Oliver Black

Software IPv6 Router in Rust

Computer Science Tripos – Part II

Selwyn College

April 24, 2019

Proforma

Name:	**Oliver Black**
College:	**Selwyn College**
Project Title:	**Software IPv6 Router in Rust**
Examination:	**Computer Science Tripos – Part II, July 2001**
Word Count:	**"FIX ME"** [1] **(well less than the 12000 limit)**
Project Originator:	Oliver Black & Richard Watts
Supervisor:	Andrew Moore

Original Aims of the Project

To implement a simple IPv6 router using Rust that behaves as specified in the IPv6 RFCs

To write a demonstration dissertation[2] using LaTeX to save student's time when writing their own dissertations. The dissertation should illustrate how to use the more common LaTeX constructs. It should include pictures and diagrams to show how these can be incorporated into the dissertation. It should contain the entire LaTeX source of the dissertation and the makefile. It should explain how to construct an MSDOS disk of the dissertation in Postscript format that can be used by the book shop for printing, and, finally, it should have the prescribed layout and format of a diploma dissertation.

Work Completed

All that has been completed appears in this dissertation.

Special Difficulties

Learning how to incorporate encapulated postscript into a LaTeX document on both Ubuntu Linux and OS X.

[1] This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

[2] A normal footnote without the complication of being in a table.

Declaration

I, [Name] of [College], being a candidate for Part II of the Computer Science Tripos [or the Diploma in Computer Science], hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

1 Introduction **9**
1.1 Overview of the files . . . 9
1.2 Building the document . . . 9
1.2.1 The makefile . . . 9
1.3 Counting words . . . 10

2 Preparation **11**

3 Implementation **13**
3.1 Verbatim text . . . 13
3.2 Tables . . . 14
3.3 Simple diagrams . . . 14
3.4 Adding more complicated graphics . . . 14

4 Evaluation **17**
4.1 Printing and binding . . . 17
4.2 Further information . . . 17

5 Conclusion **19**

Bibliography **19**

A Latex source **21**
A.1 diss.tex . . . 21
A.2 proposal.tex . . . 21

B Makefile **23**
B.1 makefile . . . 23
B.2 refs.bib . . . 23

C Project Proposal **25**

List of Figures

3.1 A picture composed of boxes and vectors. 14
3.2 A diagram composed of circles, lines and boxes. 15
3.3 Example figure using encapsulated postscript 15
3.4 Example figure where a picture can be pasted in 16
3.5 Example diagram drawn using `xfig` 16

Acknowledgements

This document owes much to an earlier version written by Simon Moore [**?**]. His help, encouragement and advice was greatly appreciated.

Chapter 1

Introduction

1.1 Overview of the files

This document consists of the following files:

- `makefile` — The makefile for the dissertation and Project Proposal
- `diss.tex` — The dissertation
- `proposal.tex` — The project proposal
- `figs` – A directory containing diagrams and pictures
- `refs.bib` — The bibliography database

1.2 Building the document

This document was produced using LaTeX 2$_\varepsilon$which is based upon LaTeX[**?**]. To build the document you first need to generate `diss.aux` which, amongst other things, contains the references used. This if done by executing the command:

```
pdflatex diss
```

Then the bibliography can be generated from `refs.bib` using:

```
bibtex diss
```

Finally, to ensure all the page numbering is correct run `pdflatex` on `diss.tex` until the `.aux` files do not change. This usually takes 2 more runs.

1.2.1 The makefile

To simplify the calls to `pdflatex` and `bibtex`, a makefile has been provided, see Appendix B.1. It provides the following facilities:

`make`
: Display help information.

`make proposal.pdf`
: Format the proposal document as a PDF.

`make view-proposal`
: Run `make proposal.pdf` and then display it with a Linux PDF viewer (preferably "okular", if that is not available fall back to "evince").

`make diss.pdf`
: Format the dissertation document as a PDF.

`make count`
: Display an estimate of the word count.

`make all`
: Construct `proposal.pdf` and `diss.pdf`.

`make pub`
: Make `diss.pdf` and place it in my `public_html` directory.

`make clean`
: Delete all intermediate files except the source files and the resulting PDFs. All these deleted files can be reconstructed by typing `make all`.

1.3 Counting words

An approximate word count of the body of the dissertation may be obtained using:

```
wc diss.tex
```

Alternatively, try something like:

```
detex diss.tex | tr -cd '0-9A-Z a-z\n' | wc -w
```

Chapter 2

Preparation

This chapter is empty!

Chapter 3

Implementation

3.1 Verbatim text

Verbatim text can be included using `\begin{verbatim}` and `\end{verbatim}`. I normally use a slightly smaller font and often squeeze the lines a little closer together, as in:

```
GET "libhdr"

GLOBAL { count:200; all  }

LET try(ld, row, rd) BE TEST row=all
                        THEN count := count + 1
                        ELSE { LET poss = all & ~(ld | row | rd)
                               UNTIL poss=0 DO
                               { LET p = poss & -poss
                                 poss := poss - p
                                 try(ld+p << 1, row+p, rd+p >> 1)
                               }
                             }
LET start() = VALOF
{ all := 1
  FOR i = 1 TO 12 DO
  { count := 0
    try(0, 0, 0)
    writef("Number of solutions to %i2-queens is %i5*n", i, count)
    all := 2*all + 1
  }
  RESULTIS 0
}
```

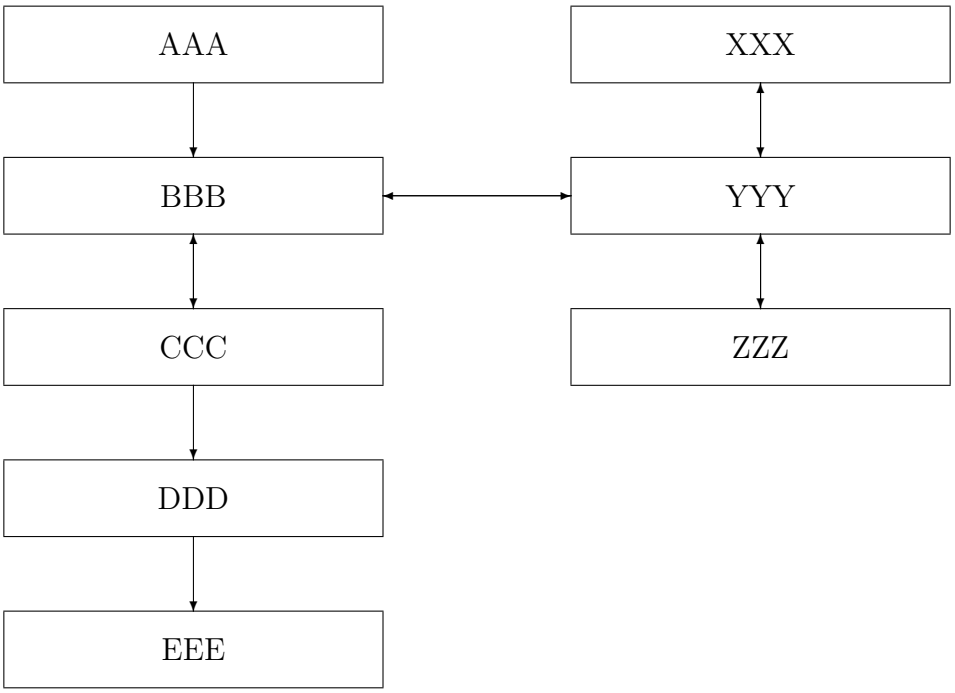

Figure 3.1: A picture composed of boxes and vectors.

3.2 Tables

Here is a simple example[1] of a table.

Left Justified	Centred	Right Justified
First	A	XXX
Second	AA	XX
Last	AAA	X

There is another example table in the proforma.

3.3 Simple diagrams

Simple diagrams can be written directly in LaTeX. For example, see figure 3.1 on page 14 and see figure 3.2 on page 15.

3.4 Adding more complicated graphics

The use of LaTeX format can be tedious and it is often better to use encapsulated postscript (EPS) or PDF to represent complicated graphics. Figure 3.3 and 3.5 on page 16 are

[1] A footnote

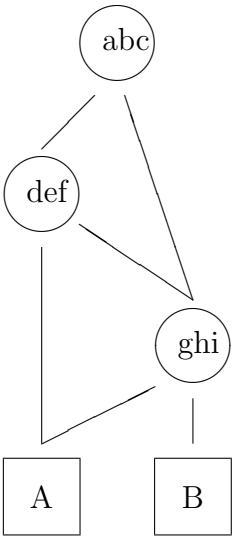

Figure 3.2: A diagram composed of circles, lines and boxes.

examples. The second figure was drawn using `xfig` and exported in `.eps` format. This is my recommended way of drawing all diagrams.

Figure 3.3: Example figure using encapsulated postscript

Figure 3.4: Example figure where a picture can be pasted in

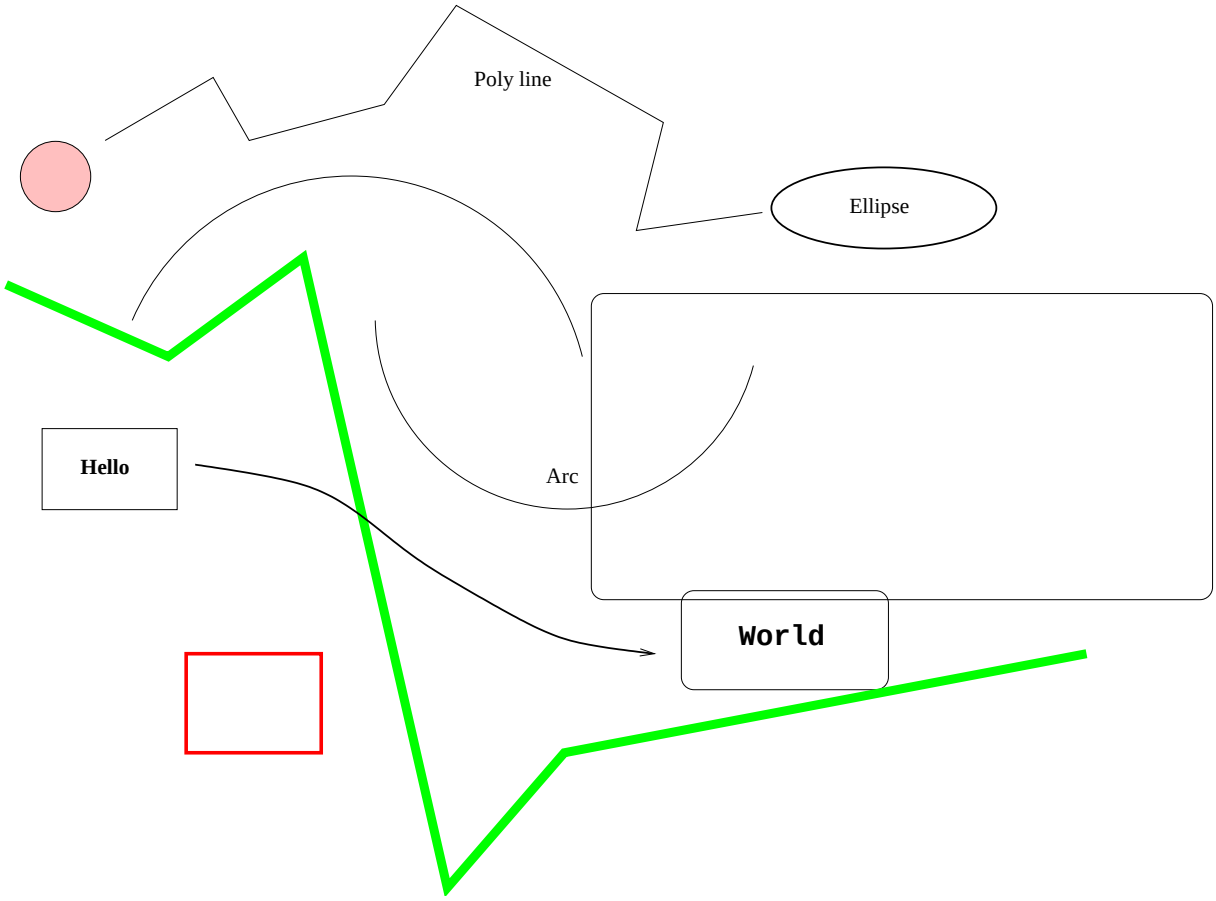

Figure 3.5: Example diagram drawn using `xfig`

Chapter 4

Evaluation

4.1 Printing and binding

Use a "duplex" laser printer that can print on both sides to print two copies of your dissertation. Then bind them, for example using the comb binder in the Computer Laboratory Library.

4.2 Further information

See the Unix Tools notes at

`http://www.cl.cam.ac.uk/teaching/current-1/UnixTools/materials.html`

Chapter 5

Conclusion

I hope that this rough guide to writing a dissertation is LaTeX has been helpful and saved you time.

Appendix A

Latex source

A.1 diss.tex

A.2 proposal.tex

Appendix B

Makefile

B.1 makefile

B.2 refs.bib

Appendix C

Project Proposal

Part II Project Proposal

Software IPv6 Router in Rust

2018-10-11

Oliver Black (olb22) - *Selwyn College*

Overseers - *Jean Bacon / Ross Anderson / Amanda Prorok*
Supervisor - *Andrew Moore*
Director of Studies - *Richard Watts*

Originator - *Oliver Black / Richard Watts*

Introduction & Description of Work

IPv6 is the next generation internet addressing standard, it solves numerous problems with IPv4, such as the shortage of IPv4 addresses. It does more than just increase the number of addresses though, many advanced features not in IPv4 are added with IPv6, full details of the current IPv6 standard can be found in IETF RFCs [1.1]. Mininet [1.2] is a virtual network simulator (supporting IPv6) that was developed at Stanford and until 2016 was used in the 1B Computer Networking Course [1.3] (the year before I attended it).

This project will make use of Mininet to write a software implementation of an IPv6 router in Rust [1.4]. This will begin with gaining an understanding of how Mininet works, then doing research into the IPv6 specification, and finally developing a router and a test bench. Initially that router will implement a minimum amount of IPv6 functionality (as per the IPv6 standards), moving on to more advanced functionality if time permits.

Resource Declaration

No special resources will be required. Work will be undertaken on a personal laptop, with git used for source control, and github for cloud backup. This will enable work to be seamlessly continued on an MCS machine if the laptop is taken out of action.

Starting Point

I took the 2017 1B Computer Networking course, so have a good overview of what a router is meant to do. I have spent around 1 hour fiddling with Mininet, and reading up on the 2016 Computer Networking course, to check what I want to do is feasible. An implementation of an extremely simple router already exists as an optional extension of that course, and I have looked at it briefly [3.1]. I have attended a Rust talk at the CL, and have done my own brief research into the programming language. I have used continuous integration testing methodologies during my summer internship at Solarflare.

Substance & Structure of the Project

The aim of this project is to write a software IPv6 router that complies with the IPv6 RFC standard [4.1]. It'll begin by complying with all the 'must's mentioned in the main IPv6 RFC standard, with other aspects of the standard being extensions, see success criteria for more details.

The first stage of the project will be research and refining requirements, this will be two fold. On a practical level, understanding how the Simple Router [4.2] project works and interfaces with Mininet, also gaining a working knowledge of Rust. On a non-practical level, researching IPv6 and gaining a deeper understanding of the requirements listed in success

criteria. From these requirements a detailed plan will be written, including the router's system architecture.

The next stage of the project will be development and testing. A test bench will be created as the project proceeds, with tests being added for each new requirement that is implemented. The test bench will take the form of scripts that set up a Mininet environment, and then have IPv6 nodes sending packets to each other and the router. All the tests will be run every time a feature is added, the result being a rudimentary form of manual continuous integration.

The development itself will primarily be in Rust [4.3], with bits of C and Python. Rust is a safe modern low level language, and will be an interesting learning experience, the project can always be completed entirely in C if there are insufficient libraries available, or the learning curve is too steep. Interfacing with Mininet at a high level (for the test bench) will be done in Python, as that is the language the API is written in. Exploratory work will be done in C due to preexisting code bases, such as the Simple Router example, being in C.

The final stage will be the verification of the test bench, with additional end-to-end tests being performed. This will be followed by an evaluation to demonstrate the implementation has been successful, and then by the writing of the dissertation.

Success Criteria

To have a software IPv6 router and accompanying test bench running on top of Mininet. The router will comply with all of the core requirements of an IPv6 router, and the test bench will test each of these requirements.

The core requirements are divided into two parts: basic and advanced. The basic requirements are an implementation of a control and data plane in software that can forward packets to the correct unicast addresses, with static address allocation. The advanced requirements are an implementation of ICMPv6 (i.e. proper error handling), and handling anycast and multicast addresses.

Testing is required for all of these, and will meet the success criteria if there is a unit test for each implemented bit of functionality and the relevant requirements listed in the specification, and if the router passes all these tests. This includes tests for edge cases, for example when TTL is 1.

The extension requirements are implementing IPv6 extension headers (both those included in the main IPv6 RFC, and those with their own RFC), implementing SLAAC & DHCPv6 (stateless and stateful), compatibility features with IPv4 addresses, and checking addresses comply with IPv6. Other stretch goals include optimisation, these will be tested through load testing and comparisons with non-optimised versions. Additionally any optional minor features encompassed by the core requirements are extension requirements. A further potential extension would be to pull the software router out of mininet, and get it running on a switch, testing it with real machines.

All of these requirements are based on the relevant RFCs[1.1]. Where applicable, the core requirements only include aspects of the RFC prefaced with ‘must’, whereas the extension requirements include all functionality specified.

Plan of Work

Note: Throughout, any work on extension goals can be replaced with work from a previous 2 week slot, making the plan more flexible and responsive to unexpected changes.

20th October

Start of project - Time since last entry/special events
<> - Work that is completed/stopped on this date
Kick off, begin research - Work that is begun on this date
Milestones: - List of milestones expected by this date

3rd November

2 weeks
Research completed.
Start implementation of core requirements.
Milestones:
List of core and extension requirements mapped out in detail from IPv6 standards and documentation, including system architecture.
Simple Router implementation and interface with Mininet understood.
Basic Rust concepts understood.

17th November

2 weeks
Basic routing framework completed.
Start working on the advanced core requirements.
Milestones:
Router software that can perform basic packet forwarding.
Test bench that can perform basic tests for packet forwarding.

1st December

2 weeks - End of Michaelmas term
Core criteria all met.
End-to-end testing begins, more comprehensive test cases to be added to test bench, small problems fixed as testing continues, big problems documented and listed. Implementation evaluated based on success criteria.
Milestones
Router software that meets all of the core success criteria, with accompanying test bench.

15th December

1 week - 1 week holiday
List of big problems completed.
Big problems fixed in order of severity.
Milestones:
List of remaining identified problems with core functionality.

29th December

2 weeks
All major issues with core functionality resolved.
Begin work on extension goals.
Milestones:
Stable router with comprehensive test bench covering all core functionality.

12th January

2 weeks - Start of Lent Term
Extension work suspended.
Begin evaluation and writing dissertation.
Milestones:
List of implemented extension functionality, with accompanying router software and test benches.

26th January

2 weeks - 1st February: Progress Report Deadline
Evaluation completed.
Continue writing dissertation, write a progress report for presentation.
Milestones:
Evaluation and test report.
(midway) Progress report completed and submitted

9th February

2 weeks
Draft dissertation completed, dissertation work suspended.
Resume work on extension goals.
Milestones:
Completed draft dissertation.

23rd February

2 weeks
Extension work suspended.
Resume work on dissertation - get friends to read.

Milestones:
List of implemented extension functionality, with accompanying router software and test benches.

9th March

2 weeks
<>
Based on feedback received begin work on weaknesses in dissertation.
Milestones:
Improved dissertation based on feedback received.
List of areas of weakness to be worked on.

23rd March

2 weeks - End of Lent Term
Areas of weakness resolved/explained.
Exam revision.
Milestones:
Dissertation submitted to overseers and supervisors for review

6th April

2 weeks
<>
Exam revision.

20th April

2 weeks - Start of Easter Term
<>
Alter dissertation based on comments from from overseers and supervisor
Milestones:
Completed Dissertation

4th May

2 weeks
<>
Exam revision

17th May

2 weeks - 17th May: Dissertation Deadline
<>
Dissertation reread and then submitted.
Milestones:
Submitted dissertation and source code.

Links

1.1: IPv6 https://tools.ietf.org/html/rfc8200
- Addressing: https://tools.ietf.org/html/rfc4291
- ICMPv6: https://tools.ietf.org/html/rfc4443
- DHCPv6: https://tools.ietf.org/html/rfc3315
- SLAAC: https://tools.ietf.org/html/rfc4862
- Authentication Header: https://tools.ietf.org/html/rfc4302
- Encapsulating security payload: https://tools.ietf.org/html/rfc4303

1.2: http://mininet.org/
1.3: https://www.cl.cam.ac.uk/teaching/1617/CompNet/handson/
1.4: https://www.rust-lang.org/en-US/
3.1: https://github.com/mininet/mininet/wiki/Simple-Router
4.1: 1.1
4.2: 3.1
4.3: 1.4