMSs are commercial and cost quite a lot; these are relatively quite simple. One nice free MIB browser for Windows is from iReasoning.

Screenshot

Practice how to perform the following, taking a screenshot to show your results for each:

- Getting the value of 'sysName.0'. You will need to add the .0 to the OID.
- Walking the 'system' sub-tree.
- Viewing the description of ipInDiscards. Why might you be interested in graphing this value?

  You're also shown that the status is Deprecated. Reading the text you find that it has been replaced by the IP version-neutral ipSystemStatsInDiscards (which you will find inside ipTrafficStats). However, you will probably find that vendors support may be slow in arriving, so it really pays to test.

  Note that graphing support in **tkmib** is not very usable in this version, so we suggest you don't try graphing with **tkmib**.

Finally, you may recall that we have set up a Squid proxy-cache, which allows querying statistics over SNMP. First, here is what you would need to edit in `/etc/squid/squid.conf` to have Squid support statistics over SNMP.

```
…  Search in the file for where to put these
snmp_port 3401
…
snmp_access allow localhost
…
```

Here is an example of walking Squid's embedded SNMP agent.

```
$ snmpwalk -m +SQUID-MIB localhost:3401 SQUID-MIB::squid
SQUID-MIB::cacheSysVMsize.0 = INTEGER: 104
SQUID-MIB::cacheSysStorage.0 = INTEGER: 11096
SQUID-MIB::cacheUptime.0 = Timeticks: (19894) 0:03:18.94
SQUID-MIB::cacheAdmin.0 = STRING: webmaster@localdomain
SQUID-MIB::cacheSoftware.0 = STRING: squid
SQUID-MIB::cacheVersionId.0 = STRING: "2.7.STABLE7"
…
SQUID-MIB::cacheRequestHitRatio.1 = INTEGER: 0     1 minute average
SQUID-MIB::cacheRequestHitRatio.5 = INTEGER: 0     5 minute average
SQUID-MIB::cacheRequestHitRatio.60 = INTEGER: 0    60 minute average
SQUID-MIB::cacheRequestByteRatio.1 = INTEGER: 0
SQUID-MIB::cacheRequestByteRatio.5 = INTEGER: 0
SQUID-MIB::cacheRequestByteRatio.60 = INTEGER: 0
…
End of MIB
```

Looking at the MIB (`/usr/share/snmp/mibs/SQUID.txt`), perhaps the most immediately useful things to look at is the cacheRequestHitRatio and cacheRequestByteRatio), as these give you a good indication of how effective the cache is at saving you traffic.

25.5. Long-Term Graph Trending

In this section, you will intrepret some long-term graph data captured from real-life situations. The practice in this section is in intrepreting the graphs, writing down everything you can observe from it.

Much of the value of long-term graphing is in seeing behaviour over periods such as a day, week, month and year. Daily patterns are often fairly easy to spot this way. The yearly graphs often show some useful trends. The website MRTGHelp.com [http://www.mrtghelp.com/examplemrtg.htm] has a page of useful examples of things you can see in MRTG graphs; have a look at them so you can recognise particular events. Note that MRTG can also be used to graph things like system load, mail-server throughput, and a host of others.

25.6. Trap Notifications

In this section, we enable the receipt of trap messages, which can help us to understand problems that our network elements may want to tell us about.

Network elements, such as wireless access points, have no user interface on which to print messages that might be useful for diagnosing problems — such as why the access-point might need rebooting — although it's not going to be able to give you much detailed information. Most of the time, the interesting trap messages will be either about authentication problems or interfaces going up or down. Such devices can and do often also have the capability to log to a remote syslog server as well.

Trap messages often end up either getting logged to somewhere (such as being forwarded to syslog), or conditionally being actioned (such as paging an administrator).

To set up the Net-SNMP trap receiver, edit `/etc/default/snmpd` and change `TRAPDRUN` to `yes`. However, if this is the only thing you do, you will see this in your system logs.

```
... snmptrapd[...]: Warning: no access control information configured.
  This receiver will *NOT* accept any incoming notifications.
```

So clearly there is more work to do; we at least need to configure the community string for traps. See the manual page snmptrapd.conf(5) for the specifications of what can go into `snmptrapd.conf`. Have a brief read of that document to ensure you understand what the following configuration does, and put this configuration into `snmptrapd.conf`.

```
snmpTrapdAddr udp:0.0.0.0:162,udp6:[::]:162
disableAuthorization yes   This is a potential security problem
doNotRetainNotificationLogs yes
doNotLogTraps no
doNotFork no
authCommunity log,execute,net public
```

Restart the SNMP services using the init-script, and check the logs for any errors. Check that **snmpd** and **snmptrapd** are listening:

```
# sudo lsof -Pni | grep snmp
snmpd     31823   snmp   7u  IPv4 3810858  UDP *:161
snmpd     31823   snmp   9u  IPv6 3810863  UDP *:161
snmpd     31823   snmp  10u  IPv4 3810864  TCP *:161 (LISTEN)
snmpd     31823   snmp  11u  IPv6 3810866  TCP *:161 (LISTEN)
snmptrapd 31826   root   8u  IPv4 3810878  UDP *:162
snmptrapd 31826   root   9u  IPv6 3810882  UDP *:162
```

Okay, so now we have our trap receiver running we should now test it by sending a trap message. We could generate a trap message ourselves using **snmptrap**, but that gets complicated and confusing; typically we would be using something that can generate its own traps. Agents typically are able to generate traps for authentication failures; agents can also generate traps for when an interface changes operational state (goes up or down). Net-SNMP can also monitor the various tables it supports, such as the process table, and send a trap whenever something is in an error state, which could be useful. However, that requires the use of SNMPv3 internally, which I'm avoiding for this lab, so we shall simply use the authentication failure traps as an example.

So, configure **snmpd** with the following section in its configuration file `snmpd.conf` and restart it. Here I have configured trap messages to simply be sent to localhost; typically, you would configure a manager that is on an element to send to some remote NMS which is operating a trap receiver.

```
###############################################
# SECTION: Traps and Notifications
#

trapsink       localhost   public

authtrapenable  1
```

Because **snmptrapd** also uses TCP Wrappers, you will need to add a suitable line to `hosts.allow`; the service name is, as you should expect by now, `snmptrapd`.

Testing is easy, simply try to retrieve something using the wrong community string. Remember that `-c` specifies which community string to use. We're deliberate using the `wrong` community string, which should generate an authentication trap:

```
$ snmpget -c wrong localhost SNMPv2-MIB::sysUpTime.0
… should eventually timeout, probably resending …
```

Now check your logs, you should see messages similar to the following, which I have reformatted to flow nicely onto multiple lines:

```
Jan 8 15:54:00 server1 snmptrapd[21918]: 2015-01-08 15:54:00 ↩
    server-ip-address(via UDP: [127.0.0.1]:39352->[127.0.0.1]:162) ↩
    TRAP, SNMP v1, community public ↩
    #012#011.iso.3.6.1.4.1.8072.3.2.10 Authentication Failure Trap (0) ↩
    Uptime: 0:00:23.44#012
```

When you can, I suggest you configure an SNMP trap handler in a real network and get any network elements such as routers and wireless access points, to send any notications to it.

We've only touched on traps in this brief introduction. Our trap-receiver can also run handlers when particular traps arrive; a handler may do something like send an e-mail or page, which is much faster than waiting for any log-processing to happen periodically.

25.7. Optional: Installing the Microsoft Windows SNMP Agent

Windows comes with an optional component which allows you to query information about the host using SNMP. This section shows you how you can install and configure the required components. By the end of this optional section, you should have gained some small amount

of Windows service administrative ability and have gained experience with another piece of agent software.

The instructions for this procedure can be found in Microsoft's Knowledge Base article Q324263 [http://support.microsoft.com/kb/324263].

25.8. Self-Assessment

This is an optional lab, therefore there are no marks available for this lab. However, it would be useful to check your knowledge, as having done this lab will cement your knowledge about the topic, which can appear in the exam.

1. Briefly describe the following terms as they relate to the SNMP conceptual model and architecture: 'agent', 'network element', 'manager', 'network management station', 'community string', 'SMI tree', 'OID', 'MIB' and 'trap'.

Describe how the process of authentication is handled in SNMP versions 1 and 2c.

Describe the SNMP messages used to walk a part of the tree for both SNMP versions 1 and 2c.

2. What is the difference between **snmpget** and **snmpgetnext**?

Why would an OID specified as 'sysContact' *not* work with **snmpget**? What would be needed instead?

Does this mean we should use **snmpgetnext** instead of **snmpget**?

3. Ensure you have taken screenshots of using an MIB browser to do the following operations:

- Getting the value of 'sysName'.
- Walking the 'system' sub-tree.
- Viewing the description of 'ipInDiscards'. Why might you be interested in graphing this value?

4. When would short-term graphing be more useful compared to long-term graphing?

5. The teaching staff should have pointed you to some real-life MRTG graphs which may (or may not) show some useful information.

Look at the daily, weekly, monthly and yearly graphs provided, and write down as many things as you can reasonably infer from them.

25.9. Final Words

There are a number of useful things we have not looked at in this lab. The first is SNMPv3, which changes the authentication model to something much more secure. The second is extending the Net-SNMP agent to enable monitoring of things not otherwise supported by the agent. The Net-SNMP web-site has a number of tutorials, such as for SNMPv3 Options [http://www.net-snmp.org/wiki/index.php/TUT:SNMPv3_Options] and a useful page with examples [http://www.net-snmp.org/wiki/index.php/Net-snmp_extensions] of the various extension mechanisms.

Finally, SNMP is only one part of a network monitoring solution. Take a look at a more full-blown solution such as OpenNMS or Nagios.

Lab 26 [Optional] IPv6 Firewalls

Old Lab

This lab hasn't been run recently, so no guarantees are made regarding the commands and their output.

The format of this lab is more like a self-guided tutorial. Part of the difficulty of learning about firewalls is now how to implement the rules, which we have done in the previous firewall lab, but rather figuring out what sorts of attacks you need to defeat.

The basic attacks don’t really change much between IPv4 and IPv6, so this tutorial applies to both. IPv6 does have some extra cautions, mostly because of tunnelling, but that is also present in IPv4 as well.

One of the things that is often a target for firewalling is ICMP and ICMPv6. However, care needs to be taken, especially with ICMPv6, as you can easily break communication when filtering the Internetwork *Control* Message Protocol (for v4 or v6). So some of this tutorial will be spent looking at the suggested best practices in this regard.

By the end of this tutorial, you should have a good idea of some of the types of attacks experienced today, and what sorts of ICMP and ICMPv6 traffic should be able to make it through your firewalls. You should also be able to develop an understanding of common limitations in firewalls.

1. Research and briefly describe the following sets of terms (the list is certainly not exhaustive). Where multiple terms are listed, you should be able to explain what is similar about the terms in each group, as well as be able to explain what is different.

1. Ingress filtering, Egress filtering

2. IP Spoofing

3. Directed broadcast, Backscatter

4. Smurf attack, Fraggle attack

5. BotNet

6. Denial of Service (DoS), Distributed Denial of Service (DDoS)

7. SYN flood, Ping of death

8. Source routing (IPv4 and IPv6)

9. ARP Poisoning, (IPv6) Neighbour Discovery attacks.

10. Connection hijacking, session hijacking.

11. Deep Packet Inspection (DPI) aka Layer 7 filtering, [Network] Intrustion Prevention System (IPS).

12. 6-in-4 tunnel aka Protocol 41 tunnel, 6to4, Toredo, IPSec ESP, VPN, Covert tunnel.

2. Summarise RFC 2827 “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing”, briefly describing how ICMP should be handled by a firewall.

(To make it easier on yourself, you do not need to consider the issue of multihomed networks).

3. Summarise RFC 4890 “Recommendations for Filtering ICMPv6 Messages in Firewalls”, briefly describing what should be allowed through a firewall.

Lab 27 [Optional] Network Visibility with NetFlow

> **Old Lab**
>
> This lab hasn't been run recently, so no guarantees are made regarding the commands and their output.

In this lab, we shall investigate network visibility using the NetFlow architecture. There are many tools available for NetFlow, so we shall first review the NetFlow architecture and then perform some requirements analysis to see which set of tools will be most suitable for us, looking briefly at some of the tools available to us.

27.1. NetFlow Architecture

NetFlow is conceptually very simple: devices such as routers or switches have some *probe* configured which looks at the packets going through its configured interfaces. These observed packets are collected into *flows*. A flow, in the case of TCP, is all the packets that make up a particular connection; for UDP, a flow is everything with the same source and destination address and port. This description of a flow is a simplification, but is intuitively what is meant.

Periodically, such as when a flow is completed (eg. a TCP connection has been torn down) or after some period of time, the flow is exported to a configured *collector*. This is where the NetFlow protocol is used, which we shall discuss in more detail in a following section.

The collector process will generally run on a powerful machine with a lot of available disk space. The collector will often simply just write the exported data to disk, perhaps filtering to capture only certain data, and typically performing maintenance such as rotating files and expiring (deleting) old capture files. You may need to have multiple collectors in order to process all the data exported from all of your probes, so you can end up with some relatively more complex architectures to ensure you can process the incoming data fast enough. NetFlow is a UDP protocol, so one of the things a NetFlow administrator will want to do is monitor how many UDP packets are being dropped by the collector[1]; this can be in indication that the collector host is either not fast enough or that the collector process is not given a high enough scheduling priority.

The data stored by the collector(s) can be used for various purposes, such as accounting, traffic analysis and diagnosis and for investigating security incidents such as Denial of Service (DoS) attacks. We shall look further at some of these later in this lab.

27.2. NetFlow Versions

There have been numerous versions of NetFlow over the years, many of these were to add extra fields to the flows exported by the probe, which were needed for grouping (*aggregating*) based on various attributes, such as which switch port the data entered on. This led to an explosion of various sub-versions (8.x), leading finally to a more flexible version 9 and finally to the open standard IP Flow Information eXchange IPFIX, which is still yet to see much support in vendor offerings.

[1] This can be done using **netstat -s**.

To keep this section free of unneeded detail, here are the versions most commonly used:

Version 5
This is perhaps the most common and is generally the default version configured.

Version 7, 8
These are generally Cisco specific and add various fields. Versions 9 and 10 (IPFIX) have replaced them.

Version 9
This includes support for IPv6 and MPLS (outside the scope of this paper). The protocol format is also more flexible.

Version IPFIX (aka. Version 10)
Based on version 9, this is the IETF-based version. Allows for some enterprise-defined extensions and enhanced packet format.

27.3. Requirements Analysis

There is a lot of software available for implementing various parts of the NetFlow architecture. Most of the products are for various analysis tasks, but there is still a wealth of choices for collectors; a bit less for probes, since most probes will come with routers operating system (eg. Cisco IOS), although this feature may be limited to higher-end devices in some vendor offerings. Our problem is to figure out which software is best for us; to do this we need to do some requiremnts analysis. To answer this question, we need to first figure out what features are important, and which software satisfies the various features.

Table 27.1. NetFlow Probe Selection

Software	v.9?	Linux?	Debian?	Maintained?	Comm. Opinion	Sampling?	Source	Cost
fprobe-ulog	N	Y	Y	2005	dead	Y	F/wall	Free
softflowd	Y	Y	Y	~	Good	Y	pcap	Free
nProbe	Y	Y	nTop	Y	Y	Y	pcap	€99.99
pmacct	Y	Y	Y	Devel	Y	Y	either	Free
ipt_NETFLOW	N	Y	N	Y	~	N	F/wall	Free

Considering that we really want version 9 support for IPv6, we can choose from softflowd, nProbe or pmacct. pmacct is developmental, so may be harder to support and may have less stable community experience. nProbe seems very good, particularly aimed at performance, but the cost is quite high; although nProbe is available free for universities and research, I'd rather not have to deal with licensing issues. That leaves us with softflowd, of which I found at least one comprehensive article about building an integrated NetFlow system around, and is also widely available.

Retrospectively, softflowd does do a good job, although I do wish it would export incoming and outgoing interface; unfortunately libpcap based probes can't really get that information; it would need to be more tied to the routing subsystem.

softflowd had NetFlow version 9 support added two released ago, so should be fairly stable. It is also available via package repository, so keeping up-to-date should be less of an issue,

although the software seems to be in a stable-but-slowly updated phase of its life, due to developer priorities. **softflowd** also has one extra feature that may be rather nice: the ability to read in a pcap file (as produced by **tcpdump**) and emit NetFlow records based on it, which is useful for testing and evaluation.

Okay, so we have an apparent winner in the selection for the 'probe' component of our NetFlow architecture: we still require a 'collector' and some analysis or reporting tools, which there are numerous commercial and open-source products for.

Using NetFlow version 9 as a must-have requirement, we can immediately discount much of the field, much of which may be otherwise commonly used. One fairly serious contender, which is free, is **nfdump**. This is a tool written by people actively working and using the product in projects such as Internet2, which feature technologies such as IPv6 and other assorted goodies, so it is reasonable to think that it should be capable of satisfying our needs — but note that we don't yet know how hard it might be to use. A look at resources such as mailing list archives, on-line bug reports and release history confirms that it is actively maintained and developed, with a stable and developmental branch and a friendly and responsive developer. **nfdump** can be integrated with a larger product, called NfSen, which is a web reporting frontend, though it is not required.

So now we have some components that should be suitable for the 'probe' and 'collector'… but what about the reporting and analysis tools? To answer that question, we need to do some further requirements analysis as to what we want to do with the data.

Some likely activities include network accounting / monitoring / diagnosis. Using NetFlow we can ask questions such as:

- Who (is/has been) using the bandwidth the most?

  This could be used as a data-source for targetted education of, say, users that are using resources inappropriately.

- How much data did user/computer/group X send/receive internationally/domestically/locally?

  This could be used for billing or, if such reports are generated frequently, to alert someone about potential budget blowouts.

- How much of our bandwidth use is comprised of application X?

  Perhaps I could use this data to make an informed decision about deploying caching data to move data off my expensive WAN links.

- Has our change to X reduced/impacted network load as expected? Are there any unexpected consequences?

  This is generally, like many of these, a graphing/trending application, but may also be facilitated by a weather-map.

- Are the any unusual spikes that might indicate security incidents such as Denial of Service or internet worms?

  This is a common use of NetFlow accounting data in a security setting.

As we only have limited time available, we shall concentrate on the easier areas of accounting, although there is plenty of scope to expand. It is also an area that is to be most likely in

smaller networks. Note that many areas commonly require a mix of pre-determined and ad-hoc reporting facilities:

- "Hello, I'd like to know why my network bill is so high…"

  The network bill would be a pre-determined report, but the ability to drill down (ie. look at increasing levels of detail) may very likely need to be ad-hoc.

- "Okay, so it seems host X was broken into by an exploit on port Y… what other machines may have been affected this way?"

  The detection of an exploit may not have come from a NetFlow based report, but investigating traffic to port Y would generally require some degree of ad-hoc reporting, which could be done by a pre-defined parameterised report.

- "Hmmm, can we recognise new applications on our network that we don't currently track?"

  It is common to try and track the amount of data used by protocols such as HTTP, SMTP, FTP (in its various modes) and others, but there will be some Other category, which you might like to take a closer look at occassionally.

- "Argh! What is causing these spikes in traffic?"

  In this situation, NetFlow itself may not be enough, because NetFlow only gives you packet-level statistics, it doesn't give you the packet data itself. But that said, NetFlow can give you enough visibility on the network to begin making intelligent hypotheses as to what might be happening, and then you can introduce more ad-hoc monitoring in the parts of the network which are seem more able to answer the question. Finally, note that many of these LAN-activity questions will be best answered using NetFlow data from switches, of which there is likely to be an excessive amount of data.

- "Why are my Voice-over-IP sessions so slow sometimes?"

  You might use NetFlow to determine what else was happening in the network when the problem manifests itself. Perhaps there is a lot of traffic from machines automatically downloading patches or submitting backups, perhaps there are people viewing YouTube videos at the time, or perhaps someone is sending photos via e-mail which is choking upstream bandwidth. These are all questions that NetFlow can help you answer quickly.

A suite of programs called flow-tools are a common free solution for these tasks; generally coupled with other tools to do things like graphing. FlowScan is another common free tool that can do some useful reporting such as producing a graph showing the composition of traffic — such as web traffic, FTP, etc. — over time. Unfortunately, neither of these tools yet support NetFlow version 9, so we shan't cover them.

Fortunately, **nfdump** has some rather useful querying facilities, including a filter mechanism similar to **tcpdump**, that supports many of the ad-hoc queries we may wish to make, including Top-N style questions. Therefore, we shall use **nfdump** for some of our reporting and querying examples as well as our 'collector' component.

This leaves us with at least one part of our reporting system with which there are surprisingly few available tools: the accounting system. This is because such systems are commonly: complex, customised, vary greatly according to a) how the accounting data is collected, b) policies by which accounting is done, c) integration with billing and payment systems. Note that there can be a reactive quality to b and c, such as what may happen when you introduce a quota policy. As such, accounting systems are often developed in-house; we shall see how

we could use **nfdump** to collect the basic data, but processing the data and making that into a report is an exercise left to the reader.

27.4. Install, Configure and Test the Probe

*In this section, we shall install **softflowd** from a package repository, configure it appropriately and test that it is working.*

The probe needs to be installed either on a router, switch, or attached to a port on said device though which a copy of every frame is sent; such a port is commonly referred to as a 'mirror' or 'SPAN' port. On our Linux router, we shall be installing the probe on the router itself. Most commercial routers will have a probe facility available already installed.

Install the **softflowd** package on your router using the standard tools available.

```
# apt-get install softflowd
… note that it doesn't start yet …
Not starting softflowd
Edit /etc/default/softflowd and define the INTERFACE variable
```

The software does not yet run because it cannot run without being configured to listen on a particular interface. Because the software is based on the same software (libpcap) as **tcpdump**, we cannot listen to multiple interfaces at once. If you wanted to, you would need to start two instances of the program, which would generally involve duplicating and changing the startup script.

But which interface should we listen on? This will depend on what sort of questions we want to ask, as this affects the sort of traffic that we see. If we listen on the outside, then we see all of the traffic coming to us including the traffic that gets dropped by a packet filter on the outside interface. This is useful for determining DoS activity etc, but will not be useful for answering questions about traffic purely on the inner network or directed to the inner interface of the router.

Another, rather major, implication is the affect of NAT. If we listen on the outside interface, we see the packets after they have been Source-NAT'd, which means we see all traffic as coming from the router, not the machine itself. Clearly this is not useful for tasks such as accounting, so we shall listen on the inside interface.

As root, edit the file `/etc/defaults/softflowd`. Set `INTERFACE` to `inside` and `OPTIONS` to `"-n 127.0.0.1:9995"`.

Start **softflowd** using the standard startup script.

```
# /etc/init.d/softflowd start
Starting softflowd: softflowd.
```

Verify that it starts, and that it logs nothing of concern to syslog. You should see something like the following:

```
… softflowd[…]: softflowd v0.9.9 starting data collection
… softflowd[…]: Exporting flows to [127.0.0.1]:9995
```

We can verify that it is sending data using a tool such as **tcpdump**. After you start this, you will need to either wait for a while for some expired data to be sent, but it is easier to simply generate some yourself. You should see something like the following:

```
# tcpdump -n -i lo port 9995
listening on lo, link-type EN10MB (Ethernet), capture size 96 bytes
16:16:01.830727 IP 127.0.0.1.54991 > 127.0.0.1.9995: UDP, length 600
16:17:01.490783 IP 127.0.0.1.54991 > 127.0.0.1.9995: UDP, length 1464
16:17:01.491298 IP 127.0.0.1.54991 > 127.0.0.1.9995: UDP, length 984
```

Further activities regarding the probe would be related to performance monitoring and tuning, but that is an activity best left till the other components have been put into place.

27.5. Install and Test the Collector

*In this section, we install the **nfdump** product, test that it is receiving and storing the data, and verify that the data from the probe appears reasonable.*

The **nfdump** package is a suite of tools, one of which is **nfcapd**, which is the collector, and **nfdump** which is the display and analysis program. There are some other tools included as well, but those are the major commands we need to know about.

Install the **nfdump** package. After you install, check to see if any new processes are running, you should see a new process called **nfcapd** running. Let's see how it was started.

```
$ ps axo command | grep '[n]fcapd'
/usr/bin/nfcapd -D -l /var/cache/nfdump
```

Okay, and which port is it listening on (different probes default to different port numbers)?

```
# lsof -Pni | grep nfcapd
nfcapd    25755        root    3u  IPv4 2570199       UDP *:9995
```

Great, so now we have verified that the collector is working and listening on the same port as the probe is exporting to. Optionally, we could also verify that **nfcapd** is receiving the data, but you would only typically bother doing that if you were diagnosing any problems, or if you were just curious.

```
$ pidof nfcapd
25755
# strace -p 25755
Process 25755 attached - interrupt to quit
recvfrom(3, "\0\5\0\31\0\36J\217K\0273\211…"..., 65535, 0, NULL, NULL) = 1224
time(NULL)                              = 1259811721
recvfrom(3, ^C <unfinished ...>
Process 25755 detached
```

Note that you would need to wait a while for some data to be produced, when there is some data produced, it will be returned via the second argument to `recvfrom`. But what is the data saying? To answer that, let's use some very basic querying to check what it is recording.

```
$ nfdump -R /var/cache/nfdump/ | head -5
Date flow start        Duration Proto Src IP Addr:Port     Dst IP Addr:Port      Pkts   Bytes Flows
2015-01-08 16:19:14.592 35.895   TCP  139.80.206.169:3142 ->192.168.1.11:60041   1736   5.0 M  1
2015-01-08 16:19:14.592 35.895   TCP  192.168.1.11:60041  ->139.80.206.169:3128   460   27291  1
2015-01-08 16:20:16.625  8.619   TCP  139.80.206.169:3142 ->192.168.1.11:60041  27142  49.0 M  1
2015-01-08 16:20:16.625  8.619   TCP  192.168.1.11:60042  ->139.80.206.169:3128  1946   91175  1
```

Screenshot

Take a screenshot to record what you see here. Does the data make sense? Well, we have meaningful source and destination addresses which are not unduly influenced by

NAT, so that is good, and the port numbers indicate some proxy traffic, so I'm happy that it is working for now. Time to look at some reporting.

But are the collection statistics accurate? Think about how you could determine this; we'll look at it in the assessment.

27.6. Basic Reporting

*In this section, we use the **nfdump** product, installed in the previous section, to query the collected data and answer some basic questions about network activity which we generate. This practice is fundamental to understanding what we can do with NetFlow, so don't rush.*

Screenshot

Let's practice querying the recorded data using the **nfdump** tool. All similar tools should be able to produce much of the same, although there will be some large differences in user interface and user-friendliness for various tasks. As you do each item in the list, take a screenshot.

- First some very basic usage. Look at the manual page for **nfdump** to determine what these options mean.

```
nfdump -R /var/cache/nfdump/ -c 5
```

Screenshot

This should print out the first five flows stored in the directory `/var/cache/nfdump/`. Take a screenshot showing the output of that command. Have a look in the directory that is used. Briefly describe how the data is stored; you will need this for the assessment.

Have a look at the summary line. What useful information is listed here?

- Often we are concerned about a particular time slice. Let's repeat the previous example using a particular timeslice; you'll need to determine a suitable example timeslice of which you will have traffic recorded.

```
$ nfdump -R /var/cache/nfdump/ -c 5 -t 2014/12/04-2014/12/06
… first five flows of a two day period …
$ nfdump -R /var/cache/nfdump/ -c 5 -t 2014/12
… first five flows of December 2014 …
$ nfdump -R /var/cache/nfdump/ -c 5 -t 2014/12/04.12-2014/12/04.13
… first five flows of 4th December 2014, 12:00-13:59 …
```

- We can select a different output format that is useful for either casual use, closer inspection, or for use with other programs such as for generating billing data. We can even specify a custom output format with **nfdump**.

```
$ nfdump -R /var/cache/nfdump/ -o long
```

Have a look at the manual page for **nfdump** and play with different arguments to the `-o` option.

- Notice that in the previous example, the rightmost column of the 'extended' output mode shows the number of flows that are aggregated in that line, and that they are all 1.

Aggregation is an important concept when generating reports, as it aggregates the data from individual flows into various groups we are interested in.

Some connections, such as SSH connections, can be long-lived and so may easily be exported from a probe due to time, rather than due to flow completion. For this reason, one very simple form of aggregation is simply to aggregate based on ‘connections’, so individual flows which may have been exported because they spanned a long-time get aggregated together, which is a rather more ‘natural’ way to look at it.

```
$ nfdump -R /var/cache/nfdump/ -a -o long
```

Now you should notice that there are numerous lines that have numerous flows aggregated.

- What are the top ten hosts with regard to bandwidth use? This is another form of aggregation, only based on a particular NetFlow attribute.

```
$ nfdump -R /var/cache/nfdump/ -s srcip/bytes -n 10
```

The size of the report can be parameterised using `-n` for the number of reports, and/or using something like `-L +40M` to limit those results to those over 40 MB.

We can use two queries, one with srcip and the other with dstip, to look at data both incoming and outgoing. You can start to get a feel that we are approaching what looks like an accounting report.

- However, for accounting purposes, there is a lot of stuff that is duplicated. If we want stuff going to/from the internet and our private network, we only want stuff that goes in one interface and out another. Typically, we might use the incoming and outgoing interface ID, but our probe doesn’t supply us with that information.

  This provides us a useful opportunity to look at the filter capability of **nfdump**. This gives us great ad-hoc flexibility. The syntax is similar, but sufficiently different enough to be a little annoying, to the filter syntax used by libpcap (ie. **tcpdump** and friends).

```
… This is our upload report …
$ nfdump -R /var/cache/nfdump/ -s srcip/bytes -L +10M 'src net 192.168.1.0/24'
Byte limit: > 10485760 bytes
Top 10 Src IP Addr ordered by bytes:
Date first seen          Duration  Proto   Src IP Addr  Flows  Packets  Bytes   …
2014-12-03 16:09:07.812 96171.835  any    192.168.1.51  10074   446939  34.6 M  …
2014-12-03 16:21:44.166 104711.381 any    192.168.1.52  6561    57491   16.4 M  …

Summary: total flows: 23357, total bytes: 57.6 M, total packets: 544964,
         avg bps: 4578, avg pps: 5, avg bpp: 110
Time window: 2014-12-03 16:09:07 - 2014-12-04 21:26:55
Total flows processed: 40795, Records skipped: 0, Bytes read: 2125540
Sys: 0.016s flows/second: 2549687.5  Wall: 0.011s flows/second: 3677875.9

… This is our download report …
$ nfdump -R /var/cache/nfdump/ -s dstip/bytes -L +10M 'dst net 192.168.1.0/24'
Byte limit: > 10485760 bytes
Top 10 Dst IP Addr ordered by bytes:
Date first seen          Duration Proto    Dst IP Addr  Flows  Packets    Bytes …
2014-12-03 16:09:07.812 96170.221 any     192.168.1.51   9787   549401  710.0 M …
2014-12-03 16:21:44.166 104711.381 any    192.168.1.52   6420    73105   52.0 M …
2014-12-03 16:19:20.083 104837.615 any    192.168.1.145  2373    24907   19.7 M …

Summary: total flows: 23035, total bytes: 789.8 M, total packets: 662711,
         avg bps: 62818, avg pps: 6, avg bpp: 1249
```

```
Time window: 2014-12-03 16:09:07 - 2014-12-04 21:26:55
Total flows processed: 40795, Records skipped: 0, Bytes read: 2125540
Sys: 0.016s flows/second: 2549687.5  Wall: 0.010s flows/second: 3717084.3
```

Have a think about what you see in this report and in particular the filtering expression used. What might be missed?

27.7. Advanced Reporting

We don't have time in this laboratory session to work through an install of a anything particularly complicated, so we shall simply look at what one example of a NetFlow analysis and reporting product can do for us.

There are a wealth of available tools for doing reporting and analysis; many are commercial because they help provide greater business intelligence and so is something companies are willing to pay for.

Do some on-line searching (suggested terms: 'netflow white paper') and find a product that does analysis and reporting. Choose a product that is different than what others are covering. What are some of the features this product provides? You will need this for the assessment.

27.8. Assessment

1. When we installed the collector, you were asked to think about how you would determine if the results were accurate. Briefly describe an experiment you could undertake to evaluate the accuracy of the data.

2. Describe how the data in `/var/cache/nfdump/` is organised. What limits, if any, are in place to prevent the data from growing to fill the disk? If needed, take steps to constrain the amount of disk used; document how this could be done. You will need to do some self-guided research. (Hint: see the manual page for **nfdump** and learn about expiration of old files).

3. Earlier we practiced using **nfdump** for generating upload and download reports. What may be missing in that report. Why is input and output interface preferable?

4. What might happen to accounting when we introduce a proxy cache? What are the issues here? What could be done about this?

5. In the Advanced Reporting section, you were asked to do some on-line searching (suggested terms: 'netflow white paper') and find a product that does analysis and reporting, which is different from a product covered by your peers.

Print the whitepaper. On a single sheet of paper, answer the following, referring freely to the whitepaper or the vendor literature:

1. Which product did you look at?
2. What are the major features of this product? (ie. what makes this product better than others?)
3. What versions of NetFlow does it support? Does it support sFlow ("sampled flow") which is an improvement on NetFlow?
4. What are the suggested uses of this product? (ie. why should a business use this product?)

When you have been marked, pin the whitepaper and your answers on the board in the classroom in the place provided. In this way, we can survey the marketplace and get a feel for the various things that can be done using NetFlow data.

6. Why is it useful to have different graphs of flows, packets and bytes?

How might you recognise a Syn-Ack denial-of-service (DoS) attack by time-series graphs of flows, packets and bytes?

Lab 28 Directory Services

Old Lab

This lab hasn't been run recently, so no guarantees are made regarding the commands and their output.

Setting up an LDAP (Lightweight Directory Access Protocol) under Linux has often been a problem for me… there is much that can go wrong. So, I have decided to keep this lab particularly simple; as simple as I can make it and still be useful.

In todays lab, we shall first learn about using LDAP querying tools, as provided by OpenLDAP. These will be useful to us in testing and diagnosing any problems. We shall do this using a virtual lab I have set up you that has the server already configured and populated. We shall configure another client in that lab to use the server. Finally we shall use a different Vinel virtual environment and set up a server *and* client for basic authentication. In this way, we shall have less places where things can fail during each stage.

28.1. Linux Authentication

In order to understand how LDAP works, or to understand why it doesn´t if it is broken, we need to understand the Linux authentication sub-systems. This primarily involves three things: the Name Service Switch (NSS); the Name Service Caching Daemon **nscd**; and Linux Pluggable Authentication Modules (Linux-PAM).

Before PAM was introduced, all the software on a Unix system had to take care of its own authentication. If a system administrator wanted to add a different authentication mechanism —typically LDAP—than what was typically supported (looking in local files such as `/etc/passwd` or using network authentication such as Sun´s Network Information Systems (NIS), then all of the software that needed to do authentication would have to be re-built (typically with third-party patches applied) to be able to use it.

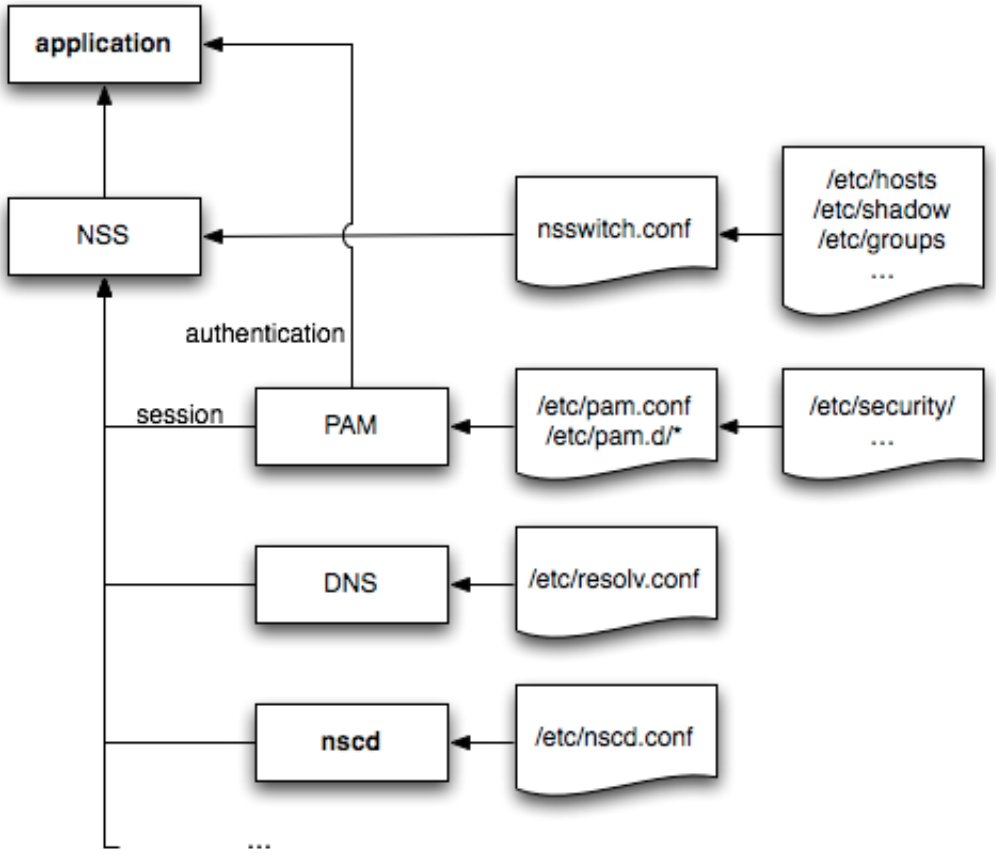
The Linux-PAM effort was created to provide a dynamic library that software could link against, and allow the authenticating software to be isolated from the process of authentication. With PAM, you don´t even need to assume username/password semantics. We can even use Single Sign-On (SSO) services such as Kerberos.

The Name Service Switch is part of the core C language library of the system (`libc`). Its purpose is the management of various system databases to do with system and user identities, such as hostnames, users, groups and a bunch of other, less-important things. Have a look at the documentation [http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html#Name-Service-Switch], particularly the NSS Basics.

One of the things that can make LDAP client configuration on Linux tricky is that the NSS LDAP and the PAM LDAP file use the same syntax, but does not guarantee that this will stay the same. So some vendors, such as Redhat, just use one file `/etc/ldap.conf`, while others, such as Debian, use different files[1], but use the debconf (Debian Configuration) framework to manage them, although you can manage them yourself if you prefer; but it would more reliable to let debconf manage them and keep them in sync.

The relationship between PAM and NSS can be summed up in the following diagram.

[1] `/etc/pam_ldap.conf`, `/etc/libnss-ldap.conf`, and some utilities may refer to `/etc/ldap/ldap.conf`. It's all really depressing.



NSS is a common library that applications can use to query the system databases. Which backends are available for each database, and the order in which they are queried, is configured in `/etc/nsswitch.conf`. Because some lookups can tax network resources if they are done often, a caching daemon **nscd** works to cache results for a configured period of time. Note that applications also interact with PAM directly, not just through NSS.

Rather than stumble through an overview of what happens when a user logs in, I shall instead point you to an overview [http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/sag-overview.html] in the official PAM documentation. I highly recommend reading this overview; the rest of the document is also a worthwhile read. The Linux-PAM System Administrators Guide [http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/Linux-PAM_SAG/] is one of three major documents; the other two being for PAM Module writers and Application writers.

When you are done, use this test-yourself to ensure you understand the material.

1. Give three common examples of a *service*

2. In your own words, briefly describe each of the following four service *types*: *account*, *auth*, *password* and *session*. Note that *password* can be missleading, as it updates authentication tokens; it might not represent a username/password authentication scheme.

3. In your own words, give a brief explanation of each of the *control* keywords: *required*, *requisite*, *sufficient* and *optional*.

4. What happens if both `/etc/pam.conf` and `/etc/pam.d/` exist? Explain the purpose of `/etc/pam.d/`

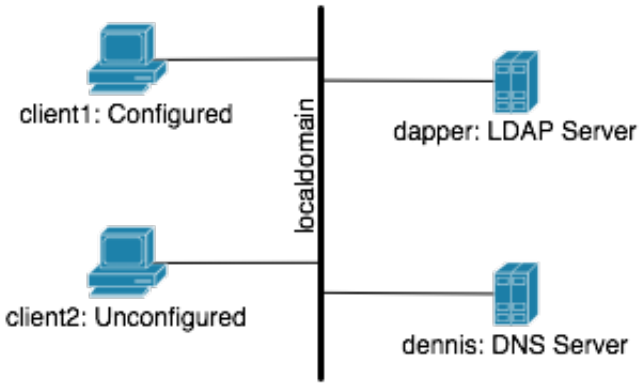
5. Explain the purpose of the *other* service.

6. Briefly explain the purpose of each of the following common PAM modules[2]: `pam_unix.so`, `pam_deny.so`, `pam_access.so`, `pam_time.so`, `pam_cracklib.so` and XXX.

This is certainly not a complete list of PAM modules; rather it is just the set of PAM modules that come with PAM. Try **apt-cache search libpam | grep libpam** on an Ubuntu or Debian machine to see what other PAM modules are readily available. One particular module that we haven't yet come across is the LDAP PAM module. We will also need the LDAP NSS module, but we'll come to that later. Right now, let's try querying an LDAP server.

28.2. Practicing querying and navigating with LDAP

In this lab, I have provided a different Vinel lab environment, consisting of an OpenLDAP server and a client. In this section, you will need to use the `ldap-query` Vinel lab (so use **vinel start ldap-query**). The layout of this lab is as follows:



The layout of the `ldap-query` lab, which will be used for practing querying an established LDAP server.

28.3. Client Installation and Configuration

Okay, with the objective of keeping this really simple, do the following:

Procedure 28.1. Installing libpam-ldap and libnss-ldap

1. Ensure that `/etc/pam_ldap.conf` and `/etc/libnss-ldap.conf` *do not* already exist. This is not necessary, but if you had the software installed before, and your configuration was faulty, then you ought to delete the faulty configuration.

```
# apt-get install libpam-ldap libnss-ldap
```

The software will install and you will be asked the following questions:

[2] See `/lib/security/pam_*` for the modules currently available on your system.

2. Please enter the URI of the LDAP server used. This is a string in the form `ldap://hostname-or-IP:port/`, `ldaps://` or `ldapi://` can also be used. The port number is optional.

   Note: It is usually a good idea to use an IP address; this reduces risks of failure in the event name service is unavailable.

   LDAP server Uniform Resource Identifier:

   Before we go on, let's understand what `ldaps://` and `ldapi://` are. `ldapi://` allows you to connect to an LDAP server running on the same machine. It uses what is called a *named socket*. It is faster than an IP connection to the same machine, and it allows for more access-control.

   `ldaps://` first negotiates a SSL or TLS connection, and runs LDAP on top of that. This is much like `http://` versus `https://`.

   Respond with **`ldap://IP-address-of-dapper/`**

3. Please entre the distinguished name of the LDAP search base. Many sites use the components of their domain names for this purpose. For example, the domain "examples.net" would use "dc=example,dc=net" as the distinguished name of the search base.

   Distinguished name of the search base:

   Respond with **`dc=localdomain`**

4. Please enter which version of the LDAP protocol should be used by ldapns. It is usually a good idea to set this to the highest available version number.

   LDAP version to use:

   Respond with **`3`**

5. Choose this option if you can't retrieve entries from the database without logging in.

   Note: Under a normal setup, this is not needed.

   Does the LDAP database require login?

   Respond with **`No`**

6. This option will allow tools that perform requests to the NSS system with libnss-ldap as backend to return more information when called as root.

   If you are using NFS mounted `/etc/` or any other custom setup, you should disable this.

   Special LDAP privileges for root?

   Respond with **`No`**. Responding with **`Yes`** would imply we have some special access to the LDAP server.

7. If you use passwords in your libnss-ldap configuration, it is usually a good idea to have the configuration set with mode 0600 (readable and writable only by the file's owner).

Note: As a sanity check, libnss-ldap will check if you have **nscd** installed and wil only set the mode to 0600 if **nscd** is present.

Make the configuration file readable/writable only by its owner only?

Respond with **No**, as we don't need any passwords stored in the configuration file. Answering **Yes** won't hurt though.

8. Please choose which account will be used for NSS requests with root privileges.

   Note: For this to work the account needs permission to access the attriutes in the LDAP directory that are related to the users' shadow entries as well as users' and groups' passwords.

   LDAP account for root:

   You should only be asked this if you said **Yes** to the question about special access for root. If you did, finish the configuration questions, and then restart the questions with **sudo dpkg-reconfigure libnss-ldap libpam-ldap**.

9. `nsswitch.conf` not managed automatically

   For the libnss-ldap package to work, you need to modify your `/etc/nsswitch.conf` to use the `ldap` datasource. There is an example file at `/usr/share/doc/libnss-ldap/examples/nsswitch.ldap` which can be used as an example for your nsswitch setup, or it can be copied over your current setup.

   Also, before removing this package, it is wise to remove the `ldap` entries for `nsswitch.conf` to keep basic services functioning.

   Switch over to another terminal on the same virtual machine, and edit `nsswitch.conf`. The new parts are shown in bold. Not all of the file is shown.

```
passwd:         files ldap
group:          files ldap
shadow:         files ldap
```

10. You will then be asked some extra questions for the libpam-ldap package, many of which will be the same, but will already have your answers from the previous question, so just accept the defaults for those ones. The first new question will be:

    This option will allow you to make password utilities that use PAM, to behave like you would be changing local passwords.

    The password will be stored in a seperate file which will be made readable to root only.

    If you are using NFS mounted `/etc/` or any other custom setup, you should disable this. Make local root Database admin.

    Respond with **No**. This might be something consider later, but right now, we want to get basic authentication working.

11. The PAM module can set the password crypt locally when changing the passwords, this is usuall a good choice. By setting this to something else than clear you are making sure that the password gets crypted in some way.

    The meanings for the selections are:

    clear
    Don't set any encryptions, this is useful with servers that automatically encrypt userPassword entry.

    crypt
    (Default) make userPassword use the same format as the flat filesystem. This will work for most configurations.

    nds
    Use Novell Directory Services-style updating, first remove the old password and then update with cleartext password [which the server hashes in multiple ways].

    ad
    Active Directory-style. Create Unicode password and update unicodePwd attribute.

    exop
    Use the OpenLDAP password change extended operation to update the password.

    md5
    [No help available. Use a salted MD5 password, like a modern Linux system with shadow passwords.]

    To answer this effectively, you need to know what kind of server is being used. Best to talk with your system administration team.

12. One thing to check. Ensure that **nscd** is running. I like to use **pstree** to do this sort of thing.

Well, that was a lot of questions, but hopefully, everything should be working. Better test it and see.

28.3.1. Testing the Client

Okay, before we attempt something big, like logging in, let's verify some basics. We shall first test out the NSS subsystem and ensure that we can a) see groups that are defined only in LDAP, and b) see users that are defined only in LDAP. We can use the **getent** command to do this.

```
$ getent group groupname
groupname:*:gid:member1,member2,…memberN
```

```
$ getent passwd username
ckerr:x:15994:11400:Cameron Kerr:/home/cshome/c/ckerr:/bin/bash
```

Good, if you made it this far, NSS appears to be working okay. Lets have a quick poke around the PAM configuration before testing any further.

Lab 29 Ethernet Practical

> **Not offered in 2018**
>
> This lab is not offered in 2018.

This practical comprises two elements. First, you will learn how to build a simple ethernet cable by installing an RJ-45 *plug*. Then, you will take a virtual tour of a structured cabling plant, and record details particular to a certain machine, as if you'd just installed it into the network.

You will take a little time to learn about cabling tools and concepts such as cable types and wiring standards. If you are already familiar with them, you can skip over to Section 29.1.7, "Building the Cable" to follow the steps for building an Ethernet cable.

We also provide information regarding how to install an Ethernet Network Interface Card (NIC) in Section 29.2, "Install an Ethernet Network Interface Card (NIC)" and Ethernet autonegotiation and flow control in Section 29.3, "Autonegotiation and Flow Control". Much of the time you will not need to know much about them, but occassionally they can be useful for your understanding and future career.

Finally, Section 29.4, "Structured Cabling" provides you information regarding the assessment of structured cabling after watching the video.

You must complete the two parts of the laboratory to attain the marks for this lab.

29.1. Build an Ethernet Cable

Your teacher will give a demonstration of the procedure for installing an RJ-45 plug onto a UTP cable.

29.1.1. Tools

You will need the following tools and materials, shown in Figure 29.1, "Ethernet Cabling Equipment", which will be provided for you. For your reference, I have provided the approximate cost of them for you.

Figure 29.1. Ethernet Cabling Equipment

There are a number of tools that are useful to have when doing Ethernet cabling.

1. UTP Cable Stripper. (optional, ~ $15)

2. RJ45 plug. (~ $0.30 each in bulk)

3. Serrated Snips. (optional)

4. Cable Crimper. (~ $50, buy quality)

5. Ethernet Cable Tester. Very Basic (~ $15+)

6. (Not shown) Cat5e or better UTP cable. (~ $1/m wholesale)

If you look at five different videos of how to do this procedure, you will find about five different methods that people use, but the basics are the same: strip the cable, sort the pairs, crimp the cable.

What method you use will depend on the tools you have available, and the amount of money you are willing to spend on your tools. Professional cable engineers have this as part of their ‘plant’, which means from a taxation point, they receive depreciation on their tools.

Professional grade tools can be expensive, but they are worth it if you are doing this regularly. This is particularly good advice when buying a crimper; buy one with a metal body, as a sloppy hip joint will give mostly bad crimps.

29.1.2. Cable Types

There are a few possibilities as far as Ethernet cables are concerned. The most common is Cat6, Cat5E, Cat5, or Cat3 in slightly older (10Base-T) networks. All of these are UTP. New structured cabling installations will have at least Cat5e.

In very old Ethernet networks, you also see Thin-net and Thick-net, examples will be shown in the lab. We will not be concerning ourselves with such antiquated hardware.

Nor will we be concerning ourselves with Optical Fibre, as it takes specialised tools and training to deal with, and you're not likely to need it in a small network.

There are three types of cables you might see in Ethernet networks. The first is often called Silver Satin, and is a flat cable suitable only for 10Base-T networks. It's not used in modern networks, as it's more susceptible to noise, because the pairs are not twisted[1].

Cat3 and above cables come in twisted pair format. They are generally unshielded, although you can get Shielded Twisted Pair (STP) for environments where noise may be an issue, such as factory environments. They have 4 pairs of wires. Each pair is twisted, and the twists should be maintained as far as possible to the plug for Fast Ethernet and above. The number of twists/foot is what helps it to be less susceptible to interference, and is one of the things that is checked when a new installation is certified.

29.1.3. Cable Cores

The core of each wire can be either solid or stranded. Solid core is used for permanent installations, as it has better signal characteristics. Stranded core is often used for patch cords, or anywhere where the cable may be moved a lot. Stranded cores are less susceptible to breaking when they are bent, and are quite whippy, whereas solid core cable tends to keep it's shape when you bend it.

Stranded cable requires a slightly different RJ45 plug than solid core. The differences are very subtle, and affect how the pins on the plug bite into the cable.

29.1.4. Performance Specifications

Cables have certain performance specifications, and are graded by the baud (signalling) rate, and other signal characteristics, such as susceptibility to noise, attenuation, etc. For 10Base-T networks, Cat3 cable is used, it can carry 10Mbps. Cat4 cables can carry 16Mbps, and is used in older Token Ring. Cat5 is common today, and can be used with 100Base-Tx networks. Cat5e and Cat6 are higher grade cables that are suitable for Gigabit applications and for 'future applications'. All of these cables have four pairs of cores. However, only Gigabit Ethernet uses all four pairs.

Naturally, you can use a higher grade cable than you need to and everything will still work fine, but the reverse is not true. If you were to use Cat3 in a 100Mbps application, you would get a lot of errors, and the network would be even slower.

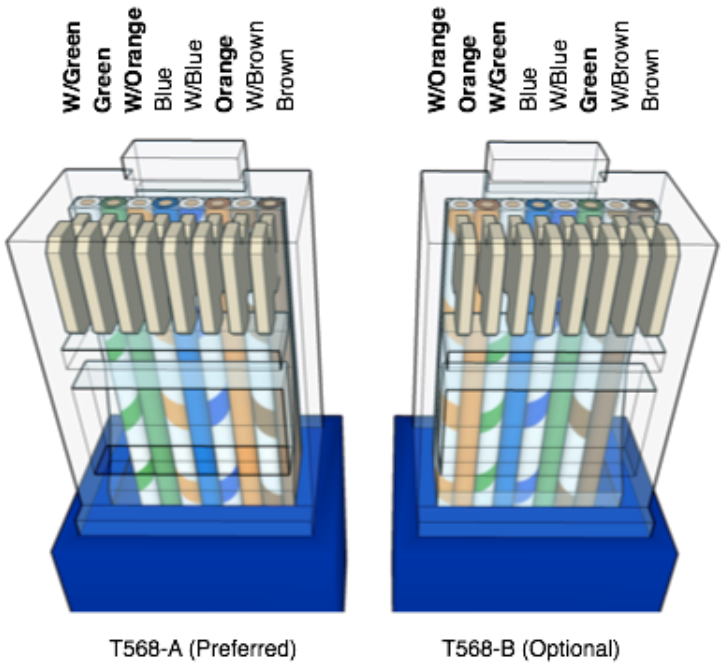
29.1.5. Wiring Standards

There are two standards for wiring twisted pair ethernet cable: T568-A (Preferred) and T568-B (Optional). Both can be used together, but for optimal performance, the same should be used throughout the network. In fact, in Cat5 installations and above, the standards say not to mix the two.

We have listed the Preferred and Optional wiring standards in Figure 29.2, "TIE-EIA T568 Ethernet Wiring Standards", as laid out by the TIE-EIA standards body. The optional scheme comes about from the prominence of AT&T wiring schemes, and is still very common.

[1] Although you *can* get flat cable that has twisted pairs, such as Flat Cat 6, which is useful for laptops.

Figure 29.2. TIE-EIA T568 Ethernet Wiring Standards



There are two acceptable wiring standards, TIE-EIA T568 A and B.

Note the the words Preferred and Optional may not reflect reality on your site.

29.1.6. Straight-through and Crossover

To connect from a hub or switch to a computer (*non-like* devices), you use a straight-through cable. To connect a *like* devices, such as hub/switch to a hub/switch (without using an *uplink port* on the hub or switch), or a computer to another computer, you would use a *crossover cable*. Note that a router interface would be wired the same as the computer.

If you don’t get a link light when you plug in the cable, you’re probably using the wrong cable for the port you’re plugging into. Crossover cables are sometimes marked with an X on the ends to note that it is a crossover cable. Get used to looking at the both ends of the cables to judge reliably if you have a straight-through or crossover cable. A lot of switches and laptops being sold on the consumer market today are *autosensing*, which means it will detect if you’re using a straight-through or crossover cable.

A crossover cable for 10Base-T and 100Base-T (i.e., Fast Ethernet) is constructed simply by wiring one end in Preferred, and the other in Optional, as shown in Figure 29.2, “TIE-EIA T568 Ethernet Wiring Standards”. However, for Gigabit Ethernet and 10Gigabit Ethernet, you have to follow the standards to make crossover cables. Fortunately, such network cards generally have have an autosensing feature to detect which type of cable is plugged in and whether a crossover needs to be added internally or not. Therefore, there is no particular need to use crossover cables for Gigabit and 10Gigabit Ethernet.

Straight-through (normal) cables have both ends the same, either both Preferred, or both Optional. Short cables that are wired this way are generally referred to as *patch cables*, as they are used for wiring patch panels and for the computer-WAO (Work Area Outlet) link.

29.1.7. Building the Cable

> **Important**
>
> You must check with the demonstrator before criping your cable end.

The demonstator will show you in detail the steps for putting a RJ-45 plug onto the end of the cable. Here are the steps in brief so you can prepare.

1. In case you want to use a protective boot for the cable, thread this onto the cable first, ensuring you put it on the right way. Boots are useful, because they prevent the snap on the plug from breaking off when the cable is pulled backwards, often from a tangle of other cords. However, we won't be using boots in this lab.

2. **Either** use a suitable cable stripper to remove the cladding. You don't need to cut more than 1 inch of cladding away. The tool should ideally cut almost through the cladding. There is often an adjustment mechanism (usually a screw) for tuning this,

   or use a sharp knife, such as a craft knife or Stanley knife. *Score* a ring around the cable, about 25mm (1 inch) from the end. If done well, the cable casing should snap off when pulled or twisted at the score line, leaving a tell-tail light-blue ring around the inner-wall of the cladding where the blade did not penetrate. This provides a handy guarantee that you didn't cut into the actual cable pairs.

3. Cut off the ripping cord (a thin string), if present. This is present so that cable installers can open up a large section of the cable if needed.

4. Untwist and sort out the pairs and cores, according to whatever wiring scheme you are creating on the cable. Do not untwist further than the edge of the casing; in fact, try to stop untwisting a little bit before it. Practice will tell you when to stop. See Figure 29.2, "TIE-EIA T568 Ethernet Wiring Standards" for the wiring sequence. Looking at the natural lay of the pairs will help to create a tidier cable. Remember, if it is a normal (straight-through) cable, both ends must be the same. For crossover they must be different.

5. Straighten and flatten the cores so they lie side by side, and parallel to each other. Massaging them between finger and thumb while swaying from side to side (the cores, not you) is a technique I find effective. You want to get them nice and straight close to the bottom also. Rotating the cable cladding to and fro while holding the cores between finger and thumb can be useful here.

6. Using your cable stripping tool, or serrated snips, cut off the cores 12mm from the edge of the cladding, being careful that the cut is square. If its not square, some of the cores may pull free during its working life. I don't recommend using diagonal pliers (sidecutters), because it's harder to make a level cut. Cable crimpers, and also strippers often have a part of the tool for this, which can be useful for getting the correct length.

   Another thing you can do here, if you cut them about 1mm or so longer first, is to insert the cores into the plug, so that they straighten, then take the plug off. Trim the cable to the correct length and straight across.

7. Holding the cable so the green (or orange depending on the wiring scheme) core is on the left, place the plug into the cable, with the snap of the plug *away* from you. Push the plug in, gently but firmly if required. Make sure the cores go all the way into the plug. You should be able to see the copper cores when you look at the end of plug.

You want to push the cladding up into the cable. When the plug is crimped, part of it will pinch the cladding, making sure the cladding takes the strain and not the cores.

8. Double-check the pinning. This is your last chance.

> **Important**
>
> Remember, you must check with the demonstrator before criping your cable end.

   Place the cable and plug into the crimper, making sure it is in the correct way (there should be a notch for the snap to fit into). Squeeze firmly on the crimper. With some plugs you may hear a snapping noise. Don't be alarmed.

9. Take out the cable, and check that all of the brass pins at the top are well pushed in. If the plug fails to go into a socket, or is a tight fit, it's likely that it needs recrimping.

10. Once the cable is complete, put both ends in a tester and ensure that the wiring is correct.

    If you have the type of tester that has all lights on at the same time, firmly hold the plugs into the sockets, and give each end of the cable a wiggle. Look for any flickering of the lights, indicating a poor crimp which will soon fail.

    If a plug is found to be faulty, cut it off and start again.

Once you have completed a cable, use it to assemble a very small network comprising a couple of switches and stations; there should be a physical-layer network map shown in the classroom, which you will need to replicate using the hardware provided.

29.2. Install an Ethernet Network Interface Card (NIC)

In previous years this was a required activity, but the practical side of it has been removed due largely to two reasons. The first reason is that the skills were generally not new to students. The second reason is that most computers have on-board NICs for performance. Therefore, the reading material of this section has been retained for your reference. You should read it to check your own skills; you may indeed find some information that is new to you.

First, a short note on screwdrivers. The screwdriver is one of the most important tools for a computer technician, and as such must work well. Most screwdrivers you see on the market fit fairly sloppily into the screw head. In particular, they sit too high because of the point on the end of the screwdriver. You can file the tip so it becomes a little stubby. This will help the screwdriver make better contact with the walls of the screw. This simple technique prevents damage to screws and makes it much easier to remove stubborn screws.

Procedure 29.1. Install a PCI Ethernet NIC

1. Shut down the computer, and turn it off at the power supply. Disconnect the peripherals, and turn off the switch at the power-supply, and at the wall. This will disconnect the power but leaves the system earthed.

2. Remove the cover of the case, giving you access to the expansion card slots.

3. Fasten an anti-static strap to an unpainted part of the case, the other end to your wrist. There are other solutions apart from straps, such as mats that you stand on.

   You should be aware of the kinds of clothing you are wearing. Remove articles such as polar fleeces and polyester, as these gain a lot of static charge when moving around. Avoid wearing silk boxers too, especially if you're going to be doing this profesionally[2]. Computer component manufacturers often have humidors to keep the air humid, which helps to draw out surplus charge, protecting the components.

> **Important**
>
> Component damage from static discharge is generally subtle, often resulting in degradation, rather than complete and noticeable failure. This leads people to form a *blasé* attitude towards it.

4. Place the unopened antistatic bag on the case, giving it a moment or two to equalise the charge. Antistatic bags work by passing the charge *around* the bag, rather than through it, so it's important that they are closed. A small piece of cellotape is sufficient for this.

5. Remove a faceplate from the computer. Have a look at the card to determine the orientation.

> **Warning**
>
> Some edges of faceplates and computer cases can be quite sharp and vicious: handle with care.

6. Open the static bag and remove the card. Do not touch the copper terminals where it plugs into the slot on the motherboard, both for reasons of static safety, and because the grease from your fingers can cause a poor contact. Contacts can be cleaned of dust with a can of compressed air, or with some IsoPropyl Alcohol (IPA, or rubbing alcohol) which will remove any oil.

7. If you were dealing with an older ISA card, you might need to change the jumpers on the card to select the location in memory the card is to found (IO Port), and the hardware interrupt it will use (IRQ). This is commonly done if the machine will have multiple cards, such as a router.

   Many ISA cards are configured using a special program that comes with the card. This generally runs in pure DOS, and is used to configure the IO and IRQ, and run diagnostics on the card.

8. Insert the card into the slot. PCI cards usually go in relatively easy[3], while ISA cards can be more firm, or sometimes rather sloppy. Check that the card is seated correctly in the slot.

9. Check that a network cable can be pushed into the socket, and you can hear the RJ45 plug snap into place. Once you can do this, insert the housing screw and tighten with a gentle torque.

[2] My boss warned me about this on my first day as a computer technician.
[3] Some can be quite stiff.

10. If you have the diagnostics disk, boot it and start the program. The name of the program will depend upon the card or the chipset. A good place to look for unsupported cards is Network Drivers [http://www.network-drivers.com/].

    Diagnostics often require two stations to make full use of the test, having one station set up as a master and another as a slave. This means it can test all components on the card.

11. You will often need to update an inventory or network configuration database. This is used for such things as: DHCP (assigning network addresses based on the MAC address; port security on Ethernet switches (to prevent spoofing and the attachment of unauthorised devices).

    You will also need to record the WAO identifier of the jack (socket) the machine is plugging into to ensure that the port is ‘live’.

29.2.1. Switches and Hubs

There are two types of *hub*, repeating and switching. Repeating hubs are often just called hubs, while switching hubs are often called switches. You can tell whether a device is a repeating or switching hub just by looking at it. For the benefit of coherency, when we talk of ‘hub’, we mean repeating hub, and when we talk of a ‘switch’, we mean switching hubs.

Hubs and switches have *uplink ports*, which are either normal ports, or activated via a switch. Using a normal port on a hub to connect to another hub as an uplink means you don’t need to use a crossover cable.

Some network adapters (such as those found in modern laptops) are *autosensing*, which means that you can use either a standard or crossover cable from the hub to the laptop without any worry. Most new switches sold today are autosensing. You can tell that because there are no uplink ports on them.

Repeating Hubs (Hubs)

Hubs have a *collision* indicator light, whereas switches do not, this is because all ports on a repeating hub share the same transmission channel (*collision domain*). Hubs have up to two lights per port, each one either on or off, usually. They are generally a link light, and a link speed light, to indicate the speed that a port is running at (10Base-T, or 100Base-Tx). If it only has one light, then its a 10Base-T hub.

In the time when switched ports were expensive, a device commonly known as a *concentrator* was used to attach machines onto the building backbone (as you will see in the later part of this practical, between the horizontal and vertical cable). The stations, would plug into the hub ports, while the switched port would be attached to the backbone. This would help to isolate the traffic, and help to prevent the network from flooding.

Traffic going through a repeating hub can be seen on all ports using a network analyser, which is a security concern with passwords and any unencrypted traffic, such as network filesystems. We shall see how a switch can help to prevent this.

Switching Hubs (Switches)

Switching hubs are full-duplex capable devices, and each port usually has two or three lights: link/activity (these are often put into the one light) and speed. The link/activity light will shine

when it has a link, and blink when traffic goes through that port, although the specifics differ between switches[4].

It is harder to sniff on a switch, although manageable switches (read ‘expensive’) often have a *mirror port* that you can attach a network sniffer to. There are also tools that use *ARP spoofing* and *broadcast storms* to fill up the tables in the switch, causing it to send out all ports, like a hub.

In modern enterprise networks, another common security feature is *port security*, which means only specified MAC addresses may be seen on a specified port. This can help to prevent unauthorised attachment of devices, but is not a foolproof, as MAC addresses can be changed, and techniques such as NAT (discussed much later in the paper) can hide devices.

29.2.2. Installing the Machine into an Ethernet Network

In a commercial setup, with a properly installed network with wall sockets, installing a machine into an ethernet network should be a simple matter of plugging in a patch cable (a cable with stranded core, wired straight-through, and no longer than 5 metres) between the wall socket and the network card in the computer. There are some gotchas however, which you shall learn about in the third part of this practical.

For now, just remember that you need to record the WAO identifier and the MAC address of the station so a connection can be made and an address assigned.

When you have plugged the computer in, and activated the port, turn on the computer. At some time during bootup, the link light on the network card should turn on, and stay on. However, the semantics of the light behaivour may differ from card to card, but that is usually the case. If you don’t get a link light, one of the following may be happening:

- You are using a crossover cable when you should be using a straight through cable, or vice versa.
- The cable is not wired correctly, broken, or is not of sufficient quality or grade.
- The port is not activated, or has been labelled incorrectly.
- The switch you are connecting may require reconfiguration of port security or Virtual LAN (VLAN) assignments. The port may be administatively disabled for some reason.
- Autonegotiation (see the next section for details) of the NIC and the switch or hub has failed.

Whatever the problem, keep calm. Whatever is causing the problem, it is somewhere in the link, and therefore lies between (and includes) the NIC and the switch. Section 29.4, “Structured Cabling” will show you some of these possible points of failure.

29.3. Autonegotiation and Flow Control

What happens when a NIC with a faster line speed, say 1000Mbps, communicates with something on the same link running a different speed, say 100Mbps? What happens when

[4]For this reason, a 24-port switch is fun to watch when you’re bored, waiting for something to happen. This is commonly referred to as “blinkenlights”

a NIC on one side can operate in full-duplex (sending and receiving on different pairs and thus simultaneously) while its link partner (such as a repeating hub) can only do half-duplex? Obviously, link partners need some mechanism to help them advertise their capabilities to each other so the highest common denominator can be used; this is what is called Autonegotiation.

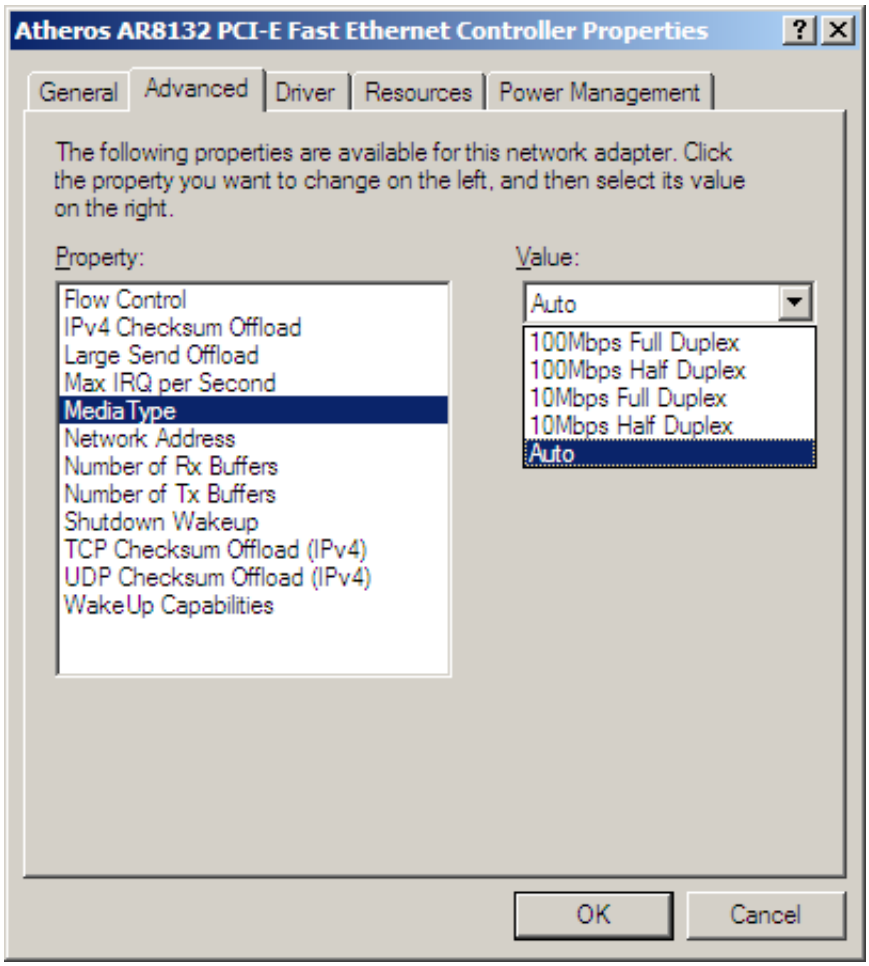
Autonegotiation was introduced into the 802.3 family in 802.3u, which introduced 100Base-TX; therefore anything that doesn't offer autonegotiation on twisted pair is assumed to be 10Base-T. Occassionally, autonegotiation may fail on some older (implemented according to 802.3 prior to 1998 and thus prior to Gigabit Ethernet) products, so you can also set the capabilities manually, forgoing autonegotiation. However, autonegotiation is mandatory for Gigabit Ethernet running over copper.

Autonegotiation is implemented using something called the "Medium Independent Interface", or MII for short. On Linux, we can inspect and configure the status of the MII using a tool such as **mii-tool**, although not all device drivers will expose this functionality. To give you an idea, here is what you would see on Client1. You can see from the "link ok" that a link has been established.

```
mal@client1:~$ sudo mii-tool --verbose
eth0: no autonegotiation, 1000baseT-FD flow-control, link ok
  product info: Yukon 88E1011 rev 4
  basic mode:   autonegotiation enabled
  basic status: autonegotiation complete, link ok
  capabilities: 1000baseT-FD 100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD
  advertising:  1000baseT-HD 1000baseT-FD 100baseTx-FD 100baseTx-HD 10baseT-FD  ↵
10baseT-HD flow-control
  link partner: 1000baseT-FD 100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD
```

On Windows, you can inspect and modify these settings by Configuring the adaptors Properties, as shown in Figure 29.3, "Inspecting MII Properties in Windows".

Figure 29.3. Inspecting MII Properties in Windows



It is generally possible to configure the Medium Independent Interface and other adaptor properties for a particular device under Windows XP. Note however, this dialog does not tell us what the result of the negotiation was.

But autonegotiation by itself is not entirely sufficient; what happens when one station is sending at the full capability of a 1000Base-T but another station in the segment (not its link partner, perhaps connected to another port on a switch) is only 100Base-TX? Typically, we would expect a higher layer protocol, such as TCP, to handle this flow-control. 802.3x is a standard that specifies Ethernet flow control using a special type of frame called a PAUSE frame. However, this can prove problematic as it can end up slowing down a significant portion of a network when one station is receiving too much data. Flow control is generally not used in the core of a network, and it can interfere with other performance measures such as Quality of Service (QoS). It is useful to know the position of various vendors [http://www.networkworld.com/netresources/0913flow2.html] on the issue of flow control.

For someone beginning network administration, it is enough to realise that it exists, that it can be problematic, and that if you see a lot of PAUSE frames on a segment, you should probably redesign some part of your network.

29.4. Structured Cabling

> **Note**
>
> Please bring headphones to the lab for this activity.

In this task, you will tour a virtual mock-up of a structured cabling plant in a small building. You will take note of various identifiers in order to 'make live' the Work Area Outlet (WAO).

Note that the cabling plant is generally a passive component, but today devices such as patch panels are becoming more intelligent, in order to reduce the cable clutter and reduce electrical noise.

Finally, you will answer some questions about what you have seen and about structured cabling in general. Watch the provided video[5], and record the following identifiers, for use in a cable identification scheme used by Pandiut Corp.

- Work Area Outlet (WAO).
- Telecommunications closet, rack (aka frame) and row.
- Patch panel and port.
- Switch and port.
- MAC address of the station.

Some things are not shown in the video, but it should be sufficient to give you a good idea.

There are many different cable identification schemes. The scheme you use will depend on the installer and the size of the installation. A work area for example, could be an entire building, or it could be one part of a floor.

A cable can be identified using the identifiers of both its end-points. This is particularly useful when you are putting labels on the cable, which should be done as they are installed.

Assume you have a cable that terminates on Floor 2, telecommunications closet A, rack 1, row 2, port 12. You might identify that end of the cable as 02A-1-2-12. The other end of the cable might terminate at the work area outlet A27 on that floor, and so you might refer to the *basic link* (the cable between the wall outlet and the patch panel) as 02A-1-2-12/A27. Putting this identifier on both ends of the cable will be useful for maintenance.

Assessment

1. Identify the particular *basic link* (eg. 02A-1-2-12/A27) that leads to the computer in the video.

2. Although it is not shown in the video, each WAO might have a label identifying the other end of the cable. Why can this be useful? It's not often done, probably because of the amount of labelling required. How else might this task be accomplished?

3. Describe what might happen when a duplex mismatch occurs: one end of a link is half-duplex, the other full-duplex. Remember that in full-duplex operation, operation can go in both directions *simultaneously*.

[5] There should be a CD available in the lab, otherwise you can find it from the course webpage.

Would a ping test be likely to demonstrate the presence of a duplex mismatch? Why or why not. To help you, consider the network traffic that ping generates. If a ping test would be insufficient describe a test that would detect the presence of a duplex mismatch.

29.5. Final Words

> **Important**
>
> Make sure you have had all of the practical activities marked off and your marks recorded.

Chances are you will have learned something in this practical, and I hope you have enjoyed it. However, the material presented today is the tip of a very large iceberg. If you would like to learn more on this topic, I suggest the following resources, some of which may need to be persued off-campus in order to appreciate.

Multimedia Resources

You should look at these from off-campus.

The History of Ethernet [http://youtube.com/watch?v=g5MezxMcRmk] (YouTube.com)
: Bob Metcalfe talks about how he and David Boggs invented Ethernet. Includes some insight into the areas in which Ethernet is developing.

Introduction to a Career in Network Support [http://www.youtube.com/watch?v=HCLDKaId09w] (YouTube.com)
: **Recommended**. So what if Server administration isn't interesting to you? One of the other options is Network Support. In this episode TechAnvil talks about the jobs that are available in the field, touches on certifications, and offers some commentary of the importance of understanding the basics of Network Operation.

Lab 30 Wireless Networking

> **Note**
>
> You are supposed to read the lab sheets before starting this lab. If you have done so, jump directly to Section 30.5, “Configuring Access Points” to start configuring a wireless access point. By the way, VirtualBox is not used for this lab.

What we are aiming for in this lab is to give you some experience configuring and using wireless networking, and give you an understanding of its benefits and limitations. This should be useful to you in developing or understanding a wireless network solution. You should be able to directly use the knowledge gained in this lab to set up a simple home wireless network.

30.1. About the IEEE 802.11 Standards

There are multiple varieties of 802.11 standards. Here are the most common standards you will need to know about, in the order they were released to the market. The letter indicates the time order in which the design of the particular standard began.

802.11b
: At a theoretical maximum of 11Mbps, the 11b standard is the one that started to capture the market, due largely to the success of Apple’s Airport.

802.11a
: The first of the 54Mbps standards, the 11a series uses a much higher frequency, in the 5GHz ISM (Industry, Scientific & Medical) band. There are three unlicenced bands. 2.4Ghz is the most common.

  This means there is less interference with other consumer devices, giving better performance, since most other consumer devices use the 2.4GHz frequency. However, because of its shorter wavelength, it is affected by obstacles to a greater degree, and uses more battery power when transmitting. Also, because of the different frequency, different antennas must be used, which is a major pitfall of 802.11a, due to the upgrade cost from 802.11b.

  Shortly after 11a was released, 11g was released, which is now the *de facto* 54Mbps standard. Most cards you buy new are B and G compatible. Some cards are tri-mode (dual-band), which means that can do all of B, A and G, but are more expensive.

802.11g
: The defacto 54Mbps standard, this runs at 2.4GHz, and, like its 11a cousin, runs at a theoretical maximum of 54Mbps, though is not quite as high-performance. Because it runs in the 2.4GHz ISB band it, like 802.11b, is susceptible to interference from devices such as Bluetooth devices, various cordless phones (not cell-phones) and microwave ovens. In order to support 11b clients, 11g access points can slow the network down to 11Mbps when an 11b client is present. To avoid this, a network can be made exclusive, meaning only 11g clients can connect.

802.11n
: 802.11n is an enabling technology for wireless multimedia, particularly for HD (High Definition) video. Some of the associated WiFi standards such as WiFi MultiMedia (WMM), which provides performance guarantees for multimedia, and the up-and-coming

WiFi Voice-Personal, which can be seen as a refinement of WMM for very low-jitter traffic for WiFi handsets, signal the readyness and acceptance of the use of WiFi for real multimedia delivery, and the performance levels associated with that.

802.11n introduces numerous innovations to achieve performance roughly five times that of 802.11g. Consistent numbers for throughput of the standard are hard to find and not particularly meaningful, because some of these innovations may not be suitable or applicable in particular deployments.

The first innovation that 802.11n brings is that it can use *both* the 2.4GHz and 5GHz frequency bands. To co-operate with other WiFi deployments, typically for transition periods, a 802.11n access-point may be configured not to use a particular band.

Related to this is the second technique, which standardises a practice already seen from some vendors[1] such as D-Link's Turbo range of products, whereby twice as many channels are used. So 802.11n *can* use 40MHz channels *in the 5GHz band* instead of the 20MHz channels that have been used previously and will still be used in the 2.4GHz band. This feature is commonly called Channel Bonding

You will find many 802.11n devices easy to recognise because they will typically (not always) have perhaps three antennas; this is used for something called MIMO (Multiple Input Multiple Output). MIMO takes advantage of *spatial multiplexing* to send two different frames at the same time, thus doubling the throughput. Previously, signals experienced interference by reflections off things like metal furniture; this was called *multi-path inteference*: 802.11n turns this liability into an asset and manages to actually use multi-path to its advantage. We can therefore expect better performance indoors compared to outdoors; opposite to what we would expect with previous WiFi technologies.

All of these added features come with a cost of extra power draw, which can cause problems in enterprise environments where the access-point is powered using IEEE 802.3af Power over Ethernet (POE). Enterprises will have some other considerations to make as well. *802.11n: Enterprise Deployment Considerations [http://www.wi-fi.org/knowledge_center_overview.php?docid=4568]* from the Burton Group gives a good overview of the issues that an Enterprise should consider when moving to 802.11n. The same report is also the source for the following comparison. Remember though, that 802.11n performance depends on a number of factors; some of the numbers will depend very much on equipment and environment, and these 802.11 numbers are "theoretical maximums".

Table 30.1. Comparison of Different Wi-Fi Protocols

	802.11b	**802.11g**	**802.11a**	**802.11n**
Maximum signaling rate	11Mbps	54Mbps	54Mbps	300Mbps[a]; up to 600Mbps[b]
Operating frequency band	2.4GHz	2.4Ghz	5GHz	2.4GHz & 5GHz
Range	100 m	100 m	100 m	150 m[c]
Non-overlapping channels (varies by country)	3	3	23	3 (2.4GHz), 23 (5GHz)

[1] It is useful to realise that standards are simply a means of codifying existing practice.

	802.11b	802.11g	802.11a	802.11n
Interference sources	Bluetooth, microwave ovens, baby monitors etc.	Same as 802.11b	Cordless phones	Same as 802.11b/g at 2.4GHz. Same as 802.11a at 5GHz.
Standard approved	Yes	Yes	Yes	Yes
WiFi CERTIFIED™	Yes	Yes	Yes	Yes

[a]Assumes 40MHz channel and 2×2 MIMO
[b]Assumes 40MHz channel and 4×4 MIMO
[c]Assumes 50% range improvement over 802.11g/a

For further authoritative information of what makes 802.11n different, I strongly encourage you to read a whitepapeg (registration required) released by the Wi-Fi Alliance on *Wi-Fi CERTIFIED™ 802.11n Draft 2.0: Longer-Range, Faster-Throughput, Multimedia-Grade Wi-Fi® Networks [http://www.wi-fi.org/knowledge_center_overview.php?docid=4590]*

30.2. When to Use 802.11 Wireless

Wireless is a great technology, I use it every day with my Mac laptop that has inbuild wireless. I don't even need to think about it. It just works, and that's a sign of a successful technology, that is successfully integrated in the user experience. Here's some trivia from the Wi-Fi Alliance's *Did you know… [http://www.wi-fi.org/knowledge_center/knowledge_center-didyouknow]* page:

- 68% of users would rather give up chocolate than do without Wi-Fi
- Users ranked Wi-Fi above YouTube®, MySpace®, Facebook® and iTunes® as technology having the biggest impact on their online experience.
- 56% of users will seek Wi-Fi in their next cell phone purchase.
- 1 in 3 look for it in music players, digital cameras and gaming devices.
- 1 in 4 would switch cellular carriers to get Wi-Fi data or voice plans on a converged phone.

30.2.1. Mobile or Roaming Networks

Naturally, any wireless technology, by definition, is best suited for when you don't want to be tied down with wires. Typical examples are laptops and hand-held computers. Sure, laptops can plug into Ethernet, and it's good to do so when you want to transfer large files, as there is still a great speed difference between wireless and switched 100Mbps Ethernet.

Roaming is being able to move between the presence of multiple access points, and still be able to access the network. Recently, the 802.11f standard was ratified to create a standard IAPP (Inter Access-Point Protocol). However, not all equipment supports this, so it is still useful to use equipment from the same vendors. Previously, you had to have the same vendor for wireless roaming to work.

Mobile networking is similar, and can support layer 3 changes (by using Mobile-IP), but not as seamless, so you can't maintain your address and TCP connections, whereas in a roamable

network you can. This is because roaming means a layer-2 transition, which is transparent to IP.

30.2.2. LAN-Size Networks

802.11 protocols are a LAN based protocol. If you want to connect on the road, a more appropriate, and usually far more expensive, solution would be to use a cell-phone modem, using a technology such as GPRS. There are other technologies, but that is outside the scope of this paper.

The 2.4GHz 802.11b/a/g protocols are, when using an omni-directional antenna, limited to about 30m indoors and 100m outdoors; but these are ideals and you'll find performance at these distances to be quite unsatisfactory. 802.11n has a range typically 150% that of 11b/a/g protocols. The range can be extended by using different antennas, which change the shape of the coverage volume, usually either making it into a cone (directional), or a flattened sphere (omni-directional).

30.2.3. Directional Links

Directional antennas are all the rage in networking applications where you want to hook up branches of your network that are not in the same building. For instance, wireless links could be used to cheaply connect sites in a Metropolitan Area Network; running cable through streets can be very expensive. Alternatively, you could connect sites using a Virtual Private Network (VPN) over the Internet or a "leased line" from an telecommunications provider.

Wireless provides a cheap solution but is *considerably* more difficult to troubleshoot. Consider trying to move a building out of the way to see if the link quality improves. It is this reliability issue that may rule out wireless for primary links, but still be useful as a backup or transient link.

There are systems other than those in the 802.11 suite of protocols that are designed for high-speed point-to-point wireless networking. 802.16 (WiMax) is a Metropolitan Area Networking (MAN) standard designed for point-to-point applications at a speed of up to 280Mbps. There are also laser-based point-to-point networking technologies, but I won't go into those here.

30.2.4. Backup Links

When an important link goes down, the routers at either end of the link could switch to using a wireless link as a backup, until the primary link comes back up. Network downtime can cost thousands of dollars per hour[2], so if someone severs a cable (*backhoe interference*) running under the streets between buildings, the network can still function, perhaps not as well depending on performance requirements, but the mission critical work can still go on.

30.3. When *Not* to Use 802.11 Wireless

Like all technologies, there are scenarios where 802.11 wireless is a good fit, and scenarios where it is quite unsuitable. Unfortunately, wireless has a lot of times where it can be quite unsuitable, especially when its weaknesses are not well understood, or the skills to properly manage it are not available.

[2]Something less than number of staff × average salary per hour × hours network is unavailable + lost earnings.

I've seen a lot of people in newsgroups seriously thinking about using a wireless solution to network their home. The main rational was that it was cheaper than getting in a builder to lay the cable. People in this position often don't know what they are letting themselves in for.

Wireless certainly *is* useful for everyday home networking, but it is good to wire where you can. Wireless can be very difficult to troubleshoot when you're dealing with reception and interference issues, especially when they are temporal.

Professionally retro-wiring a house can be expensive (especially if it is a historic building), so its a very good idea to make sure you get your house wired either when you build a new house or renovate. If you use conduit, rewiring will be much cheaper, and provides an easier upgrade path.

Even if you use older Cat5/100Mbps equipment to build you home network, if you can lay the cable yourself you'll save yourself a lot of money and time. A cheap, though aesthetically unappealing home network can cheaply be made with a few lengths of cable and a switch. You do need to aware of though the issues that can pop up with poor quality Ethernet cable when you have gigabit interfaces on the network. Keep an eye on the error counters during heavy traffic.

Access points and Wireless NICs can be expensive, although they are getting to a price point comparable with an 24 port Ethernet switch and this doesn't include the greater price of the cabling! However, despite the monetary parity, the performance and robustness that an Ethernet network gives you is very much superior. This is especially true if the users in that network want to use protocols that are sensitive to packet loss, or do a lot of large file transfers, although 802.11n offers Quality of Service (QoS) improvement using Wi-Fi MultiMedia (WMM) and Wi-Fi Voice-Personal.

So a wireless home network can be a welcome *addition* to a home network and it can be useful for the entire network in many SOHO networks. It can also be useful for transient networks such as a mobile computer lab.

30.3.1. Reliability

Wireless link quality can vary quite largely, both in space and time. This is influenced by environment, equipment and use. Isolating the cause of wireless failure can be very difficult, with a lot of guess work and costly experimentation.

When you have a solution prone to reliability issues, the perceived value of that solution plummets. Nobody wants to use a solution that is prone to breaking down often, especially when they don't have the knowledge to fix it themselves, so its very important that a thorough site survey is carried out and the access point is located in a suitable place.

30.3.2. Security

Finally, wireless networks are a security risk, unless proper precautions are put in place to strengthen it. Some people believe that it is a good thing to let the public use their connection; and some people will happily use any network they can get on. Let's pause for a moment and think about the possible negative aspects of this. Some of the risks of running an access point when you don't understand the security aspects are as follow:

- People could steal your Internet bandwidth, using it for sharing illicit and illegal material. This can have an enormous impact on your Internet bill or available quota. Some people may use your bandwidth for casual Internet activity too, but it's still dishonest if not illegal.

Because it can be traced back to your IP address you are legally held responsible for whatever happens, unless you can prove that it wasn't you.

- People could use your wireless network to break into the computers inside the network, which can be (mis)used as above, or for stealing data, spreading viruses, etc, etc. If this doesn't scare you, consider the case of an Internet café, which was struck by a virus that installed a keyboard logger that activated whenever someone visited their Internet banking site. Encryption between the computer and bank doesn't help there, as keyboards aren't encrypted.

In order to protect the wireless network, and any networks beyond, from access by unauthorised users/devices a number of security protocols have been used. Some, such as WEP have been found to have major flaws and should not be used; updated protocols such as WPA2 should be used instead. We will cover these options in much greater detail when we look at configuring access points.

Unfortunately, although all access-points support some sort of security protocol, very few actually ship with it enabled or in some secure-by-default configuration. Even more unfortunately, according to *Wi-Fi Protected Setup [http://www.wi-fi.org/wifi-protected-setup]*, recent Wi-Fi Alliance research indicates that 44% of Wi-Fi users report that enabling security features on their Wi-Fi networks is moderately or very difficult. This has two major results: a lot of product returns affecting the sales chain, and a lot of networks not having security enabled. We shall also have a brief look at how this issue is being resolved.

30.4. Interference and Absorption

The two common problems to radio signals are interference (including reflection, although 802.11n turns this liability into an asset) and signal absorption. There are many possible sources of interference, and these may aggregate to give you problems such as a link failing or performing very badly periodically. This is a real-world issue. Nearly every consumer grade wireless ISM-band product (not including televisions, radios and cell-phones) can cause interference, or otherwise interfere with the clean propogation of signals. Examples are as follows.

- Cordless telephones (not cell-phones). This will appear as an audible buzzing during a call. Try pressing the channel button on the handset a few times until it gets to a relatively quiet channel. Older cordless phones may not exhibit this, as they use the lower ISM band, about 700 MHz. Many modern cordless phones use the 5 GHz spectrum, which has less use and more available channels.

- Microwave ovens. Pretty much all of this will be shielded, so if its giving you problems, get it tested or replaced.

- Remotely controlled toys.

- Wireless headphones (not the Infrared ones, obviously).

- Cosmic radiation (no, I'm not joking).

- Metal window blinds.

- Polarised windows.

- Multipath reflections/fading.

- Metal office furniture.
- Walls and floors/ceilings, especially concrete walls, as they have metal reinforcing mesh.
- Early Bluetooth equipment. Versions earlier than 1.1 had problems co-existing with Wi-Fi equipment. However, although they are now aware of each other they still contend for access to the spectrum and so can slow down the wireless network.
- People.

Most things absorb radio frequencies, the largest thing we most need to contend with are walls and metal furniture. Concrete walls are a large problem, as they have steel reinforcing mesh, and the steel is particularly good at absorbing radio frequencies. When you are surrounded on all sides by reinforced concrete, including the ceiling and floor (as many office spaces are), you effectively have an imperfect *Faraday cage*, which is particularly good at isolating isolating wireless communications. The effectiveness depends on the wavelength and the spacing of the mesh.

Interference can come from unlikely sources too. Polarised window glass can absorb microwave frequencies, as can various plastics, wood, and human flesh, which you notice when you use a PCMCIA wireless card on your lap (which is an excellent reason to buy a laptop with a builtin wireless antenna, even if you don't yet have a NIC installed). One way of estimating if a particular material will absorb 2.4GHz frequencies is to give it the microwave test: will it get hot (absorb energy) if I put it in the microwave for about 30 seconds? This is why flesh is such a good absorber of radio energy in the microwave part of the radio spectrum, as we are primarily composed of water, which microwave ovens were designed to heat by excitation.

Warning

Never put metal in the microwave.

For outside networks, especially for unidirectional links that span a distance, weather can present a problem, so be sure to test it during rainy or foggy conditions. Because of this, humidity is also a factor.

30.4.1. Interference Robustness

One common feature of wireless access points and client devices is the option to enable *Interference Robustness*. This essentially tells the device not to use the 1Mbps bit rate[3], which, due to the longer timeframe of a single packet, can't help but be interfered with during interference commonly caused by microwave ovens.

30.4.2. RTS/CTS

This is the Request To Send / Clear To Send type of mechanism that is used for serial communications. It is used differently however. It is not used for flow control, rather it is useful when you have two nodes N1 and N2, which can both see an access point AP1, but cannot see each other. This is called the *hidden node* problem. If N1 sends a packet, and N2

[3] 1Mbps and 2Mbps are called the basic rates, and are used for management frames such as access-point beacons etc. They come from the very first 802.11 protocol (simply "802.11") before 802.11b was introduced.

sends a packet while N1's packet is still in transit (N2 cannot see N1s transmission, so the channel appears available), they will interfere with each other when AP1 sees the packet. So, by turning on RTS/CTS, N1 sends a short RTS, and may only send its data packet when AP1 send back a CTS packet. This does drop performance a little, so is only useful when the number of users and traffic increases.

One option that is available is to use an RTS/CTS threshold, which means RTS/CTS will be used only for packets larger than a configure threshold, giving the best of both worlds.

30.5. Configuring Access Points

There are a few ways that an access point can be configured: an embedded web-server; telnet; and SNMP (Simple Network Management Protocol, covered later in this paper), though usually read-only in consumer grade equipment. Finally, some, notably the Apple Airport family of base stations, use proprietary software and appear to use no standard mechanism to configure themselves. However, they are much easier to find on the network.

30.5.1. Resetting Access Points to a Factory Defaults

There are two ways to set the D-Link access points to factory defaults, and this is true in general for most devices of this class. You can use the configuration interface to restore factory defaults, or you can tell the hardware to reset the defaults. The former method, of using the configuration interface, is only of use when you a) can figure out which IP address the device is to to be found, and b) have sufficient administrative access to the device.

What follows are some instructions for particular types of devices. Notice that this may or may not be documented in the manual that comes with the product, but it would be very rare for this support not be available in this case.

Procedure 30.1. Resetting D-Link Access Points to Factory Defaults

1. Power-on the device, and wait for it to complete boot-up.

2. Take an un-bent paper-clip, or similar, and hold in the reset pin at the back of the unit for 10 seconds, or thereabouts.

3. The device should now be set to defaults. The default IP address is 192.168.0.50/24 for access-points (typically a single Ethernet port) or 192.168.0.1/24 for access-points with router functionality (typically multiple "LAN" ports and a "WAN" port), and to access the configuration interface, you can browse to it, and authenticate as "admin" with no password.

   If you have difficulty getting to the device after you have reset or changed the settings, check that your local interface (still) has a compatible configuration and that you are navigating to the correct address.

4. Note the some D-Link access points can also be configured via telnet or SSH, although configuration over the Web interface is most usual.

5. If you have difficulty getting the device to reset itself, the device may have a bad power-supply. Hopefully this should not be the case for this lab.

Procedure 30.2. Resetting Cisco Access Points to Factory Defaults

Cisco devices, like other Enterprise class devices do not have a reset button. This is because they have a boot manager in the firmware. To reset such a device you access the firmware (Cisco calls this the “ROM Monitor” and change the startup settings.

1. Connect to the device using a serial terminal cable, typically at 9600 8N1.

2. Use a serial terminal emulator such as HyperTerminal, **minicom** or even **screen**[4], access the device.

3. Reboot or power-on the device. Before it starts loading the operating system you need to send a serial *break* signal. You will need to check your serial terminal emulators instructions to find out how to send this. This causes the *ROM Monitor* (which is a bit like a PC’s BIOS) to not load the operating system and instead interact with the user.

4. Change the *configuration register* so it doesn’t use its default configuration upon startup. The configuration register is a set of flags that is stored in non-volatile memory. One of the flags says whether to load the stored configuration or to use factory defaults. This is commonly used to reset a device after you have forgotten or otherwise lost the password to access or configure the device.

   A suitable command might be **confreg 2142** in order to boot the device ignoring its configuration, and **confreg 2102** for setting it back again after you have reinitialised its settings. Note that the specific command will depend on other factors, but this is fairly default behaviour. *The Purpose and Use of the Configuration Register on All Cisco Routers [http://www.cisco.com/application/pdf/paws/50421/config-register-use.pdf]* gives further details.

I mention this because this is a semi-typical way of resetting devices with no reset-button. Since we don’t have any Cisco Access Points or equipment in this lab, I have not gone into details regarding this.

30.5.2. IP Settings

Even if configured as a bridge (a layer 2 device much the same as a switch, only it translates wireless to Ethernet in this case) access points will still have an IP address for management purposes. As such, it has all the usual IP configuration details you would find for an Ethernet card, including hostname, address, subnet mask, gateway and DNS details.

30.5.3. ESSID / SSID

The two acronyms refer to the same thing. The Wi-Fi Alliance seems to prefer SSID.

The ESSID is the name of the wireless network. Clients will specify this name when they want to *associate* with your network. Alternatively, a client may specify the ESSID “any”, meaning they will try to associate with any access point. Such scenarios may be useful for hotspots, where the user will not know what ESSID they would have to use.

In a network set up for roaming using a distribution system[5], a client device can roam (and maintain it’s connections) between access-points with the same ESSID. Each access-point has its own hostname which should be registered in DNS using some suitable naming scheme.

[4] **screen /dev/device 9600**
[5] The Distribution System could be based on wired or wireless networking.

30.5.4. Channel

Depending where you live in the world, there will be a certain number of channels that you may operate on, between 7 and 15. 802.11b/g devices use three contiguous channels at any time, so if you select to use channel 4, then the AP will also use channels 5 and 6; you need to ensure that access points use channels at least three channels apart, so they don't interfere with each other, reducing the bandwidth.

30.5.5. WEP Settings

WEP (Wired Equivalent Privacy) is a very poor security protocol that is one great reason why Wi-Fi deployment has been very slow in the commercial sector. There is a newer, temporary enhancement called WPA (Wi-Fi Protected Access) that has been introduced.

WEP uses a Pre-Shared Key to control access. This is used to encrypt the packets over the air. It is *not* end-to-end encryption, as the access point will decrypt it when it puts it onto the wired network behind it (usually Ethernet, but it could also be a dial-up connection.)

Using a pre-shared key, means that if you want to update the key on the access point, you need to update the key on all of the devices associating with it, which is not feasible for large networks. WPA-Enterprise offers a much better solution for such environments.

Access points generally offer two or more key-sizes for encryption, 64-bit (crippled to make it effectively 40 bits), and 128-bit (crippled to 104 bits). Since there is now a known attack against WEP that means the keys can be easily recovered after about a million packets[6], each is equally useless. 64-bit is the only really standard size, so is more compatible.

Apple software uses a passphase to make WEP easier. This just means that a particular transformation (hash) is applied to the passphrase to make it generate the key. Vendors are free to implement this as they see fit, so it will pay to find out what the WEP key is once you have configured a pass-phrase.

30.5.6. WPA

802.11i was still being worked on at the time when WEP was utterly broken, so some urgent work was needed to patch up the crumbling confidence in Wi-Fi networking at the time. Thus, WPA was created, which is a subset of the 802.11i standard, made to be fairly compatible with most existing hardware and drivers. Windows XP users will require at least Service Pack 2 for WPA functionality, as well as updated drivers and access-points etc. Not all devices can be migrated.

WEPs biggest weakness was that it did not change its keys periodically, so an attacker armed with software that made use of the cryptographic weaknesses in the RC4 cipher, could easily break in once they had collected enough packets (that had all been encrypted with the same key). WPA, in order to provide a firmware-upgrade path, still uses RC4, but changes the keys periodically, using a protocol called TKIP (Temporal Key Integrity Protocol). How this works is outside the scope of this lab. The key timeout, which is set on the access point, is often defaulted to 60 minutes; it is considered good practice to lower this to 15.

WPA, unlike WEP, has two authentication architectures: a pre-shared-key variety called WPA-Personal, and a centralised username/password[7] variety called WPA-Enterprise.

[6] The program **airsnort** can do this.
[7] Actually, it doesn't have to be a password, other authentication technologies are regularly used.

WPA-Personal Settings

This is easier, and naturally much safer, than WEP. All you need is a passphrase of no more than 63 characters. WPA-Personal is also often called WPA-PSK or “Pre-Shared Key” in many vendor offerings. Unlike WEP, the pass-phrase is handled uniformly. Because there are few limits on the passphrase, it is easier to remember. For example, with WEP the key-length must be either 5 or 10 ASCII characters, depending on the key-length. It’s much easier to remember a passphrase such as “Carrots & Cabbage”.

It appears that some access-points offer the ability to use a “hex” key instead of a passphrase. Do not use this, as it is non-standard behaviour and you will find that various devices will not be able to join the network.

WPA uses a standard key derivation function which uses the passphrase as well as using the SSID as a “salt” value; this means that two networks with the same passphrase (but different SSID) will have a different key.

In order to determine what the WPA passphrase is, the attacker would, in effect, factorise a very large number. This is, by design, very expensive to do, requiring a lot of time. It is possible to change this time tradeoff for a space tradeoff and pre-calculate a “rainbow” table of keys with a passphrase that generates said key. Because the SSID also enters into this calculation a rainbow table can be pre-calculated for common SSID values. Indeed, there exist DVDs of these rainbow tables in order to break this security fairly easily.

The catch is that it takes a very long time and a lot of effort to calculate such tables although you can speed this up enormously such as by using the power of modern graphics cards. To prevent this from becoming a problem it is a good idea not to give your network a common or default network name, and to change your passphrase ocasionally. When this is a problem due to the number of users it would affect, then WPA-Enterprise should be used.

WPA-Enterprise Settings

WPA-Personal still shares one potential issue with WEP; it still uses a pre-shared key, which makes it unsuitable for use in Enterprise deployments: consider the cost of updating the keys if it is compromised. This is where WPA-Enterprise comes into it. WPA-Enterprise uses 802.1X and RADIUS to authenticate users, not machines. It doesn’t have to use username/password though, or even authenticate users; the EAP (Extensible Authentication Protocol) that employs 802.1X and RADIUS allows the use of other authentication mechanisms, such as TLS certificates.

Some of this alphabet soup will be covered later in the paper when we investigate authentication in greater depth. In this lab, we shall just concern ourselves with the basics suitable for understanding the SOHO networking environment.

First some basic 802.1X terminology: the client is called the *Supplicant*, as it supplies information to the *Authenticator* (the access-point), which talks to the *Authentication Service* (the RADIUS server).

WPA-Enterprise still uses TKIP, so its good to reduce the key timeout.

802.1X is not specific to wireless. Indeed, it was originally designed for port-based security, where a port is a physical device, such as an Ethernet switch-port. The port is abstract when it applies to wireless, so a dummy value may need to be supplied, depending on your RADIUS configuration.

Before we leave the Enterprise world, let’s just try and confirm our understanding of the various components mentioned.

EAP
: The Extensible Authentication Protocol comes from the world of dial-up networking (in particular the Point-to-Point Protocol or PPP). It was used to provide greater flexibility for creating authentication mechanisms. EAP is not tied to PPP so EAP can be used for other network authentication tasks. EAP is a framework-based protocol, and gets its flexibility from the use of different modules, such as EAP-TLS, EAP-TTLS and EAP-PEAP. EAP can also use its own authentication mechanisms for existing authentication mechanisms.

802.1X
: This is an IEEE protocol that was adapted from the IETF's EAP. So we don't hear much of EAP with regard to wireless security, but we do hear a lot of the EAP modules such as EAP-TTLS.

  In WPA-Enterprise 802.1X is used between the client ("supplicant") and the access-point ("authenticator").

RADIUS
: The Remote Authentication Dial-In User Service is used between the access-point ("authenticator") and the "authentication service" that knows about the users and authentication mechanisms etc.

EAP-TLS
: One of the common EAP authentication mechanisms that are used in wireless security.

  This protocol uses digital certificates (the same technology used to protect secure web-sites) and authenticates both the client and authentication-server (to ensure the user is sending credentials to a legitimate server. Both the client/user and the authentication servers need certificates, which is a lot of work, so EAP-TLS is not used that often.

EAP-TTLS
: EAP Tunnelled TLS supports "legacy" authentication mechanisms that are commonly used to authenticate users. It uses an "outer" and "inner" authentication protocol. The outer layer uses TLS to verify the legitimacy of the authentication server and the inner layer is an existing authentication mechanism that is encrypted by the outer layer.

  This is a commonly used module which is quite similar and a bit more flexible compared to EAP-PEAP.

EAP-PEAP
: Protected EAP is another very common EAP module. It is a recommended upgrade for anyone using the older and less secure LEAP.

LEAP
: Cisco's proprietary Lightweight EAP was used prior to WPA to enable user-based authentication in the days of WEP, which didn't support it at all. Because it uses a defunct vulnerable protocol (MS-CHAP) it's use is strongly discouraged and a stronger protocol should be used, such as EAP-PEAP.

As you can see, Wi-Fi networking is aflood with acronyms. The really scary thing about this is that users are often exposed to these details. Users have difficulty with just SSID and PSK. Wi-Fi suffers from a serious lack of user-friendlyness.

WPA2

This is the Wi-Fi Alliance's term for conforming 802.11i equipment. Conceptually it is very similar to WPA except WPA2 implements the full 802.11i standard, and some older interface

cards may be incompatible. Technically, it is a higher level of protection compared to WPA as it uses the Advanced Encryption Standard AES.

Windows XP users will require Service Pack 2 as well as the following patch [http://support.microsoft.com/kb/893357].

30.5.7. Bridging or NAT

Every access point I've seen has the capability to act as a bridge, or to share one address between everything behind a NAT (the latter requires an AP-router, not just an AP, but most APs include routing support for about the same price). We will have a lab on NAT later. Configuring it as a bridge can be preferable in a small network, mainly because resource discovery tasks such as network browsing relies on broadcast traffic, which would otherwise not be passed through on a router, making nodes on the wireless network have difficulty easily locating devices on the wired network.

30.5.8. Port forwarding for NAT

As is common with NAT, you can elect to forward certain ports to machines inside the NAT. This may be called NAPT or Port Forwarding, or something else. Beware that port forwarding will interfere with various Peer-to-Peer (P2P) programs, including various multimedia communications.

We shall talk about NAT more in detail later in the paper.

30.5.9. Offer Addresses Using DHCP

Access points offer simple DHCP services for distributing addresses to clients. Access points can also generally use DHCP to get an address for themselves.

Because DHCP service is generally enabled by default in an AP, it is *very bad* to plug an AP into a production network without it being configured first. This is because you'll end up interfering with any existing DHCP server. We'll cover this problem in more detail when we talk about DHCP later in the paper.

30.5.10. Access Control

Consumer-grade devices usually offer some access-control based on the MAC addresses, which is usually printed on a sticker on the wireless card. This is okay for casual security, but you should be aware that there are ways to change the MAC address for cards, and that it's not difficult to sniff the wireless traffic and find out what MAC addresses are in use on the network.

Some access points offer the ability to import a list of MAC addresses from a RADIUS server.

30.5.11. Less Common and Non-Standard Features

Many access points offer a non-standard feature called a *closed network*, which just means that the network ESSID is not advertised, so it won't be seen in browse-lists. This does *not*

mean that your network is invisible, as anyone with a wireless packet sniffer, such as Kismet or NetStumbler can easily detect it when packets are sent over the network. Therefore, it is only of use on very quiet networks.

Some vendors, most notably D-Link in our region, offer a Turbo mode, that is non-standard and requires D-Link equipment on all devices concerned. It works by using more than the three channels, so it will likely cause interference or contention when you have two or more access points in range, due to a lack of channels.

30.5.12. SNMP Authentication Traps

If someone tries to log in unsuccessfully to the access point to configure it, an SNMP trap (an alert message) can be sent. We'll learn more about SNMP in a following lab.

30.5.13. Syslog

Many access-points offer the ability to log events, such as association attempts and various errors to a syslog server. We shall learn of this type of remote logging in a later lab.

30.5.14. Broadband Router/Wireless Combos

More and more commonly, access points also act as broadband routers. Some may have an Ethernet port for connecting to a cable modem, others may offer DSL or dial-up connectivity. Naturally, the specifics will depend on the broadband technology used. Often, a username and password is required.

The Apple Airport has an interesting feature whereby a user can dial-up to their own home from somewhere else to access their network.

30.5.15. Multicast Rate

Apple Airports have a feature to allow you to set the rate at which multicast and broadcast traffic will be sent by the access point. Because broadcast and multicast traffic go to multiple users, they must go essentially at the slowest pace. The best way to configure this is to take into account the lowest bit rate you achieve in your coverage areas. This will be useful in the future when applications such as Video-over-Wi-Fi become more prevalent, as will likely happen with emerging devices such as the AppleTV.

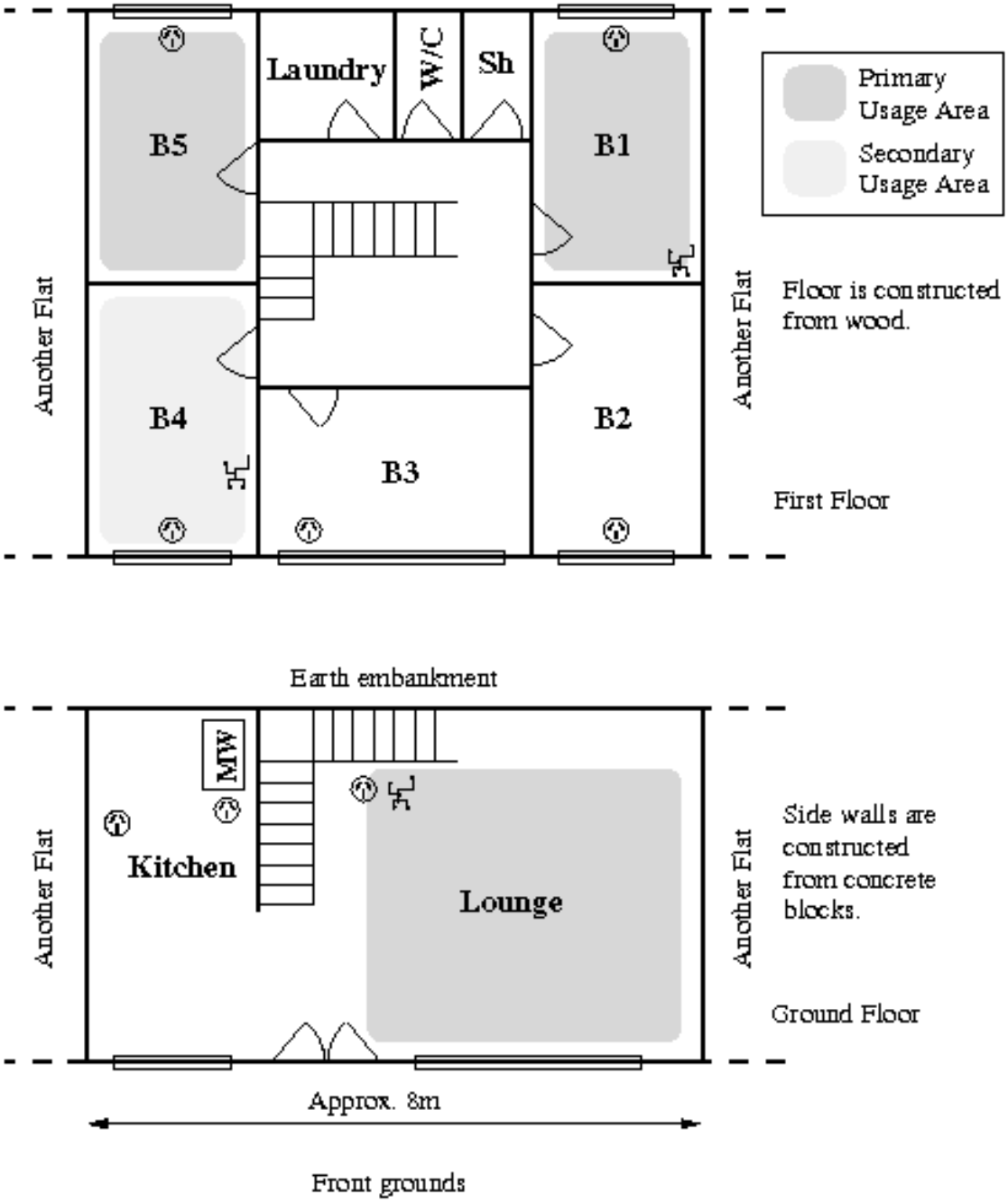
30.6. Locating Access Points

1. Make a floor plan, taking note of the location and composition of walls, office furniture, network and power outlets, etc.

2. Find out where on the plan coverage is required, and how many users will be using the access point in a each area.

3. Try to find out where you can put your access point(s) so that you maximise your coverage and minimise obstacles. You may want to use different kinds of antenna for certain areas. Figure 30.1, "Wireless Site Survey" shows you an example of what your floor plan might look like. Remember that walls may present a very thick obstacle depending on the angle the signal enters the wall.

Figure 30.1. Wireless Site Survey



Locating an access-point can involve a lot of tradeoffs including power and network connectivity, range and coverage areas.

4. Site your access point, pointing it in the right direction if required.

5. Go around with a laptop, at various places in the coverage areas. Try not to let the link quality drop below half for good results. Try putting it down on the ground, sitting down with it, placing yourself between the access point and the card, turning the computer (PCMCIA cards have a poor *polar diagram*, meaning at some angles, reception lulls badly.)

If you want to roam on your wireless network, you need to move around and test the handoff between access points.

6. Its a slow process, involving a lot of fiddly playing around, but the golden rule would be "lather, rinse, repeat". But if you only need the wireless access to access the Internet, you can be more forgiving of poorer signal quality, depending on the speed of your Internet connection.

30.7. Assessment

1. You will be given an access-point of unknown configuration. Your task is to do the following:

1. Reset the access-point to factory defaults.

2. Connect the access-point to your management computer with an Ethernet cable.

3. Configure your management computer to have a different address that is suitable for use with the access-point's default configuration. For example, if the access-point defaults to 192.168.0.50/24, then set the ethernet port of your computer to have an address of 192.168.0.51/24 or similar.

   Note that the D-Link access points default to 192.168.0.50/24, while the D-Link wireless routers default to 192.168.0.1/24.

   Note also that if you use **ifconfig** to temporarily configure the IP address of the configuring computer, you may find that the address goes away whenever the access-point reboots (as the cable appears to be unplugged briefly when the device reboots). Thus, you will need to reconfigure the IP address every time you reboot the access-point or change the access-point's IP. If you are annoyed by this, you may choose to disable the Network Manager as we did in a previous lab.

4. Start up a web-browser and connect to the management interface of the access-point. eg. http://192.168.0.50/

   The user is "admin" and there is no password by default.

5. As this lab is largely about playing around with wireless networking, take a look throughout the interface and see what you can configure. Check the help or ask a demonstrator if you are unsure of a particular configurable.

6. Affect the following configuration:

   - Change the administrative password to a good password, and document it.

   - Set the wireless security to WPA or WPA2, whichever is highest. Use the Pre-share-key (aka WPA Personal), and select and document a *good* passphrase.

   - Select a suitable SSID. Choose one such that it is likely to be unique.

   - Select an unused starting channel. There should be a list of available channels on the whiteboard. Write your group-members names on the table to co-ordinate spectrum management in the class.

- Configure the access-point to have the IP address 192.168.1.2/24. Set your configuration machine's IP address suitably. Don't forget that you will need to change which IP address you point your browser to.

  Configure no gateway (you may need to enter 0.0.0.0) and no DNS. If 0.0.0.0 is not accepted, enter the IP address of the access point.

- Enable the DHCP server to distribute addresses in the range 192.168.1.128—191. This corresponds to the third quarter of the 192.168.1.0/24 address space. This would mean we can use 192.168.1.128/26 whenever we want to match "wireless DHCP clients".[8]. Don't forget you need to enable the use of the pool addresses, as well as the use of DHCP in general on the D-Link access-points.

7. Test the configured access-point by connecting two computers to the wireless network and pinging each others IPv4 address. You can look up each hosts IPv4 address using **ifconfig** and ping them with the **ping** command, as practiced in the lab on basic interface management.

8. Ask your lab demonstrator for marking once you have answered all the questions in this assessment. You will be shown how you can use a tool for finding out what access-points are available, and how to connect to your network.

   If you have a laptop or other wireless client device available, try to connect yourself. Ensure that you are given an IP address via DHCP.

2. Imagine you are setting up a wireless network for a client. How might you evaluate the reliability of your network? You should think about the factors that impact the performance of the network in comparison of wired networks.

3. Add notes for 802.11ac to Table 30.1, "Comparison of Different Wi-Fi Protocols".

4. Suggest a location for a single access point to satisfy the needs of the site survey shown in Figure 30.1, "Wireless Site Survey".

5. [Optional] If you have a WPA2-capable wireless-enabled laptop, try registering it for use on the student wireless network. This will be using WPA2-Enterprise.

6. Explain the relationships between the following terms, you may need to do some research: WEP, WPA, WPA2 and 802.11i.

Explain the relationships between the following terms, you may need to do some research: RADIUS, 802.1X, EAP and EAP-TTLS.

7. Some new Wi-Fi devices often come with a feature called "Wi-Fi Protected Setup" make it easier to set up a secure wireless connection. Do some research (you must consult and cite only authoritative sources) and describe how this system works; in particular, describe the PIN and PBC methods. The Wi-Fi Alliance have a useful downloadable presentation *Wi-Fi Protected Setup [http://www.wi-fi.org/wifi-protected-setup]* covering this technology. There is also a whitepaper available, which requires registration, but goes into more detail than the downloadable presentation.

[8] Use **ipsc -a 192.168.1.128/26**

Lab 31 Effective Use of an Editor (vim)

> **Note**
>
> Virtualbox is not used for this lab. You can use any machine which has the **vim** program installed, such as your local workstation.

Today's lab is intended to be of the self-guided variety. The reason this lab is included in the schedule this year is to ensure that you have a certain degree of productivity in the environment we provide to you. There are a number of features any Unix power-user or administrator needs to have in their chosen editor in order to do things quickly.

Vim is a member of **vi** family. Due to the licensing of the original **vi** editor, all **vi** *work-alikes* on Linux systems now tend to be a subset of **vim** or another **vi**-like editor, such as **nvi** or **elvis**.

The reason I want to teach you **vim** here is because you already have some (albeit minimal) exposure to GNU Emacs, and I wish you to train your mind to be open to different approaches, and of not fearing things because they are different or unfamiliar. People with some fluency in **vi** already will likely find something useful in this lab also. If you are already well achieved in using **vim**, you can do the assessment in Section 31.4, "Assessment" directly.

31.1. Vimtutor

You can complete this section on your local workstation. Because we aren't using Client1 for this lab, you can work on this lab while you work on another lab at the same time, in order to schedule your time more effectively.

The default behaviour of **vi** can be rather awkward for the beginner, so I recommend you put the following in your `~/.vimrc` to remove some of the awkwardness of **vi** from **vim** before you dive into the tutorial. You will likely need to create this file. Since you have not used **vim** yet, you may prefer to use **nano** to edit the file.

```
set nocompatible
```

When you have made the change, from a terminal, run the command **vimtutor**. This will start up **vim** on a copy of a tutorial file. This file has teaching instructions inside it which will get you well on your way to mastering the basics. Follow the instructions till you have completed the tutorial.

> **Note**
>
> At the end of the tutorial, you don't need to change `~/.vimrc` as suggested. You may like to read Section 31.3, "Customising Vim" to find some suggestions as to what you can put in your `~/.vimrc` instead, if you are interested.

31.2. Some Vim Tricks

Copying and Pasting

On a Mac, you can use **⌘C** and **⌘V** to copy from elsewhere and paste into Terminal.

On Linux systems, using the Terminal application, you can use **Ctrl+Shift+C** and **Ctrl+Shift+V** instead.

Additionally, when using VirtualBox on a Mac, you can configure your mouse so that clicking on the scroll-wheel will send a Button 3 event (which is a middle-click). This way you will be able to select and paste into the terminal with a middle-click. Instructions for configuring this behaviour can be found in the Pre-lab material material near the start of the lab-book.

Pasting in pre-formatted text into **vim** can be troublesome if you just paste while in insert mode, because the auto-indenter may get in your way, and sometimes you end up with stair-stepped text, such as this:

```
Line 1
      Line 2
            Line 3
```

Other symptoms can appear, such as auto-expansion of particular phrases. You need to remember that when you are in 'Insert' mode, and you paste into the terminal window, Vim has no real of knowing the difference between a paste operation and a lot of typing.

Although these appear not to be a problem on the particular version and configuration of Vim you will be using, it can be problematic on other systems or in other accounts with different Vim configurations. To prevent this, use `:a` in **Vim** instead (remember to hit **return**). Then paste with **⌘V**, press **Enter** to ensure the cursor is at the beginning of the line, and then hit either **Esc** or **^C**. The pasted text will then be input into your document without any auto-indenting.

You can **include the output of a command** by writing the command into the text, selecting the lines of text with `V` (Visually select by lines) and then typing `|sh` to run the command, replacing it with the output of the command.

More generally, we can see that the selected text is input to a particular command; it needn't be shell commands written in the text. For example, if you wanted to filter the selected text such that only lines matching a particular pattern were retained, you could use something like `|grep pattern`. We learn more about such commands in the lab on scripting, and regular expressions are covered in the lab on post-installation (log filtering).

Global search and replace is a very useful feature when paired with the power of regular expressions. A Vim command such as `:%s/regex/replacement/flags` will replace text on all (`%`) lines. The flags that are common are: `g`, to replace all occurances on a line, not just the first one; and `c`, to ask for confirmation. Use `:he :s` to find more help information. You may have noticed that commands can be abbreviated, so long as they are not ambiguous.

Document-specific configuration is often seen when working with other people's source code. In powerful text-editors, such as Vim and Emacs, you can instruct the editor to set particular preferences for how a file should be edited, by putting such instructions in a

comment. Here is an example, which you might find in a C or Java file to tell the editor to set the tab-width to 4 spaces when editing this file:

```
/* vim: softtabstop=4 shiftwidth=4 expandtab
 */

#include <stdio.h>

int main(int argc, char *argv[]) {
    puts("Hello, world!");
}
```

Note that for this to work, the directive `set modeline` needs to be specified in your `~/.vimrc`, otherwise it won't be processed. For security reasons, not all Vim commands can be specified in a modeline. For more information, see `:he modeline` and `:he design-not` inside **vim**.

31.3. Customising Vim

This section is optional. It gives you some ideas about what can be done through configuring **vim**. If you don't have much time now, you may skip over to Section 31.4, "Assessment".

The behaviour of **vim** can be customised by putting vim commands in `.vimrc`. This can be very useful to remove some of the original **vi**'s awkwardness. Here is a (slightly simplified) version of my own `.vimrc` to show you some of what is possible, and to give you ideas.

Inside **vim**, you can get help on a topic by using the `:help topic` command. So for example, `:help autowrite` will tell you about the third line below. Use `:q` to exit the help.

> **Important**
>
> You are *not* expected to put all of this into your `~/.vimrc`. It is there to give you ideas, and to help send you on your journey.

```
set nocompatible
set scrolloff=5
set autowrite
set softtabstop=4
set shiftwidth=4
set expandtab
set viminfo='20,\"50
set history=50
set modeline
set modelines=5
set backspace=indent,eol,start
set hlsearch
set incsearch
set ignorecase

filetype on
syntax on

filetype plugin on

" By the way, this is a comment. A bit unusual to use
" a double-quote for a comment, but there you go...

" Set an auto-command when we enter a buffer
" to use a particular indent style for different
" file extensions.
au BufEnter *.c set cindent
au BufEnter *.h set cindent
```

```
au BufEnter *.cpp set cindent
au BufEnter *.java set cindent

" Specify auto-formatting options
set formatoptions=tcroqn

" Setting different options depending on the file type
au FileType html set smartindent
au FileType tex set smartindent
au FileType text set smartindent
    \ formatoptions+=a
au FileType perl set cindent
    \ formatoptions-=t
    \ cinkeys-=0#

" When in normal mode, map some keys to particular actions.

"   F5 runs the make command (see also: autowrite)
nmap <F5> :make<cr>
"   Shift-F5 runs a full make
nmap <S-F5> :make clean ; make<cr>

"   Many of the : commands accept a range specifier to select which
"   which lines to operate on. % selects all lines.
"   The selected region (ie. all the lines) are then fed into the
"   indent(1) command, which indents C code according to a specification
"   which is typically stored in a file called .indent.pro
nmap <F6> :%!indent<cr>

"   When editing multiple file (vim file1 file2 ...), set F7 and F8 in normal
"   mode to browse to the previous and next file.
nmap <F7> :previous<CR>
nmap <F8> :next<CR>

" Here is an example of creating a useful macro to simplify common workflows.
" This one is useful for swapping a line with a line two lines down.
" I used it when editing HTML code such as the following, and wanted
" to swap easily between having a link activated or not.
"
" <!--
"   foo                                 cursor would be on this line
" -->
"   <a href="foo.html">foo</a>
"
" Ensure you can understand how the macro below works to do this.
"    dd = delete a line
"    j  = down (if you have trouble remembering the diff. between j and k,
"               j looks closer to a down arrow.)
"    p  = paste deleted line after cursor
"    k  = up a line
"    dd = delete a line
"    k  = up a line
"    P  = paste deleted line before cursor (capital P)
nmap <F2> ddjpkddkP

" When in normal mode, I want space to be like PgDn (or C-d rather)
nmap <Space> <C-d>

" The Q command falls into the old ex mode, which is a common trap.
" I prefer to remap it to something more useful, such as justify.
map Q gq

" Rejustify a paragraph of text (VERY useful)
nmap <F4> gqap

" This will make the default lexical highlighting colours more readable if
" you use a dark terminal background (programs can't figure this out for
```

```
" themselves)
" All I ever use is a black background, so this is ok, but you may want to
" uncomment it. You can always enable it in-session using ':set bg=dark'
set bg=dark
"set bg=light

set ruler  " show the cursor position all the time

" Suffixes that get lower priority when doing tab completion for filenames.
" These are files we are not likely to want to edit or read. These  are
" mostly related to LaTeX editing.
set suffixes=.bak,~,.swp,.o,.info,.aux,.log,.dvi,.bbl,.blg,.brf,.cb,
set suffixes+=.ind,.idx,.ilg,.inx,.out,.toc
```

31.4. Assessment

The assessment for this lab is in the form of a “follow me” video which is made available in the “Resources” share. You basically practice what is shown in the video until you are confident enough to do it without referring to the video, then when you are confident, have a demonstrator come and watch you do it.

31.5. Final Words

That should be enough for this lab. I expect pretty much everyone will have gotton *some* benefit from this lab, even people who have used **vi** or **vim** for a long while.

In the next section, we shall list just a few little Emacs tricks, as it’s likely that you won’t have met those yet, and it would be a little disingenuous to come away thinking that Emacs is not very powerful, merely because you have not been introduced to it thoroughly. A lot of people use both commonly, perhaps using **vi** for small jobs, such as editing config files, and Emacs for the larger jobs.

> **Note**
>
> There is a strong parallel here for System Administrators to learn objectively about multiple different solutions/products/etc. so they can make a wise decision.

One last point (actually, second to last) is that you shouldn’t try to learn all of your editor’s commands at one time. You have to learn it slowly, because much of the common things that are done, are learned via *muscle-memory*, which is learned by repetition. Aim to develop a habit of using new commands. Always be mindful of what you are doing in an editor, so you can ask yourself “How can I do this better?”

My final word is addressed to your little pinky finger, and will be of more importance to Emacs and Unix people, rather than just **vi** people. In order to reach the **Ctrl** key, people often have to curl their pinky finger. Ergonimically, this is a bad idea. It is much better to remap your keyboard, so the big **Caps Lock** key which should hardly ever be used, gets re-assigned to the **Ctrl** key. You might also decide to change **Ctrl** to **Caps Lock** as well. On Macs, you can do this via System Preferences > Keyboard & Mouse > Modifier Keys. Software support for remapping **Caps Lock** to **Esc**, which is nice for heavy **vi** users, is harder to come by on platforms other than X11 (the graphical infrastructure commonly used on Linux and UNIX systems); GNOME even makes this easy and exposes this in the Layout Options of the Keyboard preferences.

31.6. Emacs tips

This section is entirely optional. You've learned a few interesting tricks in this lab, and perhaps you're wondering how you can do them in Emacs, which you may be a little more familiar with at present. This section is aimed at addressing that desire, albeit very briefly.

To process selected text, such as a shell-command fragment, and replace it with its output, use `C-u M-|`. If you use just `M-|`, the results will be shown in a new buffer. Similar to that is `C-c M-!`, which you simply give a shell-command, and it will input the command's output into your document.

Global search and replace, with confirmation, can be very easily achieved with `M-%`. If you want to use a regular expression instead, use `C-M-%`. If nothing is selected, it operates from the current point to the end of the buffer.

Vim's modelines are referred to as "file variables" in Emacs, and can be specified alongside Vim modelines. For further help, you can use the menu entry Help > Search Documentation > Look up Subject in User Manual... > **`file variables`**. Here is an example showing both Vim and Emacs:

```
/* vim: softtabstop=4 shiftwidth=4 expandtab
 */

#include <stdio.h>

int main(int argc, char *argv[]) {
    puts("Hello, world!");
}

/*
  Local Variables:
  mode:c
  c-basic-offset:4
  compile-command: "make hello CC='gcc -std=c99' CFLAGS='-O2 -Wall -Werror'"
  End:
*/
```

However, for stylistic sort of settings that generally get applied to multiple files, it can be better to put such material into a directory variable [http://www.gnu.org/software/emacs/manual/html_node/emacs/Directory-Variables.html] instead, which goes into a file `.dir-locals.el`.

Part V. End Notes

Lab 32 Final Words

Here we provide some additional directions to foster the inner sys-admin.

32.1. Home Lab

We have been asked on several occasions about how to set a web/ssh server at home. Here's a list of steps that you'll need to complete:

- Service setup (ssh, http, etc.).
- Port forwarding (from your router's external port to your server's internal port).
- DNS setup and update (so you can find your server from the public internet).

If you want to run on a dedicated piece of hardware, we suggest you buy a Raspberry PI (around $NZ60 or so, the power adapter and SD card are a little extra). You can install raspbian (a variant of debian--the ancestor of ubuntu).

32.2. Automation and Devops

Over recent years there has been a move towards automation of infrastructure. It's a philosophy of combining devopment (i.e. programmatically specifying infrastructure) and operations (making sure services are running correctly) and often is abbreviated to devops.

- Investigate Ubuntu's MAAS (other systems have other offerings).
- Investigate Puppet, Chef, or Ansible.

Another approach is to deploy applications as containers. This is almost like having a virtual machine for the components of your application, but is much lighter weight (it's similar to the distinction between threads and processes).

- LXC/LXD (native containerisation on Ubuntu)
- Docker
- Kubernetes

32.3. Previous contributors of the labbook

Cameron Kerr and Paul Crane

Colophon

This book was created using Docbook XML 4.4.0 with XSL stylesheets provided by the Docbook XSL and the **xsltproc** XSL processor.

The PDF versions (student and teacher editions) are created using a custom stylesheet to convert from Docbook XML to TeXML which is an XML vocabulary of TeXML. TeXML then converts this from to XeLaTeX input. XeLaTeX is a version of TeXML that has better Unicode and Font support. The result of XeLaTeX is a PDF.

Editing was done using GNU Emacs with Norm Walsh's nXML mode which performs on-the-fly validation of input, making it much more pleasant to input XML documentation.

Working with XML also allows for greater Quality Assurance. For example, checks can be made to ensure every question has an answer; that answers don't leak into the student edition and that the teachers edition has teachers notes for each laboratory. The **xmlstarlet** tool has been shown to be useful for this.

The other major reason for using XML is to allow for publication in different formats, such as an on-line labbook. For this the standard Docbook XSL stylesheets are used.

Images are done using a mixture of PNG images for screenshots, and now also SVG for vector diagrams, which are exported in PDF format. Some older diagrams are still in a proprietary format, however.

The entire product uses liberal amounts of scripting to build the different versions and perform the various quality checks.