ELEC436 Portfolio

Christian Nassif-Haynes

16 June 2014

Preface

It is a software engineer's responsibility to keep her or his skill set up-to-date, be it by independent research or undertaking a formal course. Throughout my degree I have done the latter. In ELEC436, additional emphasis has been placed on the former, more self-guided form of learning which will be required during my career.

To undertake this independent research most fruitfully, ELEC436 students familiarised themselves with the ACS-EA Joint Board software engineering competencies, then independently (of course, we could have asked for help along the way from Mike, our lecturer) satisfied those competencies which had not yet been covered. This document describes that research (Chapters 1 and 2) in addition to work covered in class (all other Chapters).

Contents

Chapter 1

Distributed Computing

In this chapter I completed part of MIT's 6.824 2014 spring unit[1]. (My original intention was to use OpenCourseWare version of the unit, however the content was not new and relied on old software that I could not get to run on my machine.)

1.1 Introduction

Before talking in-depth about the work I did on distributed computing, it is wise to ask what the topic is about. I have knowledge on parallel computing, which is similar, from a previous unit (COMP226), so I will contrast the two here:

- In parallel computing, processors are connected to each other directly. Examples include multi-core CPUs or GPUs. In distributed computing, the processors are kept on separate machines connected via a network. One example is Folding@home, in which work units are sent to each volunteer's personal computer via the Internet to be computed and returned to the server.
- In parallel computing, the same memory pool is usually shared by all processors, unlike in the distributed case where each CPU core gets its own.

1.1.1 Setting Up

Version 1.2 of the Go programming language was required to complete the lab. I obtained a copy for Linux from `http://golang.org/doc/install`.

[1] `http://css.csail.mit.edu/6.824/2014/index.html`

1.2 Lab 1

The link for the lab on MIT's website can be found here `http://css.csail.mit.edu/6.824/2014/labs/lab-1.html`. The full code I wrote can be found at the following repository: `https://bitbucket.org/DoxasticFox/lab-1/src/`

1.2.1 Overview

In this lab the aim was to implement the MapReduce algorithm. An overview of the algorithm is as follows [1]:

1. The initial input is split up into pieces so that it can be distributed to workers.
2. The workers are assigned pieces of the input by the master on which they execute a `map` function. This function takes the input and expresses it as a set of key/value pairs.
3. Sets of pairs from the `maping` step are forwarded to workers which execute a `reduce` function on the sets.
4. The pieces of output from the `reduce` stage can be collated to form the final output.

1.2.2 Some Definitions

To discuss the lab in further detail, it is useful to define some associated terms first:

Mutex Short for "mutual exclusion". In terms of go, this is a programming language construct designed to prevent sections of code from being executed, or resources from being accessed, at the same time. In other words, this prevents race conditions.

Race Condition What occurs when a resource is accessed at the same time.

Go Channels An alternative to mutexes which works by passing messages.

1.2.3 Part I

In this part, the MapReduce algorithm was used to in a non-distributed manner (`map` and `reduce` jobs were not sent to workers) to calculate a word count for each word in the King James Bible. The code I wrote to achieve this can be found in the repository at `src/main/go.wc`. The written code consists of three functions:

- NotLetter—A helper function to segment the text into words.

- Map—Takes a chunk of text as its argument and returns a list of pairs of the form $\langle w, 1\rangle$, where w is a word in the text.

- Reduce—Takes a word w and list of words l and returns the number of times w occurs in l.

In the MapReduce library implementation provided by MIT, the algorithm proceeds by splitting the Bible's text into pieces, calling DoMap (the caller of Map) on those pieces to produce intermediate files, calling DoReduce (the caller of Reduce) on those files, then merging the files produced to generate a list of word counts.

1.2.4 Part II

This part concerned the implementation of a master to allocate map and reduce jobs. What the Map and Reduce functions do are unimportant in this exercise. The implementation of this part can be found in the repository at src/mapreduce/master.go. (Actually this is the code from part III.) The code I wrote is simply the RunMaster function. An abridged version is shown here:

```go
func (mr *MapReduce) RunMaster() *list.List {
    mr.Workers = make(map[string]*WorkerInfo)
    mr.WorkersMtx = &sync.Mutex{}

    /* incessantly listen for new workers and add them
        to mr.Workers */
    go func() {
        for {
            var wi WorkerInfo
            // wait for worker address to come down channel
            wi.address = <-mr.registerChannel

            mr.WorkersMtx.Lock()
            mr.Workers[wi.address] = &wi
            mr.WorkersMtx.Unlock()
        }
    }()

    /* Map... */
    for i := 0; i < mr.nMap; {
        mr.WorkersMtx.Lock()
        for _, w := range mr.Workers {
            // try assigning reduce job. If successful do i++
            if i >= mr.nMap { break }
        }
        mr.WorkersMtx.Unlock()
    }

    /* ...reduce... */
    for i := 0; i < mr.nReduce; {
        mr.WorkersMtx.Lock()
        for _, w := range mr.Workers {
            // try assigning reduce job. If successful do i
                ++
            if i >= mr.nReduce { break }
        }
        mr.WorkersMtx.Unlock()
    }
    return mr.KillWorkers() // ...termination.
}
```

mr.Workers is a list of workers to which remote procedure calls (RPCs) may be made. The library provided by MIT made RPCs using Unix domain sockets. A mutex was used here as the goroutine (starting "go func() {...") runs concurrently with the Map and Reduce functions, adding new workers to mr.Workers when they are registered.

1.2.5 Part III

This part built on part II to handle the failure of workers. (In this case workers failed simply: they did not respond when assigned a job.) In the provided implementation, each worker had a set of properties associated it. To detect failures, a failed property was added which was permanently set to true if the worker did not respond. As well as this, the inner loops in the map and reduce stages of the algorithm were changed from, for example, this:

```
for _, w := range mr.Workers {
    // try assigning reduce job. If successful do i++
    if i >= mr.nMap { break }
}
```

to this:

```
for _, w := range mr.Workers {
    if w.failed { continue }
    // try assigning reduce job. If successful do i++
    // else w.failed = true
    if i >= mr.nMap { break }
}
```

1.3 Lab 2

The link for the lab on MIT's website can be found here: http://css.csail.mit.edu/6.824/2014/labs/lab-2.html. The full code I wrote can be found at the following repository: `https://bitbucket.org/DoxasticFox/lab-2/src/`

1.3.1 Overview

In this lab a primary/backup replication service for a key/value database was set up. Clients who needed to access (read or write) a key/value pair were required to do so by making a RPC to a primary server, which was paired with a backup. Which server was the primary and which was the backup was decided by the *viewservice*, which replied with numbered *views* containing the primary and backups' Unix sockets.

This lab spanned two weeks (one for each part) in the MIT course and provided hands-on knowledge of at-most-once semantics, mutexes, go channels and fault tolerance in distributed systems.

1.3.2 A Definition

To discuss the lab itself, it is useful to define the associated terminology:

At-most-once Semantics An RPC is said to have this if the call executes exactly once or not at all. In order to implement this semantics duplicate RPC packets must be detected.

1.3.3 Part A

In this part, the key/value service was given fault tolerance by adding a viewservice which designated the primary and backup roles for two subordinate servers. This was achieved by performing the following steps every time the viewservice's `Ping` function in `src/viewservice/server.go` was remotely called:

1. **Initialisation (`Ping` function variables)**—A snapshot of the current view was recorded along with the time of the `Ping` RPC.

2. **Initialisation (viewservice)**—It is checked if this is the viewservice's first received `Ping`. If it is, the first pinging server is unconditionally set as the primary.

3. **Detection of Primary's Acknowledgement**—An "acknowledged" flag is set if the primary server received the latest view. The view can only be updated if the primary has acknowledged the previous update. This rule prevents the viewservice from getting more than one view ahead of the key/value servers.

4. **Update of the View**—Dead servers are removed, promotable servers are promoted, then the view number is incremented if the view changed.

Note well that, to save resources, I should have written the code so that the view is updated every `Tick`, however I did not have enough time to learn how to do so correctly.

1.3.4 Part B

In this part, fault tolerance was given to the key/value service: it was required to continue operating normally so long as there was never a time at which no primary or backup server was alive and; even if they were unable to access the viewservice momentarily.

To help give fault tolerance, `tick` in `src/pbservice/server.go` was written so that the key/value pairs were sent to the (new) backup if it changed since the previous tick. The backup was kept up-to-date by forwarding key/value pairs to it when they were sent to the primary with a `Put` RPC.

A `Commit` flag was given to `common.go` to ensure at-most-once semantics:

```go
type PutArgs struct {
    Key string
    Value string
    DoHash bool // For PutHash
    Commit bool // Whether to commit changes to the database
}
```

In my implementation, to perform a put RPC, the client would send the datagram with the `Commit` flag set to false, indicating that the key/value pair should not be stored. Upon receiving a reply, the client would resend with the flag toggled at which point the pair would be stored.

In retrospect, at-most-once semantics could have been achieved better (that is with less network utilisation and more reliability) by using an acknowledgement and sequence number-style mechanism like in TCP (see Section 2.3). Nonetheless my server implementation still passed the unit tests.

Chapter 2

Networking

In many web browsers, pressing F11 enters full screen mode—something which I do from time to time. The keyboard's function keys are somewhat awkwardly placed however, making it easy to press an adjacent key. Inevitably, I have pressed F12 by mistake. In modern browsers (I use Firefox at the moment) doing so pops up a pane with curiously labelled tabs, including one which says "Network" as in Figure 2.1. One might wonder what a feature like this is doing in a program like a browser. Indeed, this common, inconspicuous feature has the ability to evoke a substantial line of questioning.

In this chapter I aim to satisfy some of those more salient questions that arise as I experiment with freely available networking-related programs.

2.1 How to Use This Chapter

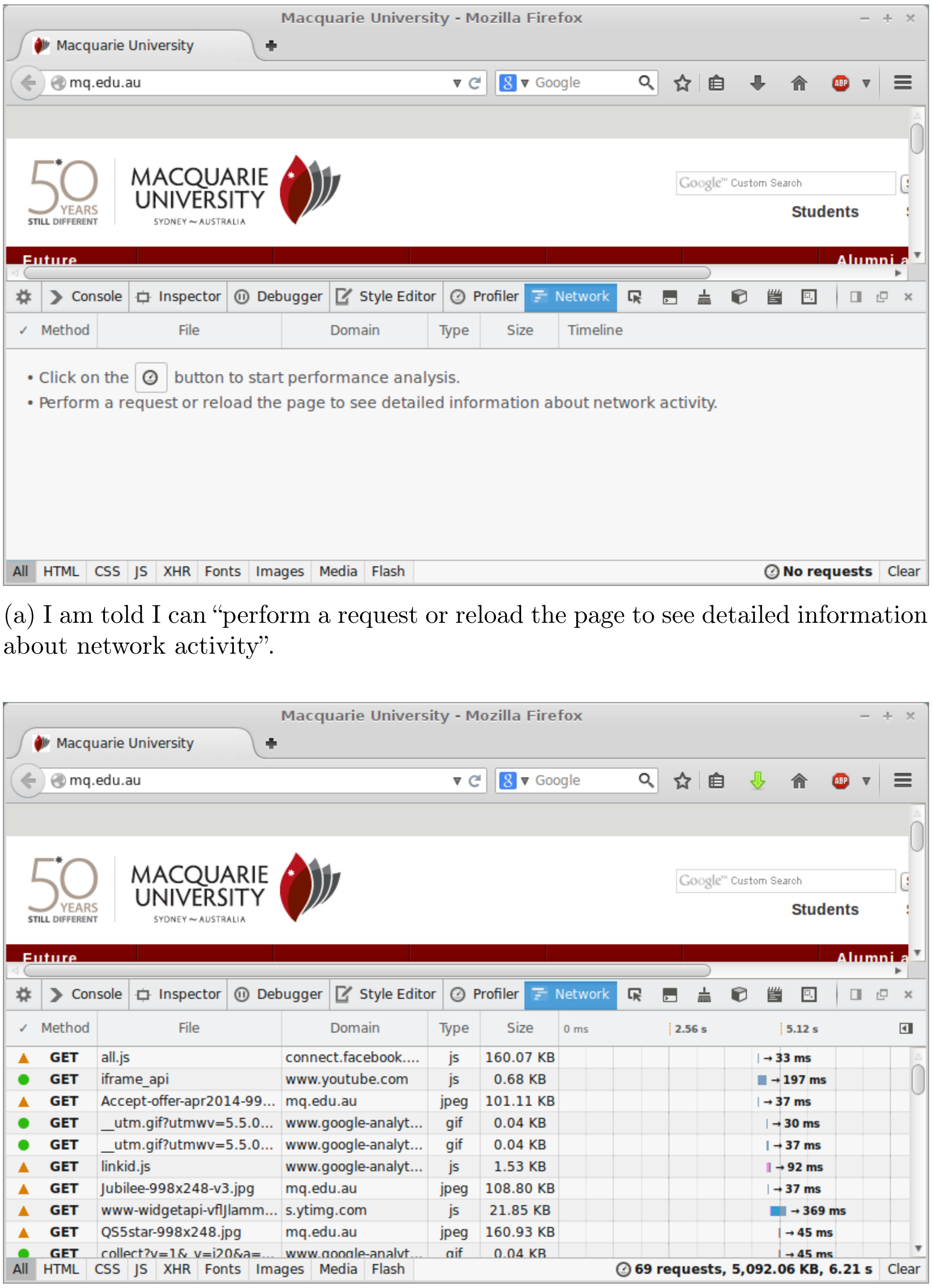
In the margin I have included numbers for the many questions that arise, according to the section they appear under, and in the order which they are asked. For the remainder of this chapter, when I make reference to a question without mentioning the section number it is understood that I am talking about the question in the same section it is referenced.

2.2 Exploring Firefox's Developer Tools

A few curious features and questions on them can be noted about Figure 2.1b:

1. What does each row represent? **Q2.1**
2. Why does each row have a triangle or circle? **Q2.2**
3. The "Method" column says **GET** a lot. What does that mean? **Q2.3**

The answer to Question 1 is not of prime importance but helps to contextualise answers to other questions, so I will provide it briefly here. Taking



Method	File	Domain	Type	Size	Timeline

- Click on the button to start performance analysis.
- Perform a request or reload the page to see detailed information about network activity.

No requests

(a) I am told I can "perform a request or reload the page to see detailed information about network activity".

	Method	File	Domain	Type	Size	0 ms 2.56 s 5.12 s
▲	GET	all.js	connect.facebook....	js	160.07 KB	→ 33 ms
●	GET	iframe_api	www.youtube.com	js	0.68 KB	→ 197 ms
▲	GET	Accept-offer-apr2014-99...	mq.edu.au	jpeg	101.11 KB	→ 37 ms
●	GET	__utm.gif?utmwv=5.5.0...	www.google-analyt...	gif	0.04 KB	→ 30 ms
●	GET	__utm.gif?utmwv=5.5.0...	www.google-analyt...	gif	0.04 KB	→ 37 ms
▲	GET	linkid.js	www.google-analyt...	js	1.53 KB	→ 92 ms
▲	GET	Jubilee-998x248-v3.jpg	mq.edu.au	jpeg	108.80 KB	→ 37 ms
▲	GET	www-widgetapi-vflJlamm...	s.ytimg.com	js	21.85 KB	→ 369 ms
▲	GET	QS5star-998x248.jpg	mq.edu.au	jpeg	160.93 KB	→ 45 ms
●	GET	collect?v=1&_v=j20&a=...	www.google-analyt...	gif	0.04 KB	→ 45 ms

69 requests, 5,092.06 KB, 6.21 s

(b) What I see when I reload the page.

Figure 2.1: The Network tab under Firefox's Developer Tools when visiting `http://mq.edu.au`.

a look at the source code for site shown in Figure 2.1 (by right-clicking the page and clicking "View Page Source"), it can be seen that the "File" and "Domain" columns show URLs that appear in the source code. For example, the file `Jubilee-988x248-v3.jpg` appears in Figure 2.1b because it is mentioned at line 302 in the source. (A snippet from it is shown as Listing 2.1.) Each URL in the source is, ostensibly, a resource which must be *requested* from a web server in order to load the web page in full. This leads to a question which might be better than that in Question 1:

4. What is a request and what does my computer send and receive when my browser makes one? **Q2.4**

Clicking on a row gives a hint towards the answer to Question 2. Upon doing so the pane pictured in Figure 2.2 appears, showing that the triangle represents the `304 Not Modified` status code and the circle, `200 OK`. What do these codes mean? Searching Google for "304" returns The Internet Engineering Task Force's (IETF) RFC 2616 [2] as the first result, in which these status codes were officially defined. From this I can conclude that:

- Status codes (like `304` and `200`), requests and GET methods are part of the Hypertext Transfer Protocol (HTTP).
- HTTP is a text-based protocol.
- This line of questioning is so far leading to an insight of networks which is abstracted away from, for example, how data find their way from one computer to the next.[1]

Additionally, in regards to Question 3, RFC 2616 states fairly plainly that:

- "The GET method means retrieve whatever information ... is identified by the Request-URI[2]" [2, Section 9.3].

Interestingly, knowing that HTTP is a text-based protocol and the form of GET requests, I should be able to retrieve a web page using only the `telnet` program. `http://www.w3.org/pub/WWW/TheProject.html` is given in RFC 2616 as an example [2, Section 5.1.2]. If a user was to type this URL into a browser, the following lines would be sent, as text, to the server which hosts it:

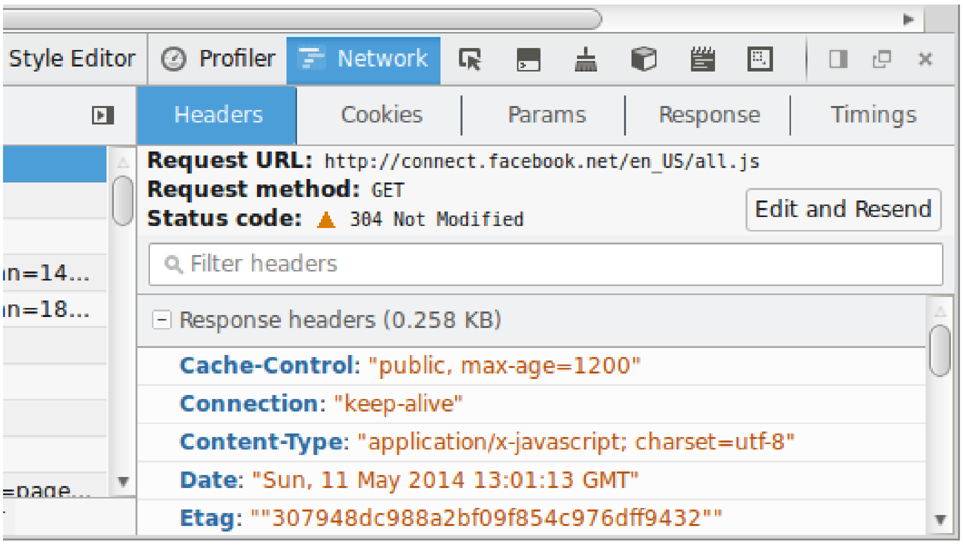
```
GET /pub/WWW/TheProject.html HTTP/1.1<CRLF>
Host: www.w3.org<CRLF>
<CRLF>
```

[1] This is because the explanation so far has been centered on HTTP, which operates at the application level [2, Section 1.1].

[2] This is roughly equivalent to a URL.

```
    <div class="next-btn"></div>
   </div>
   <div class="thumbnails">
    <ul>
    <li><a href="mq_templates/corporate/
      homepage_banner_rotator/images/2014/Tokyo-MY-
      v2-homepage998x248banner.jpg"></a><a href="
      http://mq.edu.au/courses/areas_of_study/#ac=
      mid-year2014_tokyo_creative" onclick="_gaq.
      push(['rollup._trackEvent', 'MQ Homepage', '
      banner', 'Tokyo Creative']);"></a></li>
    <li><a href="mq_templates/corporate/
      homepage_banner_rotator/images/2014/Accept-
      offer-apr2014-998x248.jpg"></a><a href="http
      ://students.mq.edu.au/student_admin/
      enrolmentguide/acceptyouroffer" onclick="_gaq.
      push(['rollup._trackEvent', 'MQ Homepage', '
      banner', 'Accept your offer']);"></a></li>
     <li><a href="mq_templates/corporate/
       homepage_banner_rotator/images/2014/Jubilee
       -998x248-v3.jpg"></a><a href="http://jubilee
       .mq.edu.au/homepage/1" onclick="_gaq.push(['
       rollup._trackEvent', 'MQ Homepage', 'banner
       ', 'Jubilee']);"></a></li>
     <li> <a href="mq_templates/corporate/
       homepage_banner_rotator/images/2014/QS5star
       -998x248.jpg"></a> <a href="http://www.mq.
       edu.au/future_students/international/
       why_choose_macquarie/our_reputation/
       university_rankings/" onclick="javascript:
       _gaq.push(['rollup._trackEvent', 'MQ
       Homepage', 'banner', 'QS 5 star']);"></a></
       li>
    </ul>
   </div>
  </div>
 </div>
</div>
<!-- endMain Gallery container -->
```

Listing 2.1: A code snippet showing the source for the home page of Macquarie University.



Request URL: http://connect.facebook.net/en_US/all.js
Request method: GET
Status code: ▲ 304 Not Modified

Response headers (0.258 KB)

Cache-Control: "public, max-age=1200"
Connection: "keep-alive"
Content-Type: "application/x-javascript; charset=utf-8"
Date: "Sun, 11 May 2014 13:01:13 GMT"
Etag: ""307948dc988a2bf09f854c976dff9432""

Figure 2.2: The HTTP headers for a request.

(Here <CRLF> is used to denote a carriage return followed by a linefeed.) These two lines form a *request*. Using this example I can GET http://mq.edu.au via the terminal as shown in Listing 2.2. Here the three lines ending with <CRLF> (the return key) are my input. I set telnet to use port 80 as RFC 2616 mentions that it is the default port, citing Reynolds and Postel [3]. I would like to know:

5. What is a port? **Q2.5**

The RFC also mentions that "HTTP communication usually takes place over TCP/IP connections":

6. What is TCP/IP (or, for that matter, either one)? **Q2.6**

After all this, I have addressed Question 4 in part, but the explanation so far is too high-level to be satisfying. Additionally, I noticed that telnet has found the IP address for mq.edu.au, but:

7. How did telnet find the IP address? **Q2.7**

8. Why is it necessary to find it in the first place? **Q2.8**

Moreover, now that I have seen some HTTP message text, I wonder:

9. Why does the response in Listing 2.2 appear to have sections (lines 8-16, lines 18-26)? **Q2.9**

Turning again to RFC 2616, I have my answer to Question 9 [2, Section 4]. The three main sections shown in Listing 2.2 are:

```
christian@christian-pc ~ $ telnet mq.edu.au 80<CRLF>
Trying 137.111.223.158...
Connected to mq.edu.au.
Escape character is '^]'.
GET / http/1.1<CRLF>
Host: mq.edu.au<CRLF>
<CRLF>
HTTP/1.1 200 OK
Date: Mon, 12 May 2014 13:21:18 GMT
Server: Apache
Cache-Control: max-age=7200, proxy-revalidate
Expires: Mon, 12 May 2014 15:21:18 GMT
Vary: Accept-Encoding,User-Agent
Transfer-Encoding: chunked
Content-Type: text/html

4b7d
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:
    lang=
"en">
```

The response body (HTML code) has been omitted for brevity.

```
</html>

0



Connection closed by foreign host.
```

Listing 2.2: Performing a HTTP request in telnet.

Request (lines 5-7) *As above.* Headers (additional information about the request) may be included, but lines 5 and 6 are mandatory.

Status-line and **response message headers** (lines 8-16) The first line in this section is the status-line. It gives a HTTP status code like the ones I noted earlier. The subsequent lines are the response message headers, which contain any additional information about the response which cannot be placed in the status-line (e.g. things like the age of the requested resource and how long the client should cache it).

Response message body (lines 17-26) In this case, the message body is simply the HTML code of the requested page, however other media like images may be sent instead.

It should be noted that lines 17 and 23 are not HTML because the server is using chunked encoding[3]. These lines are the size of the chunks [2, Section 3.6.1] that the client should expect. The last chunk (line 23 here) is necessarily of size zero.

2.3 What is "Underneath" HTTP? (The TCP Section)

In the previous section I asked some foundational questions about networks and found upon answering some that browsers access web pages by performing HTTP requests. Continuing along this line I found that generally more than one request is performed and usually by the GET method. The response to the request is sent back as text which includes not only the requested file, but metadata like status codes and headers as well. While learning of this, Questions 2.5, 2.6, 2.7 and 2.8 arose, but were left open.

To answer Question 2.6, I turned to IETF's website again. Searching for "TCP/IP" conveniently returns RFC 1180, "A TCP/IP Tutorial" [4]. Now, in Section 2, "TCP/IP" has been given the unfortunately vague definition of "anything and everything related to the specific protocols of TCP and IP". Fortunately, the situation is elucidated somewhat by Section 2.1, which gives the diagram in Figure 2.3a, showing common Internet protocols, in relation to each other, as a hierarchy. In particular it shows that TCP works "over the top of" IP. This diagram raises obvious questions which I will address later:

1. What is ARP? **Q3.1**
2. How does Ethernet work? **Q3.2**

[3] Initially I thought they were not HTML because the encoding of the message sent by the server did not match the encoding used in my terminal.

```
--------------------------
|                        |
|  network applications  |
|                        |
|... \ | /  ..  \ | / ...|
|    -----      -----    |
|    |TCP|      |UDP|    |
|    -----      -----    |
|        \      /        |
|        --------        |
|        |  IP  |        |
| -----  -*------        |
| |ARP|   |              |
| -----   |              |
|     \   |              |
|     ------             |
|     |ENET|             |
|     ---@--             |
---------|----------------
         |
    -----o--------------
           Ethernet Cable
```

(a) The original ASCII diagram from RFC 1180. The layout has been slightly modified here.

```
--------------------------
|     ------             |
|     |HTTP|             |
|     ------             |
|       |                |
|     -----              |
|     |TCP|              |
|     -----              |
|         \              |
|         --------       |
|         |  IP  |       |
|  -----  -*------       |
|  |ARP|   |             |
|  -----   |             |
|      \   |             |
|      ------            |
|      |ENET|            |
|      ---@--            |
----------|---------------
          |
     -----o--------------
            Ethernet Cable
```

(b) The diagram modified to show the place of HTTP (and the scope of this chapter).

Figure 2.3: Schematics showing a hierarchy of commonplace protocols though which data travel. The lines represent the flow of data, the asterisk an IP address, the ampersand an Ethernet (MAC) address and the "o" the transceiver (which converts electrical signals to Ethernet frames and vice versa). The boxes represent parts of the network which process data. "ENET" is one part which processes Ethernet frames.

Now that I have the diagram in Figure 2.3a I can place HTTP at the top of the entire stack, at the application-level. Additionally, I have an overview of common network protocols in the hierarchy so that I can deal with TCP and IP separately. I will start with the former as it is the next lowest in the hierarchy after HTTP. RFC 793 gives the purpose of TCP along with an overview of its operation [5]. A summary of TCPs basic functions is as follows:

- **Basic data transfer**—TCP is able to be used to transfer a continuous stream of octets (bytes) via the Internet by grouping them into *segments*.
- **Reliability**—TCP is able to deal with segments which are damaged, lost, duplicated or delivered out of order. Damage is dealt with by giving each segment a checksum. The remaining issues are dealt with by giving segments ACK and sequence numbers which count the number of octets transmitted. The receiver must correctly ACKnowledge each sent segment within a timeout period, else it is resent.
- **Flow control**—Using TCP, the receiver can set the maximum allowed number of octets sent without replying with an ACK. This is known as the *window*.
- **Multiplexing**—By providing a set of ports to each host[4] TCP allows that host to use or provide multiple services at the same address. For example, in principle it is possible for the host at the single address, `mq.edu.au` to be running two services, `telnet` and the Macquarie University website, by having the former on port 23 and the latter on 80. (This answers Question 2.5 directly!)

Figure 2.4 shows the parts of the TCP header which store the aforementioned sequence numbers and other fields [5, Section 3.1].

To make this concrete I have performed the same GET request as in the previous section of this chapter while running Wireshark, a segment analyser, to view the TCP segments. Figure 2.5 shows the Wireshark window after having performed the request. In particular, in Figure 2.6, I can now see firsthand the way in which the segments are ACKed:

- Packet 1—The connection with the server (137.111.223.158) is established by sending a SYN segment. The ACK number does not matter as there is nothing for the client to acknowledge. Notice that although the sequence number is shown as zero, this is not the way it appears in the segment. Wireshark only shows the starting number as zero for readability.

[4] This is roughly the same as a computer or some other network device.

	0								1								2								4							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	Source Port																Destination Port															
32	Sequence Number																															
64	ACK Number																															
96	Data Offset				Reserved for Future Use						Control Bits						Window Size															
128	Checksum																Urgent Pointer															
160	Options (Padded to 4-byte Boundary)																															
160-480 ...	Data																															

Figure 2.4: The TCP packet format as defined in RFC 793. The numbers along the top row are the byte and bit numbers. Those along the left side are also for bits. Notice that the data can start *between* bits 160-480 because the options section is of variable size. The payload (data) *length* is similarly variable.

- Packet 2—The server ACKnowledges the first SYN segment by taking the previously sent sequence number and incrementing it by one. Because the server has not previously sent anything in this connection, it sends a sequence number of zero.

- Packet 3—The client sends a segment with a sequence number of one indicating the SYN it sent just before. It also ACKnowledges the SYN by taking the sequence number sent with it, incrementing it and sending it as an ACK.

The client and server continue to communicate like this throughout the duration of the connection, only, just like SYNs have a "weight" of one, segments containing data are weighted by their length. Once the connection closes, the systems which were involved can free resources associated with maintaining it.

As well as the ACK and sequence numbers, Wireshark can be used to show the TCP segment as binary[5]. The following is the first 44 bytes of frame 22 from Figure 2.5:

```
0000000: 00000000 01010000 11000001 11111001  .P..
0000032: 01001001 10000111 01010101 00101110  I.U.
0000064: 00000110 00000111 10000010 00111110  ...>
0000096: 10000000 00010000 00000000 00101110  ....
0000128: 11101000 01010001 00000000 00000000  .Q..
0000160: 00000001 00000001 00001000 00001010  ....
0000192: 11010101 10011101 10101110 11101010  ....
0000224: 00001100 11100101 00100111 10111011  ..'.
0000256: 01001000 01010100 01010100 01010000  HTTP
0000288: 00101111 00110001 00101110 00110001  /1.1
0000320: 00100000 00110010 00110000 00110000   200
```

The left-most column shows the bit numbers. Alternating grey and white has been used here to show field boundaries. (Compare this to Figure 2.4). Binary conversions for some fields are as shown here:

- Source port: $00000000\ 01010000_2 = 80_{10}$

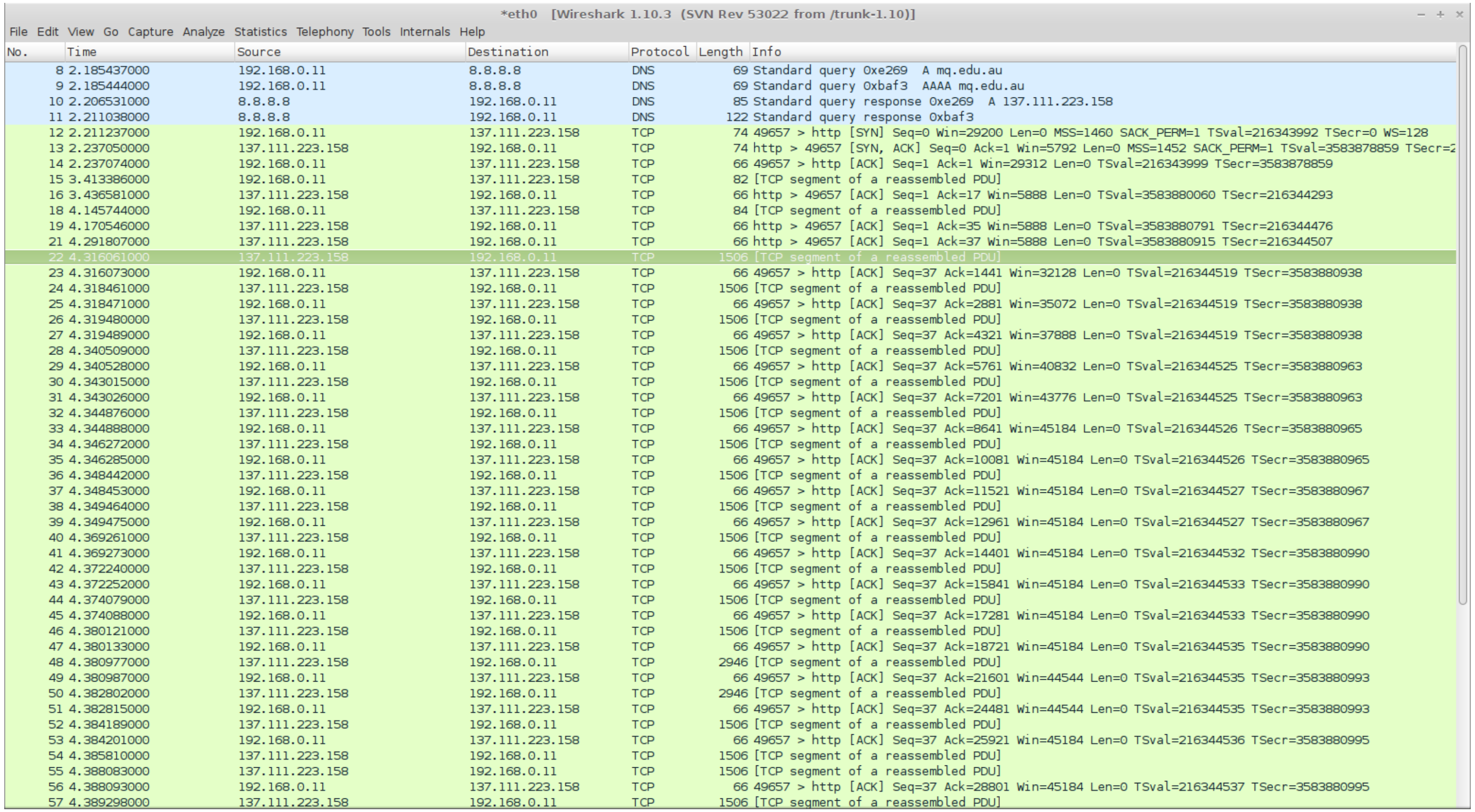
- Destination port: $11000001\ 11111001_2 = 49{,}657_{10}$

- Sequence number: Notice that as this is the 1st (relative) sequence number, the 0th must have been $1{,}233{,}605{,}933_{10}$. The conversion is as follows: $01001001\ 10000111\ 01010101\ 00101110_2 = 1{,}233{,}605{,}934_{10}$.

- ACK number: $00000110\ 00000111\ 10000010\ 00111110_2 = 101{,}155{,}390_{10}$

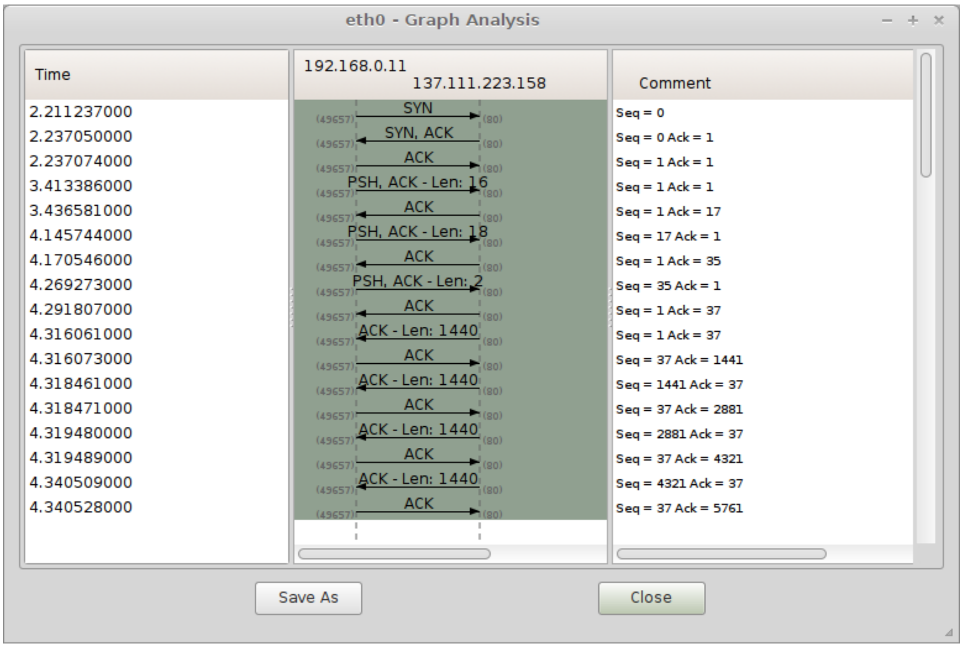
Notice that the options field is 12 bytes long in this case. The data field, which contains some of the HTTP text, begins at the 256th bit and continues on further than shown here.

[5] Wireshark does not show the output exactly as is done here. The formatting has been changed for easy comparison with Figure 2.4.


*eth0 [Wireshark 1.10.3 (SVN Rev 53022 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

```
No.   Time           Source            Destination       Protocol Length Info
    8 2.185437000    192.168.0.11      8.8.8.8           DNS          69 Standard query 0xe269  A mq.edu.au
    9 2.185444000    192.168.0.11      8.8.8.8           DNS          69 Standard query 0xbaf3  AAAA mq.edu.au
   10 2.206531000    8.8.8.8           192.168.0.11      DNS          85 Standard query response 0xe269  A 137.111.223.158
   11 2.211038000    8.8.8.8           192.168.0.11      DNS         122 Standard query response 0xbaf3
   12 2.211237000    192.168.0.11      137.111.223.158   TCP          74 49657 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=216343992 TSecr=0 WS=128
   13 2.237050000    137.111.223.158   192.168.0.11      TCP          74 http > 49657 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1452 SACK_PERM=1 TSval=3583878859 TSecr=2
   14 2.237074000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=216343999 TSecr=3583878859
   15 3.413386000    192.168.0.11      137.111.223.158   TCP          82 [TCP segment of a reassembled PDU]
   16 3.436581000    137.111.223.158   192.168.0.11      TCP          66 http > 49657 [ACK] Seq=1 Ack=17 Win=5888 Len=0 TSval=3583880060 TSecr=216344293
   18 4.145744000    192.168.0.11      137.111.223.158   TCP          84 [TCP segment of a reassembled PDU]
   19 4.170546000    137.111.223.158   192.168.0.11      TCP          66 http > 49657 [ACK] Seq=1 Ack=35 Win=5888 Len=0 TSval=3583880791 TSecr=216344476
   21 4.291807000    137.111.223.158   192.168.0.11      TCP          66 http > 49657 [ACK] Seq=1 Ack=37 Win=5888 Len=0 TSval=3583880915 TSecr=216344507
   22 4.316061000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   23 4.316073000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=1441 Win=32128 Len=0 TSval=216344519 TSecr=3583880938
   24 4.318461000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   25 4.318471000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=2881 Win=35072 Len=0 TSval=216344519 TSecr=3583880938
   26 4.319480000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   27 4.319489000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=4321 Win=37888 Len=0 TSval=216344519 TSecr=3583880938
   28 4.340509000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   29 4.340528000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=5761 Win=40832 Len=0 TSval=216344525 TSecr=3583880963
   30 4.343015000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   31 4.343026000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=7201 Win=43776 Len=0 TSval=216344525 TSecr=3583880963
   32 4.344876000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   33 4.344888000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=8641 Win=45184 Len=0 TSval=216344526 TSecr=3583880965
   34 4.346272000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   35 4.346285000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=10081 Win=45184 Len=0 TSval=216344526 TSecr=3583880965
   36 4.348442000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   37 4.348453000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=11521 Win=45184 Len=0 TSval=216344527 TSecr=3583880967
   38 4.349464000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   39 4.349475000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=12961 Win=45184 Len=0 TSval=216344527 TSecr=3583880967
   40 4.369261000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   41 4.369273000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=14401 Win=45184 Len=0 TSval=216344532 TSecr=3583880990
   42 4.372240000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   43 4.372252000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=15841 Win=45184 Len=0 TSval=216344533 TSecr=3583880990
   44 4.374079000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   45 4.374088000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=17281 Win=45184 Len=0 TSval=216344533 TSecr=3583880990
   46 4.380121000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   47 4.380133000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=18721 Win=45184 Len=0 TSval=216344535 TSecr=3583880990
   48 4.380977000    137.111.223.158   192.168.0.11      TCP        2946 [TCP segment of a reassembled PDU]
   49 4.380987000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=21601 Win=44544 Len=0 TSval=216344535 TSecr=3583880993
   50 4.382802000    137.111.223.158   192.168.0.11      TCP        2946 [TCP segment of a reassembled PDU]
   51 4.382815000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=24481 Win=44544 Len=0 TSval=216344535 TSecr=3583880993
   52 4.384189000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   53 4.384201000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=25921 Win=45184 Len=0 TSval=216344536 TSecr=3583880995
   54 4.385810000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   55 4.388083000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
   56 4.388093000    192.168.0.11      137.111.223.158   TCP          66 49657 > http [ACK] Seq=37 Ack=28801 Win=45184 Len=0 TSval=216344537 TSecr=3583880995
   57 4.389298000    137.111.223.158   192.168.0.11      TCP        1506 [TCP segment of a reassembled PDU]
```

Figure 2.5: Wireshark after having performed a GET request to `mq.edu.au` using `telnet`. Some segments have been filtered from the output you see.



Time	192.168.0.11 137.111.223.158	Comment
2.211237000	SYN	Seq = 0
2.237050000	SYN, ACK	Seq = 0 Ack = 1
2.237074000	ACK	Seq = 1 Ack = 1
3.413386000	PSH, ACK - Len: 16	Seq = 1 Ack = 1
3.436581000	ACK	Seq = 1 Ack = 17
4.145744000	PSH, ACK - Len: 18	Seq = 17 Ack = 1
4.170546000	ACK	Seq = 1 Ack = 35
4.269273000	PSH, ACK - Len: 2	Seq = 35 Ack = 1
4.291807000	ACK	Seq = 1 Ack = 37
4.316061000	ACK - Len: 1440	Seq = 1 Ack = 37
4.316073000	ACK	Seq = 37 Ack = 1441
4.318461000	ACK - Len: 1440	Seq = 1441 Ack = 37
4.318471000	ACK	Seq = 37 Ack = 2881
4.319480000	ACK - Len: 1440	Seq = 2881 Ack = 37
4.319489000	ACK	Seq = 37 Ack = 4321
4.340509000	ACK - Len: 1440	Seq = 4321 Ack = 37
4.340528000	ACK	Seq = 37 Ack = 5761

Figure 2.6: The sequence and ACKnowledgement numbers for a GET request to `mq.edu.au` (137.111.223.158).

2.4 Internet Protocol

In the previous section I established that TCP acts as a wrapper around HTTP (or some other application-level protocol) which provides reliability over unreliable networks, and allows for multiplexing by ports, amongst other benefits. I also discovered that IP further wraps around TCP segments. Now what remains in answering Question 2.6 is to find more thoroughly out what the Internet Protocol (IP) is. At the moment Questions 2.7, 2.8, 3.1 and 3.2 are still open.

I begin by consulting RFC 792 which states that the purpose of IP is the move datagrams through a network [6, Section 2.3] by its two main functions, addressing and fragmentation [6, Section 1.4]. It explains that data is split into fragments so that they may be routed through a computer network before being reassembled at the destination determined by an *Internet address*[6] [6, Section 2.3]—but:

1. Why fragment data to route over a network? Why not just send them in one stream? **Q4.1**

What IP does not provide are reliable communication (there are no ACKs, retransmissions or reordering of out-of-order datagrams) and flow control [6, Section 1.4]—functions which I found, in the previous section, TCP provide.

[6] Earlier in this chapter I mentioned “IP addresses”, which are the same thing.

Four key mechanisms by which IP operates are identified in the RFC [6, Section 1.4]:

Type of Service This indicates desired characteristics—such as latency, throughput and reliability—of the provided service. Since its specification in 1981 it has been split into the Differentiated Services Code Point (DSCP) [7] and Explicit Congestion Notification (ECN) [8] fields in the IP header.

Time to Live The IP packet's maximum lifespan as set by the sender. If the time to live of the packet is exceeded before it reaches its destination, the packet is destroyed.

Options This is usually not used, but allows each packet to have a timestamp, and desired security and routing constraints.

Header Checksum Using this, it can be checked that the packet was correctly transmitted. If it was not, it is discarded. TCP (or some other protocol) is responsible for detecting the lost packeted and arranging for it to be resent.

As well as these mechanisms, IP provides the two primary functions which were briefly mentioned before:

Addressing Each host on a network has a unique address. Addresses are simply four-byte numbers by which hosts are identified. In RFC 791 these are contrasted with (domain) names and routes—a name indicates what is sought, an address indicates where on the network it is [9] and a route indicates how to there. This answers Question 2.8.

Fragmentation Not only do data need to be broken into packets to be sent through a network, they may need to be further divided as they pass through from one network to another. This latter sense, the further division, is what is meant in RFC 791 by "fragmentation". In short, this is done by splitting the original packet into two others, copying their headers and updating the total length and fragment offset fields shown in Figure 2.7, and the more fragments flag appropriately.

In the above point on addressing, I used the word "route" without immediately questioning it further:

2. How is a route decided? **Q4.2**

Before I start answering another question, I would like to take a look at an IP packet in Wireshark. Just like I did in the previous section of this chapter, I will examine the 22$^{\text{nd}}$ frame from Figure 2.5 as binary:

	0								1								2								4							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	Version				IHL				Type of Service								Total Length															
32	Identification																Flags			Fragment Offset												
64	Time to Live								Protocol								Header Checksum															
96	Source Address																															
128	Destination Address																															
160	Options (Padded to 4-byte Boundary)																															
...	Data																															

Figure 2.7: The IP packet format as defined in RFC 791. Those along the left side are bit numbers. The options field may have zero length so that the data can start from the 160th bit onwards. The data field's size is variable.

```
0000000: 01000101 00000000 00000101 11010100  E...
0000032: 11111110 01010101 01000000 00000000  .U@.
0000064: 00110100 00000110 00011001 00001101  4...
0000096: 10001001 01101111 11011111 10011110  .o..
0000192: 11000000 10101000 00000000 00001011  ....
0000224: 00000000 01010000 11000001 11111001  .P..
0000256: 01001001 10000111 01010101 00101110  I.U.
```

For this packet the options field has zero length. The data field begins at the 256th bit and contains the entire TCP segment (only the first 8 bytes are shown here). Binary conversions for some interesting fields are thus:

- Internet Header Length (IHL): $01010_2 = 5_{10}$ 4-byte words. To get the length of the header in bytes, multiply the IHL by 4.
- Total Length: $0000010111010100_2 = 1492_{10}$ bytes.
- Protocol: $00000110_2 = 6_{10}$. The number 6 corresponds to TCP [3, Page 8], which is the protocol used to encode this packet's data field.
- Source: $10001001.01101111.11011111.10011110_2 = 137.111.223.158_{10}$
- Dest.: $11000000.10101000.00000000.00001011_2 = 192.168.0.11_{10}$

2.5 Address Resolution Protocol

In the previous section I answered Question 2.6 by finding that IP allows hosts on a network to be addressed and packets fragmented so that they can be successfully delivered. In this section I intend to *briefly* answer Question 3.1. Questions 2.7, 3.2, 4.1 and 4.2 remain open.

To address this question I take another look RFC 1180 which says that the Address Resolution Protocol (ARP) is used to translate from IP addresses (logical) to MAC addresses (physical). More precisely, it is used in networks of computers connected by a length of Ethernet cable [10, Abstract] (or equivalently, on the same local network). Only once a host's physical address is known can data be sent to it.

If the IP address has already been translated by a host, it will appear in a table of cached IP-MAC address pairs stored on that host. On my computer, I can access the table using the `arp` command[7] as shown in Figure 2.8.

The IP address `192.168.0.1` refers to my network's modem/router. The other two addresses are computers connected to it. My computer is not shown on the list. As an example, if my computer needs to send a datagram to `192.168.0.12` it will first have to check the table to find the corresponding MAC address, which, for that item of hardware on my network, is `bc:77:37:0f:3e:17`.

[7] I discovered this by running `man -k arp`.

```
[127]christian@christian-pc ~ $ sudo arp
[sudo] password for christian:
Address          HWtype  HWaddress           Flags Mask      Iface
192.168.0.1      ether   00:1e:2a:18:d4:2c   C               eth0
192.168.0.12     ether   bc:77:37:0f:3e:17   C               eth0
192.168.0.13     ether   bc:77:37:87:6c:20   C               eth0
```

Figure 2.8: Running `arp` in Unix.

If the IP address has not been translated by a host, it will try to find the MAC address for the IP address only once a packet is needed to be sent there. This is done by broadcasting an ARP request packet to all hosts on the local network which contains fields for the requester's IP and MAC address, and a destination IP and MAC address. This last field will be blank as that is what needs to be discovered. If a host has an IP address which matches the destination in the packet, a similar one will be sent in reply that has all its fields filled in [4, Section 4.3].

This is all well and good, except for the case when data needs to be sent to an IP address outside the local network:

1. How does a computer find the MAC address for a server on the Internet? **Q5.1**

2.6 Ethernet

In the previous section I learned that ARP allows hosts to find each other's physical address on a network. In this section I intend to satisfy Question 3.2. At this point Questions 2.7, 4.1, 4.2 and 5.1 remain unanswered.

When "Ethernet" was mentioned in Figure 2.3, ostensibly this was a reference to Ethernet frames as opposed to the physical medium (whose discussion is outside of the scope of this document). After researching other protocols, I feel familiar enough with Ethernet to discuss it only shortly here.

In Wireshark, "Ethernet II" frames can be seen enclosing all the data my computer sends and receives. As I have in previous sections, I will examine the 22nd frame shown in Wireshark:

```
0000000: 00011100 01101111 01100101 00110111  .oe7  1c 6f 65 37
0000032: 11010101 00110111 00000000 00011110  .7..  d5 37 00 1e
0000064: 00101010 00011000 11010100 00101100  *..,  2a 18 d4 2c
0000096: 00001000 00000000 01000101 00000000  ..E.  08 00 45 00
0000192: 00000101 11010100 11111110 01010101  ...U  05 d4 fe 55
```

Here a header specification is not necessary to make out a few of the fields. The second is the MAC address of my router which was shown in Figure 2.8. If that's the case, the 6 bytes before it probably make up the destination address. Indeed, is it the MAC address for my computer's network adaptor:

```
christian@christian-pc ~ $ sudo ifconfig | head -n 2
eth0      Link encap:Ethernet  HWaddr 1c:6f:65:37:d5:37
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.25
5.255.0
```

Wireshark tells me the third field indicates that the frame contains an IP packet (so presumably this is like IP's protocol field). The last field shown is the beginning of the IP data I saw in section 4.

Looking again at Figure 2.5, it can be seen that the length of the 22$^{\text{nd}}$ frame is 1506 bytes. Using this and the total length field for the IP packet from section 4, I can do a simple exercise to find that the length of this frame's header is $1506-1492 = 14$ bytes, which can be confirmed by manually counting them in the binary above.

2.7 Conclusion and Future Work

In this chapter I have familiarised myself with HTTP, TCP, IP, ARP and Ethernet by experimenting with freely available software and consulting relevant RFC documents from the IETF.

These protocols work in a stack with the more high-level HTTP at the top and the low-level Ethernet at the bottom. The lower layers help give essential network attributes such as routing and reliability, whereas the higher level protocols like HTTP are more application-specific and abstracted away from the task of transmitting data through the network.

Questions 2.7 (how did `telnet` find the IP address?), 4.1 (why fragment data?), 4.2 (how is a route decided?) and 5.1 (how does a computer find a webserver's MAC address?) remained unanswered in this document due to time constraints. Nonetheless, in my research on other questions, I believe that I have gained at least some degree of knowledge on them. Still, I feel that it is incumbent on me as a prospective software engineer to answer these important questions thoroughly in the future.

Chapter 3

Databases

3.1 Introduction

Databases are organised collections of data. This is definitely a broad definition. Indeed, a database may be something as rudimentary as a pen-and-paper phone book. In this Chapter, a computer science-centric view of databases is considered.

It should be noted that the order in which topics appear in this chapter is reminiscent of the steps taken to implement a database: it is written on paper so that flaws in its design may be most easily seen before being implemented in software, after which time changes are usually costly and time consuming.

3.2 ER Modelling

In entity-relationship (ER) modelling, there are many notational variants including Bachman notation [11], Barker's notation [12], and crow's foot notation (which is part of Barker's notation). Here, conventions from set theory will mostly be used.

3.2.1 A Mathematical Approach

Databases can be modelled using mathematics. To see this, first some definitions and notation need to be established.

Functions

A function f is a mapping from an element of one set $\mathcal{X}$ to an element of another (not necessarily distinct) set $\mathcal{Y}$. Such a mapping may be denoted $f : \mathcal{X} \to \mathcal{Y}$. Functions have the following properties:

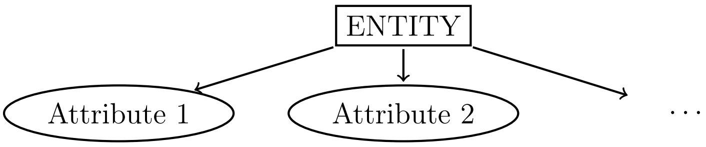


Figure 3.1: An entity-relationship diagram.

Well-defined For every element in the function's domain, the function maps to an element in its range. Symbolically, $\forall x \in \mathcal{X}(\exists y \in \mathcal{Y} : f(x) = y)$.

Single-valued Each element in the function's domain maps to only one element in its range. $\forall x \in \mathcal{X}\ \exists y_1, y_2 \in \mathcal{Y}(f(x) = y_1, f(x) = y_2 \implies y_1 = y_2)$.

Some liberties have been taken in this definition of a function—for example a set is never defined—however I am only interested in those properties pertinent to this discussion.

In addition to the above properties, a function necessarily falls into at least one of the following three classes:

Injection A function which maps to *at most* one element of its range is said to be injective.

Surjective A function which maps to *at least* one element of its range is said to be surjective.

Bijection A function which is both injective and surjective is a bijection.

Databases as Graphs

Now that functions have been defined, databases can be modelled as entity-relationship diagrams whose arrows are functions, as shown in Figure 3.1. Injective functions are denoted by arrows with tails $\rightarrowtail$. As well, each attribute of an entity is enclosed by an ellipse, whereas the entity itself is enclosed by a box and its name is written in capitals in its singular form.

Figure 3.2 shows a more concrete example of a graph taken from Johnson, Rosebrugh and Dampney [13] and slightly modified here. Note in it the injective functions. Intuitively, for example, the function SPECIALIST $\rightarrowtail$ MEDICAL PRACTITIONER may be read, "each medical practitioner is at most one specialist", which makes sense as a practitioner cannot be more than one person. On the other hand, the SPECIALIST → SPECIALISATION function does not impose an at-most-one *cardinality constraint* as it is quite possible, "a specialisation may have more than one specialist"[1].

[1] Actually, the relationships between entities are usually read in the direction of the

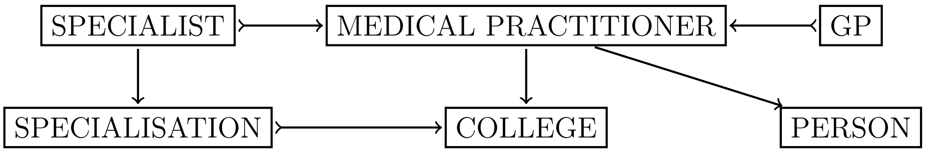


Figure 3.2: Part of a graph for a health informatics database.

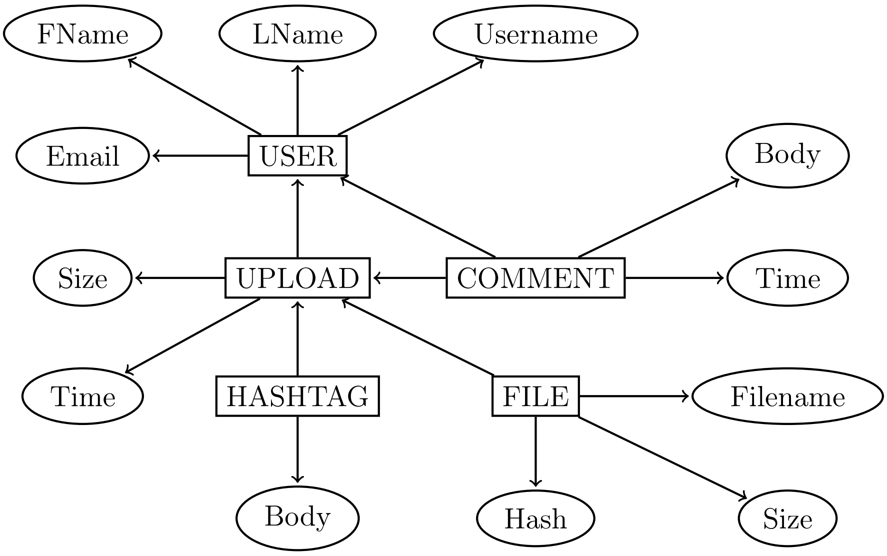


Figure 3.3: An ER diagram for a (feature-impoverished) file sharing website.

3.2.2 Example: A File Sharing Website

Figure 3.3 shows a graph for a file sharing website. In it we see there are many USERs, who each are able to make multiple UPLOADs and COMMENTs. As well, many COMMENTs and Twitter-style HASHTAGs can be made on UPLOADs. Unlike COMMENTs which are made by identifiable users, HASHTAGs are not (unless it is a rule that they can only be left by the uploader, which is a constraint that is awkward to show using the ER model). An UPLOAD can have many files.

It is important to check that the functions used are appropriate. For instance, while a MEDICAL PRACTITIONER can only be one SPECIALIST, the injection COMMENT $\rightarrowtail$ USER would be inappropriate unless for some reason a user should make no more than one comment.

Additionally, any parts of the graph which (do not) *commute* should be carefully checked. A diagram is said to do so when the composites of the functions along two paths are necessarily equal. In this example there is what

arrow—for example, "a SPECIALIST is a MEDICAL PRACTITIONER. I have avoided that here to make my discussion of the cardinalities clearer.

looks like a commuting diagram formed by the paths COMMENT → UPLOAD → USER and COMMENT → USER. This is not a commuting diagram however because users are not restricted to commenting on their own uploads.

Compare this to Figure 3.2 where a SPECIALIST is a MEDICAL PRACTITIONER who is a MEMBER of a COLLEGE, and a SPECIALIST has a SPECIALISATION which is a COLLEGE. In this case, the MEDICAL PRACTITIONER and SPECIALISATION must refer to the same COLLEGE, given a particular SPECIALIST, so the diagram commutes.

3.3 Relational Databases

A relational database is one which uses a collection of tables to store and organise data. The name comes from the fact that data in a table is called a relation. This usage of this word is distinct from, but related to, the sense used in mathematics where, for example, a function is kind of relation.

3.3.1 Tables

A database's table is similar to one you might create in a spreadsheet, with a few possible differences:

- Data in the table is changed only by `INSERT`, `DELETE` and `UPDATE` *queries* (see Section 3.3.2) which add, remove and change rows, respectively.
- Columns (attributes) cannot usually be easily added or removed once the database is implemented in software. Indeed, if this needs to be done, not enough time was spend designing the database.
- Rows must be unique.

To express an ER model as a table, one would consider each entity as a different table. The entity's attributes become the columns (or fields) of the table and rows symbolise instances of an entity. Functions between entities are encoded using primary keys and foreign keys:

Primary Key This is one or more columns in the table used to uniquely identify rows. Often each row would be assigned an arbitrary integer.

Foreign Key This is one or more columns containing the primary keys from another table.

Considering Figure 3.2, each PERSON would be given a primary key which would be included in the MEDICAL PRACTITIONER table as a foreign key. The reason the keys would not be assigned the other way around is that the function needs to map from a MEDICAL PRACTITIONER to a PERSON.

3.3.2 SQL

SQL (Structured Query Language) allows users to view rows from tables using the SELECT statement. INSERT, DELETE and UPDATE operations are also possible (though these are not queries proper).

Most SQL queries are of the following form

```
SELECT column1, column2, ..., columnn FROM table1 WHERE
    constraints_here;
```

which would show the columns column1, column2—all the way up to columnn—from table1 which meet the given constraints (which are optional). An asterisk can be used in place of the columns to show all rows. For example, if the database represented by Figure 3.2 was implemented, the query

```
SELECT * FROM specialists;
```

would get every row from specialists, showing every column.

3.3.3 Example: The File Sharing Website Revisited

Now that I know something about relational databases, I can express the file sharing site as one. I am using MySQL as my DataBase Management System (DBMS), which can manage multiple databases simultaneously, so I need to explicitly create the database and give it a name:

```
CREATE SCHEMA `filesharing_website` DEFAULT CHARACTER SET utf8 ;
```

Then the tables should be created. I will only show a few here starting with the USER table:

```
CREATE TABLE `filesharing_website`.`user` (
    `username` VARCHAR(20) NOT NULL,
    `first_name` VARCHAR(45) NULL,
    `last_name` VARCHAR(100) NULL,
    `email` VARCHAR(256) NULL,
    PRIMARY KEY (`username`));
```

Now the UPLOAD table is made:

```
CREATE TABLE `filesharing_website`.`upload` (
    `idupload` INT NOT NULL,
    `fk_username` VARCHAR(20) NOT NULL,
    `time` DATETIME NOT NULL,
    `size` INT NOT NULL,
    PRIMARY KEY (`idupload`),
    INDEX `username_idx` (`fk_username` ASC),
    CONSTRAINT `username`
        FOREIGN KEY (`fk_username`)
        REFERENCES `filesharing_website`.`user` (`username`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION);
```

Notice that the idupload field was not present in Figure 3.3, but appears here. This primary key ensures each row can be uniquely identified. The fk_username field was also added, which is a foreign key used to refer to a row in the USER table that allows us to say which user made the upload.

The above SQL statements were generated automatically in MySQL Workbench, which is part of the reason lines like ON DELETE NO ACTION can be seen. They refer to the fact that the DBMS can have DELETE and UPDATE operations CASCADE changes through to other tables. If MySQL did not allow this, to delete a user account for instance would require all the following steps be done manually:

1. The corresponding row in the USER table is deleted.
2. Any rows in the UPLOAD table with a foreign key from the now deleted user must be deleted.
3. Any rows in the COMMENT table with a foreign key from the now deleted user *and* upload must both be deleted.
4. The same is done for other tables with newly orphaned foreign key references.

Some Queries on The Database

A well-designed database should allow most queries to be expressed succinctly. It is then fitting that I test my new database with some queries. For example, *how many uploads have a size greater than 10 megabytes?*

```
mysql> select * from upload where size > 10*1024*1024;
+----------+-------------+---------------------+----------+
| idupload | fk_username | time                | size     |
+----------+-------------+---------------------+----------+
|        1 | dave42      | 2014-12-12 00:09:31 | 19975316 |
|        2 | dave42      | 2014-12-12 00:09:31 | 15254584 |
|        3 | dave42      | 2014-12-12 00:13:01 | 22986333 |
+----------+-------------+---------------------+----------+
3 rows in set (0.00 sec)
```

This answers my question reasonably well (assuming size is measured in bytes), but I know I can do better as an UPLOAD table with many entries poses a problem—the output from my above query would be too big to count manually. You might reasonably guess that the last idupload shown is the number of rows, but this will probably not be true if one of the rows above it was deleted, because the IDs will not be automatically updated then. MySQL's aggregate function COUNT can be used to solve this issue:

```
mysql> SELECT COUNT(*) FROM upload WHERE size > 10*1024*1024;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

```
 1  SELECT  COUNT(*)
 2  FROM
 3  (
 4          SELECT  user.username
 5          FROM    user
 6          INNER JOIN
 7                  upload
 8          ON      upload.fk_username = user.username
 9          INNER JOIN
10                  hashtag
11          ON      hashtag.fk_idupload = upload.idupload
12          WHERE   UPPER(hashtag.body) = 'YOLO'
13          UNION
14          SELECT  user.username
15          FROM    user
16          INNER JOIN
17                  comment
18          ON      comment.fk_username = user.username
19          WHERE   comment.body LIKE '% ur %'
20  )
```

Listing 3.1: An example showing the use of nested queries, aggregate functions and SQL joins.

One thing which is apparent is that an SQL query will always return a table, even if the result is just a number.

Now imagine that I am an elitist and that I habitually correct and criticise others' use of language so that I do not like users of my site writing the phrase "YOLO"[2] in comments or using "ur" to mean "your". I would like to ask *how many users' accounts should I delete?* What results is a fairly complicated query, however I would argue that this is not because of a poorly designed database as the question itself it complicated, consisting of two main parts: how many users made bad hashtags, and how many users wrote bad comments. Lines 4-12 and then 14-19 answer each of these respective questions in part, finding *which* users performed the bad activities. Conceptually[3], the `UNION` keyword then combines the `usernames` from each of the resulting tables, creating a third one, which importantly does not have any `username` repeated. The rows in this table are finally counted. One issue is that the query in Listing 3.1 will not match instances of "ur" in comments unless they are surrounded by spaces.

[2] This is an acronym meaning, "you only live once", from a song by rapper Drake. It gained infamy after being commonly used to justify extremely poor decisions.

[3] A query may execute differently to the way it is written because of the *query optimiser*, which can internally rewrite statements to improve performance.

3.3.4 Caveats with Multiple Concurrent Users

It is worth mentioning that when a database has multiple concurrent users, the same sorts of issues evoked in Chapter 1 can occur—namely race conditions. This is an issue especially when multiple tables need to be modified to make the database consistent, like when deleting a USER. For this reason practical DBMSs need to be able to perform multiple queries as a single *transaction* which can be *rolled back* (undone) if it cannot be executed completely. While the transaction is being *committed*, any resources involved (tables or rows) need to be *locked* (the idea is the same as mutexes mentioned earlier) to prevent changes. This locking usually happens implicitly, though the user can manually request a lock.

Chapter 4

Operating Systems

4.1 Introduction

Modern Operating Systems (OSes) are extremely complicated things, managing a computer's hardware and software. To talk about an OS in its entirety in just the few pages I have here would be ambitious to say the least. Instead, I have a C program which hints at the underlying complexity of an OS while demonstrating important security concerns and the operation of the stack.

4.2 A Buffer Overflow Attack

Buffer overflow occurs when there are too much data for a *buffer* (a region of memory) to store them, resulting in them *overflowing* into an adjacent part of the memory.

4.2.1 Some Preliminaries

Computers deal with function calls using a *call stack*, which is a last-in-first-out (LIFO) data structure that stores *stack frames*, one containing the data associated with each function call. Stack frames can be added to or removed from only the top of the stack. This property makes it well-suited to coordinating calls, because the function which was last called should be the first to return.

Inside each stack frame are the following, ordered from the higher in the stack to lower:

- Space for local variables.
- The return address, which is the memory location that the program should resume executing code from once the function has completed.
- Arguments passed to the function.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void do_work (char * typed)
5  {
6      char array[20];
7      printf("What's on the stack?\n%p\n%p\n%p\n%p\n%p\n%p\n%
           p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n")
           ;
8      strcpy(array, typed);
9      printf("%s\n", array);
10     printf("Now the stack looks like:\n%p\n%p\n%p\n%p\n%p\n
           %p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\
           n\n");
11 }
12
13 void never_called (void)
14 {
15     printf("how could I be executing?\n");
16 }
17
18 int main (int argc, char * argv[])
19 {
20     printf("Where are my functions?\n");
21     printf("do_work is at %p\n", do_work);
22     printf("never_called is at %p\n", never_called);
23     printf("main is at %p\n", main);
24     if (argc != 2) {
25         printf("usage: name string \n");
26         return -1;
27     }
28     do_work(argv[1]);
29 }
```

Listing 4.1: The C code of a program vulnerable to a buffer overflow attack.

As an example, consider the code in Listing 4.1. When the do_work function is called, the argument (a pointer) will be placed on the stack, then the return address (which roughly corresponds to line 29), followed by the array (line 6). The array is stored with the higher-indexed elements lower in the stack.

4.2.2 The Attack

If a program has a buffer which can overflow, (part of) the running instance of it can be modified. In this case, the return address in do_work's stack frame can be changed to redirect to never_called. This attack is of such concern that most modern (circa post-2005) operating systems and compilers have features built in to prevent them.[1] In any case, upon running the code in Listing 4.1, I received the following output:

```
[139]christian@christian-pc /tmp $ ./a.out 0123456789
Where are my functions?
do_work is at 0x40058d
never_called is at 0x4005d8
main is at 0x4005e8
What's on the stack?
0x7fffffec
0x7ffff7dd77f0
0x13
0x6
0x7ffff7dd46a0
0x7ffff7dd77f0
0x7fffffffe595
0x4007ec
0xffffffff
(nil)
0x7ffff7ffe1e8
0x7fffffffe1c0
0x400667
0x7fffffffe2a8
0x200000000
(nil)
0x7ffff7a50995
(nil)

0123456789
Now the stack looks like:
0x7ffff7ff7000
0x7ffff7dd77f0
0xffffffffffffffff
0x7ffff7dd77f0
0x7ffff7dd46a0
0x7ffff7dd77f0
0x7fffffffe595
0x3736353433323130
0xff003938
(nil)
0x7ffff7ffe1e8
0x7fffffffe1c0
0x400667
0x7fffffffe2a8
0x200000000
(nil)
0x7ffff7a50995
(nil)
```

I noted a return address of `0x400667`, which I needed to change to `0x4005d8`. Now, running

[1] These include Address Space Layout Randomization (ASLR) and canaries, which were briefly mentioned in class. On my machine, the exploit discussed here did not require these features to be disabled.

```
./a.out 0123456789012345678901234567890123456789$(cat a.bin)
```

in the terminal (where `a.bin` is a binary file containing `0xd80540`) modified the address appropriately, however I did expect to require only 20 characters (the number of elements in the array) of padding before the address instead of 40. Moreover, changing the size of `array` by only a small amount did not usually result in me requiring more or less padding. Presumably this had something to do with alignment of data structures in memory and/or the size of the assembler instructions, however I did not have time to investigate.

If the program had been written in a safer, higher-level language like Java, it would (probably) not have been vulnerable to this kind of attack.

Chapter 5

This Week in The News

The ICT industry is new, complex and rapidly changing. As a software engineer, I will be expected to provide special skills and advice in this industry. It is therefore appropriate that I remain up-to-date with associated developments. This chapter is a collection of software engineering-related news that occurred during this semester, along with comments on their socio-technical relevance.

5.1 Primary Students Learn to Program

25 March. Key points [14]:

- Representatives from Sydney University, the Westpac Banking Group and Telstra have held talks with Education Minister Christopher Pyne and Communications Minister Malcolm Turnbull to provide funding ideas on how to train teachers in programming.
- 23,500 teachers would need training to introduce the new curriculum at a cost of $23 million.
- NSW could have the new curriculum taught by 2017 at the earliest.
- Advanced programming will be an elective subject in Years 9 and 10 in Australia.
- Britain is making computer coding compulsory in all primary schools in September.

5.1.1 Comments

It is in the best interests of industry to have a large, highly skilled workforce at their disposal. Robust software and good-quality, professional colleagues are what I would like as a software engineer as well. However, one possible issue with teaching programming at a primary-school level is job market saturation, which could devalue the work of good programmers.

Now, software engineers are not programmers; the latter do not necessarily take a disciplined, methodical approach to creating software. However there is significant enough overlap in the two areas to be concerned about the future of Australia's ICT job market. One of the many roles of professional societies is to ensure fair pay.

5.2 UK Report Finds Cost of Data Breaches Doubled

29 April. Key points [16]:

- Though the number of companies hit by information security breaches was down on the previous 12 months, the PriceWaterhouseCoopers survey found that the severity and impact of attacks has increased.
- The average cost of the worst cyber security breach for big companies was between $1.1 million and $2.2 million, up from the 2013 survey.
- Cyber attacks on companies in Europe and the United States have increased in regularity and severity in recent years as "hacktivists" and criminals become increasingly sophisticated, requiring businesses to strengthen their defences.
- In spite of this, the survey found that 81 percent of big companies suffered a security breach in the past 12 months, down from 86 percent in the 2013 study. It also found most companies had increased their spending on IT security.

5.2.1 Comments

The 80% figure on security breaches is concerning. In banks and infrastructure (e.g. energy and transport) such a breach could cost anywhere from millions to lives. Software engineering then has far-reaching ethical implications.

As well as the responsibilities of industry the story also highlights that society is changing. The fact that activism by hacking ("hacktivism") has a name suggests that many not-necessarily-good people are becoming more technologically literate, and therefore increasing the frequency and severity of security breaches.

It is therefore important for me to follow software engineering best practice principles and encourage my colleagues to do the same.

5.3 Software Bug Disrupts E-vote Count in Belgian Election

26 May. Key points [17]:

- A bug in an e-voting application halted the release of European, federal and regional election results in Belgium, the country's interior ministry said Monday.
- The fault appeared in the system despite the fact that the application was especially developed for these elections, was "tested thousands of times" and was certified by PriceWaterhouseCoopers, he said.
- Kommer Kleijn, spokesman for VoorEVA.be, a Belgian organization that rejects the e-voting system because "it deprives voters from effectively verifying the elections in which they partake" called the problems "a catastrophe." "They claim that the recording of the votes was done flawlessly, but who can verify that? We can't," Kleijn said.
- In Germany, the Federal Constitutional Court banned the use of electronic voting machines in 2009 because results from the machines were not verifiable. The Netherlands banned the practice in 2008 after a group of activists successfully demonstrated that electronic voting machines then in use could be tampered with.

5.3.1 Comments

It is difficult to say if e-vote system could have been improved by better software engineering techniques. If the Secure Embedded L4 (seL4) microkernel[1] can be mathematically proven, why can the voting system not be verified and validated with the same degree of rigour?

[1] `http://ssrg.nicta.com.au/projects/seL4/`

Postface (Self-Evaluation)

Part of being a professional involves evaluating one's conduct so that it can be improved where necessary. Throughout ELEC436, I feel that I have been a decent student, but there are areas I could improve upon. For instance, I have not been very vocal during class discussions. In the workplace I will not be reprimanded if I do this. Rather, I will likely go unnoticed and fail to thrive in my profession.

About this report in particular, I think I have not shown very good time management—Chapter 2 is 18 pages long, but no other chapter is more than 8 pages. A Gantt chart or some kind of agenda would have been useful but limited as, to thoroughly plan the research portions of this report, I would already require a working knowledge of the fields.

Additionally, I have not discussed professional societies or the expectations of the ACS-ES Join Board enough.

Otherwise, my technical knowledge is good. Perhaps, once marks have been deducted for the above shortcomings, my work amounts to a low-to-mid-range distinction.

Bibliography

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, 1999.

[3] J. Reynolds and J. Postel. RFC 1700, Assigned Numbers. `http://www.ietf.org/rfc/rfc1700.txt`, 1999.

[4] T. Socolofsky and C. Kale. RFC 1180, A TCP/IP Tutorial. `http://www.ietf.org/rfc/rfc1180.txt`, 1991.

[5] Information Sciences Institute. RFC 793, Transmission Control Protocol. `http://www.ietf.org/rfc/rfc793.txt`, 1981.

[6] Information Sciences Institute. RFC 791, Internet Protocol. `http://www.ietf.org/rfc/rfc791.txt`, 1981.

[7] K. Nichols, Cisco Systems, S. Blake, Torren Networking Technologies, F. Backer, and EMC Corporation. RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. `http://www.ietf.org/rfc/rfc2474.txt`, 1998.

[8] K. Ramakrishnan, TeraOptic Networks, S. Floyd, ACIRI, and D. EMC Black. RFC 3168, The Addition of Explicit Congestion Notification (ECN) to IP. `http://www.ietf.org/rfc/rfc3168.txt`, 2001.

[9] J. Shoch. Inter-Network Naming, Addressing, and Routing, 1978.

[10] D. Plummer. RFC 826, An Ethernet Address Resolution Protocol. `http://www.ietf.org/rfc/rfc826.txt`, 1982.

[11] Charles W. Bachman. Data structure diagrams. *SIGMIS Database*, 1(2):4–10, July 1969.

[12] Richard Barker. *Case Method: Entity Relationship Modelling.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990.

[13] Michael Johnson, Robert Rosebrugh, and C. N. G. Dampney. View updates in a semantic data modelling paradigm. In *Proceedings of the 12th Australasian Database Conference*, ADC '01, pages 29–36, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Fran Foo. Primary students learn to program. http://www.theaustralian.com.au/technology/primary-students-learn-to-program/story-e6frgakx-1226863685973, 2014. [Online; accessed 25-March-2014].

[15] Sylvia Pennington. Public sector technology jobs take a dive. http://www.smh.com.au/it-pro/government-it/public-sector-technology-jobs-take-a-dive-20140402-zqpee.html, 2014. [Online; accessed 1-April-2014].

[16] Steve Slater. Cost of cyber attacks on british companies rises. http://uk.reuters.com/article/2014/04/28/uk-cybercrime-britain-survey-idUKKBN0DE21120140428, 2014. [Online; accessed 29-April-2014].

[17] Loek Essers. Software bug disrupts e-vote count in belgian election. http://www.pcworld.com/article/2159260/software-bug-disrupts-evote-count-in-belgian-election.html, 2014. [Online; accessed 26-May-2014].