ITS332
Information Technology Laboratory II

Networking Lab Manual

by

STEVEN GORDON

Sirindhorn International Institute of Technology
Thammasat University

Semester 2, 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 About ITS332 Information Technology Lab II

The course ITS332 Information Technology Laboratory II is a lab covering introductory concepts and technologies in networking. This document is the manual for the lab tasks. For information about the course structure, lab dates, instructors, assessment and email list, see the course website at:

http://ict.siit.tu.ac.th/~steven/its332/

1.2 About the Lab Manual

1.2.1 How to Use the Manual

You can use this lab manual as a reference document, rather than a set of instructions for the lab. That is, you do not have to read this manual from start to finish.

Starting from Chapter 2, each chapter roughly corresponds to a lab class (some chapters are covered across two classes). A chapter provides background on the technologies you are going to learn in the class, including examples and reference material. At the end of each chapter is a list of general tasks. The lab instructor will inform you about details of each task.

In some cases you don't have to read the entire chapter: after listening to the instructor you can get started on the tasks. Then refer back to the manual when you have problems.

You should also use this manual to record notes. See Section 1.3 for details about tasks and notes.

1.2.2 Notation

Often you will use a terminal (command line) to enter commands. This lab manual explains different commands using examples enclosed in a box, as illustrated below.

- The command prompt is where you type commands using a terminal. In this manual the prompt is shown as a dollar sign ($). You do not type this in.

- Commands that you should type are given after the command prompt.

- Variables are part of a command, but a string that you must choose. For example, `FILENAME` below is a variable: you should type in a suitable name of a file.

- Comments are shown in italics following a hash symbol (#). This is just to explain to the reader. You do not type it in.

- Output of commands is shown without a command prompt. For example, the output of the first command below has 3 lines, while the output of the second command is a single line.

```
$ cat FILENAME # this command displays the contents of a file
line1
line2
line3
$ wc example.txt # this command counts lines, words and bytes in a file
 3  3 18 example.txt
```

1.3   Completing the Tasks

1.3.1   Making Notes

It is important that you make notes of what you do and what you learn when completing tasks. The notes help you in identifying the important information, and hopefully will help you in study for exams. The things you should note include:

- Record the commands you used to perform tasks (especially if it is different from the commands given in this manual). Include a description of the important options used. An example:

  ```
  To view the routing cache run the command:
  route -C -n
  The -C option displays the cache (instead of table),
  while the -n option shows only IP addresses (not domains).
  ```

- Note important concepts learnt from the tasks and from the instructors. An example:

  ```
  Reverse DNS maps IP addresses to domain names. But not all
  organisations register the reverse mapping.
  ```

- Illustrate the operation of protocols, and the packets transferred. See Sections 1.3.2 and 1.3.3 for details.

- When building a network, record the design of the network. See Section 1.3.4 for details.

1.3.2 Drawing Message Sequence Diagrams

One method to illustrate the operation of a protocol is to draw the exchange of packets between the involved entities. Such a diagram is often called a *message sequence diagram*. Figure 1.1 shows an example message sequence diagram.
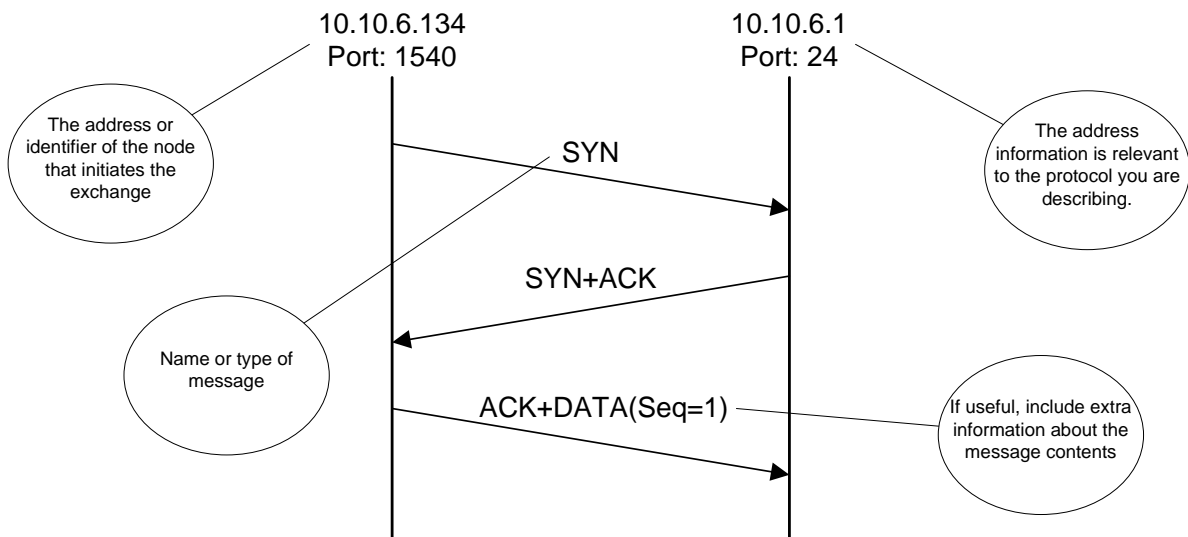
Figure 1.1: Example message sequence diagram

1.3.3 Drawing Packets

Understanding the relationship of protocols to different layers is important to understanding the role of a protocol in a communications network. As encapsulation is often used in protocol layers, drawing a packet with the headers added by the different layers is one method of visualising the layers. The headers of each layer can be drawn simply noting the name of the protocol for each header. Although sometimes you may included extra information, such as values of important fields in selected headers. Also, showing the size of headers and data can be useful. Figure 1.2 gives an example illustration of a packet.

IEEE 802.3	IP	TCP	HTTP	Data

Figure 1.2: Example packet diagram

1.3.4 Network Design

When you build a network you should record the design with enough detail such that a student next year could read your design and build the exact same network. Information you often include in the design includes:

- A diagram of the network topology. This should show the devices and links in the network, with each clearly labelled with a meaningful name (e.g. client, server, switch) or technology (e.g. Ethernet crossover cable). Figure 1.3 provides an example.

- Addresses of devices, especially IP addresses. You often can include them on the network diagram.

- Commands and operations you performed to configure the network. E.g. routing tables, application configuration.

- Commands and operations you performed to test the network, as well as important results from tests. E.g. ping and the average response time.
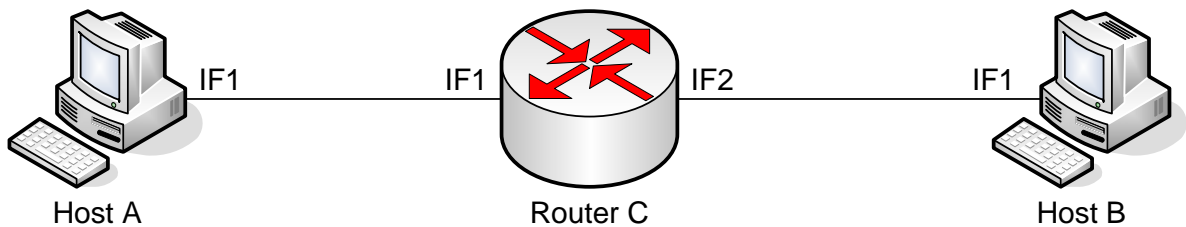
Figure 1.3: Single Router Network

1.3.5   What Not To Do

When completing tasks, often applications will produce output on the screen. Do not waste your time by copying the output from the screen to your notes. You should look at the output and try to understand the important information it tells you. If you want a record of the output, take a screenshot and save the file.

1.4   Further Information Sources

The course website, http://ict.siit.tu.ac.th/~steven/its332/, has links to numerous other sites with useful information about networking software and hardware. During the lab several sources of information you may regularly use include:

- Linux manual pages. On Linux, help can be found for almost all commands (and many important files) via the *manual pages*. Via the terminal, simply type `man` followed by the name of the command and you will see a detailed description of the command including the options available. Instead of asking your instructor about how to use a command, you should RTFM!

- Wikipedia. For details about protocols, packet/header formats, and even file formats, Wikipedia and other similar reference sites (or a search engine) is a good place to look.

Chapter 2

Ubuntu Linux

2.1 What is Ubuntu Linux?

Linux is an operating system based on Unix, one of the earlier multi-user operating systems developed in the 1970's and 1980's. Unix was originally a single operating system, but over time several commercial variants were developed. These Unix operating systems were particularly popular in the 1980's and 1990's, especially within academic and technology organisations. Some of the Internet applications and protocols were first developed on Unix, and hence Unix-based computer systems have a strong link with computer networking.

Today Unix operating systems are still used, mainly in servers and high-end workstations. In the 1990's Linux appeared, a free operating system with Unix-like functionality (or at least a kernel for an operating system). In the 2000's, Linux also became popular in typical Unix domains of servers and workstations, and also has been growing in the desktop field (however, in quantity of installs, Linux still does not compare with Microsoft Windows). As with the original Unix, there are many variants, or distributions of Linux, differing in the applications and graphical environments they provide (e.g. RedHat, Debian, Fedora, Ubuntu, Xandros). We will be using the Ubuntu Linux distribution.

Ubuntu Linux is a free, open-source Unix-based operating system, that has been developed mainly for desktop (and laptop) installations. The aim is to make a user-friendly Linux distribution. It is now one of the more popular Linux distributions. Ubuntu is installed on the Network Lab computers, and will be used extensively to demonstrate computer network operations in ITS 332. This document aims to give a quick introduction to some of the most common operations that you will need during the course.

2.1.1 Why Not Microsoft Windows?

Why use Ubuntu Linux, and not Microsoft Windows, especially since Windows is by far the most popular desktop operating system, and hence very popular with server systems? There are several reasons we will use be using Linux instead of Windows:

1. Linux is well-suited for learning of networking concepts:

File: Steve/Courses/2011/S2/ITS332/Handouts/linux.tex, r2074

(a) Linux has simple, yet powerful, operations for many networking tasks such as: changing an IP address, creating routing tables, testing network connectivity, inspecting traffic recevied/sent, and so on.

(b) Implementing and compiling simple client/server applications is straightforward on Linux.

(c) A Linux PC can easily be configured as a router (all the PCs in the Network Lab have two interface cards).

2. Experience in Unix-based operating systems is important: Although Windows is the most commonly used operating system for desktops, Unix-based operating systems (including Linux) are common for network servers, network devices and embedded systems. For example, many routers, switches and specialised computer devices use Linux.

3. Ubuntu Linux is free, as are all the applications we use (and none of them are pirated!)

2.2   Common Operations

2.2.1   Starting Ubuntu Linux

When the computer boots, within the first several seconds a program gives you the option to start Windows or Ubuntu. You should select Ubuntu, which will boot Ubuntu Linux.

2.2.2   User Accounts and Login

Once Ubuntu has started you are presented with a login screen. You should login with the username/password provided in the class.

Different users in Ubuntu have different privileges (e.g. ability to view or edit system files, view or edits other peoples files, change important operating system parameters). The user with the most privileges (that is, can do everything!) is called *root* (sometimes also called *super-user*). The problem with logging in as *root* is that a simple typing mistake may delete the entire hard drive!

The user you login as is just a normal user—lets refer to them as *network*. The user *network* has the ability to view and edit their own files in the directory `/home/network`, view most system files (that is almost all files on the hard disk, except those of other users) and view configuration options (such as IP address). You must always login as this normal user, and perform most operations as this normal user.

However, sometimes during the lab classes it will be necessary to perform tasks that require more privileges than the *network* user. For example, you require *root* privileges to install new software, change IP addresses and modify system files (such as configuration parameters for the web server). The *network* user has been configured to allow them to temporarily gain the privileges of the *root* user for these tasks. You do this using the `sudo` command.

Lets assume there is a command you need to execute in the command line shell (see Section 2.2.4). The command is:

```
$ command parameter1 parameter2
```

However, you must execute this as *root* user (since as the normal user, you are not allowed). So you would actually run the command by preceding it with `sudo`:

```
$ sudo command parameter1 parameter2
```

On the first use of `sudo` you will be prompted for a password—it is the password you logged in as *network* with. Then the command will execute. If you do not use `sudo` (and the command is privileged), the the command will not execute (usually returning an error like *Permission denied*).

Note that `sudo` should only be used for running command line applications as *root*. To run graphical applications as root (such as `gedit` or `wireshark`) use `gksudo`. The method is the same as with `sudo`, except the password will be prompted via a graphical window.

A final note on the *root* user. As we said before, you can potentially delete the entire hard drive. As we give you the access to perform operations as *root* user, you must act responsibly. Anyone caught using these privilelges incorrectly will be punished. This includes deleting system or other users files, copying other users files, changing parameters of the operating system and installing software which is not needed for the class. Punishment may range from loss of marks for the lab class, to more severe punishment in line with that for cheating (e.g. zero for the course).

2.2.3 Window System

Ubuntu has a graphical windows system like most other operating systems. It is quite intuitive. The main functions can be obtained from the *Applications*, *Places* and *System* menus from the top left corner of the screen.

Although many of the networking operations can be performed using the graphical tools, almost all have a command line interface.

2.2.4 Command Line Shell

Like almost all Unix-based systems, operations can be performed via a command line shell or terminal. In Ubuntu, to start a new terminal select *Accessories* and then *Terminal* from the *Applications* menu[1].

Some of the more common operations you will use include:

cd change directory

ls list the files in the directory

man view the manual (help) for a command

cp copy a file

mv move/rename a file

---

[1]You can also access a terminal, or `tty1` using *Ctrl-Alt-F1*. This doesn't use the windowing system. To switch back to the windowing system use *Ctrl-Alt-F7*. `tty2` through to `tty6` can also be accessed using F2 through to F6 instead of F1.

rm remove/delete a file

mkdir make/create a directory

rmdir remove/delete a directory

less display a file

cat display a file

echo print text to the screen (standard output)

pwd display the name of the present/current working directory

wc display the number of lines, words and bytes in a file

> redirect output to file

< redirect file to input

ps list the current processes running

& place process to be started into the background

Ctrl-c stop (kill) the currently active process

Ctrl-z suspend the currently active process

bg place the the just suspended process into the background

fg bring the background process to the foreground

An example of using some of these commands is shown below.

```
$ pwd
/home/sgordon
$ mkdir test
$ cd test
$ pwd
/home/sgordon/test
$ nano example.txt # use the text editor to write 'Hello, my name is Steve.'
$ cat example.txt
Hello, my name is Steve.
$ ls
example.txt
$ ls -l
total 4
-rw-r--r-- 1 sgordon sgordon 25 2009-11-06 16:34 example.txt
$ wc example.txt
 1  5 25 example.txt
$ cp example.txt copy-of-example.txt
$ ls
copy-of-example.txt example.txt
$ rm example.txt
$ ls
copy-of-example.txt
$ mv copy-of-example.txt example.txt
```

```
$ ls
example.txt
$ rm example.txt
$ ls
$ ls -al
total 12
drwxr-xr-x 2 sgordon sgordon 4096 2009-11-06 16:36 .
drwxr-xr-x 75 sgordon sgordon 8192 2009-11-06 16:33 ..
$ echo 'Hello'
Hello
$ echo 'Hello' > another-example.txt
$ cat another-example.txt
Hello
$ wc another-example.txt
1 1 6 another-example.txt
$ rm another-example.txt
$ ls
$ cd ..
$ rmdir test
```

We will introduce network-specific operations during the labs. For reference, some networking commands are listed in Appendix C.

2.2.5 Text and Source Code Editing

Although everyone has their own preferences about text and source code editors, two standard editors in Ubuntu that are recommended are:

gedit A GUI based editor, with syntax highlighting. Can be opened from *Accessories* then *Text Editor* from the *Applications* directory, otehrwise executing `gedit` from the command line.

nano A command line based editor. Provides a quick and simple way to edit a file. The main commands available to you once in `nano` are listed at the bottom of the display. Theˆcharacter means the *Ctrl* key. To save a file use *Ctrl-o*. To exit, where you are also prompted if you want to save a file, use *Ctrl-x*.

2.2.6 Applications

Some of the applications that we may use during the labs include:

Wireshark Capture and view traffic on a network interface. Command: `wireshark`. Also available via the GUI menus.

Apache Web Server A common web server.

2.3 Advanced Operations

2.3.1 Installing Software

Although it should not be required during the labs, (and you must not install any software unless asked to by the instructor!), Ubuntu has a simple command line interface to installing software, using `apt-get`:

```
apt-get install NAME
```

where `NAME` is the name of the software package you want to install. Of course, you need administrator privileges to install software (hint: `sudo`).

2.3.2   Compiling C Code

You can use the GNU C Compiler to compile C code:

```
gcc -o EXECUTABLE FILE.c
```

while compile `FILE.c` and create the executable program named `EXECUTABLE`.

2.4   Tasks

Task 2.1. *Follow the demonstration by the instructor to learn basic Linux commands and operations.*

Chapter 3

Wireshark

This lab will introduce you to an application for capturing traffic on networks. By "capturing", we mean record and view the details of every packet sent and received by the computer. The application is called *Wireshark*[1]. Packet capture applications are useful to inspect the details of the network operations being performed by your computer (and the network), thereby used to diagnose problems. We will use it in the remaining labs to understand how protocols work.

3.1 Packet Capture

The implementation of protocol layers in a network device (computer, router, switch, etc.) is done in a mix of hardware and software. Typically the Physical and Data Link layer are implemented in hardware, e.g. on an Ethernet LAN card. *Drivers* are special pieces of software that provide an interface from the operating system to a specific hardware device. That is, the Ethernet driver provides the functions for your operating system to receive Ethernet frames (and put them into memory) from your LAN card. The operating system normally implements the Network and Transport layers in software: that is, there is a software process that implements IP, as well as separate processes to implement UDP, TCP, ICMP and other transport layer protocols. Finally, each individual application (like web browsers, email clients, instant messaging clients) implement the Application layer protocols (such as HTTP and SMTP), as well as the user functionality and interface specific to that application. Figure 3.1 illustrates the layers and their implementation.

When a signal is received by your LAN card the signal is processed by the Physical and Data Link layers, and an Ethernet frame is passed to the operating system (via the Ethernet network driver). Normally the operating system will process the frame, sending it to the IP software process, which eventually sends the data to the transport layer protocol software process, which finally sends the data to your application.

In order to view all the frames received by your computer, we use special *packet capture* software, that allows all the Data Link layer frames sent from LAN card to operating system to be viewed by a normal application (in our case, Wireshark). The capturing of packets makes a copy of the exact packet received by your computer—it does not modify the original packet. This allows us to analyse data received by the computer, in

---

File: Steve/Courses/2011/S2/ITS332/Handouts/wireshark.tex, r2075

[1]Previously it was called *Ethereal*
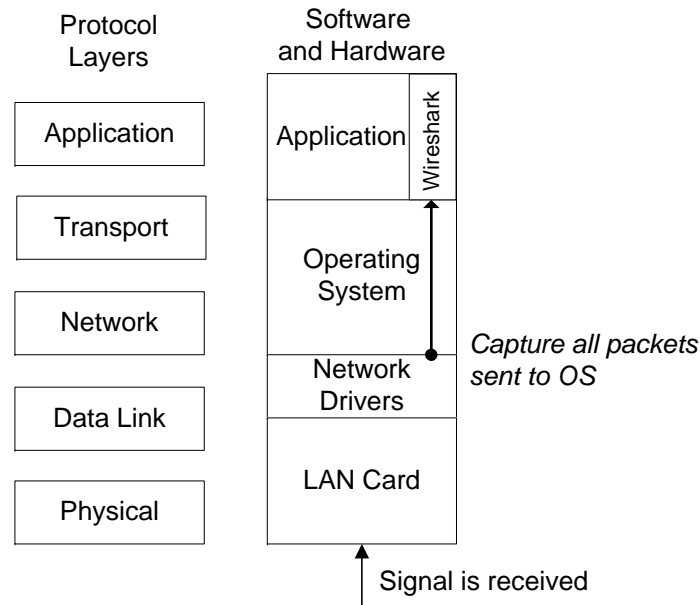
Figure 3.1: Capturing packets in the Operating System

order to perform various network management tasks (such as diagnose problems, measure performance, identify security leaks).

3.2   Using Wireshark

3.2.1   Starting Wireshark

First note that capturing traffic in Linux is a privileged operation, i.e. requires the user to be *root*. Therefore, we will in fact use two programs:

1. `tcpdump` on the command line to capture traffic.

2. Wireshark GUI to view the captured packets (although you can use `tcpdump` to view the packets, Wireshark provides many more details via a graphical interface).

To capture:

```
$ sudo tcpdump -i INTERFACE -w FILENAME
```

where `INTERFACE` is the network interface that you want to capture packets on (e.g. `eth0`) and `FILENAME` is the name of a file that you want to save the captured packets in (e.g. `capture1.cap`).

To finish the capture, press *Ctrl-C*.

Now to view the captured packets by opening Wireshark *Applications→Internet→Wireshark*. Open an existing capture—the file that you just created with `tcpdump`.

3.2.2   Viewing Captured Traffic

After a packet capture has been loaded, the main Wireshark window shows the captured packets (see example in Figure 3.2). The window is split into three sections:

1. The top section (packet list) showing the list of captured packets. Each packet has the following information:

   - Packet number (with respect to the total number of packets captured)

   - Time the packet is captured, assuming the time the first packet captured is time 0.0

   - The source and destination IP addresses of the packet

   - The highest layer protocol associated with the packet

   - Summary information about the information carried by the packet

2. The middle section (individual packet details) showing detailed information about the packet selected in the top section. This is separated based on the layers of the packet.

3. The bottom section (individual packet bytes) showing the hexadecimal and ASCII representations of the packet data.

When selecting the 12th packet (in the top section), and then selecting the Internet Protocol (in the middle section), the values of the IP datagram header fields are shown. When selecting Transmission Control Protocol (in the middle section), the bottom section shows the TCP header bytes (in hexadecimal and ASCII).

3.2.3   Analysis and Statistics

Wireshark has many in-built statistics that allow you to analyse the captured packets. This is very useful, especially if you have many packets captured (1000's to millions). You should explore (that is, view them and try to understand what they show) the following from the *Statistics* menu:

- Summary

- Protocol Hierarchy

- Conversations

- Flow Graph

- HTTP

- Packet Length

- TCP Stream Graph
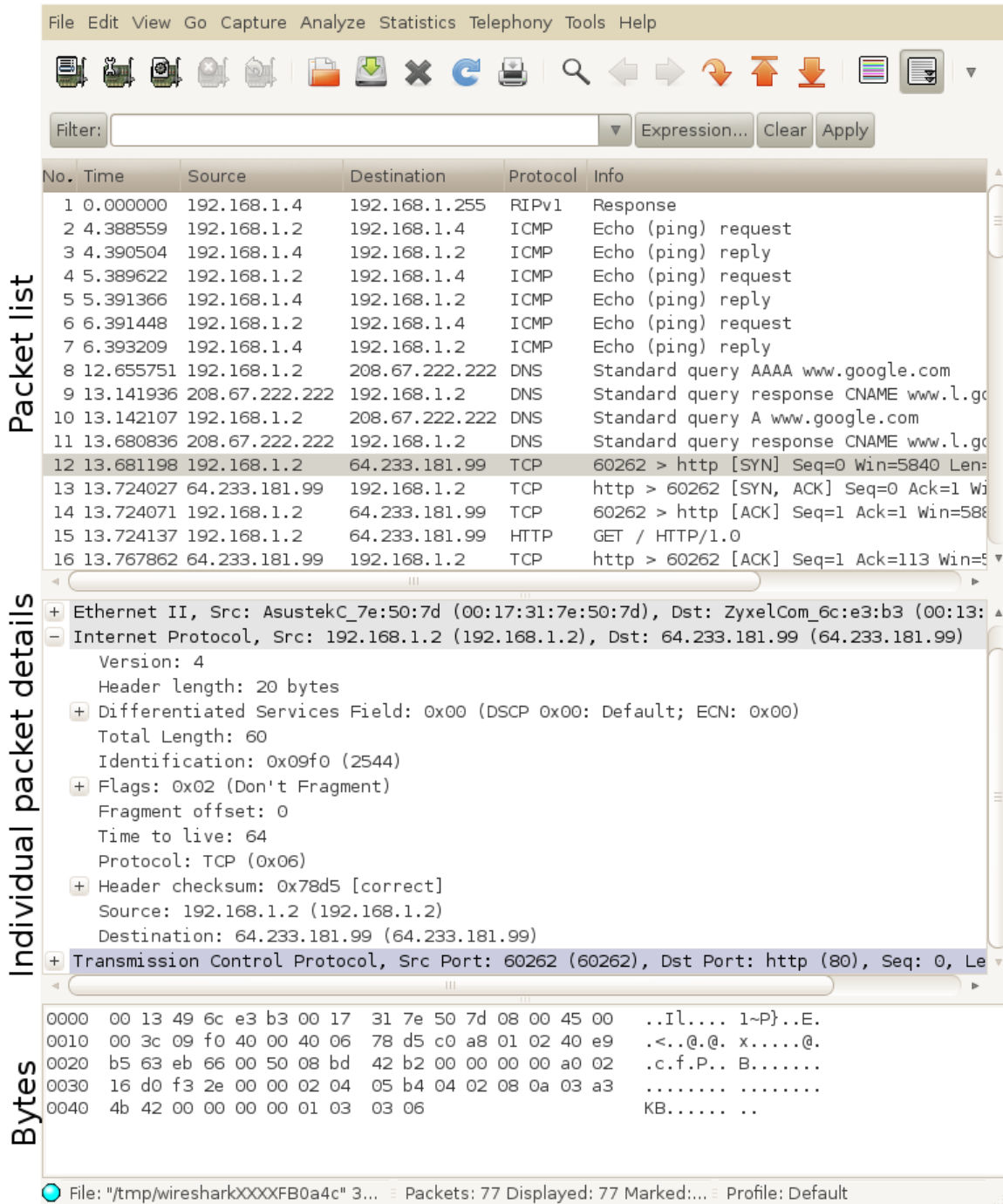
Figure 3.2: Main window of Wireshark

3.2.4 Filters

The example used above was for a small trace of less than 100 packets captured over 10 seconds. When capturing over a long time period (and hence thousands or hundreds of thousands of packets), it is often desirable to investigate a selected portion of the packets (for example, packets between certain pairs of hosts, or using a particular protocol). Hence filters can be applied during the packet capture (such that only packets that meet the specified criteria are captured - called *capture filters*) or after the capture (such that analysis is only performed on packets that meet the specified criteria - called *display filters*). In this course there is no reason for you to use capture filters, instead you can use display filters.

Display filters are used mainly to view certain types of packets. They make analyzing the data easier. One place you can enter a display filter is just above the top (packet list) section. You can either type in the filter and press Apply or create the filter using the Expression command. Some example filters are given below.

The following filter can be used to display only packets that have source or destination IP address of `10.10.1.171`

```
ip.addr==10.10.1.171
```

The next filter can be used to display only packets that have IP address of `10.10.1.127` and do not have a TCP port address of 8080.

```
ip.addr==10.10.1.127 && !tcp.port==8080
```

The next filter displays only ICMP packets.

```
icmp
```

The next filter displays only packets exchanged with a web server (assuming the web server is using port 80).

```
tcp.port==80
```

Further details of the display filter language and where it can be applied can be found in the Wireshark manual.

3.3 Tasks

For the following tasks you should understand the purpose of each of the packets captured. You should illustrate a message sequence diagram (see Section 1.3.2) as well as important packets captured (see Section 1.3.3).

Task 3.1. *Capture packets transferred while browsing a selected website (e.g. a page from the course website, a search engine home page). Investigate the protocols used in each packet, the values of the header fields and the packet sizes.*

Task 3.2. *Explore at least the following features of Wireshark: filters, Flow Graphs (TCP), statistics, protocol hierarchies.*

Chapter 4

Client/Server Applications

4.1 Clients, Servers and Addressing

Most network applications, including web browsing, email and file downloads, are implemented as *client/server applications*. For example, web browsing involves a web browser (client) retrieving web pages from a web server. The client/server model involves the server listening for new connections and the client initiating new connections. (A connection is usually needed each time we perform some operation, e.g. transfer a file, download a web page, send an email). We use IP addresses, as well as ports, to uniquely identify each connection.

4.1.1 Addresses and Ports

We know that IP addresses are used to identify computers on the Internet. This includes clients and servers. When sending data between a client and server, the source and destination IP addresses are carried in the IP datagram (see Figure E.1). These two addresses (source and destination) uniquely identify the connection between these two computers.

But what about different application programs (or processes) running on the computers? If you have one web browser connecting to a web server at www.google.com and a second web browser connected also to www.google.com, then how does your computer know which IP datagrams are destined for which instance of the web browser?

Client/server applications also use *port numbers* to identify connections between applications. Your first web browser instance uses a different port number than your second web browser instance. So in fact all communications between client/server applications can be uniquely identified by both the source/destination IP addresses and the source/destination port numbers:

For example, connection 1 between browser 1 and web server www.google.com:

Source IP 203.131.209.77

Destination IP 66.249.89.99

Source Port 47984

Destination Port 80

And connection 2 between browser 2 and `www.google.com`:

Source IP `203.131.209.77`

Destination IP `66.249.89.99`

Source Port 48032

Destination Port 80

Note that the two connections between the same computers are uniquely identified, because the source ports are different.

While the source and destination IP addresses are carried in an IP datagram header, the source and destination ports are carried in the TCP (or UDP) packet header (see Figures E.2 and E.3). Therefore every packet we send over the Internet has these four addresses. (A fifth identifier, the protocol number is also included in the IP datagram. For example, if TCP is the transport protocol being used, the protocol number field in the IP header has the value 6, representing TCP. For a list of common protocol numbers see Appendix E.2.)

4.1.2 Servers

The common structure of most network server applications is as follows:

1. The server is idle, listening (or waiting) for connection from clients on a *well known port*.

2. When a server receives and accepts a connection request (e.g. TCP SYN), it creates a child process to communicate with the client. The child process exchanges data with the client. When the exchange is finished, the child process is deleted, leaving only the original parent server process.

3. The server returns to the idle state (step 1).

In this way, a server can typically handle many connections at a time. For example, the `www.google.com` web server can handle connections from 1000's of client hosts at a time. An important aspect is a *well known port*port!well-known. Since the client initiates the connection, it has to know what is the destination IP address and port number. The client can find the servers IP address through DNS (e.g. `www.google.com` maps to `66.249.89.99`). It knows the port number because most common servers use a well known port number. Some commonly used well known port numbers are listed in Appendix E.2.

4.1.3 Clients

The common structure of most network client applications is as follows:

1. Send a connection request to a server. The client (in fact, the operating system) chooses an unused port number as the source port, and sends the connection request to the server.

2. Once connected with the server, the client and server exchange data.

So multiple instances (or processes) of one application can communicate at the same time—they just use different source port numbers.

4.2 Web Browsing

Everyone knows how to use a web browser. But what about a web server? In this lab you will gain basic experience in using Apache Web Server. Apache is free (www.apache.org) and is the most commonly used web server in the Internet.

4.2.1 Server Configuration Files

Apache (and many other Linux servers) are configured via one or more text files. You set the options in the files, and then restart the server, and then the server will run with those options.

The main configuration directory for Apache is:

```
/etc/apache2/
```

The main configuration file for Apache is:

```
/etc/apache2/apache2.conf
```

You can edit these file if you use sudo and your favourite text editor.

```
$ cd /etc/apache2
$ sudo nano apache2.conf
```

You should add the following line after the `ServerRoot` line:

```
ServerName localhost
```

This sets the name of your server (and prevents several warning messages when you start Apache). In Section 4.2.2 we will show how to apply this change. In this course we do not try to explain all the details of the `apache2.conf` file. The default settings are suitable for a basic web server.

Another file specific to the web site is:

```
/etc/apache2/sites-available/default
```

This file contains configuration options specific to a site. (You can potentially host multiple sites on the one Apache server).

The web server documents (e.g. the HTML pages that are available via the server) are stored in a *base directory*:

```
/var/www/
```

By default there is a file called `index.html` (although it may have been changed/removed my other students).

You can browse to the URL: http://localhost/ to view the web page and test that your server is working.

You can create any files/directories in the `/var/www` directory which will then be accessible by the web server. (Remember you need to use `sudo` to write to the `/var/www` directory).

4.2.2 Controlling the Web Server

You can use the `apache2ctl` command to start, stop and restart Apache. You must restart Apache if you want any changes in the configuration files to take effect. The commands are:

```
$ sudo apache2ctl restart
$ sudo apache2ctl stop
$ sudo apache2ctl start
```

If you have made a change to `/etc/apache2/apache2.conf`, you should restart the web server for that change to take effect.

4.2.3 Creating Web Pages

Basic web pages are written in HTML. In this course we do not explain the format of HTML (maybe you have covered it previously) but you should be able to create a basic web page in HTML using any text editor. With Apache, the web pages are stored in `/var/www`, and by default, if you browse to a directory (e.g. http://localhost/) then Apache will display the `index.html` file (if it exists).

You should create a simple HTML test page under the directory `/var/www/its332/` called `index.html`. An example page is:

```
<html>
<head>
<title>Test Page for ITS 332</title>
</head>
<body>
<h1>Test Page for ITS 332</h1>
<p>
This is a test page for ITS 332.
</p>
</body>
</html>
```

4.2.4 Server Logs

Another important file is the log produced by Apache. Apache logs (records) all requests for content on this server. The log is a text file:

```
/var/log/apache2/access.log
```

The format of this log file is a space separated file with each line showing details of a single request for a web page on the server. Each line has the following fields:

- The IP address of the source

- - (not used)

- The user name of the user who requested the page (only present if HTTP authentication is used, otherwise is -)

- Date and time the request was made

- The GET request, showing the path/file requested

- The HTTP status code sent back to the client (e.g. 200 is OK. See Appendix E.2 for common values)

- The size of the page/object sent back to the client

- The URL of the page that referred the request (e.g. the page that linked to the requested page)

- The user agent making the request, e.g. an identifier of the web browser

You should not edit the `access.log` file. Instead use `less` or `tail` to display its contents. `less` will display the file, page by page:

```
$ less access.log
```

The command `tail` will display the last 10 lines of the file:

```
$ tail access.log
```

4.2.5 Basic Authentication

It is common to protect some content on a web server using passwords. There are different methods of achieving this—here we will illustrate a simple (yet not too secure) method.

First you must create a username/password that Apache will use. Apache stores this information in a file. In this lab the file is:

```
/etc/apache2/passwd/passwords
```

The format of the file is a single line for each user, with the username and *hashed value of the password* separated by a colon (:). Note that the actual password is not stored: instead a somewhat secure representation of the password is stored.

To add a new user (with username *steve*) to the `passwords` file, use the command `htpasswd`:

```
$ sudo htpasswd /etc/apache2/passwd/passwords steve
```

You will be prompted for a password for the user *steve*. Enter it, and then the information will be saved in the `passwords` file.

If the `passwords` file does not already exist on your computer, you must use the `-c` option to force it to be created.

Now you must configure Apache to use a password on a specific directory. This can be achieved by editing the file `/etc/apache2/sites-available/default`. You need to add commands that inform Apache that access to a particular directory requires a username/password. An example of the code is:

```
<Directory "/var/www/its332/protected">
      AuthType Basic
      AuthName "Restricted␣Access␣to␣ITS332␣Files"
      AuthUserFile /etc/apache2/passwd/passwords
      Require user steve
</Directory>
```

The above example says only the user with username *steve* (and corresponding password stored in `/etc/apache2/passwd/passwords`) is allowed to access content in the directory `/var/www/its332/protected`.

To test that the above configuration works, you should create a file in the directory `/var/www/its332/protected`. Then restart the web server for the changes to take effect.

4.3   Remote Login

Secure shell (ssh) is a protocol for securely logging in to another computer. It is a replacement for telnet (which was insecure). OpenSSH is a free implementation of a SSH client and server. Both client and server should be installed on the Ubuntu computers.

Secure shell can be run from the command line using:

```
$ ssh DESTINATION
```

where `DESTINATION` is the IP address or domain name of the computer you want to connect to.

Optionally, you can include the `USERNAME` to log in as (otherwise it will default to the current username in use on the client):

```
$ ssh DESTINATION -l USERNAME
```

You will be prompted for the password of that user on the server. (The first time you log in you may also be prompted about unknown authentication—enter Yes to continue).

Once you have logged in, you can run commands on the server. That is, it is the same as if you are using the command line on the server.

You can log out using the `exit` command.

4.4   Tasks

In the following tasks you should record notes of the main changes made. When performing tests you should capture packets using Wireshark, recording the packet exchange with a message sequence diagram (see Section 1.3.2) as well as important packets captured (see Section 1.3.3).

Task 4.1. *Create several example files for your Apache web server to serve. Configure your web server, and then ask a friend to test your web server by accessing the files. Capture the packets and observe the log file.*

Task 4.2. *Configure authentication for a specific directory on your web server. Test, capture packets and observe the log file.*

Task 4.3. *Login to another computer in the lab, capture and investigate the data exchanged.*

Chapter 5

Networking Tools

This lab will introduce you to important software tools for managing computer networks. It will also give you an opportunity to become familiar with the ICT Networking Laboratory room, e.g. the computers, operating systems and network equipment. The software tools you learn in this lab will be used in the remaining labs in the course.

5.1 Operating Systems and Tool Interfaces

When configuring and managing a computer network, or diagnosing problems in a network, you need to use the correct *tools* for the task. Most often these tools are software applications. There are various tools available on most computers that can be used to support common networking tasks including:

- Viewing and changing the configuration of your computer's network interface, such as addresses and other protocol parameters.

- Testing your computer's network connectivity, such as ability to communicate with other computers and statistics of the communication.

- View and analyse traffic sent/received by your computer, as well as other computers on a network.

The tools that can be used to manage the network vary on different operating systems. For example, Microsoft Windows has different programs than Unix variants such as Ubuntu and Apple MAC OS. (And indeed, the programs may be different between versions: Windows 7 may be different from Windows Vista, and Ubuntu Linux different from RedHat Linux). Combined with this, many operating systems will have two different interfaces to the same tool: a graphical user interface (GUI) and a command line (text) interface.

Although the programs may be different (including interface and options), the majority of them provide similar level of functionality. Therefore once you learn the functionality using one tool, it will not be too hard for you to perform the same functionality in another operating system.

File: Steve/Courses/2011/S2/ITS332/Handouts/tools.tex, r2075

For our lab classes, we will use Ubuntu Linux, for the reasons outlined in Chapter 2. We will show examples and expect you to use the command line interface on most occasions. This is because once you know the command line interface, it is very easy to perform the same operations in the GUI (however, vice versa is not true: if you learn the GUI, it may be hard to understand the options of the command line interface). Also note that some network equipment is managed by a command line interface: e.g. you may log on to a router or switch and set the configuration via the command line interface only.

5.2   Viewing Network Interface Information

Your computer connects to the LAN via one of its Network Interface Cards (NIC) (see Appendix B for details). In the Networking Lab, each computer has three Fast Ethernet NICs, and by default one of the NICs is connected to a Fast Ethernet switch (in the switching cabinet in the corner of the room). Almost all operating systems allow the user to view information about the current NIC connection, including:

- MAC (or hardware) address

- IP address and subnet mask

- Addresses of other important nodes (servers) on the network

- Traffic sent/received by the NIC

Operating systems often allow administrator users to modify some of the above information as well. The main command to view and edit the network interface information is `ifconfig`.

To view the information for all interfaces:

```
$ ifconfig
```

The operating system assigns names to each interface, such as *eth0* for on Ethernet NIC and *eth1* for another. As the name/number assigned to an interface is automatic, you cannot assume the same scheme is used in different computers, nor can you assume it will be the same each time you start the same computer.

The special loopback interface (which isn't a real physical interface, but a virtual interface implemented in software inside the OS) is often given the name *lo*.

To view the details of a specific interface, such as *eth0*:

```
$ ifconfig eth0
```

5.3   Testing Network Connectivity

A basic task for diagnosing the connectivity of a network is to test whether one computer can communicate with another. This is normally performed using the Internet Control Message Protocol (ICMP). A user application that implements ICMP for testing connectivity is `ping`.

`ping` sends a message from your computer to some destination computer, which then immediately responds. `ping` measures the time it takes from sending the message, to

when the response is received. That is, the delay to the destination and back, i.e. the *round trip time* (RTT).

The simplest way to use `ping` is to specify the destination as the first parameter:

```
$ ping DESTINATION
```

where `DESTINATION` is the IP address or domain name of the computer you want to test connectivity with.

You can stop the ping by pressing *Ctrl-C*, or you can limit the number of messages sent by `ping` to `COUNT` messages using the `-c` parameter:

```
$ ping -c COUNT DESTINATION
```

There are other useful options for `ping`: read the manual!

5.3.1 ping at SIIT

`ping` is a very simple, but useful tool to diagnosing network problems. However, `ping` (and more generally, ICMP messages) can be used to cause problems in a network. For example, a malicious user may perform a security attack on a network by sending many ICMP messages to a router (making the router too busy to handle normal traffic, thereby restricting use of the network). Therefore, some organisations decide to not allow ICMP messages into and/or out of a network. SIIT does this: from inside the lab you cannot `ping` a computer outside on the Internet (e.g. try to `ping` http://www.google.com/). This is done for good reasons by the SIIT Network Administrators, however makes it difficult to demonstrate `ping` and other ICMP-based tools in this lab!

In addition to a network administrator blocking ICMP from leaving the network, some organisations may block ICMP from entering a network, and more specifically, block a particular computer from responding to ICMP messages. For example, the web server `www.fakewebserver.com` may be configured to not respond to ICMP messages, therefore your `ping` to such a domain would get no response.

Luckily for us, there are free web sites that allow us to use `ping` from the website to any computer that responds to ICMP messages. Note that when using these websites the source of the ICMP message is not your computer, but is the web server of the site or a router/server selected from the site.

There is an excellent list of free web-based `ping` (and other) tools at: http://www.bgp4.net/wiki/doku.php?id=tools:ipv4_ping. Several you should try include:

- Qwest Looking Glass Asia (http://stat.qwest.net/looking_glass_asia.html) - source is from Hong Kong, Singapore, Sydney or Tokyo

- Cogent Looking Glass (http://cogentco.com/htdocs/glass.php) - source from cities in North America and Europe

- Telia Looking Glass (http://lg.telia.net/) - sources from cities in Europe

- Carnegie Mellon University Network Operations (http://www.net.cmu.edu/cgi-bin/netops.cgi) - source from US

5.4   Testing a Route

Another useful network connectivity test is to determine the path (or route) that a message takes. That is, what routers does the message pass via on the way to the destination. As with `ping`, ICMP messages are sent to determine this. An application that implements this in Ubuntu is `tracepath`[1]. Like `ping`, an ICMP message is sent to the destination and returned, but with `tracepath` the set of routers along the way also send a response to the source.

The `tracepath` application can be used by giving a destination IP address or domain name as a parameter:

```
$ tracepath DESTINATION
```

As `tracepath` uses ICMP, it suffers the same drawbacks on SIIT's network as `ping`. In some cases, you may get a *no reply* message from a router. But again, you can use the free web-based applications in Section 5.3.1 to demonstrate `tracepath` (often referred to as *traceroute*).

5.5   Converting Between Domain Names and IP Addresses

We know that the Domain Name Service (DNS) is used for mapping domain names (user-friendly addresses) into IP addresses (computer-readable addres). It is also possible to do the opposite, often referred to as *reverse DNS*: map IP addresses to the corresponding domain name.

There are several tools for using DNS (or reverse DNS) in Ubuntu, all using slightly different approaches, and producing different output. In this lab we will use `nslookup`[2]. The basic use of the tools work in the same way: give a domain name as a parameter, and the corresponding IP address will be returned; or give an IP address as a parameter, and the corresponding domain name will be returned.

```
$ nslookup DOMAIN    # returns IP address
$ nslookup IPADDRESS # returns domain name
```

By default, `nslookup` will try to first use your local DNS server to retrieve the information. How do you know what your local DNS server is? On Ubuntu, the IP address of one or more local DNS servers are stored in the file `resolv.conf` under the directory `/etc/`. Consider the output of the following `resolv.conf` file:

```
$ cat /etc/resolv.conf
nameserver 10.10.10.9
nameserver 208.67.222.222
```

There are two local DNS servers configured: `10.10.10.9` and `208.67.222.222`. Requests will be sent to the first DNS server, and if no response, then the second will be tried.

---

[1]Some Unix distributions use the application `traceroute` to perform the same functionality as `tracepath`. In fact, you will see many web sites refer to *traceroute* instead of *tracepath*.

[2]The other tools are called `dig` and `host`—you can try them yourself to see the difference

If you want to retrieve the information from a specific DNS server (such as ns.siit. tu.ac.th or ns1.sprintlink.net) then you need to give an additional option:

```
$ nslookup DOMAIN DNSSERVER
```

Note that Linux typically uses (at least) two naming services: the common Internet naming service DNS, as well as a simple file that lists a set of names and corresponding addresses. This is called the `hosts` file. See Section 5.9 for further information.

5.6 Viewing the Routing Table

IP uses routing table to determine where to send datagrams. This applies to end hosts (like PCs), as well as routers, however a routing table on a host is typically quite simple, since all packets are often sent to a local (default) router.

You can view your routing table using the `route` command:

```
$ route -n
```

The `-n` option means the output will contain the numerical IP addresses (rather than the default domain names).

By default, `route` shows the main routing table. However, the operating system also maintains a cache of routing entries, which are based on where previous packets have been sent. When IP has a packet to send, it first checks the routing cache for an entry, and then (if no entry exists in the cache) uses the main routing table. You can view the routing cache using the `-C` option:

```
$ route -n -C
```

The routing cache shows the Gateway used for particular Source and Destination pairs.

In a later lab (Chapter 7) we will use `route` to modify the routing tables (like adding a new route).

5.7 Converting IP Addresses to Hardware Addresses

Remember that IP addresses are logical addresses. For a computer to send data to another computer on the same LAN/WAN they must use hardware (or MAC) addresses. For example, if computer A wants to send an IP datagram to computer B (on the same network as A) with IP address `192.168.1.3`, then computer A must know the hardware address of computer B. Hence, the Address Resolution Protocol (ARP) is used to find the corresponding hardware addresses for a given IP address.

Although we don't yet cover in detail how ARP works, we can view the information ARP has in your computer using the application `arp`. Running `arp` will return a table (called the *ARP table* or *ARP cache*) of IP addresses and corresponding hardware addresses that your computer currently knows about:

```
$ arp -n
```

ARP automatically updates the table with new entries for you. However, you can also use `arp` to delete entries from your ARP table and manually add new entries.

5.8   Network Statistics

A tool that allows you to view many different network statistics is `netstat`. For example, you can view interface statistics (similar to `ifconfig`), routing table statistics (same as `route`), connection statistics and TCP/IP packet statistics. Lets look at how to view the last two.

First, you can view the active TCP connections:

```
$ netstat -n -t
```

You can also view summary TCP/IP statistics:

```
$ netstat -s
```

5.9   Viewing More Network Information: Useful Files

Some additional networking information about your computer can be found in various files on your computer. An important directory that contains a lot of configuration details for your operating system is the `/etc` directory. Some useful files include:

/etc/hosts Set a list of local domain names and corresponding IP addresses. Used in addition to DNS. Normally this would be used to give a name to your computer, as well as other computers on your network.

/etc/resolv.conf Indicates the local DNS server for this computer.

/etc/network/interfaces Stores information about your computers' network interfaces.

/etc/services List of port numbers and corresponding servers

5.10   Automatic IP Address Configuration

5.10.1   How Does DHCP Work?

When an operating system is installed on a computer and the computer first setup (by, for example, the network administrator), the IP address and other relevant network information (such as DNS servers, subnet mask) can be manually entered. In Ubuntu, commands like `ifconfig` can be used to do this.

But with manual configuration, if any network information changes, the network administrator must then go to each computer to make the changes. With the SIIT Bangkadi network of 300 or more computers, the task of manually configuring each computer if, for example, the DNS server IP address changes, would be enormous!

Therefore, in practice there are ways to automatically configure a computers network information. The most used method is called *Dynamic Host Configuration Protocol* or DHCP. The basic process using DHCP is as follows:

1. One computer on the network is configured as a *DHCP Server*. This contains information about the possible IP addresses that can be allocated to other computers, and the DNS servers to be used. Usually, the DHCP Server is a router on the network.

2. All the hosts in the network are configured to use a *DHCP Client*. When the computers are first setup by the network administrator, no information about IP address, DNS server is given.

3. When a host boots, the DHCP Client broadcasts a request for an IP address. In other words the host sends a message to everyone else on the network saying: "I need an IP address (and other information)".

4. The DHCP Server is the only computer that responds: the DHCP Server selects an IP address for the host and sends it, including the network DNS server, subnet mask etc. to the host.

5. The DHCP Client configures its network interface using the information sent to it by the DHCP Server. The host now has an IP address.

The information assigned to the host by the DHCP Server has a lifetime. This is called a *lease*—for example, the host "leases" an IP address for 1 day. Before the lease expires, the DHCP Client will typically renew the lease. In this way, if a change of configuration information (such as DNS server) is needed, the network adminsitrator simply modifies the DHCP Server—the DHCP Clients in each host will retrieve the updated information from the DHCP Server.

Many computers now use DHCP to obtain an IP address, so the computer user does not need to worry about configuring their own IP address. For example, when you connect to the SIIT network with your laptop, typically you do not configure an IP address—DHCP is used.

5.10.2 Viewing Interface Information

By default, DHCP is used on the PCs in the Network Lab. We saw in Section 5.2 how to view the current network interface configuration using `ifconfig` (that is, the IP address *after* DHCP has obtained it). However the file `/etc/network/interfaces` indicates whether a dynamic (DHCP) IP address should be used, or some static (configured by the user) IP address should be used when the computer starts.

The format of a DHCP configured interface in `/etc/network/interfaces` is:

```
auto INTERFACE
iface INTERFACE inet dhcp
```

The interface labels (`eth0`, `eth1`, `eth2`, ...) may vary across computers and *even* when you reboot. That is, now one network card may be referred to by `eth0` and after re-booting the same card may be referred to by `eth1`.

To disable the use of DHCP and use static addresses, you can edit the file and change the `iface` section:

```
iface INTERFACE inet static
       address IPADDRESS
       netmask SUBNETMASK
```

5.10.3   Viewing DHCP Information

Now lets look at some DHCP information. The current DHCP leases are stored in `/var/lib/dhcp3/dhclient.X.lease` where X is the interface identifier (e.g. `eth2` or `eth3`). Note that the lease file may contain more than one entry—the last entry is the lease currently in use.

One way to refresh a leased IP address is to refresh the interface. Another is to use `dhclient`, where you can optionally specifiy the interface to renew/refresh th lease for:

```
$ dhclient
```

5.10.4   Setting a Static IP Address

We may not always want to use a dynamic (DHCP assigned) IP address. In the lab, the best way to assign a static IP address is using `ifconfig`. We saw before that `ifconfig` can be used for viewing interface configuration information—it can also be used for setting interface configuration information. An example to set the IP address 10.20.30.40 (with subnet mask 255.0.0.0) to interface `eth1` is:

```
$ ifconfig eth1 10.20.30.40 netmask 255.0.0.0
```

You can also use `ifconfig` to enable/disable interfaces by adding `up`/`down` to the end of the command (in Linux terminology this is referred to as "bring an interface *up* or *down*"). For example, to turn off/disable/bring down the interface:

```
$ ifconfig eth1 down
```

And to turn on the interface (add setting a different IP address at the same time):

```
$ ifconfig eth1 10.20.30.41 netmask 255.0.0.0 up
```

5.11   Tasks

Task 5.1. *View the configuration details, including addresses, of your computers network interfaces.*

Task 5.2. *Test the network connectivity between your computer and several other computers: another PC in the lab; the SIIT webserver; external web servers.*

Task 5.3. *Using one of the publicly available websites for ping/traceroute, test the connectivity to several external websites.*

Task 5.4. *Trace the path between several pairs of source/destination nodes.*

Task 5.5. *Find the IP addresses of several web servers (domains), using several different DNS servers.*

Task 5.6. *Try a reverse DNS lookup.*

Task 5.7. *View your routing table and routing cache.*

Task 5.8. *View your ARP cache. Find the hardware address of another computer in the lab using ARP.*

Task 5.9. *View the active TCP connections that your computer has, especially after you visited a website.*

Task 5.10. *View and browse through the summary network statistics.*

Task 5.11. *View the DHCP lease information for your computer, and see how it changes as you renew/refresh the lease.*

Chapter 6

Layer 2 Networking

The simplest computer network that can be created connects two computers directly together via cable (that is, a peer-to-peer network). Creating such a network will introduce you to the methods for physically connecting computers (including cable types), and configuring interfaces for computers to be on the same network.

A more useful, and much more common network is a switched Ethernet LAN, where multiple computers are connected to a switch. You will setup your own LAN, configuring all computers on the network.

This lab concentrates on Layer 2 network technologies, in particular Ethernet. The next lab will look at Layer 3 networks, that is, joining two or more Layer 2 networks together using IP routers.

6.1 Peer-to-Peer Networks

6.1.1 Background

The simplest network is one that connects two computers together, usually directly via a cable. This requires both computers to use the same Layer 2 network technology, with common options being Ethernet, a serial link, Wireless LAN or Bluetooth.
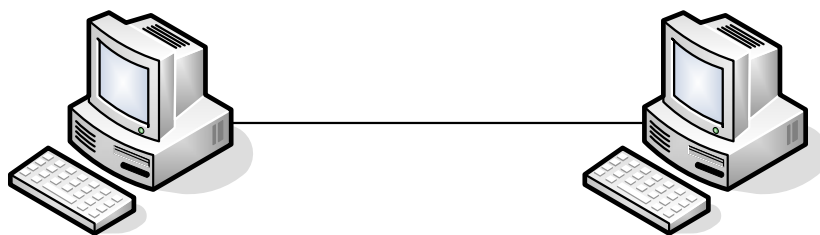

Figure 6.1: Layer 2 Peer-to-peer Network

Connecting two computers with a cable (e.g. Ethernet or serial link) is mainly useful for simple demonstrations (like this lab), simple, high speed file transfer and for troubleshooting. As an example of troubleshooting, if two computers are connected together

and can communicate with each other then that usually means the LAN cards and network software work correctly. Therefore there are other problems if one of the computers cannot successfully communicate when connected to a switched LAN or internet.

Connecting two computers via wireless technologies (e.g. Wireless LAN, Bluetooth, infrared) is commony used for temporary and/or spontaneous sharing of data.

As we will see, connecting two computers (and configuring them to communicate with each other) is quite simple. However, the applications are rather limited as you cannot communicate with anyone else.

Note 1. *(Ethernet, Fast Ethernet, Gigabit Ethernet) Although there are different variants of* IEEE *802.3, including Ethernet (10Mb/s), Fast Ethernet (100Mb/s) and Gigabit Ethernet (1Gb/s), we will often use the word Ethernet in the general sense, that refers to either of the variants. In fact, the computers in the lab are configured with Fast Ethernet LAN cards (although can operate at 1Gb/s Ethernet).*

Note 2. *(Other Types of Peer-To-Peer Networks) You may hear of Peer-to-Peer Networks or P2P file sharing to describe many applications and services in use on the Internet today. Although they are based on similar principles (all nodes are equal, no central point of control), the technologies and protocols used are not the same. In this lab we are only dealing with the method of connecting two computer together: any type of application can run on the two computers (Email, Web server, P2P File sharing), although may have limited use with only two people involved.*

6.1.2   Ethernet Cabling

Ethernet, in particular Fast Ethernet, is a full-duplex communications protocol. That is, device A can transmit to device B *at the same time as* device A is receiving from device B. This is performed by having a set of separate wires for transmit and receive. So in an Ethernet cable there are many individual wires, one set that are used for transmitting and one set for receiving.

In Ethernet networks, there are two types of devices:

1. End-user devices, such as computers and peripherals (even routers, which from a Layer 2 or Ethernet perspective, are just hosts on the Ethernet).

2. Network devices, such as switches and hubs.

To be able to transmit from one device and receive at the other device, the ordering of the transmit and receive sets of wires in the cable must be correct. The socket into which you plug the Ethernet cable must be wired correctly as well. The wiring in sockets of end-user devices is different than that in network devices. A simple way to understand this is as follows.

Lets assume the transmit wires in the *socket* of an end-user device (e.g. computer LAN card) are at the top, and receive wires at the bottom, as illustrated in Figure 6.2. And the opposite applies for the socket of a network device (e.g. switch).

Of course, the transmit wires in the socket of one device must connect to the receive wires in the socket of the other device. Hence to connect an end-user device to a network device, the *wires in the Ethernet cable* must go *straight-through* to the other side, as shown in Figure 6.3.

End-user device
(e.g. computer)

Network device
(e.g. switch)

Socket

Socket

Transmit

Receive

Receive

Transmit

Figure 6.2: Example ordering of Ethernet sockets on computer and switch

End-user device
(e.g. computer)

Network device
(e.g. switch)

Ethernet Straight-through Cable
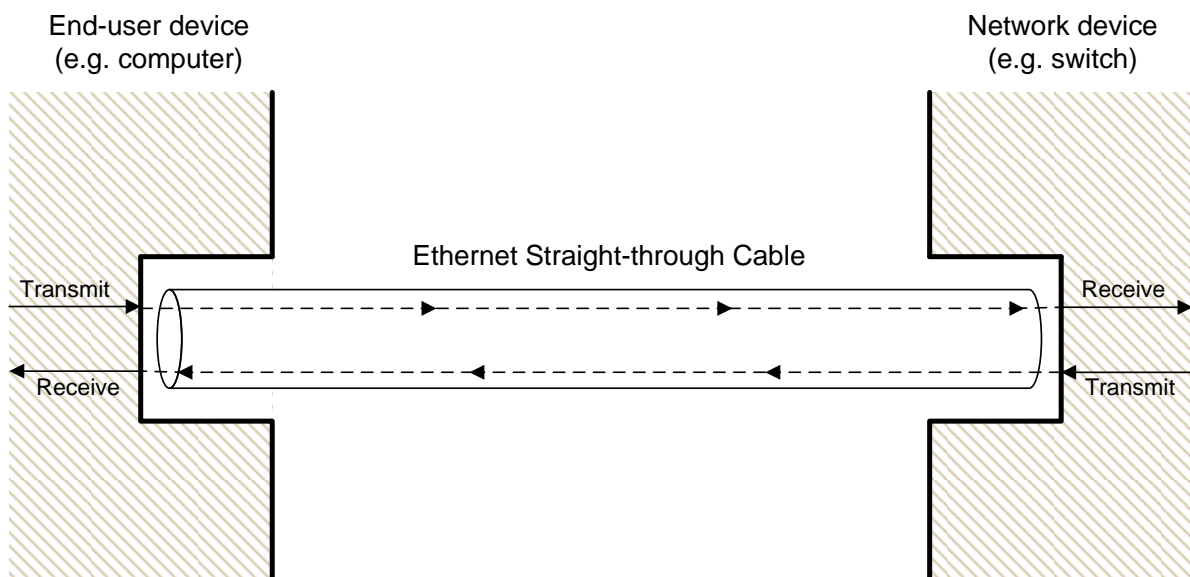
Transmit

Receive

Receive

Transmit

Figure 6.3: Ethernet straight-through cable connection

However, if we now want to connect an end-user device to another end-user device (such as in the peer-to-peer network, where we connect one computer directly to another), then the wires in the Ethernet cable must *cross-over*, as shown in Figure 6.4.
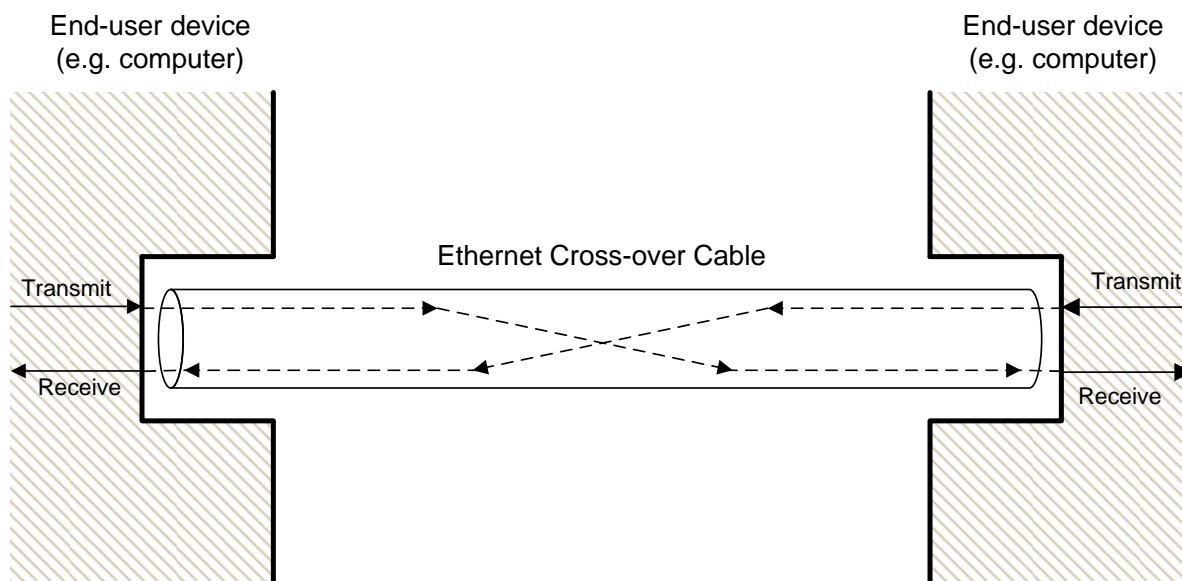

Figure 6.4: Ethernet cross-over cable connection

In summary, there are two different types of Ethernet cables for connecting devices:

Straight-through cable Connects an end-user device to a network device. For example: PC to switch; switch to router.

Cross-over cable Connects an end-user device to another end-user device. For example: PC to PC; PC to router.

How do you tell the difference between a straight-through and cross-over cable? From the colours of the individual wires that you see at the end points of the cables. For a straight-through cable, if you look at the end-points from the same viewpoint, the colours will be in the same order. For a cross-over cable, the colours will be in a different order.

6.2   Switched Network

Connecting only two computers has very limited use. Usually a user or organisation has more than two computers to connect together. There are different topologies that can be used to connect computers together, although today the most common is a *star-based topology*, where the centre of the star is a Layer 2 Switch. Hence, referred to as a *switched network*.

In a switched network, all end-user devices have individual cables to the central switch. In most cases, the connection to the switch is full-duplex. To transmit from one end-user device to another, the transmission goes via the switch. Of course, often users want to communicate with other users not on the same switched network. Hence the switch can also be connected to another switch to connect to another switched network, or to a router, to connect to any network.
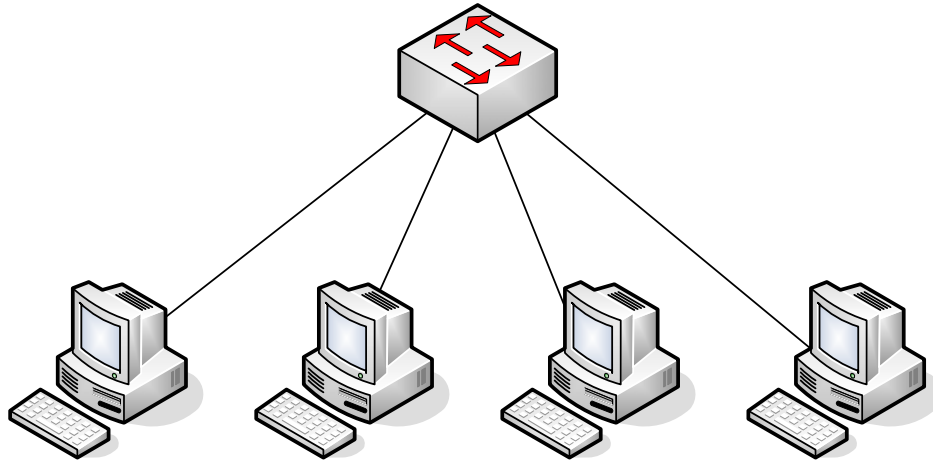
Figure 6.5: Layer 2 Switched Network

6.3 Tasks

On the following tasks you should record the design of the network, record relevant results from the tests, and demonstrate your network to the instructor.

Task 6.1. *Create and test a peer-to-peer network between two computers in the lab.*

Task 6.2. *Create and test a switched network with 4 PCs and one switch.*

Chapter 7

Layer 3 Networking

Although Ethernet is a common technology for layer 2 networks, in particular LANs, there are in fact many different technologies for layer 2 networks, including for WANs: Ethernet, ADSL, SDH, Wireless LAN, Bluetooth, Token Ring, Frame Relay, ATM, ... Therefore to allow a user to communicate with any other user, indepdenent of the LAN/WAN technology, layer 3 networking is used. Today, the Internet Protocol (IP) is the most commonly used layer 3 network technology. At layer 3, *routers* are used to connect LANs and WANs together, e.g. connect an Ethernet LAN to a SDH WAN; connect two Ethernet LANs together; connect a ATM WAN to a Frame Relay WAN; and so on.

In this lab you will create layer 3 network. That is, you will connect different LANs/WANs together using routers. The main tasks will be configuring the hosts and routers to use the appropriate addresses and routing tables.

7.1 Routers

An internet is a collection of many different computer networks (LANs and WANs) connected together. Routers are devices that connect these individual networks together.

A router has two main roles:

1. Routing. This is the process of discovering suitable routes throughout an internet. This is normally done automatically (using a routing protocol) but routes can be created manually (we will see how in this lab). Think of a route as the path through the internet.

2. Forwarding. This is the process undertaken when a router receives an IP datagram. The router looks at the destination IP address in the datagram, and from the routers routing table, determines what is the next router (or host) to send the datagram to in order to reach the final destination. Then the router sends the datagram.

7.1.1 Routers and Hosts

What is the difference between a host (e.g. your PC) and router?

- When a host receives an IP datagram destined to itself, then the host will process the datagram by sending it to the relevant application (e.g. web browser). If the host receives an IP datagram destined to an IP address other than itself, the datagram will be dropped.

- In the case of receiving an IP datagram destined to itself, the router will do the same as the host. But when a router receives an IP datagram destined to another IP address, the router will look up its routing tables and forward the datagram to another computer (host or router).

A simple example: an IP datagram with destination address `200.0.0.3` is received at a computer with IP address `192.168.1.1`. If the computer is a host, the datagram will be dropped (discarded). If the computer is a router, the datagram will be forwarded to the next router in the path. In summary, a router will forward datagrams; a host will not forward datagrams.

A router knows where to forward an IP datagram based on its routing tables. The routing tables, in their simplest form, say: *if a datagram is destined to network X, then send it to the next router Y*. In fact, both a router and a host have a routing table. The table in the router may be quite complex (with many rows or entries), whereas in a host it is usually just a single entry specifying the default router (or as we have seen, default gateway—gateway and router are the same in this context).

For a router, the routing tables are created using routing protocols. The routing protocols are implemented as software applications. During network operation, the routers in the internet communicate with each other to discover the best paths through the internet. Alternatively, the routing tables can be created manually by adding entries to the routing table.

Figure 7.1 summarises the differences between routers and hosts.

7.1.2   Enabling Routing

There is only a small difference in functionality between a router and host (however, for a real network, there may be big differences in implementation and performance: for example, a commercial router often has hardware and an operating system dedicated to the task of routing, whereas a PC uses general purpose hardware and operating systems). In practice it is easy to make a host become a router. That is, most PCs can be configured as a router if:

1. They have two or more network interfaces (as the PCs in the Lab do)

2. The operating system is configured to enable *forwarding*

On Ubuntu Linux, by default forwarding is *off*. The status of forwarding is maintained by the Linux kernel and is given in the following file—a `0` indicates *off* while a `1` indicates *on*:

```
/proc/sys/net/ipv4/ip_forward
```

To change the value you can edit the file (if you have permissions) or use `sysctl` as follows:

```
$ sysctl net.ipv4.ip_forward=1
```

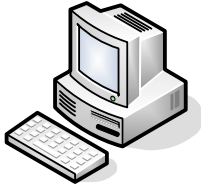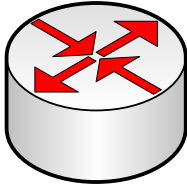Similarly you can use `sysctl` to turn off forwarding.

	Host	Router
Network interfaces	Usually 1	2 or more
Source of packets	Yes	Not common
Destination of packets	Yes	Not common
Forwards packets	No	Yes
Routing table	Yes Usually simple; 2 or 3 entries	Yes Usually complex; 100's or 1000's of entries
Routing protocol	Sometimes If so, simple protocol	Yes Complex (BGP, OSPF, RIP)

Figure 7.1: Comparison of Router and Host

7.1.3   Editing the Routing Table

In large internets, routing protocols are used to automatically create the routing tables. In small internets, we can manually configure the routes. To do this, we must add routes to the routing tables.

In Ubuntu, the `route` command shows the current routing table. Usually, for a host there will be a single entry like:

```
Destination   Gateway      Genmask        Flags Metric Ref  Use Iface
default       192.168.1.1  0.0.0.0        UG    0      0      0 eth1
```

This says, for all IP datagrams that are destined outside of this network, send them to the default router (`192.168.1.1`).

And for a simple, two interface router, there may be more entries to specify the routes to different networks. In order to add a route, you can use the `add` option:

```
$ route add -net NETWORKADDRESS netmask SUBNETMASK gw NEXTROUTER dev INTERFACE
```

where:

- `NETWORKADDRESS` is the IP address representing the destination network, e.g. `192.168.1.0`

- `SUBNETMASK` is the subnet mask for the destination network, e.g. `255.255.255.0`

- `NEXTROUTER` is the IP address next router in the path to the destination network, e.g. `192.168.3.1`

- `INTERFACE` is the network interface to send the datagram on, e.g. `eth2`

Similarly, you can delete an entry with:

```
$ route del -net NETWORKADDRESS netmask SUBNETMASK
```

7.2 Tasks

On the following tasks you should record the design of the network, record relevant results from the tests, and demonstrate your network to the instructor.

Task 7.1. *Create and test an internet that has two subnets: one subnet has two hosts connected via an Ethernet switch; the other subnet has a single host.*

Task 7.2. *Create and test an internet that has three subnets: two subnets have a single host; the third subnet simply connects two routers together.*

Task 7.3. *Create and test an internet that joins your three subnets from Task 7.2 with another groups three subnets.*

Chapter 8

Firewalls

This lab will introduce you to a common security mechanism used in networks: *firewalls*. A firewall is a device (usually implemented in software) that controls what traffic can enter and leave a network. If an organisation wants to *protect* there network, then a firewall between their internal network and all external networks ("the rest of the Internet") will be configured to inspect the traffic entering/leaving the network, and only allow the traffic that meets the organisations policies. We will setup our own simple firewall—in practice, real firewalls will be much more complex, and often require specialised network equipment.

8.1 Understanding Firewalls

Firewalls are network devices that control what packets enter and leave a computer network. Typically a company (and more recently, a home user) will use a firewall to stop people *outside* the company network (that is, everyone on the external Internet) from accessing computers and resources *inside* the company network (e.g. the SIIT network). For example:

- Stop people on the Internet from connecting to and accessing files on a SIIT computer

- Stop people on the Internet sending viruses and spam to computers in the SIIT network

The firewall can also be used to control what computers inside the network access. For example:

- Stop SIIT users from access *inappropriate* web sites on the Internet

- Stop SIIT users from sending `ping` commands to routers on the Internet

The firewall is usually a specialised router that acts as a gateway between the local network and the outside networks. That is, all traffic goes through the firewall. However, in this lab, we will see that we can configure the Ubuntu Linux computers to act as a simple firewall.
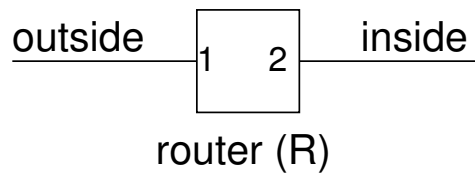

File: Steve/Courses/2011/S2/ITS332/Handouts/firewalls.tex, r2075

Figure 8.1: An organisation views their network as *inside*, and all other networks as *outside*

8.1.1   How Do Firewalls Work?

A gateway router (that is, the router between the inside and outside networks) normally receives an IP packet, looks at the destination IP address, looks up its routing table to determine where to sends the packet, and sends (or forwards) the packet.

A firewall is hardware or software running on the gateway router that provides additional functionality:

1. When the IP packet is received, the firewall looks at the packet and compares it to a set of rules stored in a firewall table. An example rule may be: "Drop all packets destined to IP address `64.233.189.104`"

2. When a rule matches, the corresponding action is taken. The action is usually DROP (discard, do not let the packet through) or ACCEPT (forward, let the packet through). In the above rule, if the IP destination address was `64.233.189.104`, then the packet would be dropped.

3. If the packet is not dropped, then the gateway router follows its normal procedures (e.g. look up routing table and send the packet).

8.1.2   Firewall Rules

The rules used by firewalls are the most important aspect. They can be very simple (e.g. "drop all packets destined to the local network") or very complex (e.g. 1000's of rules).

Packet-filtering firewalls usually create the rules using the following information:

- Packet match conditions:

    - IP source address
    - IP destination address
    - TCP/UDP source port number
    - TCP/UDP destination port number
    - Other IP/TCP/UDP header fields

- Direction of traffic:

    - Is the packet coming from outside (to inside) or is it coming from inside (to outside)

- Actions:

– ACCEPT or DROP

Using the above conditions, a reasonably good firewall can be built that can filter packets based on where the packets are coming from, where they are going to, and what applications are being used (remember, if a destination port number is 80, we can assume that a web browsing application is being used—if SIIT wanted to stop all web browsing, then they could drop all packets destined to port 80).

More complex firewalls (application-level firewalls) can be created by not only looking at the TCP/IP packet information, but also looking at the content of the messages. For example:

- Does the packet contain an email virus or spam?

- Does the packet contain spam?

- Is the web request to an unacceptable server (e.g. www.illegal-site.com)?

Figure 8.2 shows an example set of rules for a firewall (on router R). Each row in the table specifies a rule. When a packet arrives at R (the firewall) the packet will be DROPPED if a rule matches. If no rules match, then the packet is ACCEPTED (forwarded).

ARRIVES ON INTERFACE	IP SOURCE	IP DEST.	PROTOCOL	SOURCE PORT	DEST. PORT
2	*	*	TCP	*	21
2	*	*	TCP	*	23
1	128.5.0.0 / 16	*	TCP	*	25
2	*	*	UDP	*	43
2	*	*	UDP	*	69
2	*	*	TCP	*	79

Figure 8.2: Example firewall rules

The example rules specify:

- Block all packets destined to following services (applications) on internal network: FTP (port 21); TELNET (23); WHOIS (UDP port 43); TFTP (69); FINGER (79).

- Block all packets coming from internal network `128.5.0.0` (subnet mask `255.255.0.0`) and destined to external email server (port 25)

As a result, no-one outside the network could FTP to the inside network. And no-one inside the network using and address on the network `128.5.0.0` could send an email.

8.1.3   Firewalls and Servers

Most applications operate in a Client/Server mode, where a Client inside a network accesses a Server outside the network. Most computers inside the network DO NOT run servers accessible to the outside network. For example, there is no need for a SIIT staff or student's PC to run a web server accessible to someone outside SIIT.

Therefore, it is common for firewalls to be setup that will:

- Allow computers inside the network to access specific services outside the network. This is done by allowing traffic to pass from inside to outside if it is destined to a specific port (e.g. port 80 for web traffic).

- Do not allow computers inside the network to access unauthorised servers outside the network (for example, SIIT may decide that no-one inside can access FTP servers on the Internet).

- Do not allow any computers outside the network to access any servers inside the network. The only exceptions are to allow access to dedicated servers (e.g. the SIIT website).

Although the above cases can become quite complex in practice, very basic rules can be used to implement a simplified firewall that performs this functionality. You will do this in the Lab tasks.

8.1.4   Firewalls on Linux: iptables

`iptables` is a program on Linux that can be used to create a firewall. It allows the user to create a set of rules. Then when packets are received by the computer, the rules are processed. The packet is only sent if accepted by the rules.

`iptables` defines three basic classes of rules (or chains), based on where the packet is from/going to:

1. INPUT: processed if a packet is destined to this computer (e.g. the destination is this computer)

2. OUTPUT: processed if a packet is created to be sent by this computer (e.g. this computer is the source)

3. FORWARD: processed if a packet is to be forwarded by this computer (e.g. the packet is not destined to or from this computer, but this computer is acting as a router).

The most common way to use iptables is illustrated below (you need to execute with super-user privileges using `sudo`):

```
$ iptables -A CHAIN [CONDITIONS]
```

where `CHAIN` may be: INPUT, OUTPUT or FORWARD.
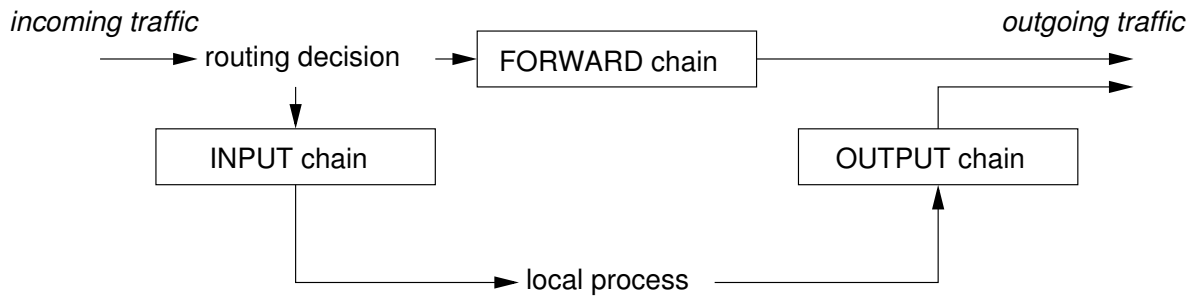Some of the optional `CONDITIONS` are:

Figure 8.3: Chains in `iptables`

```
-s source_IP_address (e.g. 192.168.1.2)
-d destination_IP_address
-i input__interface (e.g. eth0)
-o output_ interface
-p protocol (e.g. tcp, udp, icmp)
-j action (e.g. ACCEPT, DROP)
```

Each protocol (e.g. `tcp`, `udp`) also have their own set of options: e.g. `--sport`, `--dport`.

There are many other options that you can read from the `man` pages.

So one way to create the 3rd rule/row in Figure 8.2 is:

```
$ iptables -A FORWARD -s 128.5.0.0/16 -p tcp --dport 25 -i eth1 -j DROP
```

The `-A` option specifies to *append* the rule to the table. You can also use a `-I` (*insert*) and `-D` (*delete*) options in a similar way.

To view all the rules in your table run:

```
$ iptables -L [CHAIN]
```

where `[CHAIN]` is INPUT, OUTPUT, FORWARD—if omitted then all rules are shown.

To delete (or flush) all rules in your table, run:

```
$ iptables -F [CHAIN]
```

8.2   Tasks

In the following tasks you should record your firewall design, as well as tests that show that the firewall works as intended. You should also use Wireshark to observe what happens. Each task is independent: after creating firewall rules in the first task, they should be deleted before attempting the second task.

Task 8.1. *Create different firewall rule(s) that will prevent ping from working. Try using different chains.*

Task 8.2. *Create firewall rule(s) that will drop TCP packets destined to a specific computer on the Lab network (e.g. your neighbours computer).*

Task 8.3. *Create an internet with two subnets: on one subnet is a single PC1; and on the other subnet is two PCs (PC2 and PC3). PC1 should run both a web server and SSH server and a* `netcat` *server. Create a firewall on the router that allows the following: Any computer can connect to the web server on PC1; Only PC2 can connect to the SSH server on PC1; No computers can connect to any other servers (e.g. netcat, FTP, Email) on PC1.*

Chapter 9

Socket Programming

We know that many Internet applications use a client/server model for communication: a server listens for connections; and a client initiates connections to the server. How are these client and server programs implemented? In this lab you will learn the basic programming constructs, called *sockets*, to create a client and server program. You will use these programming constructs to implement your own client/server application in following labs. Example source code is given in Appendix D.

9.1 Programming with Sockets

Sockets are programming constructs used to communicate between processes. There are different types of systems that sockets can be used for, the main one of interest to us are Internet-based sockets (the other commonly used socket is Unix sockets).

Sockets for Internet programming were created in early versions of Unix (written in C code). Due to the popularity of Unix for network computing at the time, these Unix/C based sockets become quite common. Now, the same concept has been extended to other languages and other operating systems. So although we use C code and a Unix-based system (Ubuntu Linux), the principles can be applied to almost any computer system.

There are two main Internet socket types, corresponding to the two main Internet transport protocols:

1. Stream sockets use the Transmission Control Protocol (TCP) to communicate. TCP is stream-oriented, sending a stream of bytes to the receiver. It is also a reliable transport protocol, which means it guarantees that all data arrives at the receiver, and arrives in order. TCP starts be setting up a connection (we have seen the 3-way handshake in other labs), and then sending data between sender and receiver. TCP is used for most data-oriented applications like web browsing, file transfer and email.

2. Datagram sockets use the User Datagram Protocol (UDP) to communicate. UDP is an unreliable protocol. There is no connection setup or retransmissions. The sender simply sends a packet (datagram) to the receiver, and hopes that it arrives. UDP is used for most real-time oriented applications like voice over IP and video conversations.

File: Steve/Courses/2011/S2/ITS332/Handouts/sockets.tex, r2075

In this lab we are dealing only with Stream (TCP) sockets.

The basic procedure is shown in Figure 9.1. The server must first create a socket, then associate or bind an IP address and port number to that socket. Then the server listens for connections.

Figure 9.1: Socket communications

The client creates a socket and then connects to the server. The `connect()` system call from the client triggers a TCP SYN segment from client to server.

The server accepts the connection from the client. The `accept()` system call is actually a blocking function—when the program calls `accept()`, the server does not return from the function until it receives a TCP SYN segment from a client, and completes the 3-way handshake.

After the client returns from the `connect()` system call, and the server returns from the `accept()` system call, a connection has been established. Now the two can send data.

Sending and receiving data is performed using the `write()` and `read()` functions. `read()` is a blocking function—it will only return when the socket receives data. You (the application programmer) must correctly coordinate reads and writes between the client and server. If a client calls the `read()` function, but no data is sent from the server, then the client will wait forever!

9.1.1   Servers Handling Multiple Connections

It is common for a server to be implemented such that it can handle multiple connections at a time. The most common way to do this is for a main server process to listen for connections, and when a connection is established, to create a child process to handle that connection (while the parent process returns to listening for connections). In our example, we use the `fork()` system call.

The `fork()` system call creates a new process, which is the child process of the current process. Both the parent and child process execute the next command following the call to `fork()`. `fork()` returns a process ID, which may be:

- Negative, meaning the creation of the child process was unsuccessful

- 0 is returned to the child process

- A positive value is returned to the parent process—this is the process ID of the child.

Hence we can use the process ID returned from `fork()` to determine what to do next—the parent process (`pid > 0`) will end the current loop and go back to waiting for connections. The child process (`pid = 0`) will perform the data exchange with the client.

9.1.2   Further Explanation

You should read the source code for the `server.c`, and then the source code for `client.c`. The comments contain further explanations of how the sockets communication is performed.

The example code for `client.c` and `server.c` came from http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html. You may read through the details on this web page.

Most of the socket system calls are described in detail in their individual `man` pages. You should use the `man` pages for finding out further details of each function. Note that you may have to specify the section of `man` pages to use (which is section 2, the System Calls section):

```
$ man -S2 accept
ACCEPT(2)                    Linux Programmer's Manual                ACCEPT(2)

NAME
      accept - accept a connection on a socket
...
```

Note 3. *Unix man pages* `man` *pages in Unix are grouped into sections. There may be a command/file/function in Unix with the same name, but in different sections. Execute* `man man` *to get a list of the sections. For example,* `accept` *is a System Call (Section 2) as well as a System Administration command (Section 8). Executing* `man accept` *will give you the manual page for the System Administration command. To see the manual page for the System Call, you must explicitly specify the section:* `man -S2 accept`*. If you don't know the section you ar elooking for, use the* `-k` *option to search, for example* `man -k accept`*.*

9.2   Tasks

For the following tasks you should capture the tests using Wireshark to understand the relationship between the applications and the network communications. For new programs you create, you must demonstate the program and source code to the instructor.

Task 9.1. *Download, compile and test the provided client/server sockets programs.*

Task 9.2. *Modify the client/server programs to allow exchange of multiple messages. To do so, create a "fake" login mechanism. The server should ask the client for a username by sending a* username *message, and then the client will send the username to the server. Then the server will prompt for a password by sending the* password *message, and then the client will send the password to the server. Finally, the server will check if the username and password match an already known username/password pair, and send a response back to the client. Then both the client and server can finish (of course the server should still handle more connections). The output of the interaction should look as below:*

```
    Client                              Server

1.    Username:
2.    <user types in username, eg. "X">
3.                                      Client login with username "X"
4.    Password:
5. <user types in password, eg. "Y">
6. Client entered password "Y"
7. Username and password are correct.
8. You are now logged in.
```

Task 9.3. *Using the supplied client/server sockets programs, implement a third proxy server.*

Appendix A

Acronyms and Units

A.1 Acronyms

ARP Address Resolution Protocol. Given an IP address, determines the corresponding MAC address. ARP at Network Sorcery: `http://www.networksorcery.com/enp/protocol/arp.htm`. IETF RFC 826: `http://www.ietf.org/rfc/rfc826.txt`

CSS Cascading Style Sheets.

DHCP Dynamic Host Configuration Protocol. Allows IP addresses (and other information like DNS servers and netmasks) to be automatically assigned to computers in a network (rather than the user/administrator assigning the IP address manually). Requires the computer to be configured to use DHCP, and a DHCP server to be running on the local network. IETF RFC 2131: `http://www.ietf.org/rfc/rfc2131.txt`

DNS Domain Name System. Used to find the IP address that corresponds to the computer with a given domain name. Perform DNS queries at `http://www.dnsstuff.com/`.

HTML HyperText Markup Language. Language commonly used for creating web pages. W3C HTML Specification: `http://www.w3.org/MarkUp/`

HTTP HyperText Transfer Protocol. Transfer web content, often HTML but now many other formats, between a client and server. W3C HTTP Specifiaction: `http://www.w3.org/Protocols/`

ICMP Internet Control Message Protocol. Testing and diagnosing problems in the Internet. Applications such as `ping` and `traceroute` often use ICMP. ICMP at Network Sorcery: `http://www.networksorcery.com/enp/protocol/icmp.htm`

IEEE Institute of Electrical and Electronic Engineers. Professional society, which includes a standards organisation that has produced many LAN standards, as well as electrical/electronic interface standards. `http://www.ieee.org/`

IETF Internet Engineering Task Force. Technical organisation that produces standards (and other documents), known as RFCs, for many Internet technologies. Examples: IP, TCP and .

IP Internet Protocol.

LAN Local Area Network. Examples: IEEE 802.3 Ethernet, IEEE 802.11 Wireless LAN, Token Ring.

RFC Request For Comment. A type of document published by IETF. It may be an official standard, advice, experimental protocol or informal discussion document.

RTT Round Trip Time. The propagation time from a source to destination and back to source.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

W3C World Wide Web Consortium. Standards organisation that produces standards for the web, such as HTML, XML and CSS.

WAN Wide Area Network. Examples: PDH, SDH, ATM, Frame Relay.

XML eXtensible Markup Language.

A.2 Units

The following may differ from standard practice. That is, the defintions may not always be true under all conditions—they are chosen manily for the simplicity of the classes.

A Byte, normally abbreviated with an uppercase B, contains 8 bits, where a bit is normally abbreviated with a lowercase b.

The following unit prefixes are commonly used:

T tera, 10^{12}

G giga, 10^9

M mega, 10^6

k kilo, 10^3

m milli, 10^{-3}

μ micro, 10^{-6}

n nano, 10^{-9}

Appendix B

Lab Facilities

B.1   Work Stations

The Network Lab consist of 36 work stations (4 groups of 9), comprising an Intel-based computer and LCD monitor. Each computer has two operating systems installed: Microsoft Windows 7 and Ubuntu Linux. As well as all the standard devices of recent PCs, each computer has three 100Mb/s Fast Ethernet network interface cards.

B.2   Network Infrastructure

The lab computers are connected via a 100Mb/s Fast Ethernet LAN. The default configuration of the network has a single interface on each of the group of 9 computers connected to a Cisco switch in one of the four rack cabinets (that is 9 Ethernet cables into the switch). The other two interfaces on each component have "extension" cables so that the computers can be connected directly to each other in the lab. Each cabinet contains one switch and two Cisco routers. In the default configuration the routers are not used, but will be use for special confiruations during the lab class. The connections for a single computer are illustrated in Figure B.1. Note that each of the Ethernet cables go into a socket in the floor, and then to the switch. The group connections are shown in Figures B.2 and B.3.

The switch for each group is then connected to a switch in the IT Centre on the 3rd floor (and subsequently to the rest of the SIIT network, and Internet). This connects puts all computers in the lab in the same IP network. They are also connected to other computers on the SIIT Bangkadi network, and subsequently to Rangsit and the Internet.

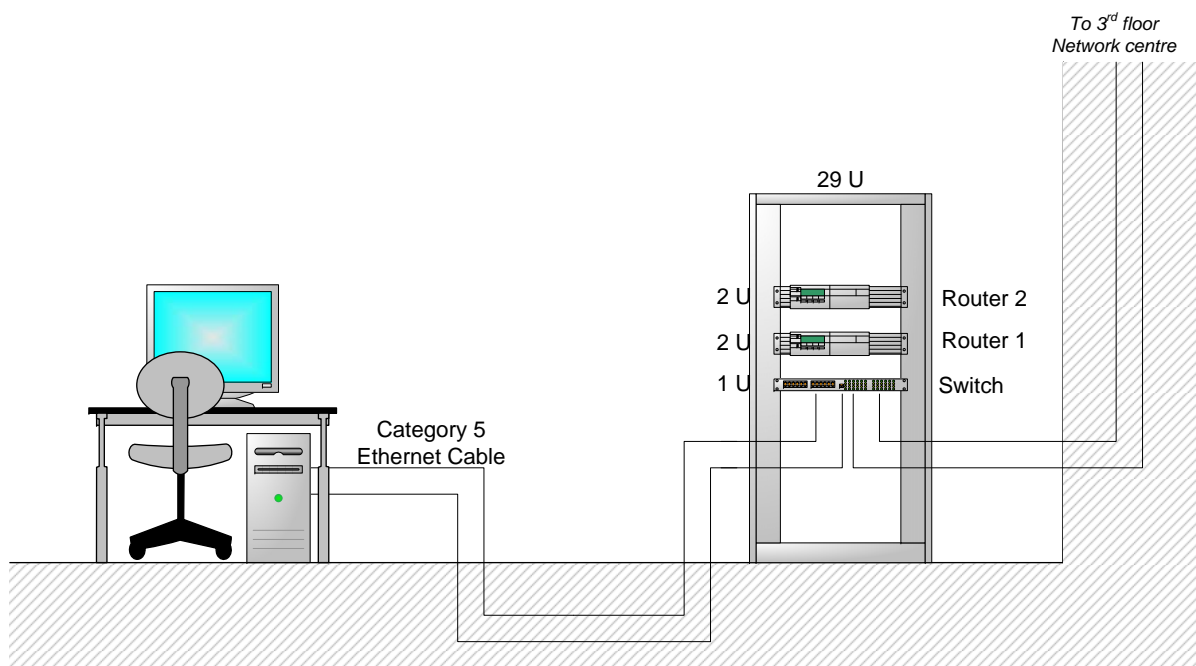File: Steve/Courses/2011/S2/ITS332/Handouts/facilities.tex, r2075

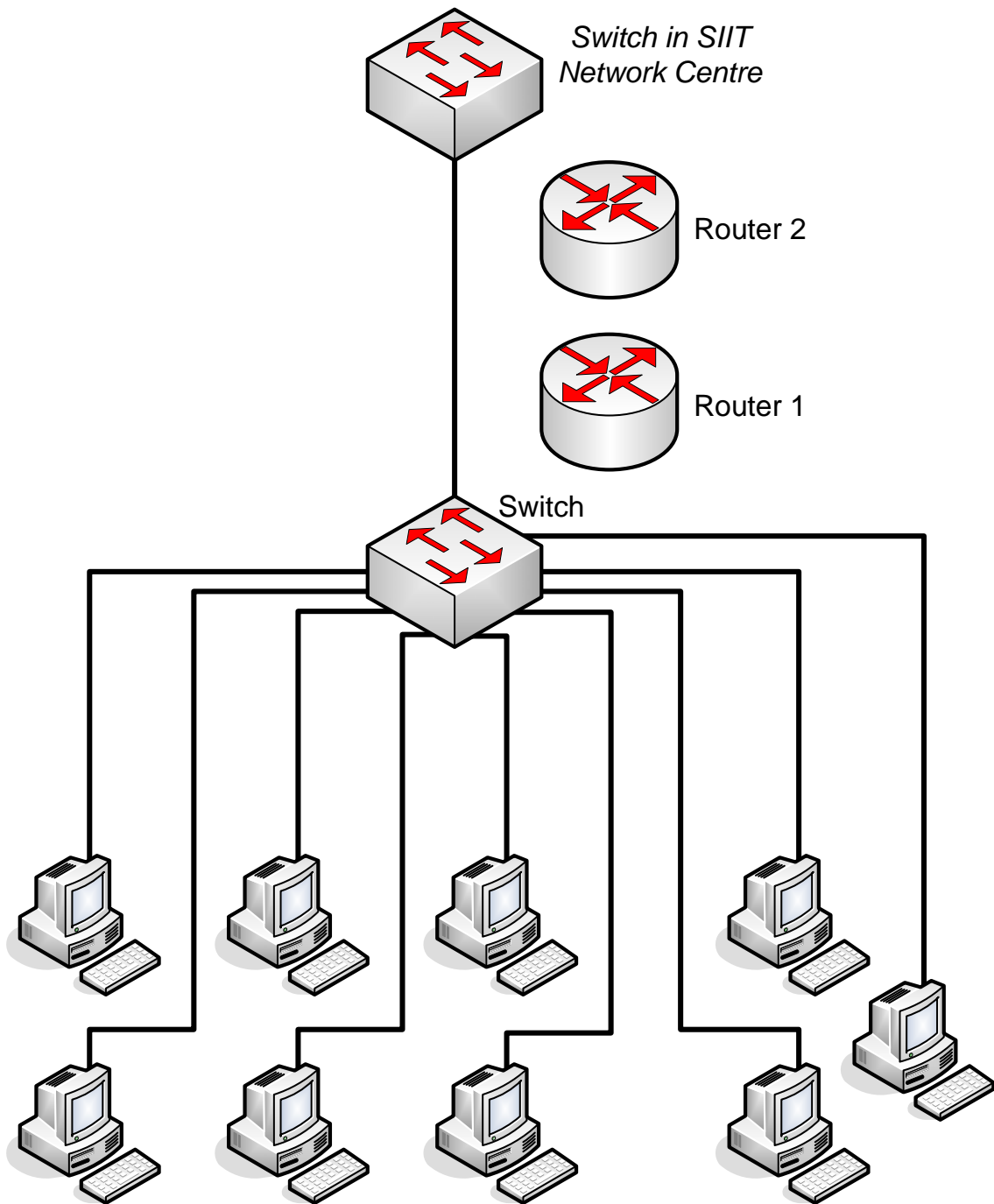Figure B.1: Network Lab: Connections for each computer

Figure B.2: Network Lab: Connections for each group of 9 computers
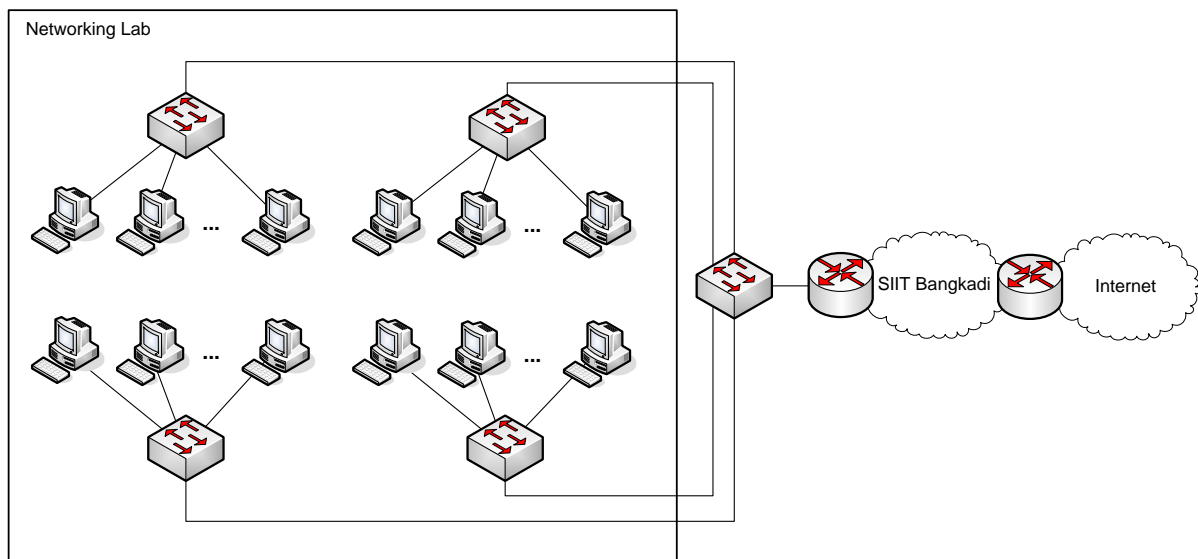
Figure B.3: Network Lab: Connections for entire lab

Appendix C

Ubuntu Reference Material

C.1   Commands

Table C.1 lists some of the general Ubuntu commands. Table C.2 lists some of the important networking commands, as well as their verison on Windows. Use `man` to see a detailed description of commands on Ubuntu.

Description	Ubuntu
List files in directory	`ls`
Change directory	`cd`
Copy a file	`cp`
Rename/move a file	`mv`
Delete/remove a file	`rm`
Create/make a directory	`mkdir`
Delete/remove a directory	`rmdir`
Display a file	`less` or `cat`
Display current working directory	`pwd`

Table C.1: General Ubuntu commands

C.2   Files and Directories

Table C.2 lists commonly accessed files and directories for the course. For some of these files, you can view a detailed description and format specification via the `man` pages, e.g. `man hosts`, `man interfaces`.

File: Steve/Courses/2011/S2/ITS332/Handouts/ubunturef.tex, r2075

Description	Ubuntu	Windows
Network interface configuration	`ifconfig`	`ipconfig`
Test network connectivity	`ping`	`ping`
Test network route	`tracepath`	`tracert`
Routing table configuration	`route`	`route`
Network statistics	`netstat`	`netstat`
ARP tables	`arp`	`arp`
DNS (simple)	`host`	`nslookup`
DNS (medium)	`nslookup`	`nslookup`
DNS (detailed)	`dig`	`nslookup`
Capture and view traffic	`wireshark`	Wireshark
Enable network interface	`ifup`	-
Disable network interface	`ifdown`	-

Table C.2: Important Ubuntu networking commands

File	Description
`/etc/hosts`	Local domain names
`/etc/resolv.conf`	Local DNS server
`/etc/network/interfaces`	Network interface information
`/proc/sys/net/ipv4/ip_forward`	IP forwarding is on (1) or not (0)
`/etc/apache2/sites-available/default`	Configuration for Apache web server
`/var/lib/dhcp3/dhclient.X.leases`	DHCP IP address leases
`/etc/dhcp3/dhclient.conf`	DHCP configuration

Table C.3: Important Ubuntu files and directories

Appendix D

Example Client/Server Programs

D.1   Client

```
1  /* ******************************************************************
2   * ITS 332 Information Technology II (Networking) Lab
3   * Semester 2, 2010
4   * SIIT
5   *
6   * Client/Server Programming Lab
7   * File: client.c
8   * Date: 24 Jan 2007
9   * Version: 1.0
10  *
11  * Description:
12  * Client to demonstrate TCP sockets programming. You should read the
13  * server.c code as well.
14  *
15  * Usage:
16  * client server_ip_address server_port_number
17  *
18  * Acknowledgement:
19  * This code is based on the examples and descriptions from the
20  * Computer Science Department, Rensselaer Polytechnic Institute at:
21  * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
22  *
23  * ****************************************************************** */
24
25  #include <stdio.h>
26  #include <string.h>
27  #include <stdlib.h>
28  #include <sys/types.h>
29  #include <sys/socket.h>
30  #include <netinet/in.h>
31  #include <netdb.h>
32
33  /* ================================================================== */
34  /* error: display an error message and exit                    */
35  /* ================================================================== */
36  void error(char *msg)
37  {
```

```
38        perror(msg);
39        exit(0);
40  }
41
42  /* ================================================================ */
43  /* main: connect to server, prompt for a message, send the message, */
44  /* receive the ack from server and then exit                       */
45  /* ================================================================ */
46  int main(int argc, char *argv[])
47  {
48      /* socket file descriptor, port number of server, and number of bytes */
49      int sockfd, portno, n;
50      struct sockaddr_in serv_addr; /* server address */
51      /* The hostent structure defines a host computer on the Internet. It
52         contains field which describe the host name, aliases for the name,
53         address type and actual address (e.g. IP address) */
54      struct hostent *server;
55      char buffer[256];
56
57      /* The user must enter two parameters on the command line:
58           - server host name or IP address
59           - port number used by server */
60      if (argc < 3) {
61         fprintf(stderr,"usage␣%s␣hostname␣port\n", argv[0]);
62         exit(0);
63      }
64      /* Get the port number for server entered by user */
65      portno = atoi(argv[2]);
66
67      /* Create an Internet stream (TCP) socket */
68      sockfd = socket(AF_INET, SOCK_STREAM, 0);
69      if (sockfd < 0)
70          error("ERROR␣opening␣socket");
71
72      /* The gethostbyname() system call uses DNS to determine the IP
73         address of the host */
74      server = gethostbyname(argv[1]);
75      if (server == NULL) {
76          fprintf(stderr,"ERROR,␣no␣such␣host\n");
77          exit(0);
78      }
79
80      /* Set the server address to all zeros */
81      bzero((char *) &serv_addr, sizeof(serv_addr));
82      serv_addr.sin_family = AF_INET; /* Internet family of protocols */
83
84      /* Copy server address obtained from gethostbyname to our
85         serv_addr structure */
86      bcopy((char *)server->h_addr,
87          (char *)&serv_addr.sin_addr.s_addr,
88          server->h_length);
89
90      /* Convert port number to network byte order */
91      serv_addr.sin_port = htons(portno);
92
93      /* The connect() system call establishes a connection to the server. The
94         three parameters are:
```

```
95            - socket file descriptor
96            - address of server
97            - size of the server's address */
98       if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
99           error("ERROR connecting");
100
101      /* Once connected, the client prompts for a message, and the users
102         input message is obtained with fgets() and written to the socket
103         using write(). */
104      printf("Please enter the message: ");
105      bzero(buffer,256);
106      fgets(buffer,255,stdin);
107      n = write(sockfd,buffer,strlen(buffer));
108      if (n < 0)
109           error("ERROR writing to socket");
110
111      /* Zero a buffer and then read from the socket */
112      bzero(buffer,256);
113      n = read(sockfd,buffer,255);
114      if (n < 0)
115           error("ERROR reading from socket");
116
117      /* Display the received message and then quit the program */
118      printf("%s\n",buffer);
119      return 0;
120  }
```

D.2   Server

```
1   /* *********************************************************************
2    * ITS 332 Information Technology II (Networking) Lab
3    * Semester 2, 2010
4    * SIIT
5    *
6    * Client/Server Programming Lab
7    * File: server.c
8    * Date: 24 Jan 2007
9    * Version: 1.0
10   *
11   * Description:
12   * Server to demonstrate TCP sockets programming
13   *
14   * Usage:
15   * server server_port_number
16   *
17   * Acknowledgement:
18   * This code is based on the examples and descriptions from the
19   * Computer Science Department, Rensselaer Polytechnic Institute at:
20   * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
21   *
22   * ********************************************************************* */
23
24  #include <stdio.h>
25  #include <string.h>
26  #include <stdlib.h>
27  #include <sys/types.h>
```

```
28  #include <sys/socket.h>
29  #include <netinet/in.h>
30
31  /* =============================================================== */
32  /* Function Prototypes                                             */
33  /* =============================================================== */
34  void dostuff(int);
35
36  /* =============================================================== */
37  /* error: display an error message and exit                       */
38  /* =============================================================== */
39  void error(char *msg)
40  {
41      perror(msg);
42      exit(1);
43  }
44
45  /* =============================================================== */
46  /* main: listen for connections, and create new process for each */
47  /* connection. Receive the message from client and acknowledge. */
48  /* =============================================================== */
49  int main(int argc, char *argv[])
50  {
51      /* file descriptors that contain values return from socket and
52      and accept system calls */
53       int sockfd, newsockfd;
54       int portno; /* port number on which server accepts connections */
55       int pid; /* process ID for newly created child process */
56       /* sockaddr_in is a structure containing an IP address - it is
57          defined in netinet/in.h */
58       struct sockaddr_in serv_addr, cli_addr;
59       size_t clilen; /* size of the address of the client */
60
61       /* The user must pass the port number that the server listens on
62          as a command line parameter (otherwise error) */
63       if (argc < 2) {
64           fprintf(stderr,"ERROR,␣no␣port␣provided\n");
65           exit(1);
66       }
67
68       /* First we must create a socket using the socket() system call.
69          The three parameters are:
70           - address domain of the socket. It may be an Unix socket, an
71             Internet socket or others. We use the Internet socket which
72             is defined by the constant AF_INET
73           - socket type. Stream (TCP), or Datagram (UDP, SOCK_DGRAM) or
74             a raw socket (for accessing IP directly).
75           - the protocol. 0 means the operating system will choose the
76             most appropriate protocol: TCP for stream and UDP for
77             datagram.
78          The call to socket returns a file descriptor (or -1 if it fails) */
79       sockfd = socket(AF_INET, SOCK_STREAM, 0);
80       if (sockfd < 0)
81           error("ERROR␣opening␣socket");
82
83       /* bzero sets all values in a buffer to 0. Here we set the server
84          address to 0 */
```

```
85          bzero((char *) &serv_addr, sizeof(serv_addr));
86
87          /* Get the port number that the user entered via command line */
88          portno = atoi(argv[1]);
89
90          /* Now we set the server address in the structure serv_addr */
91          /* Note that INADDR_ANY is a constant that refers to the IP
92             address of the machine the server is running on. */
93          /* Note that the port number must be specified in network byte order.
94             Different computer systems represents bytes in different order:
95              big endian - most significant bit of byte is first
96              little endian - least significant bit of byte is first
97             For this reason, everything must be converted to network byte
98             order (which is big endian). htons does this conversion. */
99          serv_addr.sin_family = AF_INET; /* Protocol family: Internet */
100         serv_addr.sin_addr.s_addr = INADDR_ANY; /* Server address */
101         serv_addr.sin_port = htons(portno); /* Port number */
102
103         /* The bind() system call binds a socket to an address. This
104            may fail if for example the port number is already being
105            used on this machine. */
106         if (bind(sockfd, (struct sockaddr *) &serv_addr,
107                 sizeof(serv_addr)) < 0)
108                 error("ERROR on binding");
109
110         /* The listen() system call tells the process to listen on the
111            socket for connections. The first parameter is a file descriptor
112            for the socket and the second parameter is the number of
113            connections that can be queued while the process is handling this
114            connection. 5 is a reasonable value for most systems */
115         listen(sockfd,5);
116         clilen = sizeof(cli_addr);
117
118         /* Now we enter an infinite loop, waiting for connections from clients.
119            When a connection is established, we will create a new child process
120            using fork(). The child process will handle the data transfer with
121            the client. The parent process will wait for another connection. */
122         while (1) {
123         /* The accept() system call causes the process to block until a
124                client connects with the server. The process will wake up once
125                the connection has been established (e.g. TCP handshake).
126                The parameters to accept are:
127                 - the file descriptor of the socket we are waiting on
128                 - a structure to store the address of the client that connects
129                 - a variable to store the length of the client address
130                It returns a new file descriptor for the socket, and all
131                communication is now done with this new descriptor. */
132           newsockfd = accept(sockfd,
133                 (struct sockaddr *) &cli_addr, &clilen);
134           if (newsockfd < 0)
135               error("ERROR on accept");
136
137         /* Create child process to handle the data transfer */
138           pid = fork();
139           if (pid < 0)
140               error("ERROR on fork");
141           /* The process ID in the child process will be 0. Hence the child
```

```
142                 process will close the old socket file descriptor and then call
143                 dostuff() to perform the interactions with the client. When
144                 complete, the child process will exit. */
145             if (pid == 0) {
146                 close(sockfd);
147                 dostuff(newsockfd);
148                 exit(0);
149             }
150         /* This is called by the parent process only. It closes the
151             new socket file descriptor, which is not needed by the parent */
152             else close(newsockfd);
153         } /* end of while */
154         return 0; /* we never get here because we are in infinite loop */
155 }
156
157 /* ================================================================ */
158 /* dostuff: exchange some messages between client and server. There */
159 /* is a separate instance of this function for each connection. */
160 /* ================================================================ */
161 void dostuff (int sock)
162 {
163     int n;
164     char buffer[256];
165
166     /* Set a 256 byte buffer to all zeros */
167     bzero(buffer,256);
168
169     /* Read from the socket. This will block until there is something for
170         it to read in the socket (i.e. after the client has executed a
171         write()). It will read either the total number of characters in the
172         socket or 255, whichever is less, and return the number of characters
173         read. */
174     n = read(sock,buffer,255);
175     if (n < 0) error("ERROR reading from socket");
176
177     /* Display the message that was received */
178     printf("Here is the message: %s\n",buffer);
179
180     /* Write a message to the socket. The third parameter is the size of
181         the message */
182     n = write(sock,"I got your message",18);
183     if (n < 0) error("ERROR writing to socket");
184 }
```

Appendix E

Packet Formats and Constants

E.1 Packet Formats

| 0 | 4 | 8 | 14 | 16 | 19 | | | 31 |

Version	HLength	DiffServ	ECN	Total Length
Identification				Flags	Fragment Offset
Time To Live		Protocol		Header Checksum
Source IP Address
Destination IP Address
Options + Padding (optional)
Data

(20 Bytes)
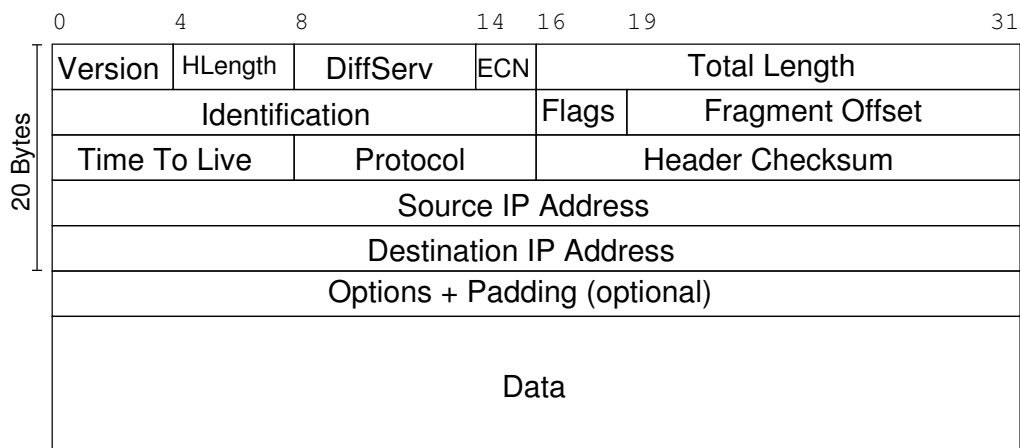
Figure E.1: IP Datagram Format

| 0 | 4 | 8 | 16 | 31 |

Source Port	Destination Port
Sequence Number
Acknowledgement Number
HLength	Reserved	Flags	Advertised Window
Checksum	Urgent Pointer
Options + Padding (optional)
Data

(20 Bytes)
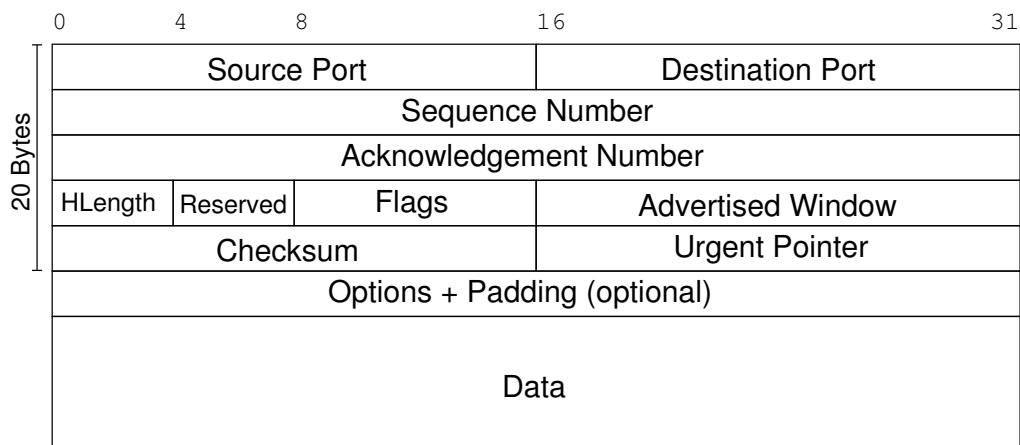
Figure E.2: TCP Segment Format
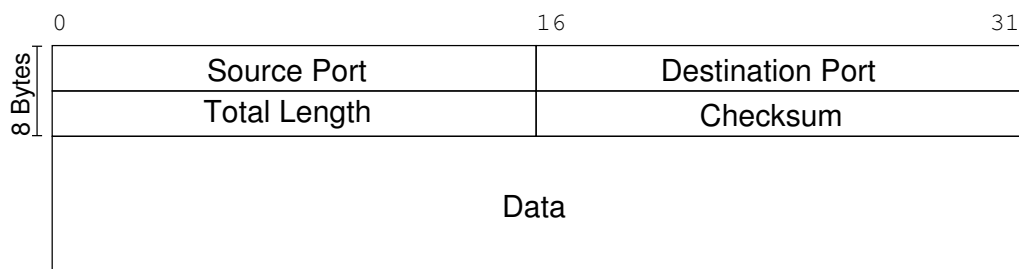
Figure E.3: UDP Datagram Format
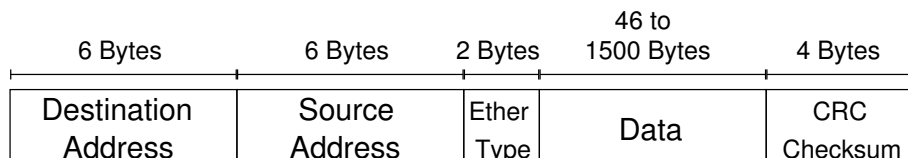
Figure E.4: Ethernet Frame Format

E.2   Port Numbers and Status Codes

Port numbers used by common applications include:

20 FTP data transfer

21 FTP connection control

22 SSH, secure remote login

23 TELNET, (unsecure) remote login

25 SMTP, email transfer between servers

53 DNS, domain name lookups

67 DHCP server

68 DHCP client

80 HTTP, web servers

110 POP3, client access to email

123 NTP, network time

443 HTTPS, web servers with secure access

520 RIP, routing protocol

631 IPP, Internet printing

1503 Windows Live Messenger

1512 WINS, Windows naming service

3306 MySQL database server

3723 Blizzard games

5060 SIP, voice/video signalling

5190 ICQ, instant messaging

8080 HTTP proxy server

Protocol numbers for commonly used transport protocols include:

1 ICMP

2 IGMP

6 TCP

17 UDP

33 DCCP

41 IPv6 encapsulation

47 GRE

89 OSPF

Status codes and their meaning for common HTTP responses include:

100 Continue Client should continue to sent the request

200 Ok Requested content is included in response

301 Moved Permanently This and all future requests should be redirected to the given URL

304 Not Modified Requested content has not been modified since last access

401 Unauthorized Requested content requires authentication that has not been provided or is incorrect

403 Forbidden Request is ok, but not allowed to access the requested content

404 Not Found Requested content could not be found on server

503 Service Unavailable Requested server is currently unavailable

Index