Oliver Black

Software IPv6 Router in Rust

Computer Science Tripos – Part II

Selwyn College

April 28, 2019

Declaration of Originality

I, Oliver Black of Selwyn College, being a candidate for Part II of the Computer Science Tripos, 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 Oliver Black

Date April 28, 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& footnote"** [1]
Project Originator:	Oliver Black & Dr Richard Watts
Supervisor:	Andrew Moore

Original Aims of the Project

The IPv6 standard contains a large number of complex requirements, complicating understanding. I aim to design and implement a simple IPv6 router using Rust[1] that behaves as specified in the IPv6 RFCs[4]. This router should implement the minimum functionality required by the relevant standards, yet still be functional, minimal, & stable. Rust is a new programming language that aims to be as fast as C while maintaining memory safety, I wanted to understand how practical it was to develop in.

Work Completed

Despite having initial difficulties setting up my test environment using Mininet[2], due to its lack of support for IPv6, I successfully implemented a functioning IPv6 router in rust that met almost all of my core requirements. Both the router itself, and the test bench, are available for public use. TODO mention speedup/code coverage/size/RFC coverage/throughput AND fill rest in when main body done. mention how ambitious original claims were

Special Difficulties

None. TODO fill in

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

Contents

List of Figures

Acknowledgements

Many thanks to:

- My supervisor Andrew Moore for his helpful advice.
- My Director of Studies Dr Richard Watts for his guidance.
- Friends & family for proofreading.

Chapter 1

Introduction

Slowly but surely the internet is making progress towards IPv6, and how do pages of Requests for Comments (RFCs) translate into real world network components. The aim of this project was to develop an IPv6 Router in Rust that explores the functionality of IPv6, and how different parts of the various standards fit together. The project has been a success, I have produced a functioning router and accompanying test suite.

Due to the popularity of the Internet, there are now not enough IPv4 addresses to go around. IPv6 is the incoming internet addressing standard that solves numerous issues with IPv4. Primarily it increases the number of addresses, however it also fixes many flaws in the IPv4 design, and standardises common non-standard practices. For example, the Time To Live in IPv4 was defined partly in terms of seconds left to live[6], but in practice was decremented by 1 every hop between nodes. In IPv6 the field is accurately renamed to Hop Limit, and is now defined in terms of hops between nodes (as opposed to seconds). Many subtle decisions like this have gone into the IPv6 standard, with an aim to making an internet that works well, rather than one that just works.

Rust[1] is an up and coming modern low level programming language. It aims to match the performance of C++ without sacrificing memory safety, and avoiding garbage collection. It does this through zero-cost high level abstractions such as *ownership* and *lifetimes*. For example, if you pass a struct to a function, that function then owns that struct, with it being inaccessible after the function returns. It is possible for functions to borrow values instead, using "`&`", similar to passing by reference. I chose Rust for my project as it can be easier to debug than C or C++, but mainly because I was interesting in learning Rust.

Mininet[2] is an open source virtual network simulator that was developed at Stanford and until 2016 was used in the Part 1B Computer Networking course, it is written in Python. It creates lightweight virtual networks by making use of Linux's *networked namespaces*, allowing processes to share a kernel, yet be behind different network interfaces. This made it the ideal candidate to build my router and test suite on top of.

Routers are the backbone of the internet, at the most simple level many of them have a *control plane* that deals with addressing, and a *forwarding plane* that deals with actually

sending packets. There are many open source routers out there, but almost all of them have lots of IPv4 code. This makes it difficult to isolate and understand how the IPv6 part actually works. Starting from scratch allows you to avoid having to deal with IPv4 at all. Using the IPv6 standard as a framework, combined with some knowledge about the internals of routers, it is possible to develop and IPv6 router that is stable, small, simple, & fast.

Chapter 2

Preparation

Before starting the implementation lots of research and design needed to be done. Lots of research was done into IPv6 RFCs and which aspects were required to be implemented, and which were not. The router software was then designed to provide a framework within which these aspects could be implemented. Additionally a test plan needed to be made, to enable effective evaluation of the finished product.

2.1 Research

An analogous project for IPv4 called Simple Router already exists[10] (it used to be a recommended extension task for the Part 1B Computer Networking course). It is implemented in C, but it ran on top of Mininet. The implementation didn't help at all (due to being in C and for IPv4), but it running on Mininet demonstrates that running a router on Mininet is feasible. Additionally the Mininet code that ran the Simple Router helped me in designing my test bench.

Most of my research time however was spent reading RFCs related to IPv6. The main RFC[4] specifies everything you need to know about IPv6 packets. This describes the contents of the main IPv6 header, Figure 2.1, including how the fields (e.g. *Hop Limit*) are modified for packets in transit. Additionally this includes the extension headers that need to be implemented by a router, when the packet in question is not addressed to the router this turns out to be none.

Another important RFC was the Internet Control Messaging Protocol RFC[7]. This protocol accompanies IPv6 proper, and must be implemented by all IPv6 routers. It allows, among other things, errors about dropped packets to be sent back to the source, and *Echo Request/Reply* (ping) messages to be sent. Alongside this it was important to gain an understanding of how IPv6 related to the link layer below and the transport layer above.

The IPv6 Addressing RFC[5] contains all you could want to know about the various kinds of IPv6 addresses. However, it is mainly just a list of address ranges and whether they have any special behaviour, it doesn't affect the design of a router much. I also

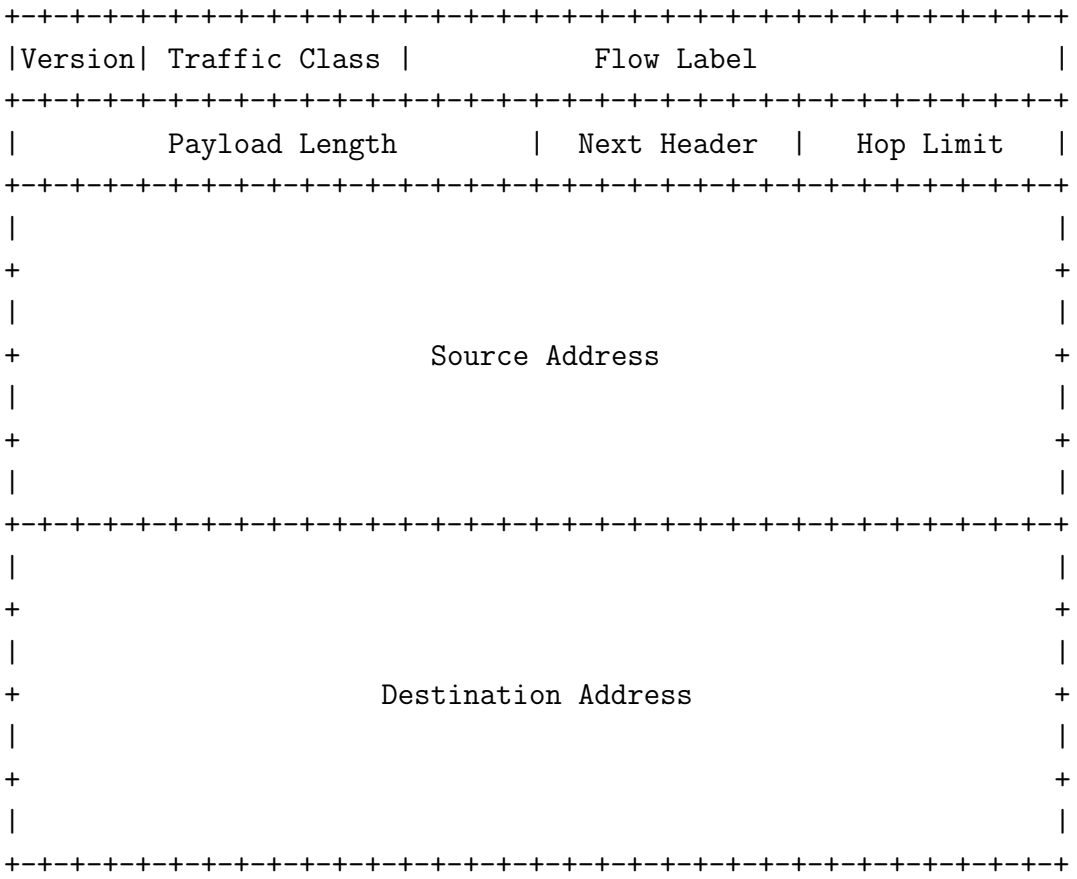

Figure 2.1: IPv6 Header Format[4]

consulted a few more RFCs related to features that didn't *need* to be implemented by an IPv6 router, more details on those can be found under Appendix A.

Finally I spent some time researching rust, as it was a new language to me, and I didn't want to make mistakes early on in my implementation that would make things much harder later on. I discovered a library called pnet[3] which implemented low level networking functions and packet abstractions, exactly what I would need.

2.2 Analysis

After finishing my research I needed to do some *requirements analysis* to workout what exactly my router needed to implement, and in what order I should go about implementing them. I divided up the requirements as recommended into *core* and *extension*, where core contained everything an IPv6 router *needed* to do (according to the RFCs), and extension things I thought I would like it to do as well. Core was further divided up into *basic* and *advanced*, with basic being everything the router required to provide some form of basic testable functionality, and extension being everything else that was *needed*.

There are three main areas of requirements:

- Addressing
- Packet inspection & forwarding
- Error reporting & ICMPv6[7]

The requirements for addressing can be divided into two parts, the address discovery mechanism (static, SLAAC[8] or DHCPv6[9]) and different address types (Unicast, Anycast, Multicast).

Although DHCPv6 and SLAAC are both practical and interesting, they aren't *needed* for an RFC router - static addressing is sufficient - so they were put as extension requirements. Static addressing means the relationship between IPv6 addresses and link layer interfaces is defined when the router starts based on a fixed configuration. In order to get the router forwarding packets as quickly as possible an additional requirement of *flooding* addressing was added. This is not defined in the RFC for IPv6, as it means the router functions instead as a link layer switch, sending all incoming packets out on all interfaces. Static addressing was *needed* by the RFCs and I decided flooding alone wouldn't really constitute basic testable functionality (of a router). So static addressing is a basic core requirement, with DHCPv6 and SLAAC being extension requirements.

Address types in IPv6 are well defined by the addressing RFC[5], hand a router *needs* to deal with all of them. However, in order to test basic functionality only Unicast really needs to be implemented, as Anycast and Multicast are just mappings from a 'Unicast' address to many Unicast addresses. So Unicast is a basic core requirement, with Anycast and Multicast being advanced core requirements. IPv6 also includes scope for local only

addresses, as well as a variety of other specific types, these could also have been extension requirements.

Every packet the router receives needs to be sent to the right destination, and any packet for which the destination is unknown should be sent to the default route. This falls under static addressing as my requirements don't differentiate between the control and data plane - see Design. However the payload length must be checked to see if it matches the actual length of the payload - and the packet discarded if not. Additionally the hop limit must be checked, if it is 1 or 0 the packet should be discarded, otherwise it should be decreased by 1. Both of these are basic core requirements, they are *needed* and without them it is hard to test basic functionality. Without hop-limit decrements the router leaves packets unaffected, so it is hard to tell if they went through the router at all.

IPv6 also has many extension headers, but they *need* to be ignored by intermediate nodes (for example, fragmentation can only be done by the source and destination nodes), except for the *hop-by-hop options header* which can be ignored by intermediate nodes. Apart from ICMPv6 packets my router does not deal with packets that are encapsulated by IPv6 packets (including transport layer protocols). This means that it does not need to process any headers at all, except for ICMPv6 packets.

ICMPv6 works alongside IPv6 to send informational and error messages between nodes. These messages include destination, packet size, hop limit, & header error messages, and echo request & reply informational messages. One slightly odd requirement relates back to hop-limit in the IPv6 header, if an arriving packet has a hop limit of 1, it should be discarded, unless the destination node is the router. This only occurs for ICMPv6 messages.

ICMPv6 is /textitneeded for any IPv6 router. However, in order to test the basic functionality of my router it was not required. This is because error reporting functions can be, in part, replaced by log output from the router itself. As such, ICMPv6 is an advanced core requirement.

For a formal list of the requirements I came up with (along with identifiers and associated tests) see Appendix A.

2.3 Design

Having completed my analysis and produced a structured list of requirements the next step was to come up with a design of router that would enable me to implement these requirements. There were two main ideas in the design of the router itself (the design of the test bench is discussed in the next section).

The first was the separation of the control and forwarding plane, which by itself is nothing special, but the design in software was slight more complex. As always, the control plane deals with the addressing (along with other aspects and the forwarding plane with link layer interfaces and individual packets. The two communicate through forwarding

tables (in the case of non static addressing communication in the other direction is also required). The forwarding plane in my case is made up of a pair of threads for each interface, one receiving and one transmitting. All of the receiving threads have read access to two objects, one is the routing table produced by the control plane, the other is a map of hardware addresses to channels. The channel each hardware address is linked to leads to the transmitting thread for the interface of that hardware address. See Implementation for more details.

The second was designing the layer separation inherent in the TCP/IP stack into the router. This was achieved by having a different function handle each layer, and only passing necessary information between the functions. There are three layers, Ethernet, IPv6, and ICMPv6 in the router, so there are three functions, with only the packets themselves and the relevant addresses being passed between the functions. The Ethernet function sends the IPv6 packet to the IPv6 function, which returns the new IPv6 packet, and the hardware address to send it to. The IPv6 function sends the ICMPv6 packet along with the source and destination address to the ICMPv6 function, which returns the new ICMPv6 packet along with the new source and destination addresses (See Figure 2.2).

Both of these design choices made the implementation much easier, separating the code into functionally separate sections, reducing the risk of introducing bugs, and most importantly providing a structured framework within which to code.

2.4 Test Plan

With requirements and a design of the router completed, I needed to come up with a plan of how I was going to verify that my router was functioning as expected. This was divided into two parts, the design of the test bench, and secondly the list of tests to be run on it.

Test bench

Test critera
Mininet - issues in implementation
Test bench packet choices
Basic requirements checked with log output
Examples Mininet: Ipv4 - see below Simple Router: https://github.com/mininet/mininet/wiki/Simple-Router
Design RFCs Router knowledge - diagram Layer separation - diagram DHCPv6, SLAAC, and Static diagram?
Test bench Test criteria (appendix) - spreadsheet More detail in implementation on mininet

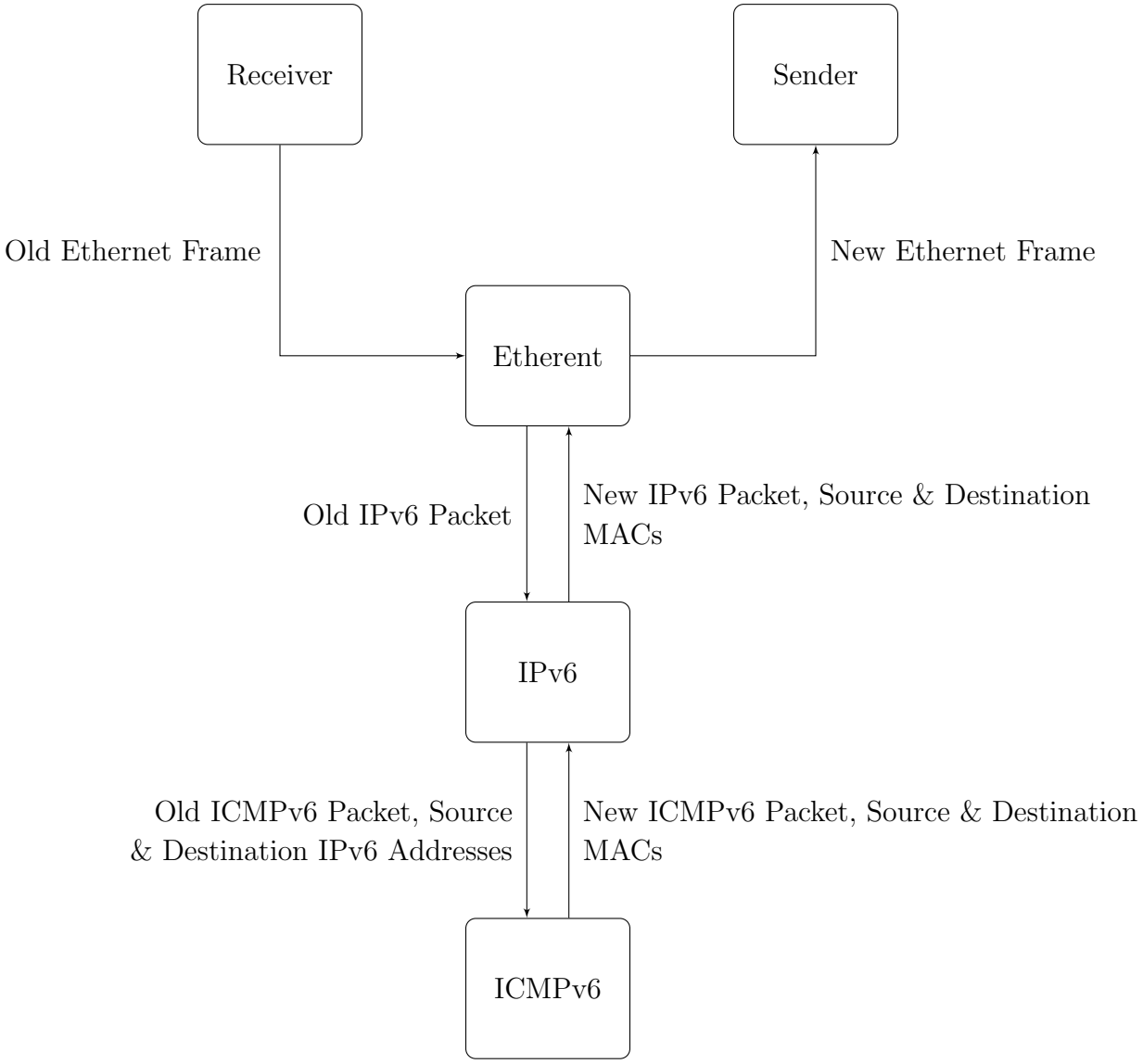

Figure 2.2: Layer Separation

Chapter 3

Implementation

3.1 Router

explain design into practice
    static addressing

3.2 Test Bench

Mininet
    Mention flooding success then test plan stuff
    Maybe some stuff from evaluation here

Chapter 4

Evaluation

Extension headers and ICMPv6 ICMPv6 erroneous header, fragment reconstruction, destination unreachable Local and subnet addresses - maybe explain more earlier

Chapter 5

Conclusion

Bibliography

[1] Rust, a modern low level programming language, `https://www.rust-lang.org/`

[2] Mininet, a network virtualisation library in Python, `http://mininet.org/`

[3] pnet, a low-level networking API for rust, `https://docs.rs/pnet`

[4] Internet Protocol, Version 6 (IPv6) Specification, https://tools.ietf.org/html/rfc8200]RFC 8200, July 2017

[5] IP Version 6 Addressing Architecture, https://tools.ietf.org/html/rfc4291]RFC 4291, February 2006

[6] INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, https://tools.ietf.org/html/rfc791]RFC 791, September 1981

[7] Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, https://tools.ietf.org/html/rfc4443]RFC 4443, March 2006

[8] IPv6 Stateless Address Autoconfiguration, https://tools.ietf.org/html/rfc4862]RFC 4862, September 2007

[9] Dynamic Host Configuration Protocol for IPv6 (DHCPv6), https://tools.ietf.org/html/rfc3315]RFC 3315, July 2003

[10] Simple Router, implementing an IPv4 router in C on Mininet, `https://github.com/mininet/mininet/wiki/Simple-Router`

Appendix A

Requirements

Requirements

By Oliver Black
The requirements below have been taken from the RFC, each requirement includes a description an example test, and in some cases a list of edge cases tests, tests and descriptions for extensions have not been given if no work has been done on their implementation.

Core - 1

Basic - 1

Control Plane - 1

Flooding - 1111

Every packet received should be forwarded to every interface but the originating one. Overridden by static.
Test: Given a packet, check every other interface receives it, and the sending interface does not
Edge cases: No other interfaces

Static - 1112

Read a configuration file with known addresses & interfaces, forward as per the file, else send to default route (also defined in the file)
Test: Given packets from every known address to every known address sent correctly, with no additional sends
Edge cases: Address not known, address very similar (last/first bit different), loopback address, multicast address, unspecified address, loopback address, link local addresses

Data Plane - 2

Header - 1

Payload Length - 11211

https://tools.ietf.org/html/rfc8200#section-3
Must correctly forward the correct amount of data based on the length in this field
Test: Given a packet containing X in this field must forward X bytes
Edge cases: Packet length 0, Packet length = packet size, packet length > packet size, packet length < packet size

Hop Limit - 11212

https://tools.ietf.org/html/rfc8200#section-3

Must correctly discard all packets to be forwarded with a hop limit of 0, and decrement the hop limit of all packets
Test: Given a packet with a hop limit of X, hop limit should be X-1 when forwarded, or not forwarded if X-1 = 0
Edge cases: Hop limit = 0, Hop limit < 0, Hop limit = 0 but router is recipient, Hop limit < 0 but router is recipient.

Addressing (Unicast) - 3

Internal Structure - 1131

Assume addresses have no internal structure - simplest
Test: Random destination addresses that are not in the immediate network are always treated the same

Advanced - 2

ICMPv6 - 1

https://tools.ietf.org/html/rfc4443

Checksum 1211

ICMPv6 packets with incorrect checksums should be dropped (only if router is destination)
Test: Random bit changes to packets so that checksum is incorrect

Unknown 1212

ICMPv6 packets of unknown type must be silently discarded (if router is destination)
Test: Packets with a variety of unknown types that are otherwise valid

Rate limit 1213

Router must apply some form of rate limiting
Test: Try and trigger more than limit of responses

Packet too big 1214

Sent to sender when a packet cannot be forwarded due to size
Test: Send a packet that is too big to be forwarded

Time exceeded 1215

Sent to a sender when a packet's hop limit is decremented to 0
Test: Send a packet with hop_limit of 0 or 1

Echo 1216

Must reply to echo request messages
Test: Send an echo request message

Parameter Problem 1217

Sent to a sender when there is an issue with an ipv6_header
Test: Send packets with an erroneous header, unrecognised next header type, and unrecognised ipv6 option

Uniquely Identify 1218

Do not send ICMPv6 responses if a packets source address is the unspecified address, a multicast address or an anycast address.
Test: Send response triggering messages with the above source addresses

Multicast - 2

https://tools.ietf.org/html/rfc4291#section-2.7
Solicited node address

(e.6) A packet whose source address does not uniquely identify a
single node -- e.g., the IPv6 Unspecified Address, an IPv6
multicast address, or an address known by the ICMP message
originator to be an IPv6 anycast address.

Anycast - 3

https://tools.ietf.org/html/rfc4291#section-2.6

Extension - 3

Optional Requirements from Core - 1

Traffic Class

https://tools.ietf.org/html/rfc8200#section-7

Flow Label

https://tools.ietf.org/html/rfc8200#section-6

Next Header

Header field, may be used to optimise

Text Representation of Addresses and Prefixes

https://tools.ietf.org/html/rfc4291#section-2.2
For logging

More Unicast

https://tools.ietf.org/html/rfc4291#section-2.5

Extension Headers - 2

https://tools.ietf.org/html/rfc8200#section-4

- Hop-by-Hop Options - https://tools.ietf.org/html/rfc8200#appendix-A
- Fragment
- Destination Options
- Routing - https://tools.ietf.org/html/rfc8200#section-8.4
- Authentication
- Encapsulating Security Payload

And order checks
Check IPv6 parameters for full list

DHCPv6 - 3

https://tools.ietf.org/html/rfc3315

SLAAC - 4

https://tools.ietf.org/html/rfc4862

TCP & UDP - 5

https://tools.ietf.org/html/rfc8200#section-8 but not 8.4

Optimisation - 6

IPSec - 7

Security - https://tools.ietf.org/html/rfc4301
Scanning - https://tools.ietf.org/html/rfc4301
Privacy - https://tools.ietf.org/html/rfc7721

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