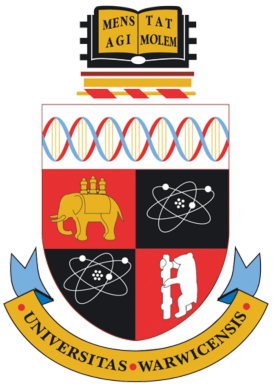


Modelling Smart Field-Programmable Gate Array Switches in the Network

CS351 Computer Systems Engineering Project

Final Report

Benji Levine
1611586

Superviser: Dr Suhaib Fahmy

Department of Computer Science
University of Warwick

2018-19

Abstract

This project investigates the design of network switches based on FPGAs. The FPGAs will perform computation on the data passing through the switches, either processing the data completely and returning it, or processing the data partially and sending it further along its original path. A system of this nature would reduce the latency of any packets which are processed to completion and returned, since the distance they will have travelled will be significantly reduced compared to having to be sent all the way to a cloud data centre.

To aid with this investigation, this project implements a system to model these switches in a virtual network. Among other customisation options, this system allows users to customise the topology used by the model, which aims to help users determine whether FPGA-based smart network switches would be suitable for their specific network.

Keywords

- Network Switch
- FPGA
- Networking
- Mininet
- Open-source
- Infrastructure

Acknowledgements

The success of this project would not have been possible without the contributions of a number of individuals. Firstly, I would like to thank my project supervisor and personal tutor, Dr Suhaib Fahmy, who has not only helped to conceive this project and to guide it through all stages of its development, but has also provided an enormous amount of time and guidance over the past three years while I have been at university. Both with respect to this project and my time at university in general, I have been exceptionally fortunate to receive the level of support I have from him, and it is greatly appreciated. I would also like to thank my personal tutor in the Department of Computer Science, Dr Matthew Leeke, for his support and constant open door policy, as well as for the time he has taken to review this report and offer advice. Finally, I would like to thank Dr Sascha Ott for taking the time to view my presentation and provide productive feedback on it.

Contents

List of Figures

List of Tables

Glossary

The following acronyms are used in the report.

- **API**: Application Programming Interface
- **ARPANET**: Advanced Research Projects Agency Network
- **AWS**: Amazon Web Services
- **ASIC**: Application-Specific Integrated Circuit
- **BCS**: British Computer Society
- **CI**: Continuous Integration
- **CPU**: Central Processing Unit
- **DARPA**: Defense Advanced Research Projects Agency (USA)
- **DSP**: Digital Signal Processing
- **EC2**: Elastic Compute Cloud
- **FPGA**: Field-Programmable Gate Array
- **GUI**: Graphical User Interface
- **HDL**: Hardware Description Language
- **ICMP**: Internet Control Message Protocol
- **IDE**: Integrated Development Environment
- **IO**: Input Output
- **IoT**: Internet of Things
- **IP**: Internet Protocol

- **LTS**: Long-Term Support
- **LUT**: LookUp Table
- **MRI**: Magnetic Resonance Imaging
- **MTU**: Maximum Transmission Unit
- **NIC**: Network Interface Card
- **OS**: Operating System
- **OSI model**: Open Systems Interconnection reference model
- **PCIe**: Peripheral Component Interconnect Express (or PCI Express)
- **SDN**: Software-defined Networking
- **SFP**: Small Form-factor Pluggable
- **TCP**: Transmission Control Protocol
- **UTP**: Unshielded Twisted Pair

1. Introduction

1.1. Background Knowledge

Prior to the commencement of this project, some existing knowledge was used to assist in choosing the direction for the project, and finding different research areas. These are detailed below.

1.1.1. Networking Concepts

The Warwick Computer Science module *Operating Systems and Computer Networks* [5] provided a background for many basic networking concepts, including the *OSI network model* and the *TCP/IP network model*, both of which are further discussed in section 2.1.2. This module aimed to provide an understanding of the purpose of each layer of the *OSI network model* (shown in table 2.1), and where this information has been of relevance to this project, it has been used as a basis for the research shown in section 2.

Working with networking hardware, including network switches, as part of *Warwick Student Cinema* [6] provided practical experience in using this equipment. This proved useful throughout the project, as it allowed for an additional viewpoint to be taken towards the project, i.e. that of someone who manages network infrastructure. Ensuring the hardware side of the project integrates sufficiently with existing infrastructure is important, since the marketability of the project will be limited if it fails to do so.

This experience, enhanced by a recent visit to a datacentre, provided a basic understanding of the different physical media used in network cables. The primary mediums are UTP made of copper, and fibre optics, often referred to simply as "fibre". This has been researched further and is discussed in more detail in section 2.1.4.

1.1.2. FPGAs

The Warwick Engineering modules *Digital Systems Design* [1] and *High Performance Embedded Systems Design* [7] provided the core concepts of FPGAs. This included the FPGA's basic structure of LUTs, block memories, DSP blocks, and IO, how FPGAs compared to ASICs and CPUs in terms of speed, flexibility, and price, different applications of FPGAs, and more. This background was crucial for the development of the project, and motivated many of the initial targets and aims it.

1.2. Project Motivation

1.2.1. Current State

Cloud computing enables computing resources to be available to users remotely, so that users do not need to actively manage the hardware. This is often motivated by economies of scale, in that the greater the quantity of computing resources being managed, the cheaper each unit becomes. Cloud computing also makes it easier for a user to scale the resources they are using, since this is simply done by increasing or decreasing a regular payment, instead of buying or selling hardware. This can be particularly attractive to small businesses or individuals.

Cloud computing has been slowly but steadily rising in popularity since the release of the first cloud computing platform: Amazon's Elastic Compute Cloud (EC2) [8]. This was released in 2006 [9], has since grown significantly, and inspired many competitors, such as Microsoft Azure [10] and Google Cloud Platform [11].

Since cloud computing intends to serve the computing requirements of large numbers of users, there are naturally hardware requirements that need to be filled by the cloud computing provider. This is normally done in the form of data centres, which are large buildings filled with servers and the appropriate infrastructure to make those servers accessible remotely. For an example of the scale that a large cloud computing provider requires, AWS has 64 "availability zones" around the world where "each AZ can be multiple data centers (typically 3), and at full scale can be hundreds of thousands of servers." [12].

When a cloud computing platform hosts a customer's data, that data is stored on one of the servers owned by the platform in one of their data centres. A good

platform will replicate the data in case of hardware failure, but generally the data would only be served from a single data centre, unless the customer pays for the data to be hosted in more locations. Whenever data is requested over a network, there is a delay involved in retrieving that data, much of which arises from the geographical distance that the data has to travel. A user located in Australia will have a longer delay after requesting data hosted in an American data centre, than someone requesting the same data who lives in America. This issue, partially caused by the rise in cloud computing and, as a result, data centres, is a key motivation for this project.

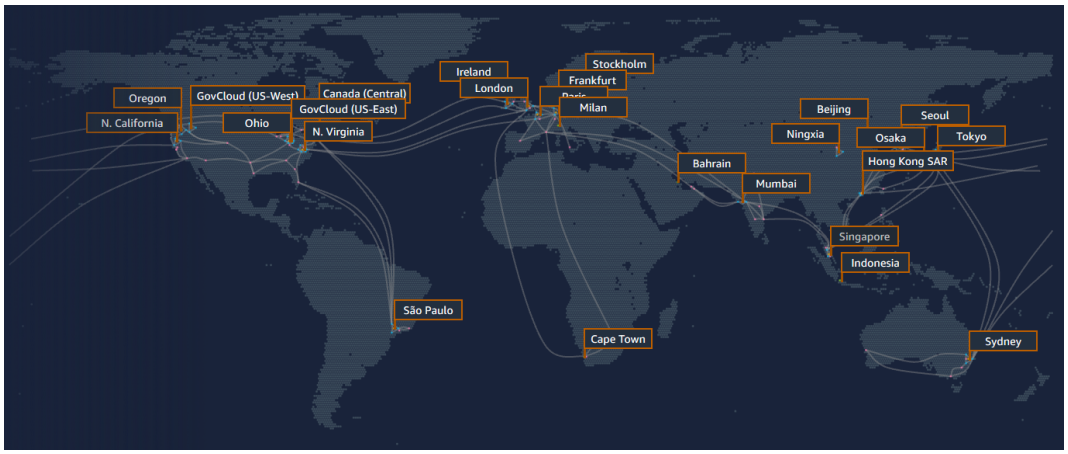


Figure 1.1.: Regions of AWS

1.2.2. Existing Solutions

Some methods have been designed to reduce the time taken for data to travel between a user and a cloud server. One of these methods is ‘edge’ or ‘gateway’ nodes. Edge nodes are comparable to data centres, with the key difference being that they are significantly smaller. They exist to hold content closer to users, so that not all content needs to be retrieved all the way from the data centre. Edge nodes are often used to host tools intended to manage the data centre, such as tools for customers to manage the data they have stored on a cloud computing platform.

While edge nodes do improve the service, they suffer from many of the same drawbacks as data centres. In addition, the more edge nodes a cloud computing platform creates, the more difficult it is for the provider to manage their hardware, as it becomes spread out over a larger geographical area.

Before cloud computing became widely popular, it was necessary for users to host their own files. For those with large data requirements, it was not uncommon to use mainframe computers. These are large, high performance server computers, generally built and sold as a unit. Those using a mainframe would process all of their data on site. This would allow for data to be processed with an extremely low latency.

However, mainframe computers became unpopular for all the reasons cloud computing grew in popularity. Mainframe computers require significant on-site management. They require a high up-front cost, constant running costs in terms of power and cooling, are physically large and noisy, and can be difficult to scale.

1.2.3. Project Solution

The approach this project has taken in solving the above problem has been comprised of two parts:

1. By not altering the path of the data, any data which would need to travel the full path would not be slowed.

2. Choosing an appropriate device to perform the computation.

Maintaining Data Path

This approach led to the idea of FPGA-based smart network switches. These could replace standard network switches, and would be able to perform small amounts of computation on data as it passed through these switches. If data was able to be fully processed by the switch, it would then be returned to the device, without ever needing to be sent to the data centre. If the switch was unable to complete the computation within a set amount of time, it would release the data in its partially processed state along its original path, where it would then meet either further FPGA-based smart network switches, or its destination server. If it met further FPGA-based smart network switches prior to reaching its destination server, these could perform additional processing on the data, leading to a higher chance of completing the computation for a set of data.

Choosing a Device for Computation

The devices considered for use as computation devices in these smart switches were FPGAs, ASICs, and CPUS. Each of these have different advantages and disadvan-

tages over one another. Three of the main areas of comparison were speed, flexibility, and cost.

The fastest of these three devices are ASICs. ASICs compute the algorithm they are designed for in hardware, and they are capable of doing so very efficiently. Since FPGAs are designed to be more flexible than ASICs, ASICs can achieve greater speeds than FPGAs for the same algorithm. CPUs are significantly slower than both ASICs and FPGAs, since they perform computation in software, rather than in hardware. In addition, unlike ASICs and FPGAs, CPUs are not standalone devices. They require additional hardware in order to function, including RAM, a motherboard, possibly storage, and often additional cooling. This would mean that any data the CPU would need to process would have to flow through an Ethernet interface which, for a high bandwidth Ethernet connection may need to be an NIC connected through a PCIe interface, then through the motherboard, then the RAM, then the CPU cache, and finally through the CPU registers, before it could be processed. While there will also be more than one stage to process data using an FPGA or ASIC, this is far more complex with a CPU.

While ASICs are the best for speed, they are the worst for flexibility. ASICs are by definition application-specific, meaning that they are designed to perform one algorithm, and that is the only algorithm which they are capable of performing. In order to achieve the flexibility required for this project, different ASICs would need to be designed for each algorithm required by a user, and the smart switches would then have to be physically swapped out to change the algorithm. This would also be inordinately expensive, and is unlikely to be manufacturable. FPGAs are significantly more flexible than ASICs. They can be programmed using either HDLs such as *Verilog* or *VHDL*, or some high-level languages such as *C* or *C++* using sophisticated synthesis tools. In addition, FPGAs support advanced techniques such as 'Partial Reconfiguration', which allow blocks of an FPGA to be declared as 'reconfigurable', and these blocks can then be swapped out during runtime, without stopping the execution of the remainder of the program running on the FPGA. CPUs are the most flexible of the devices, again due to their ability to run programs in software. This allows them to run programs using most modern programming languages, including 2019's "most loved languages", *Rust* and *Python* [13]. However, as mentioned above, CPUs are not standalone devices. Their requirement for additional hardware restricts their flexibility, since it increases their physical size.

Cost is the most difficult means of comparing devices, since there are many different parameters to take into account. The cheapest devices available are consumer

CPUs, which can cost less than £100 for a low end desktop CPU [14] [15], or over £2000 for a high end server CPU [16] [17]. However, these are consumer prices, and assume the purchase of only one product. It is more difficult to find accurate prices for FPGAs and ASICs, but a comparison of the two can be seen in figure 1.2. ASICs have a much higher initial cost, but a lower cost per unit compared to FPGAs. However, FPGAs are a very active research area, and their cost per unit is slowly decreasing, causing the break-even point shown in figure 1.2 to slowly shift to the right, making FPGAs more cost-effective than ASICs for a larger number of units.

FPGAs were chosen as the optimal computation device for this project. ASICs are insufficiently flexible, and even with their advantages in speed, they are not suitable for this task. CPUs, while more flexible than FPGAs, are significantly slower, and would be impractical to combine with a network switch. FPGAs are fast, flexible, and sufficiently small devices, and are the most suitable tool available.

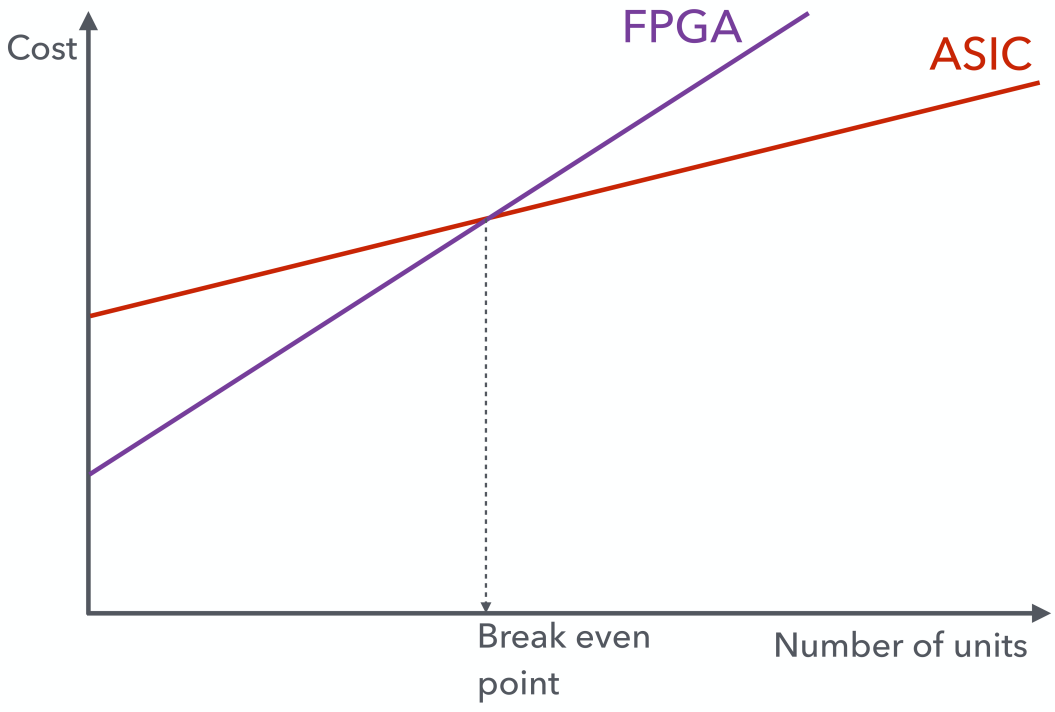


Figure 1.2.: Comparison of FPGA and ASIC cost [1]

1.3. Project Aims

The main aim of this project is to research the complexities of creating an FPGA-based smart network switch. This is a notable change from the initial aims of the project, which were more focused on implementation rather than research. However, due to the agile nature of the project, discussed further in section 8, it was possible to make this change. The initial implementation-focused aims of the project were found to limit the project from exploring certain areas of research which proved both relevant and interesting. In addition, re-evaluating the feasibility of the initial project aims after meeting certain external delays, even while using the risk mitigation strategies laid out in section 8.4, indicated that that these updated project aims would be less susceptible to similar delays.

1.3.1. Research FPGA-Based Smart Network Switches

This is the primary aim of the project. These devices are currently entirely theoretical, and as a result more research than is feasible in this project will be required before these devices are close to being commercially available. As a result, the research into these devices, along with the complications of their design and implementation is being prioritised over the actual implementation of one such device. This will allow for the project to investigate the theory of FPGA-based smart network switches in more depth.

1.3.2. Model FPGA-Based Smart Network Switches

A further aim of the project is to design and implement a model of the FPGA-based smart network switches in order to demonstrate their potential advantages over a standard connection to a server in a data centre. This should allow users to test whether FPGA-based smart network switches would be suitable in their network, and to design their network architecture accordingly if they are.

1.4. Project Stakeholders

1.4.1. Primary Stakeholders

This project had two primary stakeholders who contributed to the development of the project. These are the developer, Benji Levine, and the project supervisor, Dr

Suhaib Fahmy. Through regular meetings and collaboration, the primary stakeholders were able to produce the fundamental ideas behind the project, and develop them across the project's duration. As a result, both have a clear stake in the project.

1.4.2. Secondary Stakeholders

The secondary stakeholders are considered to be those who could benefit from the success of the project. Since this project is primarily research based, many of the secondary stakeholders would only benefit from further development of this project.

Device Manufacturers

Should FPGA-based smart network switches become commercially available, manufacturers of FPGAs such as *Xilinx* [18] and *Intel* [19], as well as manufacturers of network switches such as *Cisco* [20] and *Netgear* [21], would all benefit, both financially and in terms of additional publicity.

Users

These devices are aimed at many different types of user. However, they could perform particularly well for either users relying on cloud computing platforms for data processing, or IoT. IoT tends to involve small payloads of data, but could potentially involve large quantities of these payloads. FPGA-based smart network switches are very well suited for data formatted in this way. For example, this could be applied to tracking medical equipment in a hospital. All equipment, from small items such as scalpels, to large items such as MRI machines, could be fitted with a small, low-power IoT tracking device, and then tracking devices could be fitted throughout the existing hospital infrastructure, such as in the lighting equipment. All items fitted with a tracking device could then be located instantly, and by adding FPGA-based smart network switches into this architecture, this processing can happen much faster, and within the existing infrastructure. With the exception of the FPGA-based smart network switches, this equipment is all available, such as *Electric Imp's Asset Tracking through Ambient Lighting* [22] [23]. As discussed in *A Software Defined Networking architecture for the Internet-of-Things* [24], this can be well integrated with SDN, in order to create efficient, low-power IoT networks.

This use case is applicable to many different sectors, such as education, military, healthcare, and corporate environments, and could also be expanded out to private

individuals in future as hardware costs reduce.

Academia

FPGAs and networking are both very active areas of research, and it is hoped that this project will contribute positively. This project involved both the *Mininet* project [25] and the *NetFPGA* project [26]. Both of these projects are under active development by academic communities, and would be heavily involved in further implementation of FPGA-based smart network switches.

2. Research

This section discusses the research performed prior to the development of the project. This research was necessary to further understand the key areas involved in the project, as well as some of the required technologies.

2.1. Networking Concepts

2.1.1. Network Packets

A 'packet' in networking refers to a unit of data. This term only applies to certain units of data, since data at different stages in network models (see section 2.1.2) takes different names. A packet consists of two parts: a set of 'headers', and a 'payload'. The headers contain metadata about the packet, such as where it was sent from, where it was sent to, or a checksum of the packet used for error checking. A packet tends to have multiple headers, each stacked on top of one another. Each header is used by a different layer of the OSI model (see section 2.1.2). When a packet is being constructed, each layer prepends its header to the packet payload before sending it down to the next layer. The payload carries the actual packet data. In most cases, many packets will be required to send a message, since packets have a fixed maximum size. This limit in size is known as the Maximum Transmission Unit (MTU), and for an Ethernet packet, this is 1518 bytes including headers.

2.1.2. Network Models

The two most commonly used models are the Open Systems Interconnection Reference model (commonly referred to as the OSI model), and the TCP/IP model. Both of these models consist of layers and are intended to allow the detail of the layers beneath the layer currently being used to be abstracted away. This means that, for example, a web server will not need to know any detail about the networking hardware of the server it is running on.

The OSI model was created in 1994 [27] and consists of seven layers, shown in table 2.1. It has suffered from a few key criticisms, such as that some layers have very little use in most applications, and because the complexity of the model can result in large and slow systems. In addition, when the OSI standard was approved, the competing TCP/IP model was being used by most research universities, and so did not receive as much support as might have been expected

The TCP/IP model was developed for ARPANET, a predecessor of the modern Internet funded by DARPA. TCP was developed to serve as a protocol for communication within ARPANET from different underlying hardware (such as between satellite packet networks and ground-based radio packet networks). It initially covered both the Internet and Transport layers of what would become the TCP/IP model. However it was suggested that the protocol be separated, which resulted in the creation of the Internet Protocol (IP). These protocols went through a number of revisions before arriving at TCP/IP v4, which is still in use today.

2.1.3. Packet Switching

Layers of a model can offer either a connection-oriented or a connectionless service to the layers above. A connection-oriented service involves establishing a connection, using the connection, and then releasing the connection, similar to a landline telephone system. This requires all the parameters for the connection to be specified in advance, which may involve a negotiation between the sender and the receiver.

A connectionless service involves providing every packet with the full source and destination address for the data. Each packet is then sent independently so that each node the packet passes through chooses the next node to send it to. The current node makes this decision by choosing the node that is closest to the packet's final destination from the nodes of which it is aware. There are two main switching methods for a connectionless service: "store-and-forward" and "cut-through". "Store-and-forward" involves each intermediate node receiving a message in full before forwarding packets to the next node, while "cut-through" does not require an intermediate node to have the entire message before it starts forwarding packets [2].

Table 2.1.: The Open Systems Interconnection Reference model [3]

7	Application	The layer where user programs are located. In addition to user specified programs, there are common applications such as FTP, Telnet (virtual terminal), or e-mail, which operate at the application layer.
6	Presentation	Provides a set of data transformation services including character conversion, data compression, and data encryption.
5	Session	Designed to manage an entire conversation, consisting of a number of dialogue units which can be suspended and restarted through synchronisation points. However, client/server communication is based on an asymmetric model, in which one or more client processes can request resources from a server process.
4	Transport	Provides datagrams for both connection-oriented and connectionless protocols. In a connection-oriented service, the transport layer negotiates a suitable quality of service for the given network and application. This typically includes throughput, transit delay, and error rate among other parameters.
3	Network	Highest layer within the communication subnet. Deals with control issues such as routing, congestion control, and error recovery.
2	Data Link	Provides frames which give addressing, error control, and sequencing. Necessary to provide a reliable service.
1	Physical	Responsible for transporting bits of information. Bandwidth, signal levels, and signal coding methods are specified at this level.

2.1.4. Physical Media

There are different categories of physical media that can be used to transmit data. The two most common types are UTP and optical fibre. UTP consists of two copper wires coated in plastic and twisted together. The wires are twisted partially to keep them together, but also to help cancel out electromagnetic interference between the wires [28]. UTP is currently the dominant media, primarily since it has been commercially available for so long.

Optical fibre performs better than copper in most categories, only really falling behind when it comes to cost, although the price gap between UTP and optical fibre is narrowing [29]. Optical fibre has a significantly higher theoretical maximum

Table 2.2.: The TCP/IP model [4]

4	Application	Equivalent to the Session, Presentation, and Application layers of the OSI model. The Application layer is used to handle all process to process communication, and includes session establishment, compression, and encryption.
3	Transport	Equivalent to the Transport layer of the OSI model. The Transport layer includes message segmentation, session multiplexing, and message reordering.
2	Internet	Equivalent to the Network layer of the OSI model. The Internet layer includes traffic routing, traffic control, and logical addressing.
1	Link	Equivalent to the Data Link and Physical layers of the OSI model. Provides physical network functions like signal levels, signal coding methods, and error detection.

bandwidth, is much more durable, and can be used across much longer distances with less signal attenuation [30]. UTP and optical fibre use different connectors and so it is difficult to take full advantage of the benefits of optical fibre when using the standard RJ-45 connectors found in most consumer hardware. All of the hardware platforms offered by NetFPGA, apart from the NetFPGA 1G [31], are designed to use optical fibre.

2.1.5. Packet Headers

As mentioned in section 2.1.1, 'headers' are used to store metadata about network packets. Each layer of the TCP/IP model prepends an additional header to the data, resulting in a payload with multiple headers attached, as shown in figure 2.1. Most protocols have a set header format, making it much easier to extract data from them. As an example, the header for an IPv4 packet is shown in figure 2.2. Each row of the header represents 32 bits of data. The *Options* field of the header has a variable length, so the *IHL* field is used to record the length of the IP header. The *Total Length* field represents the total size of the packet in 32 bit words, including both the header and the payload. This information can be used to extract the data from the headers, and to remove the appropriate number of bits in order to read the payload itself. For this project the *P4* language [32] will be used to extract data from packet headers, as will be discussed further in section 2.3.

The *Version* field of the IP header refers to which version of the protocol is used by the packet. The most commonly used version is IPv4, although a newer version, IPv6, has been standardised [33] [34]. IPv6 was primarily intended to address the issue of 'addressing' in IPv4. Every device on a network using IP requires a unique IP address. When data is being sent to a particular device, its IP address is used at the Internet layer of the TCP/IP model to locate the device. An IPv4 address is 32 bits long, meaning there are a total of 4,294,967,296 IPv4 addresses for a given network. While this is more than enough for any local network, a very large network (such as the Internet) does run the risk of reaching this limit. In fact, the internet has now had all of the available addresses allocated [35]. In comparison, an IPv6 address is 128 bits long, providing a total of 3.4×10^{38} IPv6 addresses. Since IPv6 is not yet widely implemented, it is considered beyond the scope of this project.

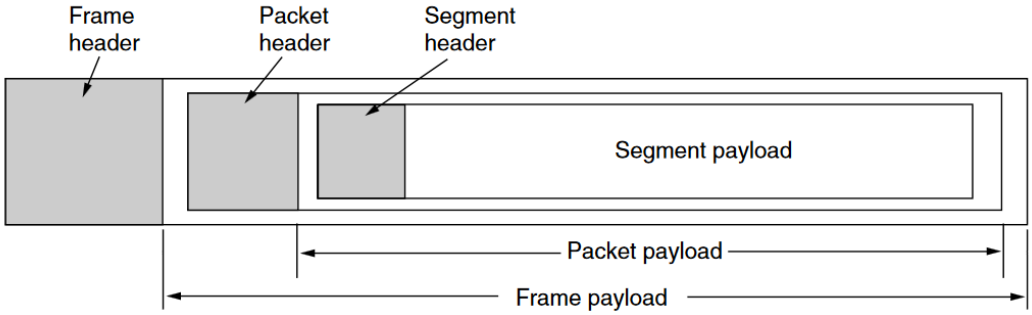


Figure 2.1.: Nesting of segments, packets, and frames. [2]

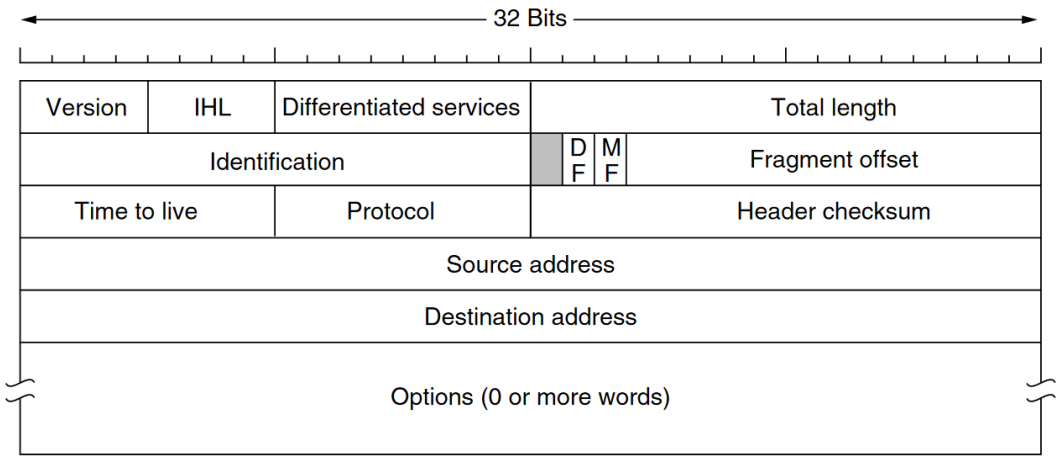


Figure 2.2.: The IPv4 (Internet Protocol) header. [2]

2.1.6. Software-Defined Networking

The primary benefit of SDN is that it gives the programmer more direct and clear control over how routing decisions are made. SDN aims to separate the network control, known as the control plane, from the forwarding process, known as the data plane [36]. The control plane makes decisions about packets which are either destined for or originally from the current host, while the data plane makes decisions about packets which are going through the host. The control plane also maintains the routing table, which is used by both planes to make routing decisions. *P4* is an example of a language which can be used to manipulate the control plane, and is discussed further in section 2.3.

2.2. Mininet

Mininet is an open-source network emulator written primarily in *Python* [25]. It is intended for use in creating virtual networks, and can be used to design custom network topologies and test different performance characteristics of these networks. *Wireshark*, the popular network protocol analyser [37], can be integrated into mininet, allowing network packets sent within a virtual mininet environment to be easily captured and analysed. Mininet also provides a *Python* API, allowing it to be easily integrated within other applications, and for complex topologies to be constructed using *Python*. It is very lightweight, with the total size of the source code being less than 1MB. Using Mininet will naturally introduce a dependency chain. However, this is the case with most open-source software, and due to its lightweight nature there are a relatively small number of dependencies. Mininet uses CI testing through *Travis CI* [38], which will run all unit and integration tests upon every commit to the Mininet *GitHub* repository [39]. These tests can be checked to improve the reliability of Mininet.

Currently, the latest stable release of Mininet is version 2.2.2, which was released on 21 March 2017. This is supported on Ubuntu 14.04 LTS, and is not supported on newer versions of Ubuntu (or other distributions) due to an incompatibility between this version of Mininet and the Linux 3.16 (or newer) kernel. As a result, software which uses Mininet 2.2.2 will also be unsupported on versions of Ubuntu newer than 14.04 LTS.

The fifth release candidate for Mininet 2.3.0 was released on 14 March 2019, and it is expected that when Mininet 2.3.0 is released, it will be supported on Ubuntu

18.04 LTS.

2.3. P4

P4 is a language used in SDN for programming the data plane of network devices [32]. The specification for $P4_{14}$ (the first version of *P4*) was released in September 2014. This went through a few minor revisions before the specification for $P4_{16}$ (the current version) was released. *P4* is a language which could be used on an FPGA-based smart network switch in order to control how packets are processed through the switch, but is not used in the implementation aspect of this project.

2.4. NetFPGA

The *NetFPGA* platform is an "open source hardware and software platform designed for research and teaching" [26]. As demonstrate in *Prototyping Fast, Simple, Secure Switches for Ethane* [40], the *NetFPGA* platforms can be used to create highly efficient and fast network hardware, and this paper influenced their consideration for this project. *NetFPGA* provides four hardware platforms, each with a different specification and target audience. As decribed in the initial specification in appendix C, the *NetFPGA 1G* was initially considered to be the most suitable hardware platform for this project, due to its compatibility with general purpose consumer hardware. This was in part due to its use of UTP copper, while all other *NetFPGA* platforms use fibre optics.

While the *NetFPGA 1G* would have been the most suitable hardware platform for a prototype model, this prototype is no longer being developed due to a change in the focus of the project, and the most suitable hardware platform for a production model would be the *NetFPGA SUME* (shown in figure 2.3). This platform is based around a *Xilinx Virtex-7 690T* [41] which contains 693,120 logic cells and 52,920 Kbit block RAM. It also incorporates 10-Gigabit Ethernet networking ports, as well as 4 external SFP+ ports for connections of up to 100Gbps. Like the *NetFPGA 1G*, it has a standard PCI form factor, and so is compatible with most consumer motherboards as an add-on card, and can also be used as a standalone board.

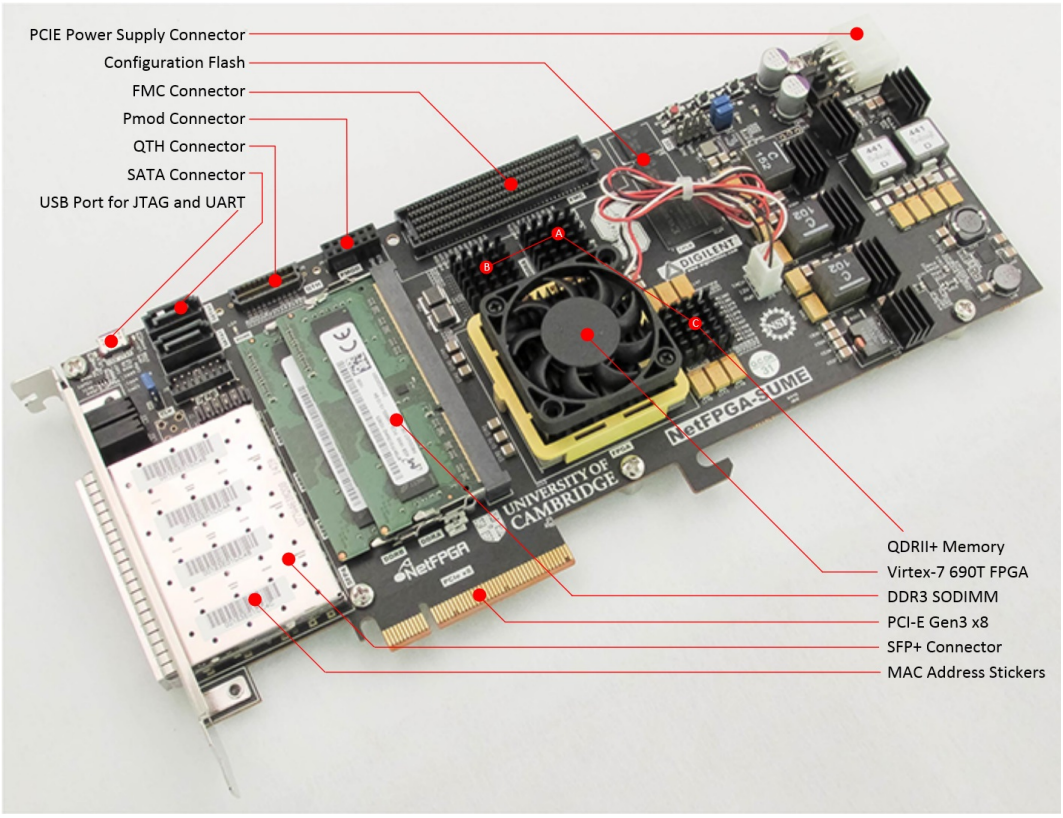


Figure 2.3.: Labelled NetFPGA SUME

3. Requirements

In order to achieve the aims of the project discussed in section 1.3, a set of requirements have been constructed. These requirements will assist in defining the scope of the project, as well as defining the aims in a more practical manner. The requirements have been written using the *MoSCoW* method [42], and have been separated into functional and non-functional requirements.

3.1. Functional Requirements

The functional requirements describe the technical capabilities of the implemented aspect of the project. They detail how the system should react in certain scenarios, and state what services the system should be able to provide. The can also state limitations in the scope of the system, i.e. routes of development that are not intended to be implemented [43].

F1: The system **must** allow the modelling of a network with at least three switches and four hosts, arranged in a tree topology as shown in figure 3.1.

F2: The system **must** allow switches and hosts to be configured in a tree topology.

F3: The system **must** allow users to specify the spread of the network when using a tree topology (i.e. how many children a switch node will have).

F4: The system **must** allow users to specify the number of levels in the network when using a tree topology.

F5: The system **must** allow users to specify which level in the network should be modelled as FPGA-based smart network switches.

F6: The system **must** allow users to run a test to ensure there is an active connection between all hosts.

F7: The system **must** allow users to run a test to display the bandwidth of the system.

F8: The system **must** allow users to run a test to display the delay between the leaf nodes and the root when using a tree topology.

F9: The system **should** allow users to specify the bandwidth of all links in the network.

F10: The system **should** allow users to specify the delay of all links in the network.

F11: The system **should** allow users to specify the chance of packet loss for all links in the network.

F12: The system **should** allow switches and hosts to be configured in a vertical linear topology.

F13: The system **should** employ multiple levels of logging.

F14: The system **should** conform to *Python*'s PEP8 code standard [44].

F15: The system **should** be made open-source.

F16: The system **could** allow users to configure the system to use a Poisson distribution for link delays.

F17: The system **could** allow users to configure the bandwidth of FPGA-based smart network switch nodes.

F18: The system **could** allow users to configure the delay of FPGA-based smart network switch nodes.

F19: The system **could** allow users to configure the chance of packet loss of FPGA-based smart network switch nodes.

F20: The system **could** allow users to display all node connections before running tests.

F21: The system **won't** be configured for IPv6 connections.

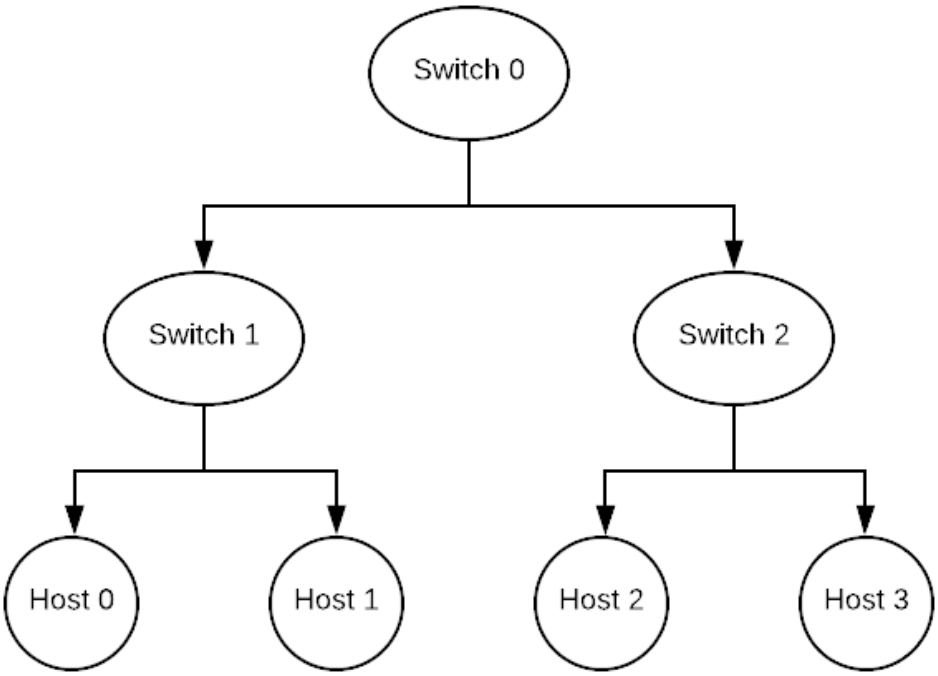


Figure 3.1.: The Required Minimum Tree Topology

3.2. Non-Functional Requirements

These requirements describe the general operation of the system produced, as well as how the project should be developed. They often apply to the system as a whole rather than targeting individual components, and they can include any legislative constraints on the system [43].

NF1: The system **must** be modular, to allow extensibility.

NF2: The system **should** be scalable.

NF3: All third-party libraries used by the system **should** be open-source.

NF4: The system **should** be efficient, able to run all tests on relatively large virtual networks in small amounts of time.

NF5: All code for the system **should** be well-documented and maintainable.

3.3. Initial Requirements

Due to the agile nature of the project, the requirements for this project have undergone a number of revisions over the course of the project. The initial requirements were written as part of the project specification (shown in appendix C) at the start of the project, and these reflect the intentions for the project at that time. The revisions made to the project since these requirements were written has rendered many of these redundant, and these are shown to illustrate some of the changes the project has undergone.

3.3.1. Initial Functional Requirements

F1: The system **must** be able to analyse packets at layer 2 of the OSI network model.

F2: The system **must** be able to send packets to the correct port based on MAC addresses.

F3: The system **must** be able to store MAC address tables.

F4: The system **must** be able to dynamically update MAC address tables.

F5: The average latency of the system **must** be measured.

F6: The throughput of the system **must** be measured.

F7: The system **should** be able to analyse packets at layer 3 of the OSI network model.

F8: The system **could** be able to analyse packets at layer 7 of the OSI network model.

F9: The system **could** be able to detect basic network attacks (such as TCP SYN Flooding [45]).

F10: The system **could** implement features of a basic firewall (such as packet filtering [46]).

3.3.2. Initial Non-Functional Requirements

NF1: The system **should** be scalable.

NF2: The system **should** be efficient.

NF3: All code for the system **should** be well-documented and maintainable.

3.4. Project Constraints

This project is subject to a number of constraints which limit the capabilities or scope of the project. These have been separated into the constraints which affected the implemented aspect of the project and the constraints which would apply to an FPGA-based smart network switch.

3.4.1. System Constraints

The constraints described in this section affected the implemented aspect of the system. As a dependency of the system, *Mininet* [25] imposed a number of constraints, largely relating to the software of the system. *Mininet*'s API is written in *Python*, which restricts the system itself to *Python*. Naturally it would be possible to combine an alternative language with *Python*, and simply use *Python* to interact with the API. However, this would introduce an additional level of complexity to the project with little potential benefit, since any improvements in speed offered by a different language would be bottlenecked by the *Python* API.

The most recent stable version of *Mininet*, 2.2.2, uses *Python* 2.7, an older version of Python due to be deprecated on 1 January 2020. While the next stable release of *Mininet* is expected to be updated to use *Python* 3, this has resulted in the project having to use *Python* 2.7.

Mininet 2.2.2 contains a known issue with Ubuntu 16.04 LTS or newer due to an incompatibility with the Linux kernel versions 3.16 and higher [47]. As a result, *Mininet* recommends using Ubuntu 14.04 LTS, which was released in April 2014. Normal LTS support for Ubuntu 14.04 ended on 25 April 2019, at which point further security updates for the OS became only available to those who purchase "Extended Security Maintenance" for it [48]. As mentioned in section 2.2, the next stable release of *Mininet* is expected to be supported on Ubuntu 18.04 LTS.

The system is intended to be lightweight and available to all users, so it needs to be capable of running on systems with a relatively low specification, and should not take a long time to run.

3.4.2. FPGA-based Smart Network Switch Constraints

The constraints described in this section would affect an FPGA-based smart network switch. One constraint on these devices is the physical size of the hardware. In order to conform to industry standards, an FPGA-based smart network switch would need to be designed so that it could be mounted in a standard 19 inch server rack. The height of these server racks and the devices mounted within them are measured in 'rack units', or 'U's, with a standard full size rack being 42U tall. Network switches are rarely largely than 1U, and as a result an FPGA-based smart network switch should also be 1U, in order to make it easier for users to transition from a standard network switch to an FPGA-based smart network switch.

As discussed in section 2.1.4, modern networks use two primary mediums for wired data transfer, UTP copper cables, or fibre optic cables. An FPGA-based smart network switch will be constrained to use one (or both) of these mediums.

4. Ethical, Social, Legal, and Professional Issues

This section discusses the different ethical, social, legal, and professional issues that may occur during the course of this project, and what measures can be taken to mitigate these. The BCS Code of Conduct [49] will be followed throughout the project as an initial approach to avoiding more common issues, and maintaining good development practice.

4.1. Ethical Issues

The primary ethical consideration for this project was privacy. While no user data is directly required for this project, a wide variety of data would naturally pass through a complete FPGA-based smart network switch. As a result, when designing these devices, or even just considering the foundation for them, it is important to ensure that they are secure, and would be resistant to malicious attempts to read data passing through the switches.

This can link to the physical media discussed in section 2.1.4. Fibre optic cable is significantly more secure than UTP copper cable, which is susceptible to 'tapping', where an additional cable can be attached to the original copper cable, and this can be used to read the data flowing through it. This technique is extremely difficult with fibre optic cable, since attempts to do so will almost always break the cable. This ethical issue should influence the design of the FPGA-based smart network switches, and the types of physical media with which they are compatible.

4.2. Social Issues

Wherever users are involved with this project, for example user testing, all user data will be entirely anonymised to prevent the possibility of discrimination or bias.

Where user data is required, all effort will be made to avoid collecting data that could identify an individual, as well as data that could be used in an inappropriate way, such as gender, sexual orientation, religion, or political status. As stated above, the BCS Code of Conduct will be used wherever appropriate throughout the project.

4.3. Legal Issues

When using third-party tools, it is important to ensure that they are being used in accordance with the licence with which they are provided. Much open-source software is provided with either the *Apache License 2.0* [50], the *GNU General Public License v3.0* [51], or other similar licences allowing conditional free use of the software. As a result, this project will aim to use open-source tools where they are available to avoid issues with copyright.

Two Regulations this project will additionally be careful not to contravene are the *General Data Protection Regulation 2016* [52] and the *Data Protection Act 2018* [53]. This legislation involves the storage, collection, and processing of data, and relates to the issues descibed in sections 4.1 and 4.2.

Wherever secondary data (i.e. data not directly collected by the developer) has been used, any conditions to the use of this data must be considered. Should any terms of use relating to the data be accepted, these must be adhered to, and the primary stakeholders outlined in section 1.4.1 should be aware of any consequences that could occur from breaching these terms of use.

4.4. Professional Issues

Since the developer is a member of the BCS, following the BCS Code of Conduct [49] is a general professional aim, not limited to the scope of this project. The BCS Code of Conduct comprises four key principles:

1. You make IT for everyone
2. Show what you know, learn what you don't
3. Respect the organisation or individual you work for
4. Keep IT real. Keep IT professional. Pass IT on.

These principles have been consciously followed throughout this project. By making the implemented aspect of the project open-source, the first principle is being followed. The IT application created as a result of this project is being made available for everyone. This project has involved a great deal of learning for the developer in order to gain the necessary skills to research and develop the project. In addition, the background knowledge discussed in section 1.1 was used as a basis for the project. This demonstrates the second key principle. The third key principle has been taken into account whenever in contact with members of the University of Warwick. Finally, this project represents a key area of interest for the developer who has been motivated to develop and share the knowledge acquired over the course of the project, exemplifying the fourth key principle.

5. Design

After completing the initial research and constructing the requirements for the system, the development of the system began. Due to the agile methodology of the project, development was conducted incrementally, with each feature being developed sequentially along with its associated tests and documentation. This provides the advantage of the system being functional throughout the process of development, and allows for flexibility as to how the development process occurs. However, this makes the separation of the design of the system from its implementation challenging, since both occurred throughout the development of each individual feature. Nonetheless, in order to provide a clear overview of the development process, they have been separated for the purpose of the report, and the design of the system is discussed in this section.

5.1. Tool Selection

5.1.1. Network Simulation

Since the primary purpose of the system is to model networks, an application needed to be sourced to simulate these networks. While this application could have been implemented from scratch as part of the project, which would have thereby eliminated any associated licencing or dependency issues, this was deemed to be an inefficient use of time since this software already exists, and would be complex to implement. *Mininet* was chosen for this purpose since it is open-source, has a wide array of features, and integrates well with SDN using its custom version of *OpenFlow* [54]. *OpenFlow* is a "flow-based switch specification designed to enable researchers to run experiments in live networks". Similar to *P4*, *OpenFlow* can be used to manually control the data flowing through a network switch, and *Mininet* supports this protocol directly.

5.1.2. Programming Language

Python [55] was chosen as the primary programming language for the project. *Mininet*'s API is written in *Python*, and *Python* includes many tools which can be used to ease the development of an application. *Python*'s style guide, *PEP 8* [44], can be used to ensure an application is written according to the standards used in industry, and tools such as *Pylint* can notify the developer if any code written does not meet this.

5.2. Open-Source

The system being made open-source contributed to the development approach. The code was made open-source using *GitHub* [39], which uses *git* [56] as its version control system. *Git* tracks changes in the form of 'commits', which constitute a defined set of changes made to the code, accompanied by a short message summarising the changes. Commits are a core concept of *git*, however *git* contains many additional features pertaining to managing the source for a project. Many of these features are complex, and as a result care needs to be taken when using *git* for an open-source project that it is used in an appropriate way. Each commit should be used for exactly one step in development, where these steps are suitably sized so that each step is as small as possible, while ensuring that the system functionality is maintained with every commit. In other words, no commit should decrease the functionality of the system, unless that is the intended purpose of the commit.

Another core concept of *git* is 'branches'. Each branch contains a replica of the project source, from the point in time when the branch was created. Each branch is totally independent of every other branch, and there is no limit to the number of branches which can be created. This means that two different features can be developed simultaneously without impacting one another, by developing each one on its own branch. Branches are also able to be 'merged' into one another, appending all of the commits from one branch onto the branch into which it is being merged. This feature was used heavily throughout the development of the project, with every feature being developed in its own branch before being merged into the primary branch of the project, known as the 'master' branch.

5.3. Dependencies

Since the project included dependencies, a reliable way of delivering the system with these dependencies was required. While the source code for all dependencies could be manually copied into the repository for the system, this would then require manually copying the code for every dependency whenever updates are released, and would also increase the size of the system. For all *Python* libraries used (which would normally be installed using *Python*'s *Pip* package manager), the *Python setuptools* [57] library was used. This is installed by default with every installation of *Python*, and allows for *Python* dependencies to be specified in a *Python* file which defines setup characteristics of the application. Whenever an application which uses *setuptools*, such as this system, is installed using *Pip*, *Pip* will then check the *setuptools* file for the list of dependencies, and also install these.

The only dependency used by the project which could not be included in this way was *Mininet*. Since *Mininet* is open-source, and its code is available in a *git* repository on *GitHub*, it was decided to use *git submodules* in order to register *Mininet* as a project dependency. *Git submodules* are a feature of *git* which allows a *git* repository to declare further *git* repositories that it uses. These additional repositories will then be downloaded with the source code for the main system whenever it is downloaded using the '–recursive' flag. *Git submodules* can also be attached to a particular commit of a repository, and this has been used to attach the submodule of *Mininet* to its version 2.2.2 release. When a new stable version of *Mininet* is released, this can be tested, and then updated simply by changing the version number in the *git submodules* configuration file.

6. Implementation

6.1. Initial System

The system was initially implemented as a wrapper for the *Mininet* API. This would utilise the relevant features of the API, while restricting access to those which were not relevant to the system. This involved declaring *Mininet* as a *git submodule* of the system, as discussed in section 5.3. The implemented features of the *Mininet* API were a simple tree topology, and a network test built into the API which would confirm that all hosts in the topology could communicate with one another. Once these had been implemented, running the system would:

1. Initialise the network
2. Create the required switches
3. Create the required hosts
4. Create the required links between switches and other switches, and between switches and hosts
5. Run the network test and display the results
6. Break down the network.

6.2. System Structure

With the core functionality in place, structure could now be added to the system to allow additional features to be added in a sustainable and efficient way.

6.2.1. User Customisation

For command line applications such as this one, customisation is commonly performed through either configuration files, or through arguments specified at the

command line. For this system, command line arguments were deemed more appropriate, since they allow for a simpler user experience, while still providing a high level of flexibility. The *Python* library *Click* [58] was used to implement these arguments. *Click* was chosen since it allows for a large amount of customisation of the system's arguments. By default, *Click* configures the '–help' flag, which will print out a formatted list of all available flags and arguments for the system, along with the type of data they expect and any explanatory messages associated with each flag or argument. For each flag or argument, the type can be configured, as can the explanatory message, and the name of the flag or argument. A default value can also be set, and custom restrictions can be placed on the type of data accepted by each flag or argument.

6.2.2. Documentation

Suitable documentation was required to ensure the usability of the system. A separate user guide was considered. However, it was concluded that this was unnecessary, due to the straightforward nature of the command line interface. Instead, a 'Readme' file was written in *Markdown* [59] and added to the *GitHub* repository for the system where it would be formatted and clearly displayed.

6.2.3. Testing Structure

Travis CI [38] is a web-based CI platform designed for running automated tests, and this was used to conduct many of the tests in the system. The testing process itself is discussed in detail in section 7. However, prior to tests being written, *Travis CI* needed to be configured to automatically run tests on changes it detected in *GitHub*. To do this, a *YAML* configuration file for *Travis CI* was added to the code. A directory was also created for tests to be added to.

6.3. Feature Implementation

Features could now be implemented on the system. The development approach up to this point would allow for each feature to be added into the system without needing to be concerned with anything other than that feature. Features were added sequentially, along with their required *Click* configuration, their tests, and their documentation.

Table 6.1.: Log Levels for *Python Logging* Package

Log Level	Numeric Value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10

6.3.1. Logging

The first feature to be added was standardised logging. This was done using *Python*'s *logging* [60] package, which is designed for this purpose and is highly customisable in how its logs are recorded. The logging package uses concepts of 'loggers', 'handlers', and 'formatters'. A logger receives log messages from a system, and dispatches these to a set of defined handlers. The handlers will then process the log messages, according to the formatters they are linked to. A formatter simply defines a format of a log message. For this system, one standard formatter was used, which was defined as `%(asctime)s - %(name)s - %(levelname)s - %(message)s`. This formatter would print the current date and time, followed by the name of the logger, followed by the log level used for the message, followed by the actual log level.

The logging package supports six log levels which are each associated with an integer value. These are shown in table 6.1. Whenever a message is logged by the system, it is logged using one of these levels. A user can specify the mininum log level to be displayed, and then all log messages of this level or higher will be displayed. If no level is specified, the system defaults to INFO.

The majority of the configuration for the logging package was placed in a *JSON* file. Three log handlers were specified in this file. These handlers logged all messages with a log level greater or equal to the system's minimum log level. Each handler logged the messages to a different location, and had a different hard minimum log level of messages it would record, shown in table 6.2. The log files used were automatically rotated by the *logging* package, such that when the log files reached 10MB in size, the data would be moved to an old file, and a new file would be created. A maximum of 20 backup log files would be kept for each of the two handlers which logged data to files.

Table 6.2.: Log Handlers Used

Log Target	Hard Minimum Log Level
Console	DEBUG
Log file ‘fpga_switch_model.info.log’	INFO
Log file ‘fpga_switch_model.error.log’	ERROR

6.3.2. Performance Tests

Functions were implemented to allow users to measure the performance of the virtual networks created by the system. The initial system described in section 6.1 included a simple test to confirm all hosts could communicate with one another. This test was based on the ‘ping’ network utility, which sends an ICMP packet from one host to another, and measures the round trip time for the packet. The test used in the initial system, known as a ‘ping-all’ test, iterates through every host in the virtual network pinging every other host, and confirms that a response is received to every ping. This test was configured so that it would only be executed if the system was started with the ‘-p’ or ‘–ping-all’ flags.

Mininet also includes a test to measure the bandwidth of the virtual network, based on the *iperf* [61] tool. This test was added to the system, and was configured so that it would only be executed if the system was started with the ‘-i’ or ‘–iperf’ flags.

At this stage of implementation, the topology of the virtual network created by the system was limited to a tree topology which could not be modified. However, by the end of the implementation stage, the depth and spread of the tree topology would be customisable, and the user would be able to specify a level of the tree to be modelled as FPGA-based smart network switches. With this in mind, an additional performance test was written which would measure the round trip time between either the base and the root of the tree, or between the base and the FPGA-based smart network switch layer of the tree. The purpose of this was to measure the time it would take a packet to be processed in the virtual network, if the root of the tree was thought of as ‘the cloud’. This test was conducted by sending 10 packets using the ping network utility, and reading the utility’s output. This consisted of the minimum round trip time, average round trip time, maximum round trip time, and the standard deviation. The test was configured so that it would be executed by

default, although it could be disabled by setting the '-c' or '–cloud-fpga' flags to 'False'.

6.3.3. Network Topology Customisation

Two options were added to customise the topology of the virtual network. The topology was still restricted to a tree, but the spread of the tree could be specified by the user with the '-s' or '–spread' flags, and the depth of the tree could be specified by the user with the '-d' or '–depth' flags. If the spread was not specified by the user, it would default to two. This value was chosen since this was the minimum spread which would still result in a tree topology. A spread of one would result in a series of switches with a single host connected at the end. If the depth was not specified by the user, it would default to four. This value was chosen since this was the minimum depth which would allow a root node, a layer of FPGA-based smart network switches, a layer of standard switches, and a layer of hosts.

6.3.4. Network Performance Customisation

Customisation options were then added to limit the performance of the virtual network, in order to bring the model closer to a real network. One option which was added was to limit the bandwidth of the virtual network. This limit would be applied to all of the links in the network, and could be specified using the '-b' or '–bandwidth' flags. If this option was not specified, a limit of 10Mbps would be applied. Another option applied a delay on all links in the network, and could be specified using the '-e' or '–delay' flags. If this option was not specified, a delay of 1ms would be applied. A futher option caused a chance for packets to be lost when passing through a link, and could be specified using the '-l' or '–loss' flags. If this option was not specified, no chance of loss would be applied.

An option was also added to configure the network delay to be selected from a Poisson distribution for which the average is the current network delay. This was designed to add an element of randomness to the system, again to bring this closer to a real network. The '–poisson' flag could be used with the system to enable the use of a Poisson distribution for the link delay.

6.3.5. FPGA-Based Smart Network Switch Customisations

This set of options introduced FPGA-based smart network switches to the system. With the '-f' or '–fpga' flags, a level of the tree topology could be specified to be modelled as FPGA-based smart network switches instead of regular switches. These switches were implemented as regular switches linked to a host, where the link between them had double the possibility for packet loss compared to that of standard links, a bandwidth of 504Mbps, and latency twice that of standard links. These hosts were treated as the FPGAs in the switches, and were used to respond to any pings directed to the switch, such as those used in the cloud-fpga test discussed in section 6.3.2. The reason for this increased possibility for packet loss was that it was seen as more likely for packets to be lost inside an FPGA-based smart network switch compared to travelling along a wire, or when travelling through a standard switch. The default bandwidth was set to 504Mbps since this is the maximum bandwidth of the PCIe interface, and it is assumed that an implemented FPGA-based smart network switch will incorporate the FPGA using an interface at least this fast. The latency was increased to account for the time it would take to perform the computation on the packets in the switch before they are returned.

All of the performance characteristics described above can be customised using options added to the system. The '–fpga-bandwidth' flag can be used to specify the bandwidth of the FPGA-based smart network switch links, the '–fpga-delay' flag can be used to specify their latency, and the '–fpga-loss' flag can be used to specify the probability of packet loss.

7. Testing

Comprehensive tests were necessary in order to ensure that the implemented system functioned as intended. The testing for this system was separated into unit, integration, system, environment, and user testing, as discussed further in the sections below.

As mentioned in section 6.2.3, *Travis CI* [38] is a web-based CI platform designed for running automated tests. It was used for many of the tests in this project due to its clear interface, and easy integration with *GitHub* repositories. Whenever code was uploaded to *GitHub*, *Travis CI* would automatically run all configured tests on the code, and would notify the developer by email if any tests failed. The testing environment used by *Travis CI* can be easily configured using a single *YAML* [62] file located in the *GitHub* repository for the project.

7.1. Unit Testing

Unit testing was conducted to ensure that each individual component of the system functioned as intended. Tests were written for function in the code using a variety of different inputs, and the output of these tests was compared with the expected output. The *Python* library *unittest* [63] was used to write these tests, as it is designed for this purpose, and integrates easily with both the existing system and *Travis CI*. The results of the unit tests can be seen in a simplifed format in table 7.1.

Table 7.1.: Results of Unit Tests

Unit Test	Success / Failure
Test *halve_delay* function	SUCCESS
Test *get_poisson_delay* function	SUCCESS
Test *test_cloud_fpga* function	SUCCESS
Test *TreeTopoGeneric* class	SUCCESS

Table 7.2.: Results of Integration Tests

Integration Test	Success / Failure
Test *TreeTopoGeneric* class with *get_poisson_delay* function class	SUCCESS
Test *TreeTopoGeneric* class with *get_poisson_delay* function class	SUCCESS
Test *TreeTopoGeneric* class with *halve_delay* function	SUCCESS
Test *TreeTopoGeneric* class with *test_cloud_fpga* function	SUCCESS
Test *TreeTopoGeneric* class with *get_poisson_delay* function class and *halve_delay* functions	SUCCESS
Test *TreeTopoGeneric* class with *test_cloud_fpga* and *halve_delay* functions	SUCCESS
Test *TreeTopoGeneric* class with *test_cloud_fpga*, *get_poisson_delay* and *halve_delay* functions	SUCCESS

7.2. Integration Testing

Integration testing was conducted to ensure the individual components of the system interacted with each other as intended. Tests were written for different combinations of functions which were used together in the system using a variety of different inputs, and the output of these tests was compared with the expected output. As with the unit tests discussed in section 7.1, the *Python* library *unittest* was used to write these tests. The results of the integration tests can be seen in a simplifed format in table 7.2.

7.3. System Testing

System testing was conducted to ensure the system as a whole performed as intended. The entire system was tested using different inputs for the available configuration options, and the output of the system was compared with the expected output. The results of the system tests can be seen in table 7.2.

7.4. Environment Testing

Environment testing was conducted to ensure the system performed as expected in different environments. The system was tested using a variety of different operating systems with both *Python* 2.7 and *Python* 3.6, and the output of the system was compared with the expected output. The results of the system tests can be seen in table 7.2.

7.5. User Testing

User testing was conducted to ensure that the system was sufficiently easy to use, and to find any issues with the system not identified through other means of testing. The system was presented to a number of individuals who were asked to use the system without further instruction, and were then observed. Feedback received through this method is shown in table 7.5, including the steps taken to resolve any issues identified.

Table 7.3.: Results of System Tests

System Test	Linked Requirement(s)	Success / Failure
Can the system function using a tree topology with a spread of two and a depth of four, with a total of 8 hosts?	F1, F2	SUCCESS
Can the system function using a tree topology with a spread of three and a depth of 8, with a total of 2187 hosts?	F1, F2, F3, F4	SUCCESS
Can the system report the bandwidth in the network?	F7	SUCCESS
Can the bandwidth in the network be configured correctly?	F9	SUCCESS
Can the link delay in the network be configured correctly?	F10	SUCCESS
Can the probability of packet loss in the network be configured correctly?	F11	SUCCESS
Can the link delay in the network be configured using a Poisson distribution correctly?	F16	SUCCESS
Can all node connections be correctly displayed?	F20	SUCCESS
Can the user specify the level of logging employed by the system?	F13	SUCCESS

Table 7.4.: Results of Environment Tests

Python Version	OS	Success / Failure
2.7	Ubuntu 16.04 LTS	SUCCESS
3.6	Ubuntu 16.04 LTS	SUCCESS
2.7	Ubuntu 14.04 LTS	SUCCESS
3.6	Ubuntu 14.04 LTS	SUCCESS

Table 7.5.: Results of User Tests

User Feedback	**Developer Response**
It is not clear how to install the system.	An installation guide was added to the public *GitHub* page for the system.
It is not clear what the available customisation options are.	While a guide to the available customisation options can be found by passing the *–help* flag to the system, an equivalent guide was added to the public *GitHub* page for the system.

8. Project Management

With a relatively long term project such as this, techniques to manage the work involved are vital in ensuring deadlines and requirements are met. This project needed to be flexible in order to respond to changes in the direction of the project, and efficient in order to complete on time. This section will discuss the management techniques which have been employed to ensure the project has progressed as successfully as possible.

8.1. Software Development Methodology

This project uses an agile methodology in order to adapt to changes arising during the project. This approach proved invaluable since the direction of the project went through a series of significant changes as it developed. Weekly meetings between the developer and the project supervisor over the course of the project allowed for frequent, short term targets to be set, and allowed for the project to be regularly re-evaluated to ensure that it was still viable in its current state, and if not, to make appropriate adjustments.

A less flexible methodology, such as a waterfall methodology, may have resulted in the project's failure, due to the significant changes to the specification which needed to be made as the project progressed.

8.2. Project Schedule

An outline of the final project timetable can be seen in figure 8.1. Due to the agile nature of the project, some tasks overlapped or could not be defined precisely in weeks. However, figure 8.1 is as accurate as possible. The initial project timetable from the specification can be seen in figure 8.2. This timetable is now redundant, but is shown to illustrate how the project has changed since it was first defined.

8.3. Tools

Many different management and development tools were used during the project. Management tools were used to ensure that the project continued according to plan, and that if changes in direction were required, they were documented and made in the most appropriate way. Development tools were used to ensure that the design, research, and implementation stages progressed as efficiently as possible, and that the quality of the produced system was as high as possible.

8.3.1. Management Tools

The management tools used in the project are shown in figure 8.3, and are discussed below.

Atom [65]

Atom was used as a text editor for writing the specification, progress report, and final report of the project. These are all key components of the project, and *Atom*'s clean, customisable interface eased the editing process of these documents.

Git [56]

Git was used for version control for all the source code and document code of the project. This allowed changes made throughout the development of the system to be tracked and recorded, and makes the full history of the code and documents available to be viewed.

GitHub [39]

GitHub is a web-based hosting service for version control using *Git*. All project code, including *LaTeX* code for documents, was stored on *GitHub*. When edits were necessary, the code was downloaded, edited, and then uploaded back to *GitHub*. This meant that *GitHub* always contained the most up to date version of the code, and could also be seen as an off-site backup repository, because the content stored there would have been uploaded from a local machine holding the same data.

LaTeX [66]

LaTeX is a typesetting system which was used for all documents in this project. *LaTeX* was used to improve the efficiency of document writing, and to ensure the produced documents were of a sufficiently high aesthetic and professional standard.

Resilio Sync [67]

Resilio Sync is a multi-platform peer-to-peer file synchronisation tool. Using a modified version of the BitTorrent protocol, *Resilio Sync* is able to synchronise files between devices more quickly than a standard file sharing application could. Using this tool, project code and documents were synchronised between the different devices of the developer. This provided additional backups as well as the ability to work on the project from any device with the application installed.

Trello [68]

Trello is a web-based Kanban board. A Kanban board is used to separate a project into small tasks, and visually separate these tasks into different lists. A common layout for a Kanban board, and the one that was used in this project, is to have one list of tasks which need completing, one list of tasks which are in progress, and one list of tasks which have been completed. This makes it clear exactly what remains to be done, and the progress of every task can be tracked. *Trello* includes additional features, such as the ability to assign labels to tasks, or to add checklists to tasks.

8.3.2. Development Tools

The development tools used in the project are shown in figure 8.4, and are discussed below.

Atom [65]

In addition to being used to write the documents for the project, *Atom* was used as a text editor for writing any of the project code which was not written in *Python*.

Bash [69]

Bash is a Unix shell and command language. It was used during the project as an Ubuntu and Debian shell, and for writing shell scripts.

Docker [70]

Docker is a containerisation platform. Containers are comparable to lightweight virtual machines, although they share a kernel with their host. *Docker* was used in this project to test the operation of the application in a clean interface.

PyCharm [71]

PyCharm is an IDE designed specifically for *Python* development, and was used for the aspects of the project code which were written in *Python*. It includes a variety of useful features, such as integrated support for *pylint* [72], which scans *Python* code for bugs and syntax errors.

Python [55]

Python was used as the primary development language for the project. This was partially motivated by *Mininet*'s API being written in *Python*. However, this is also a language with which the developer was familiar, and provides access to many additional packages through *Python*'s *pip* package installer [64].

tmux [73]

tmux is a terminal multiplexer which allows a user to switch between several programs in one terminal. *tmux* was used whenever a terminal was used, and allowed, for example, multiple programs to be viewed side by side in a single terminal window.

Ubuntu [74]

Ubuntu is a Linux-based open-source OS based off *Debian*, another Linux-based OS. *Ubuntu* was used for the development of the project, as this is the primary OS on which *Mininet* is supported.

Vim [75]

Vim is an open-source text editor. It is lightweight, and can be easily used from the terminal. *Vim* was used to edit code during the project on remote servers where more substantial text editors such as *Atom* were too resource-intensive, or were not supported.

VirtualBox [76]

VirtualBox is an open-source virtualisation product, allowing virtual machines to be created and managed. *VirtualBox* was used to create virtual machines on which the application was tested in a clean environment.

8.4. Risk Management

In order to ensure the success of the project, a number of potentially negative situations have been considered. In order to minimise the impact these situations could have on the project, strategies have been established to mitigate them.

These situations are shown in table 8.1, along with a measure of the severity of the situation and a measure of the likelihood of the situation.

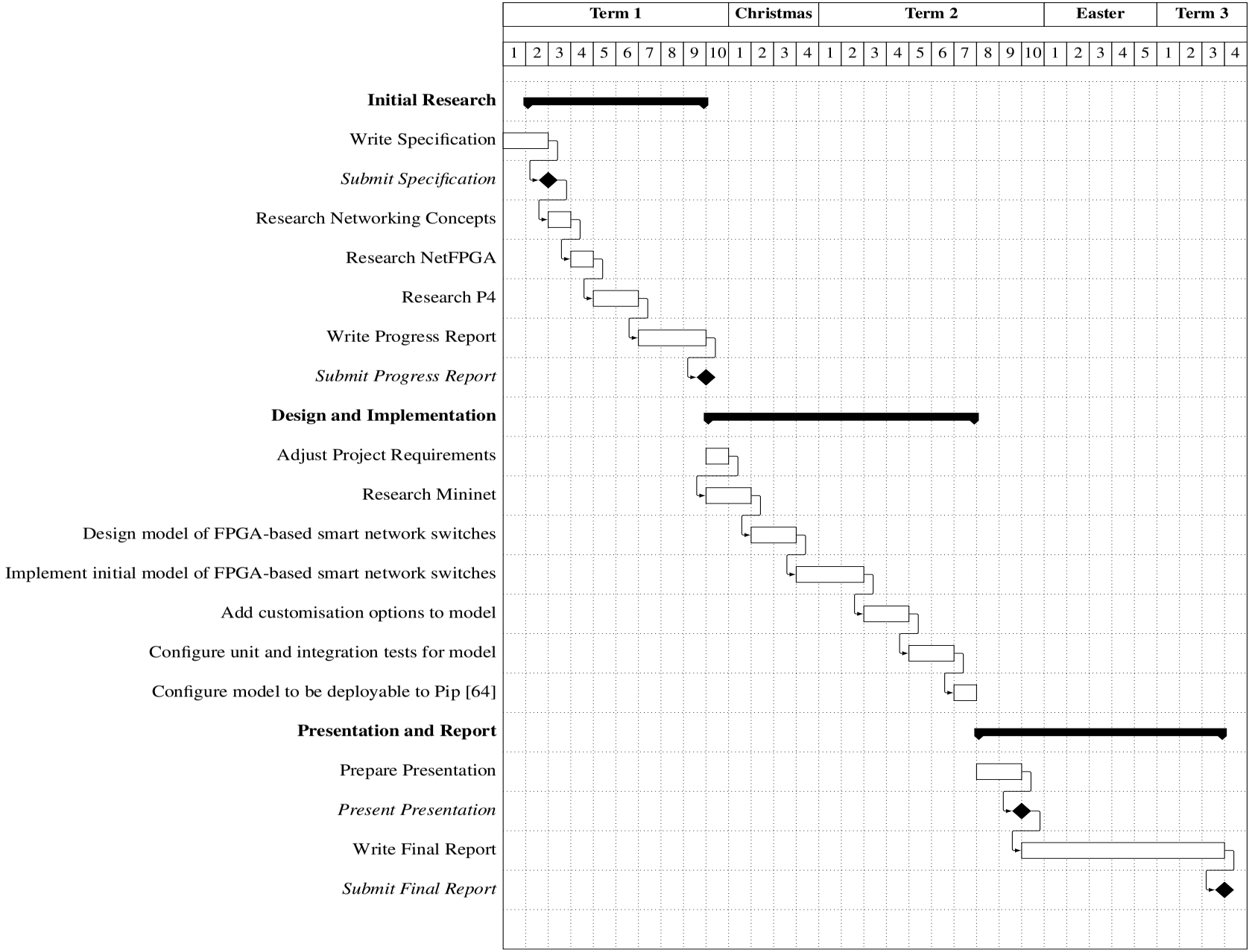


Figure 8.1.: Gantt Chart of Final Project Timetable

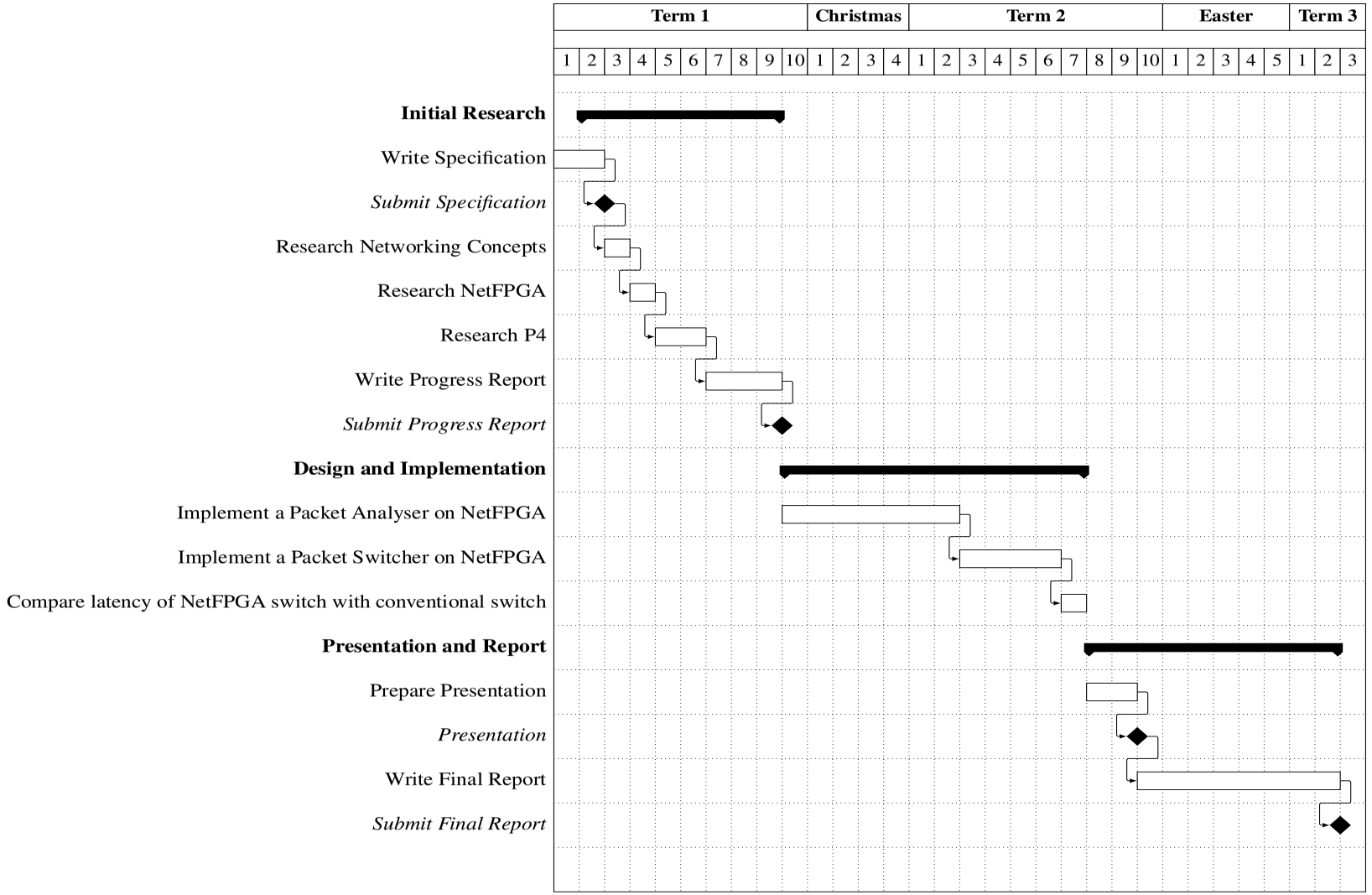


Figure 8.2.: Gantt Chart of Initial Project Timetable

Figure 8.3.: Logos of Management Tools Used

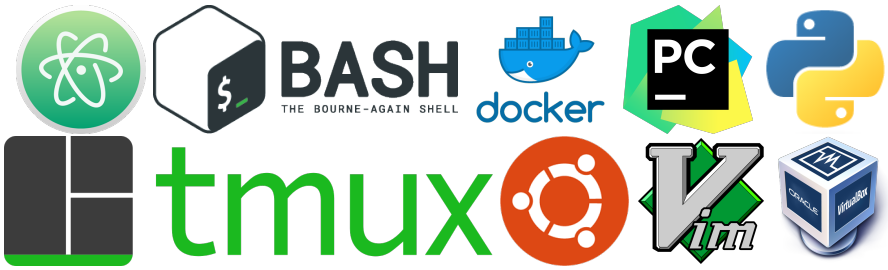


Figure 8.4.: Logos of Development Tools Used

Risk	Description	Severity	Likelihood	Mitigation
Hardware failure	Complete failure of primary machine used for project development. This would then mean an alternative device would need to be procured to use as the primary development machine, and any files existing only on this machine would be lost.	Medium	Low	All files were backed up using both *GitHub* and *Resilio Sync*. As a result, no files would be lost if the machine failed. While finding an alternative machine for development would be inconvenient, other machines are available, such as those in the Warwick Department of Computer Science.
GitHub failure	Failure of the remote repository where project code and documents are stored. This would cause the loss of any files in the *GitHub* repository.	Low	Low	Since any files in the *GitHub* repository will have been uploaded to it from a local machine, at no point should there be any data in *GitHub* which does not exist locally. In addition, *Resilio Sync* will be used to replicate files across multiple local devices, so that backups exist of all code and documents. An alternative web-based hosting service for version control using *Git*, such as *GitLab* [77], will be used in the event that *GitHub* does fail.
Mininet discontinued	*Mininet* is a significant dependency of the implemented system. If development of *Mininet* stopped, a suitable alternative may need to be found. While other network simulation software does exist, finding a replacement for *Mininet* which is open-source, and has a suitable API may be challenging. Fortunately, the implemented system would still be able to use the latest version of *Mininet*, so it may be possible to continue development with this version.	Medium	Low	A copy has been made of *Mininet*'s *GitHub* repository so that it can still be used in the event that *Mininet* is discontinued.

Table 8.1.: Possible Risks and Mitigation Strategies

Risk	Description	Severity	Likelihood	Mitigation
Single developer	The project was being developed by an individual, resulting in a greater impact to the project if the developer became temporarily unable to work, for example due to illness.	Medium	Medium	The agile methodology chosen for the project provides the project with the necessary flexibility to deal with this situation. Frequent meetings with the project supervisor to re-evaluate the feasiblilty of the project, taking into account any illness or similar circumstances, should enable the project to complete on time.

9. Evaluation

This section discusses the level of success achieved by this project. By comparing the requirements laid out in section 3, the issues discussed in section 4, the wider aims discussed in section 1.3, and the strategies discussed in section 8, with the resultant project, a detailed analysis of the project can be produced.

9.1. Functional Requirements

This section evaluates the success of the functional requirements laid out in section 3.1. Table 9.1 displays each requirement alongside a determination as to whether or not it has been successfully achieved.

Table 9.1.: Success of Functional Requirements

F#	Requirement	Success / Failure
F1	The system **must** allow the modelling of a network with at least three switches and four hosts, arranged in a tree topology, as shown in figure 3.1.	SUCCESS
F2	The system **must** allow switches and hosts to be configured in a tree topology.	SUCCESS
F3	The system **must** allow users to specify the spread of the network when using a tree topology (i.e. how many children a node switch node will have).	SUCCESS
F4	The system **must** allow users to specify the number of levels in the network when using a tree topology.	SUCCESS
F5	The system **must** allow users to specify which level in the network should be modelled as FPGA-based smart network switches.	SUCCESS

F6	The system **must** allow users to run a test to ensure there is an active connection between all hosts.	SUCCESS
F7	The system **must** allow users to run a test to display the bandwidth of the system.	SUCCESS
F8	The system **must** allow users to run a test to display the delay between the leaf nodes and the root when using a tree topology.	SUCCESS
F9	The system **should** allow users to specify the bandwidth of all links in the network.	MODERATE
F10	The system **should** allow users to specify the delay of all links in the network.	MODERATE
F11	The system **should** allow users to specify the chance of packet loss for all links in the network.	MODERATE
F12	The system **should** allow switches and hosts to be configured in a vertical linear topology.	SUCCESS
F13	The system **should** employ multiple levels of logging.	SUCCESS
F14	The system **should** conform to *Python*'s PEP8 code standard [44].	SUCCESS
F15	The system **should** be made open-source.	SUCCESS
F16	The system **could** allow users to configure the system to use a Poisson distribution for link delays.	MODERATE
F17	The system **could** allow users to configure the bandwidth of FPGA-based smart network switch nodes.	MODERATE
F18	The system **could** allow users to configure the delay of FPGA-based smart network switch nodes.	MODERATE
F19	The system **could** allow users to configure the chance of packet loss of FPGA-based smart network switch nodes.	MODERATE
F20	The system **could** allow users to display all node connections before running tests.	SUCCESS
F21	The system **won't** be configured for IPv6 connections.	SUCCESS

9.2. Non-Functional Requirements

This section evaluates the success of the non-functional requirements laid out in section 3.2. Table 9.2 displays each requirement alongside a determination as to whether or not it has been successfully achieved.

Table 9.2.: Success of Non-Functional Requirements

NF#	Requirement	Success / Failure
NF1	The system **must** be modular, to allow extensibility.	SUCCESS
NF2	The system **should** be scalable.	SUCCESS
NF3	All third-party libraries used by the system **should** be open-source.	SUCCESS
NF4	The system **should** be efficient, able to run all tests on relatively large virtual networks in small amounts of time.	SUCCESS
NF5	All code for the system **should** be well-documented and maintainable.	SUCCESS

9.3. Ethical, Social, Legal, and Professional Issues Review

9.3.1. Ethical Issues

The issue of privacy was successfully addressed in this project by not incorporating any user data. While the implemented system allows for options to be specified at the command line, none of these options could be used to identify an individual.

Should FPGA-based smart network switches become commercially manufactured, this project would recommend the exclusive use of fibre optics in order to increase the security of the data passing through the systems.

9.3.2. Social Issues

As mentioned above, since the project does not incorporate any user data, there is no possibility for discrimination or bias on this basis. By following the BCS Code of Conduct throughout the project, no other social issues have been encountered.

9.3.3. Legal Issues

The project achieved its goal of using exclusively open-source libraries in the system, and the source code for all of these libraries can be found on *GitHub*. This ensured that there would be no copyright issues arising from the project.

In addition, since no user data was required during the project, the risk of any breach of the *General Data Protection Regulation 2016* [52] or the *Data Protection Act 2018* [53] was significantly reduced, as there was no user data to misuse.

9.3.4. Professional Issues

By following the BCS Code of Conduct [49] throughout the project and engaging with the ideas and suggestions of the project supervisor, Dr Suhaib Fahmy, the project managed to avoid any notable professional issues. All research conducted throughout the project has been credited to its original founders, and the system developed by the project has been made open-source in order to share the knowledge gained from this project.

9.4. Project Management Evaluation

9.4.1. Software Development Methodology

As discussed in section 8.1, this project used an agile methodology in order to adapt to changes which occured during the project. The flexibility provided by this methodology proved invaluable, due to the changes made to the project aims over the course of the project. The methodology also encouraged the weekly meetings between the developer and the supervisor, which were crucial in keeping the project viable.

9.4.2. Project Schedule

The final project timetable is shown in figure 8.1. This changed from the initial project timetable shown in figure 8.2 due to changes in the focus of the project, and due to issues occurring which made the initial timetable no longer viable.

An example of these issues is that resources to assist with development of a *P4* based system which were due to be delivered by an external supplier early in term 2 were repeatedly delayed by the supplier, and ultimately were never delivered. Since *P4* is a relatively new language, the documentation and tutorials supplied with it still have much room for improvement. This imposed a short delay on the project timetable, as it was expected that the resources would be delivered.

Fortunately, the project was able to use its agile methodology to adapt to this and similar delays, and the final timetable was sufficient to complete the project aims.

9.4.3. Tools

A variety of different tools contributed to the success of the project, both in terms of streamlining development, and enhancing the project itself. In particular, *Mininet* was a significant component of the implemented system, and once the developer gained familiarity with this component, offered many additional benefits which improved the functionality of the system. *Git* was also a crucial management tool, allowing peace of mind during development that all code was saved and accessible, and allowing multiple components to be worked on simultaneously using *Git*'s 'branches'.

9.4.4. Risk Management

The strategies defined in section 8.4 proved worthwhile over the course of the project, as some of them needed to be actioned. While the primary development machine did not fail entirely, errors did occasionally occur which caused recently written files to be lost. Fortunately, since files were uploaded to *GitHub* regularly, the majority of these files were able to be restored. This saved a significant amount of time and effort during the project.

9.5. Author's Reflection

This section provides an evaluation of the project as a whole from the perspective of the developer. This takes into account the project aims, requirements, and motivations, and uses the questions set out below to assist with this evaluation.

What is the (technical) contribution of this project?

This project aimed to develop a system which could model FPGA-based smart network switches. This would help users understand the potential of these currently theoretical devices, and hopefully assist in their development. The system developed by the project provides an easy to use, lightweight, configurable way to model FPGA-based smart network switches, providing options to make the flow of data in the model more similar to that in real networks. The strong focus on maintaining good practice during development, as well as making the project open-source, should significantly improve the maintainability of the software, and should enable it to be used in future work where appropriate, such as that described in section 10.2.

Why should this contribution be considered relevant and important for the subject of your degree?

This project demonstrates a clear interconnection between the two departments which make up *Computer Systems Engineering*. While well used in Computer Science, the development of FPGAs is a significant field of Digital Electronic Engineering, as has been shown through the Warwick Engineering modules *ES3B2* [1] and *ES3F1* [7]. In addition, computer networks are a significant field of Computer Science. These two fields are both of great interest to the developer, which was a motivation for the initial project title and ideas. This project can demonstrate the value of the *Computer Systems Engineering* degree, since a project involving the areas examined by this project requires background knowledge in the areas, and that can only be acquired from this degree programme. It is hoped that this project will contribute to the development of FPGA-based smart network switches, which would fall well within the objectives of the developer's degree.

How can others make use of the work in this project?

Since the system developed for this project is open-source and available publicly on *GitHub*, anyone can make use of the code. This could either be by downloading and using the code as is, for which detailed instructions are clearly displayed on the webpage, or by using it as a basis for a related project.

The research conducted in this project identifies some applications of FPGA-based smart network switches, as well as a number of ways to help develop these systems. Should further research occur to develop FPGA-based smart network switches, this project could be used to provide design assistance, as well as advice on the potential pitfalls.

Why should this project be considered an achievement?

This project investigates an area which is currently entirely theoretical. While this is a common situation in academia, the constraints placed on the project, in particular the limited time frame, made finding relevant materials challenging. Regardless, relevant research has been conducted, and a modelling system has been produced which meets all of the functional and non-functional requirements defined for it. The project's agile software development methodology allowed for flexibility and change where appropriate, preventing the project from becoming stuck on a particular path if it became unfeasible. Finally, the project should be able to contribute positively to further research around the subject of the project.

What are the limitations of this project?

The primary limitation of this project is that the implemented aspect is a model, rather than a working system. Therefore, results from the model cannot be totally relied on, since they are only simulations and will never be exactly accurate compared to real data. In addition, unlike a raw instance of *Mininet*, the model does not allow users to specify completely customisable topologies. While many topologies will be a subset of a tree, the current implementation cannot produce topologies containing, for example, cycles.

10. Conclusion

10.1. Project Summary

This project investigated FPGA-based smart network switches, and implemented a system to model these in a network. The project conducted research into how these devices could be designed and constructed, and what sort of challenges may be encountered. The implemented system was developed with a strong focus on good development practice, and as a result is efficient, easy to read, and follows the stylistic guide for the language in which it is written. It has been made open-source, which encourages future work and follows the BCS Code of Conduct principle of "You make IT for everyone" [49]. While the project does not meet the original objectives or requirements of the specification, the new aims and requirements created as the project developed are sufficiently substantial to declare the project a success.

10.2. Future Work

The research and system produced by this project can be taken further in a variety of different ways. Some of these relate directly to the implemented system, while others are more abstract ways in which this work could be further developed.

10.2.1. Implementing an FPGA-based Smart Network Switch

The most obvious avenue for future work is to implement an FPGA-based smart network switch. A prototype for this type of system would require an FPGA integrated with networking hardware, such as one of the *NetFPGA* platforms described in section 2.4. It would also require a packet switching SDN language, such as *P4*, as described in section 2.3, and this language would need to be used to analyse packets up to the Application layer of the *TCP/IP network model*, as described in

section 2.1.2, and to determine whether these packets should be sent to the FPGA for processing, or simply routed through the network as usual. Furthermore, code would need to be written to perform the packet processing on the FPGA.

10.2.2. GUI for Implemented System

A GUI could make the implemented system easier to use, particularly for those without convenient access to the environment required to run the system. This could involve a drag-and-drop interface to design a network topology, which would then be converted to the *Python* format required for *Mininet*'s API. This could be deployed to a website, allowing people to save and share the network diagrams they create and the performance tests they run. This would naturally add the requirement of hosting the website.

10.2.3. API for Implemented System

An API for the implemented system would make it easier to integrate the application into further developed applications, such as a GUI, or an application to add futher functionality. This could be generated in multiple programming languages, and would make it easier for those interested in adding functionality to this system to do so.

10.2.4. Realistic Randomisation for Implemented System

In the real world, external factors have an impact on the passage of data through a network. This can have an unpredictable effect on the bandwidth, average latency, and average packet loss of a network. While the currently implemented system does allow for these attributes to be customised, they can currently only be set to a constant value or a value taken from a Poisson distribution. Introducing a greater element of randomness as an option for users into the system will bring it closer to a real network, and therefore make it more practically useful.

Bibliography

[1] "ES3B2 Digital Systems Design." https://warwick.ac.uk/fac/sci/eng/eso/modules/year3/es3b2, Sep 2018. Accessed: 2019-04-25.

[2] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*. Pearson, 5 ed., 2011.

[3] G. Martin, "Computer Networks: Introduction." University Lecture, 2017.

[4] S. Wilkins, "OSI and TCP/IP Model Layers," Nov 2011. Accessed: 2018-11-24.

[5] "CS241 Operating Systems and Computer Networks." https://warwick.ac.uk/fac/sci/dcs/teaching/modules/cs241/, Mar 2017. Accessed: 2019-04-24.

[6] "Warwick Student Cinema." https://warwick.film/, Apr 2019. Accessed: 2019-04-25.

[7] "ES3F1 High Performance Embedded Systems Design." https://warwick.ac.uk/services/aro/dar/quality/modules/undergraduate/es/es3f1, Mar 2017. Accessed: 2019-04-25.

[8] "Amazon EC2." https://aws.amazon.com/ec2/, Apr 2019. Accessed: 2019-04-25.

[9] Amazon Web Services, "Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta," Aug 2006. Accessed: 2019-04-25.

[10] "Microsoft Azure Cloud Computing Platform and Services." http://azure.microsoft.com/en-gb/, Apr 2019. Accessed: 2019-04-25.

[11] "Cloud Computing Services | Google Cloud." https://cloud.google.com/, Apr 2019. Accessed: 2019-04-25.

[12] "AWS Global Infrastructure Regions and AZs." https://aws.amazon.com/about-aws/global-infrastructure/regions_az/#Availability_Zones, Apr 2019. Accessed: 2019-04-25.

[13] Stack Overflow, "Stack Overflow Developer Survey Results 2019," Apr 2019. Accessed: 2019-04-26.

[14] "Intel Celeron G4900 Dual Core Coffee Lake Desktop CPU/Processor | SCAN UK." https://www.scan.co.uk/products/intel-celeron-g4900-s-1151-coffee-lake-dual-core-2-thread-31ghz-2 Apr 2018. Accessed: 2019-04-26.

[15] Intel, *Celeron G4900 Processor*, Apr 2018.

[16] "Intel 14 Core Xeon Gold 6132 Server/Workstation CPU/Processor | SCAN UK." https://www.scan.co.uk/products/intel-xeon-gold-6132-s-3647-skylake-sp-14-core-28-threads-26ghz-3 Jul 2017. Accessed: 2019-04-26.

[17] Intel, *Xeon Gold 6132 Processor*, Jul 2017.

[18] "Xilinx - Adaptable. Intelligent.." https://www.xilinx.com/, May 2018. Accessed: 2019-04-27.

[19] "Intel FPGAs and Programmable Devices - Intel FPGA." https://www.intel.com/content/www/us/en/products/programmable.html, Jul 2018. Accessed: 2019-04-27.

[20] "Cisco." https://www.cisco.com/, Jul 2018. Accessed: 2019-04-27.

[21] "Netgear." https://www.netgear.com/, Jan 2019. Accessed: 2019-04-27.

[22] "Electric Imp Secure IoT Connectivity Platform." https://www.electricimp.com/, May 2012. Accessed: 2019-04-27.

[23] "Asset Tracking - Electric Imp." https://www.electricimp.com/solutions/asset-tracking/, Jun 2018. Accessed: 2019-04-27.

[24] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," in *2014*

IEEE Network Operations and Management Symposium (NOMS), pp. 1–9, May 2014.

[25] "Mininet: An Instant Virtual Network on your Laptop (or other PC)." `http://mininet.org/`, Jul 2005. Accessed: 2019-04-27.

[26] "NetFPGA." `https://netfpga.org/`, Jul 2010. Accessed: 2019-04-27.

[27] I. O. for Standardization, "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model," Tech. Rep. 7498-1:1994, Nov. 1994.

[28] G. Martin, "Computer Networks: Physical Layer (Signal Transmission)." University Lecture, 2017.

[29] Universal Networks, "Copper vs Fiber," Jul 2011. Accessed: 2018-11-25.

[30] Multicom, "Copper vs. Fiber - Which to Choose?," May 2016. Accessed: 2018-11-25.

[31] "NetFPGA - NetFPGA 1G." `https://netfpga.org/site/#/systems/4netfpga-1g/details/`, Jul 2010. Accessed: 2018-10-09.

[32] "P4 Language Consortium." `https://p4.org/`, Sep 2014. Accessed: 2018-10-08.

[33] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Tech. Rep. 1883, Dec. 1995.

[34] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Tech. Rep. 8200, July 2017.

[35] "IANA IPv4 Address Space Registry." `https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml`, Sep 2018. Accessed: 2018-11-25.

[36] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833.

[37] "Wireshark - Go Deep." `https://www.wireshark.org/`, Jun 2008. Accessed: 2019-04-27.

[38] "Travis CI - Test and Deploy with Confidence." https://travis-ci.com/, May 2018. Accessed: 2019-04-27.

[39] "GitHub." https://github.com/, Sep 2009. Accessed: 2019-04-27.

[40] J. Luo, J. Pettit, M. Casado, J. Lockwood, and N. McKeown, "Prototyping fast, simple, secure switches for etha," in *15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*, pp. 73–82, Aug 2007.

[41] Xilinx, *Virtex-7 T and XT FPGAs*, Mar 2019. v1.28.

[42] R. M. Sydeek, "MoSCoW Method Explained," *Feedough*, Dec 2017. Accessed: 2019-04-28.

[43] S. Jarvis, "Software Engineering: Requirements Analysis." University Lecture, Jan 2018.

[44] G. van Rossum, V. Warsaw, and N. Coghlan, "PEP 8 – Style Guide for Python Code," *Python Developer's Guide*, Jul 2001. Accessed: 2019-04-28.

[45] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," Tech. Rep. 4987, Aug. 2007.

[46] N. Freed, "Behavior of and Requirements for Internet Firewalls," Tech. Rep. 2979, Oct. 2000.

[47] B. Lantz, "Mininet 2.2.2 Release Notes," *Mininet Wiki*, Mar 2017. Accessed: 2019-04-28.

[48] A. Conrad, "Extended Security Maintenance for Ubuntu 14.04 (Trusty Tahr) begins April 25 2019," *'ubuntu-annouce' Mailing List*, Mar 2019. Accessed: 2019-04-28.

[49] British Computer Society, "Code of Conduct for BCS Members," Jan 2019. Accessed: 2019-04-27.

[50] "Apache License, Version 2.0." https://www.apache.org/licenses/LICENSE-2.0.html, Jan 2004. Accessed: 2019-04-27.

[51] "GNU General Public License v3.0." https://www.gnu.org/licenses/gpl-3.0.en.html, Jun 2007. Accessed: 2019-04-27.

[52] Council of European Union, "General data protection regulation 2016." http://data.europa.eu/eli/reg/2016/679/oj, Apr 2016.

[53] The Government of the United Kingdom, "Data protection act 2018." http://www.legislation.gov.uk/id?title=Data+Protection+Act+2018, May 2018.

[54] "Mirror of Stanford OpenFlow 1.0 reference switch/controller." https://github.com/mininet/openflow, Feb 2008. Accessed: 2019-05-04.

[55] "Python." https://www.python.org/, Jan 1994. Accessed: 2019-04-28.

[56] "git." https://git-scm.com/, May 2012. Accessed: 2019-04-28.

[57] "Python Setuptools." https://www.docker.com/https://github.com/pypa/setuptools, Aug 2013. Accessed: 2019-04-28.

[58] "Click | The Pallets Projects." https://palletsprojects.com/p/click/, May 2014. Accessed: 2019-04-28.

[59] "Daring Fireball: Markdown." https://daringfireball.net/projects/markdown/, Dec 2004. Accessed: 2019-05-04.

[60] "logging — Logging facility for Python." https://docs.python.org/2/library/logging.html, Mar 2003. Accessed: 2019-04-28.

[61] "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool." https://iperf.fr/, Aug 2007. Accessed: 2019-05-04.

[62] "The Official YAML Web Site." https://yaml.org/, May 2001. Accessed: 2019-05-04.

[63] "unittest - Unit testing framework." https://docs.python.org/2/library/unittest.html, Feb 2002. Accessed: 2019-04-28.

[64] "pip - PyPI." https://pypi.org/project/pip/, Apr 2011. Accessed: 2019-04-30.

[65] "Atom." https://atom.io/, Jun 2015. Accessed: 2018-10-10.

[66] “LaTeX - A document preparation system.” https://www.latex-project.org/, 1983. Accessed: 2018-10-10.

[67] “Resilio Sync Home.” https://www.resilio.com/individuals-sync/, Mar 2015. Accessed: 2019-04-28.

[68] “Trello.” https://trello.com/, Sep 2011. Accessed: 2019-04-28.

[69] “GNU Bash.” https://www.gnu.org/software/bash/, Jun 1989. Accessed: 2019-04-28.

[70] “Enterprise Application Container Platform | Docker.” https://www.docker.com/, Mar 2013. Accessed: 2019-04-28.

[71] “PyCharm: the Python IDE for Professional Developers by JetBrains.” https://www.jetbrains.com/pycharm/, Feb 2010. Accessed: 2019-04-28.

[72] “Pylint - code analysis for Python.” https://www.pylint.org/, Aug 2013. Accessed: 2019-04-28.

[73] “Welcome to tmux!.” https://github.com/tmux/tmux/wiki, Nov 2007. Accessed: 2019-04-28.

[74] “The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu.” https://www.ubuntu.com/, Oct 2004. Accessed: 2019-04-28.

[75] “Vim - the ubiquitous text editor.” https://www.vim.org/, Nov 1991. Accessed: 2019-04-28.

[76] “Oracle VM VirtualBox.” https://www.virtualbox.org/, Jan 2007. Accessed: 2019-04-28.

[77] “GitLab.” https://gitlab.com/, Oct 2011. Accessed: 2019-05-03.

Appendix A.

Presentation

Modelling Smart FPGA Switches in the Network

Benji Levine

Supervised by Suhaib Fahmy

What are FPGAs?

Field-Programmable Gate Array

Can be seen as alternative to ASIC

- Flexible Logic ----------
- Flexible routing ----------
- Flexible IO ----------
- Embedded hard modules ----------

Application-Specific Integrated Circuit

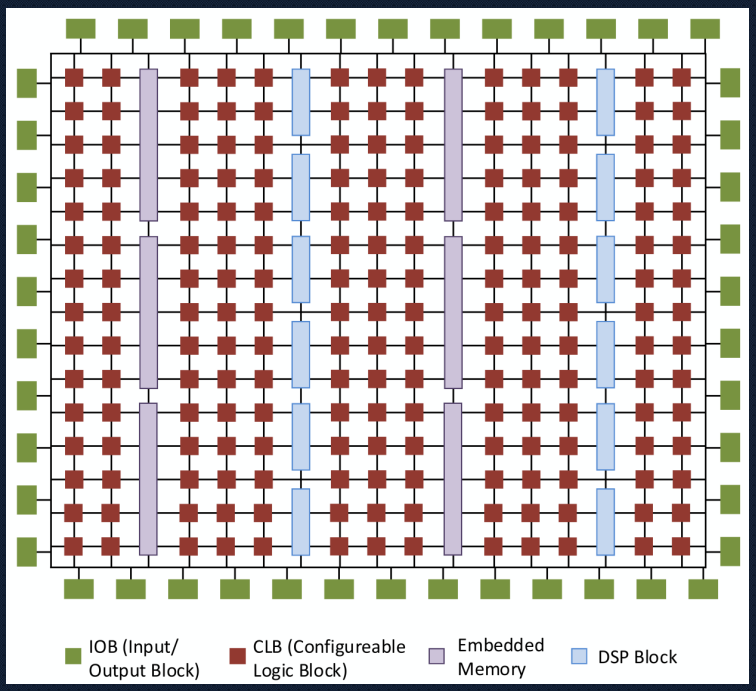
- LUTs
- Large grid of wires and switch boxes
- Support for 10G Ethernet, SATA, PCIe
- Block memory, DSP blocks

What are FPGAs?

Field-Programmable Gate Array

Seen as alternative to ASIC

- Flexible Logic
- Flexible routing
- Flexible IO
- Embedded hard modules



(1)

Networking Concepts

- OSI Network Model
- TCP / IP network stack

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

Networking Concepts

- OSI Network Model
- TCP / IP network stack
- Software Defined Networking

- Application
- Transport
- Internet
- Link
- Control plane
- Data plane
- P4
- OpenFlow (part of Mininet)

Why Smart FPGA Switches?

- Cloud computing

- High latencies
- Large data centres
- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud Platform
- Apache Hadoop

Why Smart FPGA Switches?

- Cloud computing
- Existing solutions
- "Edge" / "gateway" nodes
- Mainframes
- NetFPGA
- Software Defined Networking (SDN)

Why Smart FPGA Switches?

- Cloud computing
- Existing solutions
- Drawbacks
- Similar to data centres
- Latency
- Security
- Scalability

Why Smart FPGA Switches?

- Cloud computing
- Existing solutions
- Drawbacks
- Solution

- Do not divert data from existing path
- Appropriate device for computation

Why Smart FPGA Switches?

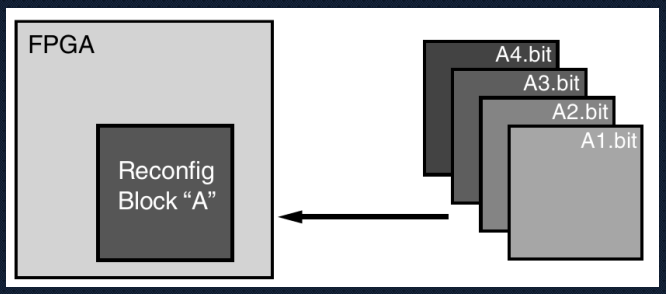
- Cloud computing
- Existing solutions
- Drawbacks
- Solution
- FPGAs

- Custom compute architectures – very low latency
- Partial reconfiguration
- Specific hardware for each algorithm

Why Smart FPGA Switches?

- Cloud computing
- Existing solutions
- Drawbacks
- Solution
- FPGAs

- Partial reconfiguration
- Specific hardware for each algorithm



Research and Implementation

- Mininet

- "Emulator for rapid prototyping of Software Defined Networks"
- Open source
- Python API
- Scalable
- Dependency chain

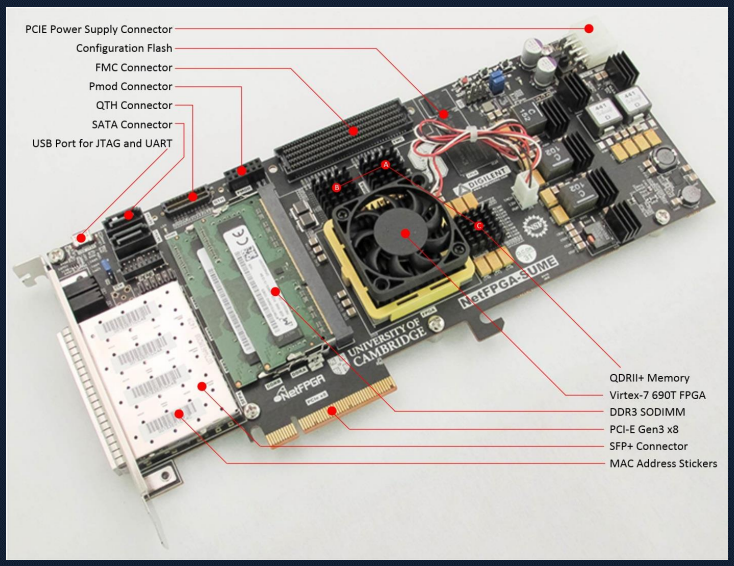
Research and Implementation

- Mininet
- NetFPGA

- PCI Express interface
- Xilinx Virtex / Kintex
- 4 x 1Gbps – 10Gbps Ethernet
- Open source software
- 1G, 10G, SUME, CML
- Standalone or integrated

Research and Implementation

- Mininet
- NetFPGA



Research and Implementation

- Mininet
- NetFPGA
- Software Development
- Python
- Open source (GitHub)
- setuptools
- click
- logging
- git submodules

(6), (7), (8), (9)

Project Management

- Agile methodology ----------
- Kanban Board ----------
- Version control ----------
- Weekly meetings with supervisor ----------
- Issues resolved quickly ----------
- Adaptive to change
- Trello
- git
- Comparable to “scrums”
- Access to tools, licensing

(10), (11)

Applications

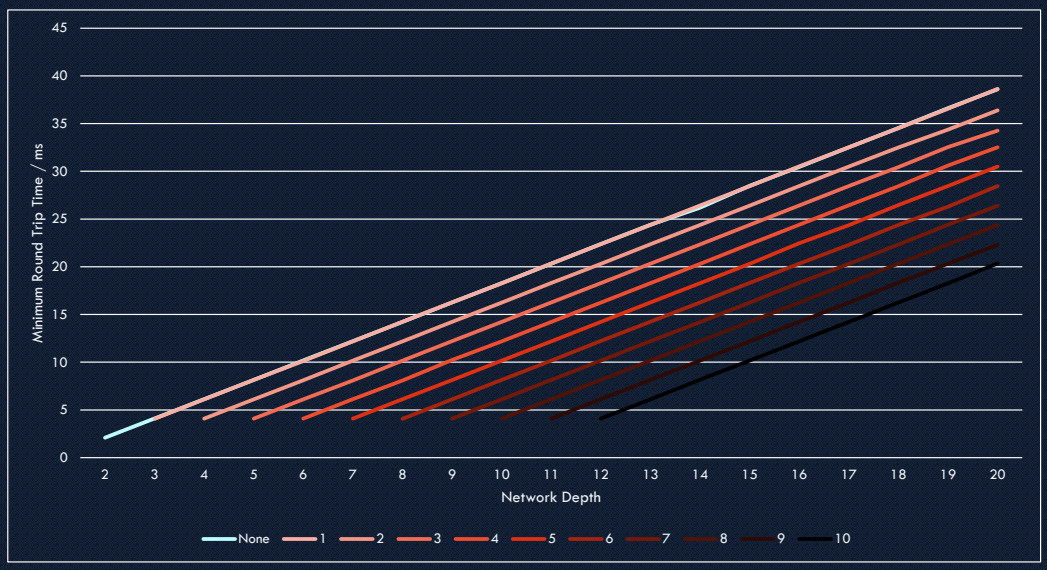
- Internet of Things (IoT)
- Education
- Healthcare
- Military
- Finance

Model Results

- Operating Parameters:
- Latency: 1ms
- FPGA latency: 2ms
- Spread: 1 (vertical linear topology)

Model Results

- Operating Parameters:
- Latency: 1ms
- FPGA latency: 2ms
- Spread: 1 (vertical linear topology)



Demo

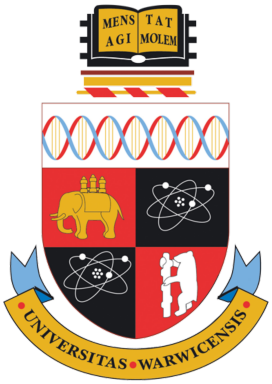
Questions

References

1. ES3F1 Lectures, Suhaib Fahmy
2. xilinx.com
3. mininet.org
4. netfpga.org
5. reference.digilentinc.com
6. python.org
7. github.com
8. https://github.com/pypa/setuptools
9. https://palletsprojects.com/p/click/p4.org
10. trello.com
11. git-scm.com

Appendix B.

Progress Report



Network Switch Design on Field-Programmable Gate Arrays

CS351 Computer Systems Engineering Project

Progress Report

Benji Levine
1611586

Superviser: Dr Suhaib Fahmy

Department of Computer Science
University of Warwick

2018-19

Contents

1 Glossary

The following acronyms are used throughout the specification.

- **FPGA**: Field-Programmable Gate Array
- **MTU**: Maximum Transmission Unit
- **OSI model**: Open Systems Interconnection reference model
- **IP**: Internet Protocol
- **TCP**: Transmission Control Protocol
- **ARPANET**: Advanced Research Projects Agency Network
- **DARPA**: Defense Advanced Research Projects Agency (USA)
- **UTP**: Unshielded Twisted Pair
- **SDN**: Software-defined Networking

2 Introduction

3 Research

3.1 Networking

Since core networking concepts form much of the background for this project, a reasonable amount of time was dedicated to researching these. These are detailed below.

3.1.1 Network Packets

A 'packet' in networking refers to a unit of data. This term only applies to certain units of data, since data at different stages in network models (see section 3.1.2) takes different names. Ultimately, a packet consists of two parts: a set of 'headers', and a 'payload'. The headers contain metadata about the packet, such as where it was sent from, where it was sent to, or a checksum of the packet used for error checking. A packet tends to have multiple headers, each stacked on top of one another. Each header is used by a different layer of the OSI model (see section 3.1.2). When a packet is being constructed, each layer prepends its header to the packet payload before sending it down to the next layer. The payload carries the actual packet data. In most cases, many packets will be required to send a message, since packets have a fixed maximum size. This limit in size is known as the Maximum Transmission Unit (MTU), and for an Ethernet packet, this is 1518 bytes including headers.

3.1.2 Network Models

The two most commonly used models are the Open Systems Interconnection Reference model (commonly referred to as the OSI model), and the TCP/IP model. Both of these models consist of layers and are intended to allow the detail of the layers beneath the layer you are working at to be abstracted away. This means that, for example, a web server will not need to know any detail about the networking hardware of the server it is running on.

The OSI model was created in 1994 [1] and consists of seven layers, shown in table 1. It has suffered from a few key criticisms, such as the fact that some layers have very little use in most applications, and the complexity of the model can result in large and slow systems. In addition, when the OSI standard was approved, the competing TCP/IP model was being used by most research universities, and so did not receive as much support as might have been expected

The TCP/IP model was developed for ARPANET, a predecessor of the modern internet funded by DARPA. TCP was developed to serve as a protocol for communication within ARPANET from different underlying hardware (such as between satellite packet networks and ground-based radio packet networks). It initially covered both the Internet and Transport layers of what would become the TCP/IP model, however it was suggested to separate the protocol, which resulted in the creation of the Internet Protocol (IP). These protocols went through a number of revisions before arriving at TCP/IP v4, which is still in use today.

3.1.3 Packet Switching

Layers of a model can offer either a connection-oriented or a connectionless service to the layers above. A connection-oriented service involves establishing a connection, using the connection, and then releasing the connection, similar to a landline telephone system. This requires all the parameters for the connection to be specified in advance, which may involve a negotiation between the sender and the receiver.

A connectionless service involves providing every packet with the full source and destination address for the data. Each packet is then sent independently such that each node the packet passes through chooses the next node to send it to. The current node makes this decision by choosing the node that is closest to the packet's final destination from the nodes it is aware of. There are two main switching methods for a connectionless service: "store-and-forward" and "cut-through". "Store-and-forward" involves each intermediate node receiving a message in full before forwarding packets to the next node, while "cut-through" does not require an intermediate node to have the entire message before it starts forwarding packets [4].

3.1.4 Physical Media

There are different categories of physical media that can be used to transmit data. The two most common types are UTP and optical fibre. UTP consists of two copper wires coated in plastic and twisted together. The wires are twisted partially to keep them together, but also to help cancel out electromagnetic interference between the wires [5]. UTP is currently the dominant media, primarily since it has been commercially available for so long.

Optical fibre beats out copper in most categories, only really falling behind when it comes to cost, the price gap between UTP and optical fibre is narrowing [6]. Optical fibre has a singicantly higher theoretical maximum bandwidth, is much more

7	Application	The layer where user programs live. In addition to user specificed programs, there are common applications such as FTP, Telnet (virtual terminal), or e-mail, which operate at the application layer.
6	Presentation	Provides a set of data transformation services including character conversion, data compression, and data encryption.
5	Session	Designed to manage an entire conversation, consisting of a number of dialogue units which can be suspended and restarted through synchronisation points. However, client/server communication is based on an asymmetric model, in which one or more client processes can request resources from a server process.
4	Transport	Provides datagrams for both connection-oriented and connectionless protocols. In a connection-oriented service, the transport layer negotiates a suitable quality of service for the given network and application. This typically includes throughput, transit delay, and error rate among other paramters.
3	Network	Highest layer within the communication subnet, deals with control issues like routing, congestion control, and error recovery.
2	Data Link	Provides frames which give addressing, error control, and sequencing, necessary to provide a reliable service.
1	Physical	Responsible for transporting bits of information. Bandwidth, signal levels, and signal coding methods are specified at this level.

Table 1: The Open Systems Interconnection Reference model [2]

4	Application	Equivalent to the Session, Presentation, and Application layers of the OSI model. The Application layer is used to handle all process to process communication, and includes session establishment, compression, and encryption.
3	Transport	Equivalent to the Transport layer of the OSI model. The Transport layer includes message segmentation, session multiplexing, and message reordering.
2	Internet	Equivalent to the Network layer of the OSI model. The Internet layer includes traffic routing, traffic control, and logical addressing.
1	Link	Equivalent to the Data Link and Physical layers of the OSI model. Provides physical network functions like signal levels, signal coding methods and, error detection.

Table 2: The TCP/IP model [3]

durable, and can be used across much longer distances with less signal attenuation [7]. UTP and optical fibre do use different connectors and so it is difficult to take full advantage of the benefits of optical fibre when using the standard ethernet con-

nectors found in most consumer hardware. All of the hardware platforms offered by NetFPGA aside from the NetFPGA 1G [8] are designed for optical fibre.

3.1.5 Packet Headers

As mntioned in section 3.1.1, 'headers' are used to store metadata about network packets. Each layer of the TCP/IP model prepends an additional header to the data, resulting in a payload with multiple headers attached as shown in figure 1. Most protocols have a set header format, making it much easier to extract data from them. As an example, the header for an IPv4 packet is shown in figure 2. Each row of the header represents 32 bits of data. The *Options* field of the header has a variable length, so the *IHL* field is used to record the length of the IP header. The *Total Length* field represents the total size of the packet in 32 bit words, including both the header and the payload. This information can be used to extract the data from the headers, and to remove the appropriate number of bits in order to read the payload itself. For this project the P4 language [9] will be used to extract data from packet headers, as will be discussed further in section 3.2. The *Version* field of the IP header refers to which version of the protocol is used by the packet. The most commonly used version is IPv4, however a newer version, IPv6, has been standardised [10] [11]. IPv6 was primarily intended to address the issue of addressing in IPv4. Every device on a network using IP requires a unique IP address. When data is being sent to a particular device, its IP address is used at the Internet layer of the TCP/IP model to locate the device. An IPv4 address is 32 bits long, meaning there are a total of 4,294,967,296 IPv4 addresses for a given network. While this is more than enough for any local network, a very large network (such as the internet) does run the risk of hitting this limit. In fact, the internet has now had all of the available addresses allocated [12]. Comparatively, an IPv6 address is 128 bits long, providing a total of 3.4×10^{38} IPv6 addresses. Since IPv6 is not yet widely implemented, it is being considered beyond the scope of this project.

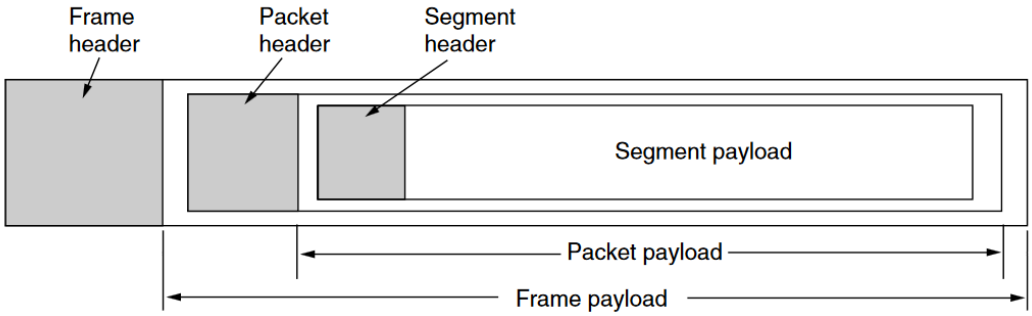


Figure 1: Nesting of segments, packets, and frames. [4]

3.1.6 Software-Defined Networking

The primary benefit of SDN is that it gives the programmer more direct and clear control over how routing decisions are made. SDN aims to separate the network control, known as the control plane, from the forwarding proces, known as the data plane [13]. The control plane makes decisions about packets which are either destined to or originally from the current host, while the data plane makes decisions

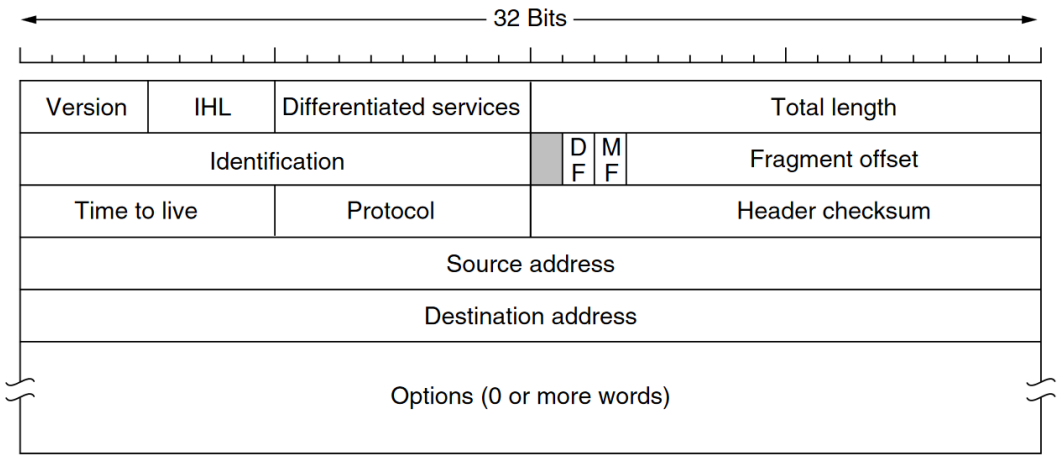


Figure 2: The IPv4 (Internet Protocol) header. [4]

about packets which are going through the host. The control plane also maintains the routing table, which is used by both planes to make routing decisions.

3.2 P4

P4 is a language for programming the data plane of network devices [9]. The specification for $P4_{14}$ (the first version of P4) was released in September 2014. This went through a few minor revisions before the specification for $P4_{16}$ (the current version) was released. This project will use $P4_{16}$.

4 Future Steps

The next steps for this project are to begin the implementation of a packet analyser using $P4_{16}$ on a NetFPGA. This will be initially completed using a virtual platform instead of on the NetFPGA directly, since access to the hardware will be limited before the start of Term 2. An updated project timetable is shown in figure 3. This has not changed significantly since the specification document was written. The specification document, along with the original project timetable, can be found in appendix **??**.

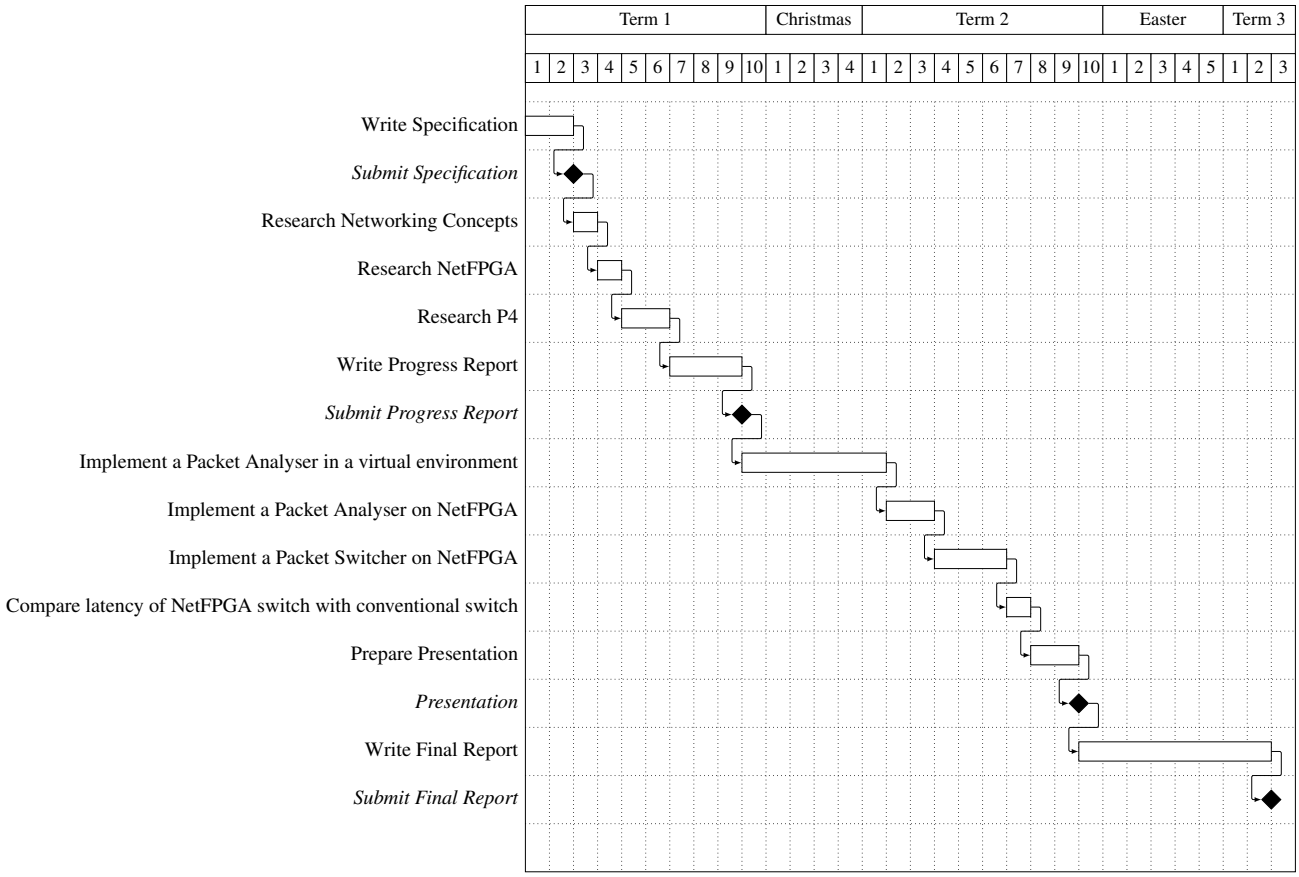


Figure 3: Gantt Chart of Project Timetable

5 Conclusion

References

[1] I. O. for Standardization, "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model," Tech. Rep. 7498-1:1994, Nov. 1994.

[2] G. Martin, "Computer Networks: Introduction." University Lecture, 2017.

[3] S. Wilkins, "OSI and TCP/IP Model Layers," Nov 2011. Accessed: 2018-11-24.

[4] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*. Pearson, 5 ed., 2011.

[5] G. Martin, "Computer Networks: Physical Layer (Signal Transmission)." University Lecture, 2017.

[6] Universal Networks, "Copper vs Fiber," Jul 2011. Accessed: 2018-11-25.

[7] Multicom, "Copper vs. Fiber - Which to Choose?," May 2016. Accessed: 2018-11-25.

[8] "NetFPGA - NetFPGA 1G." `https://netfpga.org/site/#/systems/4netfpga-1g/details/`, Jul 2010. Accessed: 2018-10-09.

[9] "P4 Language Consortium." `https://p4.org/`, Sep 2014. Accessed: 2018-10-08.

[10] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Tech. Rep. 1883, Dec. 1995.

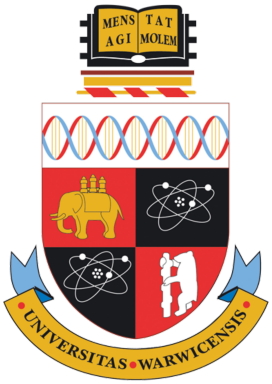
[11] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Tech. Rep. 8200, July 2017.

[12] "IANA IPv4 Address Space Registry." `https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml`, Sep 2018. Accessed: 2018-11-25.

[13] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833.

Appendix C.

Initial Specification



Network Switch Design on Field-Programmable Gate Arrays

CS351 Computer Systems Engineering Project

Specification

Benji Levine
1611586

Superviser: Dr Suhaib Fahmy

Department of Computer Science
University of Warwick

2018-19

Contents

1 Glossary

The following acronyms are used throughout the specification.

- **FPGA**: Field-Programmable Gate Array
- **ASIC**: Application-Specific Integrated Circuit
- **LUT**: Look-Up Table
- **MAC Address**: Media Access Control Address
- **OSI model**: Open Systems Interconnection model
- **TCP SYN flooding**: Media Access Control Address
- **PCIe**: Peripheral Component Interconnect Express
- **SRAM**: Standard Random-Access Memory
- **DRAM**: Dynamic Random-Access Memory

2 Problem Statement

Networking is an important area of Computing, and as network sizes and average complexity of projects have increased, the need for network connections with very low latency has also increased. For systems such as Automated Trading Systems [1], a small delay in completing a trade can result in a significant difference in profit. The faster trades can be processed, the greater the potential for increased revenue. Most conventional network switches will have a latency in the range of "hundreds of nanoseconds to tens of microseconds" [2]. In networks with a large amount of data flowing through a network switch every second, reducing this latency could result in a significant improvement to the speed of the network.

Using FPGAs to conduct switching logic can result in a switch with a very high throughput, since this switching logic will be implemented in hardware. The LUTs on an FPGA can be used to store the MAC addresses of the devices connected to the network switch. This allows an FPGA based switch to theoretically determine the appropriate port to direct a packet to, in a single clock cycle. Unlike ASICs, FPGAs can be reconfigured on the fly, so MAC address tables can be dynamically updated while still being implemented in hardware. For all of these reasons, FPGA-based network switches have the potential to create networks with a much higher throughput than traditional networking hardware, and investigation into FPGA-based switches may reveal further opportunity for other FPGA-based networking hardware, such as routers or firewalls.

3 Objectives

This project will consist of four stages: Research, Implementation, Testing, and Evaluation, and the objectives have been separated as such.

3.1 Research

- Research networking concepts (such as the OSI network model)
- Research methods of measuring latency of a network switch
- Research the NetFPGA platform [3]
- Research the packet switching language P4 [4]

3.2 Implementation

- Implement a packet analyser on a NetFPGA
- Implement a packet switcher on a NetFPGA

3.3 Testing

- Test throughput and latency of switching packets using the NetFPGA packet switcher
- Test throughput and latency of switching packets using a conventional network switch

3.4 Evaluation

- Compare performance of the NetFPGA packet switcher to a conventional network switch
- Write up performance comparison

4 Requirements

In order to measure the success of this project and to clearly define the work to be done, the following requirements have been written to support the objectives above. Since the research stage and initial implementation stage will be used to confirm the direction for the remainder project, these requirements are subject to change. They indicate a feasible path through the project which should return interesting valuable results. They have been written using the MoSCoW method [5], which is a prioritisation technique that helps write clearly defined requirements.

4.1 Functional Requirements

These requirements define the technical detail of the system produced over the course of the project, as well as the data to be analysed for the final report.

F1: The system **must** be able to analyse packets at layer 2 of the OSI network model

F2: The system **must** be able to send packets to the correct port based on MAC addresses

F3: The system **must** be able to store MAC address tables

F4: The system **must** be able to dynamically update MAC address tables

F5: The average latency of the system **must** be measured

F6: The throughput of the system **must** be measured

F7: The system **should** be able to analyse packets at layer 3 of the OSI network model

F8: The system **could** be able to analyse packets at layer 7 of the OSI network model

F9: The system **could** be able to detect basic network attacks (such as TCP SYN Flooding [6])

F10: The system **could** implement features of a basic firewall (such as packet filtering [7])

4.2 Non-Functional Requirements

These requirements describe the general operation of the system produced, as well as how the project should be developed

NF1: The system **should** be scalable

NF2: The system **should** be efficient

NF3: All code for the system **should** be well-documented and maintainable

5 Project Management

5.1 Methods

This project will use an Agile methodology so that it can adapt to changes which arise during the project. Since the research and implementation stages of the project will contribute to confirming its direction, this flexibilty is important. In addition, git [8] will be used to track changes in both written documents and code developed, such as P4 or Verilog code. Repositories will be set up using git for the different areas of the project, and these repositories will be stored primarily on an online GitHub [9] server that will be backed up regularly. The Gantt chart in Figure 1 has been constructed to show an outline of the project timetable, and is intentionally flexible for the changes that will take place.

The NetFPGA [3] platform is intended to be used for this project. It is an "open source hardware and software platform designed for research and teaching" [10]. Of the four hardware plaforms available, the NetFPGA 1G [11] is the most suitable for this project, since its hardware is most compatible with general purpose consumer hardware. The other three platforms could be used to create faster circuits than the NetFPGA 1G, however these may require additional hardware, which is not currently available for this project. This platform is based around a *Xilinx Virtex-II Pro 50* [12] which contains 53,136 logic cells and 4kb block RAM. In addition, the NetFPGA 1G contains four Gigabit Ethernet networking ports, 4.5MB of SRAM, and 64MB of DDR2 DRAM. It has a standard PCI form factor, and so is compatible with most consumer motherboards.

5.2 Timetable

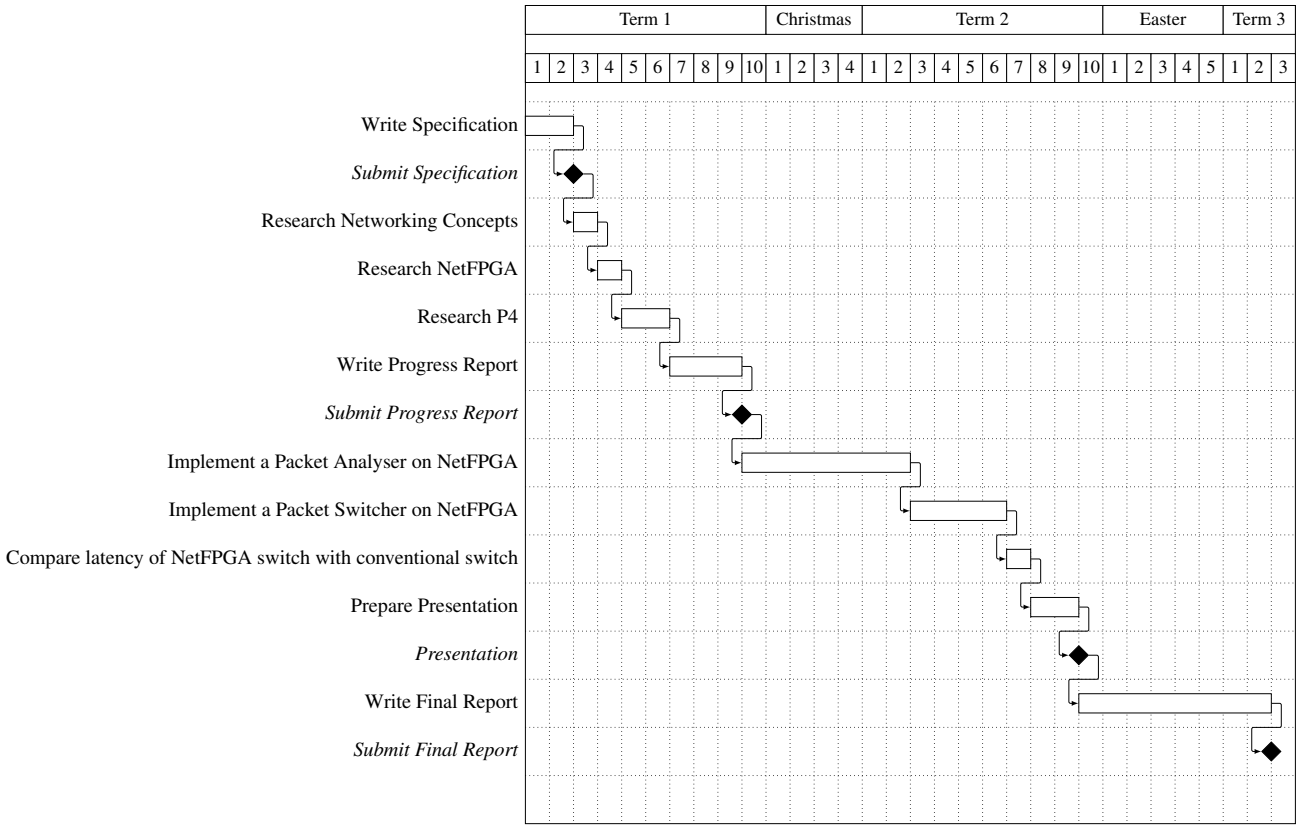


Figure 1: Gantt Chart of Project Timetable

6 Resources

This project will use a number of different resources, including hardware, software, and languages. These are listed below.

- git [8]
  - will be used for version control of all code and documents
- GitHub [9]
  - will be used as an external server to store code and documents, as well as an interface for git
- NetFPGA [3]
  - will be used as the platform on which a switch is developed
- P4 [4]
  - will be used to implement packet switching on the NetFPGA
- LaTeX [13]
  - will be used to write and format all documents
- Atom [14]
  - will be used as an editor to write code

References

[1] J. Folger, "The pros and cons of automated trading systems," *Investopedia*, Aug 2018. Accessed: 2018-10-12.

[2] Plexxi, "Behavior of and Requirements for Internet Firewalls," tech. rep., Jan. 2016.

[3] "Netfpga." `https://netfpga.org/`, Jul 2010. Accessed: 2018-10-08.

[4] "P4 language consortium." `https://p4.org/`, Sep 2014. Accessed: 2018-10-08.

[5] R. M. Sydeek, "Moscow method explained," *Feedough*, Dec 2017. Accessed: 2018-10-12.

[6] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," Tech. Rep. 4987, Aug. 2007.

[7] N. Freed, "Behavior of and Requirements for Internet Firewalls," Tech. Rep. 2979, Oct. 2000.

[8] "git." `https://git-scm.com/`, May 2012. Accessed: 2018-10-08.

[9] "Github." `https://github.com/`, Sep 2009. Accessed: 2018-10-09.

[10] “Netfpga - about.” https://netfpga.org/site/#/about/, Jul 2010. Accessed: 2018-10-09.

[11] “Netfpga - netfpga 1g.” https://netfpga.org/site/#/systems/4netfpga-1g/details/, Jul 2010. Accessed: 2018-10-09.

[12] Xilinx, *Virtex-II Pro and Virtex-II Pro X Platform FPGAs*, 5 2011. v5.0.

[13] “Latex - a document preparation system.” https://www.latex-project.org/, 1983. Accessed: 2018-10-10.

[14] “Atom.” https://atom.io/, Jun 2015. Accessed: 2018-10-10.