THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Investigation into Building an OpenFlow-enabled BNG

Craig Osborne

This report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computing and Mathematical Sciences with Honours (BCMS(Hons)) at The University of Waikato.

COMP520-14C (HAM)

Abstract

In carrier-grade Internet Service Provider (ISP) networks the Broadband Network Gateway (BNG) is a specialised network server which sits at the core of an ISP network. The responsibilities of the BNG include the facilitation and aggregation of active user sessions from the ISP's access network, and providing Internet Protocol (IP) connectivity through the ISP backbone network to the Internet. In a typical ISP carrier network, the BNG is one of the most expensive appliances and carries a high degree of responsibility. Due to the cost, it is common for there be only a single centralised BNG, which presents a singular point of failure (SPOF) for all ISP subscriber access to the Internet. This project aims to investigate whether the centralised BNG model is able to be redesigned using a new approach to computer networking called Software Defined Networking (SDN). Using SDN and the OpenFlow protocol will allow for the separation of the traffic forwarding logic of a switch from the data plane, enabling that logic to be moved into an application running in software. This results in a much more software oriented approach to network design and the control and configuration of switches. This project aims to investigate the removal of singular dependence on the centralised BNG model by distributing the functionality of a BNG among a network of commodity switches. There is a strong focus on implementing support for a number of important session establishment protocols and how these can be used to make necessary dynamic traffic forwarding decisions, as well as the investigation of a carrier-grade distributed software controller.

Acknowledgements

Firstly, I would like to thank my project supervisor Dr. Richard Nelson for his support and guidance throughout the course of this project. Additionally, I would like to thank Brad Cowie and Christopher Lorier from the WAND group for their advice and valuable input into my project as it has progressed. I would also like to thank Scott Raynel, Chris Browning and the rest of the team from Lightwire for their support and valuable insight from the onset of my project and throughout. Finally, I would like to thank Adam Coxhead for his advice and input in structuring my project and associated deliverables.

Additionally, I would like to thank Shane Alcock and Brendon Jones from the WAND group for their proof-reading and editorial input in creating this report.

Contents

List of Figures

List of Tables

List of Acronyms

AAA	Authentication, Authorisation and Accounting
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
ARP	Address Resolution Protocol
ASP	Application Service Provider
ATM	Asynchronous Transfer Mode
BNG	Broadband Network Gateway
BRAS	Broadband Remote Access Server
CFI	Canonical Format Indicator
CPE	Customer Premises Equipment
CVID	Customer VLAN Identifier
DEI	Drop Eligible Indicator
DHCP	Dynamic Host Control Protocol
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
EAS	Ethernet Aggregation Switch
HVID	Handover VLAN Identifier
IP	Internet Protocol
IPoE	Internet Protocol over Ethernet
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6

ISP	Internet Service Provider
IWF	Interworking Function
L2TP	Layer 2 Tunnelling Protocol
L2TS	Layer 2 Tunnelling Server
LAC	L2TP Access Concentrator
LFC	Local Fibre Carrier
LNS	L2TP Network Server
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
NAS	Network Access Server
NOS	Network Operating System
NSP	Network Service Provider
OAM	Operations, Administration, Management
ONF	Open Networking Foundation
ONT	Optical Network Terminator
ONU	Optical Network Unit
OVS	Open vSwitch
PPP	Point-to-Point Protocol
PPPoA	Point-to-Point Protocol over ATM
PPPoE	Point-to-Point Protocol over Ethernet
QOS	Quality of Service
RAS	Remote Access Server
ROFL	Revised OpenFlow Library
RSP	Retail Service Provider
SDN	Software Defined Networking
SLIP	Serial Line Internet Protocol

SPOF	Single Point of Failure
SVID	Service VLAN Identifier
TCI	Tag Control Information
UDP	User Datagram Protocol
UFB	Ultra Fast Broadband
VLAN	Virtual Local Area Network
WAN	Wide Area Network
WISP	Wireless Internet Service Provider

Chapter 1

Introduction

With the introduction of many new technologies and approaches to computer networking in recent years, there now exist better or smarter ways to implement existing network models and architectures. Many researchers are engaging with these new technologies in an attempt to discover ways in which common network architectures may be improved in order to better cater to the requirements of that network model. One such new technology, Software-defined networking (SDN) [7], is an emerging approach to computer networking which allows network administrators to decouple the control plane (the component responsible for a device's decision making behaviour) away from expensive specialised networking hardware. This enables the abstraction of control logic for that network hardware to a series of software controllers running remotely on cheaper commodity consumer hardware. The OpenFlow protocol [18] is one such SDN communication protocol which allows this relationship to be defined, allowing communication between these two newly distinct entities - the data plane (the component responsible for a device's traffic forwarding capabilities) and the control plane. This approach allows administrators to define their own specific traffic forwarding behaviours in the decoupled software control plane, facilitating the management of entire networks remotely with a series of custom software-defined rules that can extend the capabilities of typical commodity hardware. Such rules or behaviours are installed in network hardware as traffic flows (modifications to the device's packet forwarding table), which are a series of executable actions that are taken when incoming packets match specific series of pre-defined characteristics.

In existing Internet Service Provider (ISP) networks, one network model that may be improved upon with the advent of SDN is the Broadband Network Gateway (BNG) architecture, also known as the Broadband Remote Access Server (BRAS) architecture. In smaller ISP networks where costs may be restrictive, a centralised BNG model represents a single point of failure (SPOF) for the entire ISP subscriber base. If the BNG in this architecture were to fail, all active ISP subscribers would lose the ability to connect through the ISP's access network to the Internet or another service provider, as all subscriber sessions must traverse the BNG. If the structural principle of non-singular dependence on any critical appliance can be applied in some manner to this existing BNG architecture using SDN, it may be possible to improve the ISP carrier network's operational capabilities. By decoupling any intermediary switch's control planes away from their switching silicon and instead abstracting them to multiple SDN software controllers, it may be possible to increase the network efficiency and robustness of the ISP carrier network. This can be achieved by decreasing the dependency on specific individual appliances and spreading the existing BNG functionality among multiple devices. The ability for a SDN-capable network to distribute any of these software-defined traffic forwarding rules across the network itself may help structure the proposed architecture into a distributed non-centralised model for which dependence only exists on controllers for those packets which do not already match any existing flows. Investigating the plausibility of improving upon the existing ISP centralised BNG architecture in this manner was the primary motivation for the conception of this project.

This investigation considers and evaluates the replacement of the existing BNG architecture with a series of one-to-many software controllers running the Ryu SDN development framework [10], which implement the same BNG functionality by distributing the traffic forwarding logic across a series of OpenFlow-enabled layer 2 hardware switches, instead of relying on logical dependence on a single authoritative BNG. To evaluate the feasibility of this approach, a prototype implementation was constructed with the use of Ryu, and laboratory tests were conducted using a series of virtualised hosts interconnecting through Open vSwitch (OVS) [31].

This report first introduces the background concepts and core workings of SDN and BNG operation in Chapter 2. In Chapter 3, some of the requirements for a replacement ISP BNG system are outlined and related works are discussed. Chapter 4 presents ways in which the outlined requirements can be addressed with an OpenFlow inspired system design using constructed OpenFlow controllers and flow mechanisms. Finally, Chapter 5 contains the evaluation and discussion of presented approaches, including deployment scenarios and real-world limitations. Project conclusions, contributions and discussion of future work are provided in Chapter 6.

Chapter 2

Background

There are many concepts and protocols integral to the development of any software controller-based solution for this project, primarily revolving around the SDN and OpenFlow concepts and the BNG's current purpose and function. Relevant SDN and OpenFlow concepts which make this project possible form the basis of Section 2.1. Section 2.2 introduces the BNG, its background and general behaviours, and the rationale behind improving upon its existing architecture. Lastly, Section 2.3 briefly describes the tools which will be used to build and implement this project.

2.1 Software Defined Networking and OpenFlow

Traditional network switches are comprised of two distinct planes; the data plane which is responsible for the forwarding of packets through the switch, and the control plane which is responsible for making decisions on how to forward those packets. Traditionally, these two planes exist within the same physical switch and are tightly coupled together. SDN allows these two entities to be abstracted away from one another, separating a network switch's control plane away from its data plane. This allows for the control plane to be moved to a software application running remotely. This concept allows for a control plane to enjoy a higher degree of abstraction than is the norm in network switches, i.e. software can be written to dynamically determine the configuration of any switches being controlled, as opposed to the strict hierarchy of behaviours that a switch's control plane would otherwise use to govern data plane traffic forwarding. This remote control plane entity governing traffic forwarding behaviours is aptly named the **controller**, although it is also known as a **Network Operating System** (NOS). As a result of using a controller, network administrators gain unprecedented programmability, automation, behavioural visibility, and control over their switch's operation, enabling them to design and build highly scalable, flexible networks that readily adapt to changing needs. SDN supports these characteristics by opening access to the control plane where previously it had been considered a black box entity. Subsequently a SDN-capable network switch with an abstracted control plane would no longer be bound to the same inherent limitations traditional switches are. This can permit the extension of switch functionality without the need to modify the existing switch hardware or firmware. Furthermore, SDN enables researchers and administrators to easily experiment with new network protocols and configurations on production networks concurrently alongside existing systems, without any negative impact. This can allow network operators to introduce higher degrees of network flexibility, efficiency and provisioning agility into their networks, while keeping operating expenses and capital expenditure down.

The relative infancy of SDN means that the standardisation of its related protocols is still taking place, as contributors seek to interface with other older well-defined networking standards and protocols. This inevitably leaves room for students, researchers and administrators to address the development of new or compatible implementations of existing networking models and protocols, presenting them with the opportunity to improve the way existing network models operate [3].

OpenFlow is widely considered one of the first SDN standards and outlines the communications protocol which allows separated or decoupled control and data plane entities to communicate with one another. OpenFlow also provides a switch specification which allows the configuration of a switch via the same protocol. Originally it was designed as a tool by which researchers were able to run experimental protocols on existing networks by way of separating experimental traffic from production traffic [18]. However, as discussed in [8], [11] and [19], the advantages of SDN solutions using OpenFlow introduces viable use cases outside the research space. The primary idea behind the OpenFlow protocol is to provide an open Application Programming Interface (API) for the configuration of and interaction with the packet-forwarding hardware of a switch, regardless of the switch's vendor or underlying hardware. It allows for a controller to be logically centralised in a network, providing a basis on which new behavioural approaches to existing network architectures or models can be undertaken.

Figure 2.1 shows a basic visual representation of the relationship between an OpenFlow controller and an OpenFlow-enabled switch. Here the switch consists of two primary parts: a secure channel and a series of flow tables. The

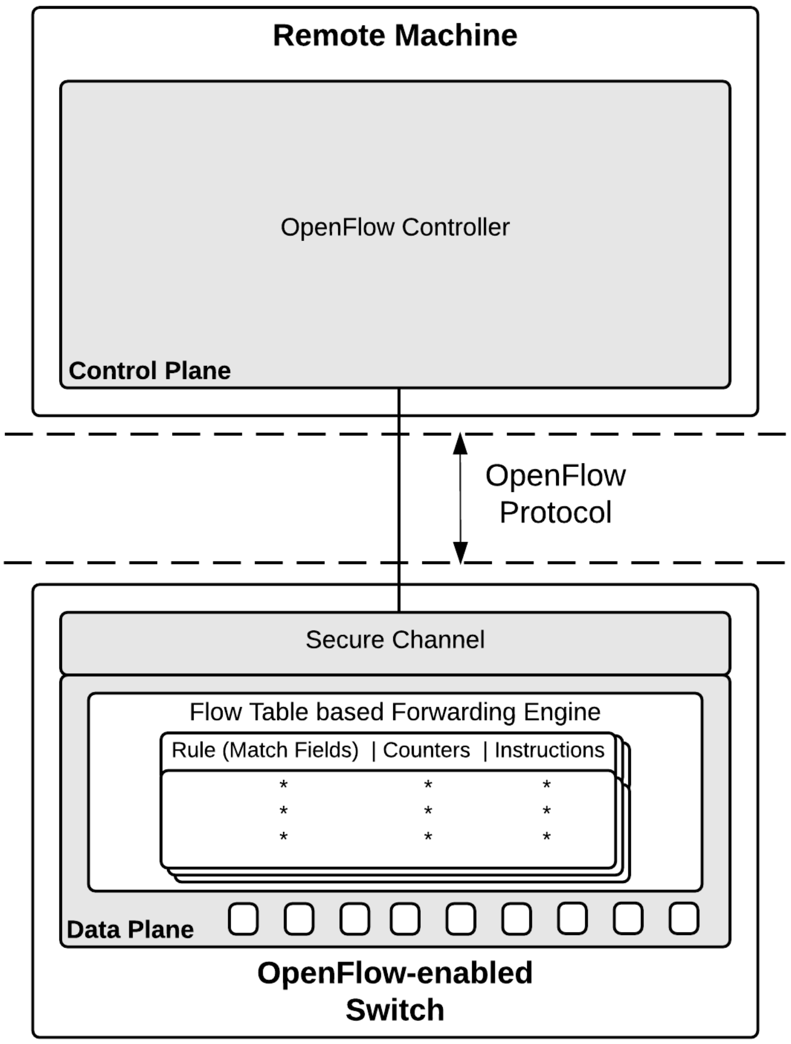

Figure 2.1: OpenFlow 1.3 Protocol and Switch Visualisation

controller manages the switch over the secure channel using the OpenFlow protocol and each flow table performs packet lookup, matching and forwarding [25]. Each flow table is used in this context to determine a packet's destination and decide how the switch subsequently forwards it on. Each flow table consists of a series of flow entries, which each consist of a series of different components. "Flow" in this context is defined as a series of packets that match a particular rule. Using OpenFlow, any packet that does not match any of the currently established flow rules held within the switch's flow table will be sent to the controller, which in turn decides how to handle the packet. The OpenFlow protocol allows a controller to insert, modify and remove these flow table entries dynamically allowing for the rapid and flexible reconfiguration of the switches. The first element in a flow table is "match fields", which are the fields of a packet header or associated meta data which determines whether a packet matches a specific flow or not. A series of different match fields may be used in conjunction with one another to define a directive or a rule for a packet to follow. An example of a rule based on match fields would be one that assigns packets to flows based on the data stored in either the source or destination Media Access Control (MAC) address fields in the Ethernet header. In such a case, any packets containing the same source and destination address as determined by the rule would belong to the same common flow and would be handled according to the instructions in the flow table. An example of what that flow entry might look like is shown in Table 2.1.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie
dl_src=00:11:22:33:44:55, dl_dst=aa:bb:cc:dd:ee:ff	priority= 100	duration=803s, n_packets=6, n_bytes=688	action= output:5	duration= 3600s	cookie= -687779960

Table 2.1: Example flow table entry

The next element in a flow table entry is the "priority" field, which determines the precedence of the flow entry, allowing flow rules with a higher priority to be chosen over those with a lesser priority. Next is the "counters" field, which updates the internal packet counters and maintains an awareness of how many packets are being forwarded through each specific flow. The "instructions" field follows afterwards, which determines what actions should occur upon receipt of packets matching the specific rule criteria. In the earlier example, the instructions field could be configured to forward packets through a specific port or interface on the switch if the previously determined matching criteria (source/destination MAC addresses) were met. In Table 2.1, the instruction field determines that any packets matching these criteria are forwarded out of port 5 of the switch. Lastly, the remaining fields in the flow table include the “timeouts” field, which determines how long a flow can exist for or remain idle in the flow table before being expired by the switch, and the “cookie” field, which is used in administrative tasks performed by the switch.

Since the OpenFlow switch specification version 1.0 released in 2009 [22], there have been four iterations of the OpenFlow standard [23, 24, 25, 27], with the next version (1.5) expected in late 2014 [4, 28]. OpenFlow switch specification 1.0 was widely considered the first official specification of the OpenFlow standard, and represented the first standardisation for which an OpenFlow switch could be constructed from. This specification remains the most common and widely supported version of OpenFlow. Each subsequent version of OpenFlow introduced additional features into the standard which would prove integral to the feasibility of this project. OpenFlow 1.1 introduced support for multiple flow tables that allowed the introduction of more complex forwarding actions inside a switch, and Virtual Local Area Network (VLAN) tag support for handling packets with a hierarchy of multiple VLAN tags [9]. OpenFlow 1.2 introduced more extensible matching criteria for packet parsing and multiple-controller functionality for maintaining simultaneous connections between multiple OpenFlow controllers using roles. OpenFlow 1.3 includes the most significant changes and improvements to the specification since OpenFlow 1.0, adding flexible table-miss support for better handling of packets which do not already match existing flows and Quality of Service (QoS) capabilities through per-flow meters. OpenFlow version 1.3 has been used for the controllers created for this project. Newer versions of OpenFlow exist, but these are either not widely supported yet or are still being written.

2.2 Broadband Network Gateway

The ISP BNG model has seen many iterative changes since originally describing a Remote Access Server (RAS) for dial-up Internet access over the public switched telephone network. The RAS was an access server that acted as a network gatekeeper for an ISP and was responsible for facilitating network access for incoming subscriber sessions off an access network using either the Serial Line Internet Protocol (SLIP) or Point-to-Point Protocol (PPP). As Digital Subscriber Line (DSL) broadband was gradually introduced, the RAS model was modified to suit the needs of the newer Asynchronous Transfer Mode (ATM) based Asymmetric DSL (ADSL) technology, in what would become the BRAS [1]. The BRAS filled fundamentally the same role as network gatekeeper and session aggregator. The BRAS served for several years before being modified to accommodate newer non-ATM technologies and as Internet Protocol (IP) QoS capabilities became necessary [39, 40]. As broadband technologies continued to evolve, Ethernet-based DSL aggregation became prevalent and the BNG was formally defined [41]. The BNG encompassed the functionality of the BRAS, but as the BNG did not form a required component of new Ethernet-based architectures, the use of the BNG title was required to clarify between these two different appliances. The new Ethernet-based networks often delegated subscriber session establishment to alternate non-PPP protocols, such as IP-over-Ethernet (IPoE) (including other protocols such as Dynamic Host Configuration Protocol (DHCP) and VLAN). As such, modern day BRASs and BNGs now require a greater range of supported protocols and session management capabilities to better serve a wide range of different network access mediums. These appliances facilitate the capability to support access-media-agnostic IP-based access networks, allowing for the extensible migration from ATM-based DSL to other technologies [12].

The BNG typically functions at the core of an ISP network infrastructure and is responsible for aggregating incoming subscriber sessions coming from an access network (i.e. ADSL, Ultra Fast Broadband (UFB) fibre or 802.11-based wireless) and enforcing subscriber-associated policy. The subscriber sessions are then forwarded through the ISP IP core by the BNG to an edge router or another termination point (i.e. a Layer 2 Tunnel Switch (L2TS), Network Service Provider (NSP) or Application Service Provider (ASP)). Figure 2.2 depicts a basic BNG architecture and shows the relationship that exists between the BNG and a number of other components in an ISP network. In this example, which has been simplified to show a single subscriber and access medium, the BNG receives the active subscriber session from an ADSL-based access network. The access network consists of a Customer Premise Equip-

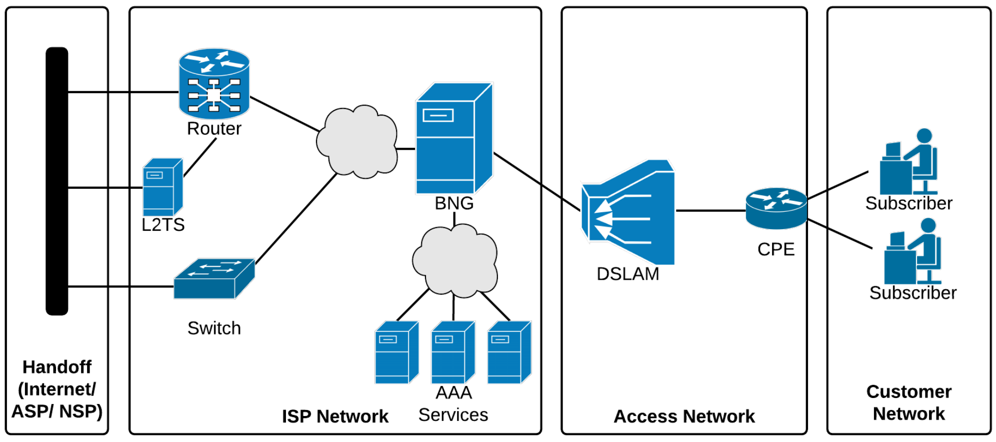

Figure 2.2: BNG operation in IP-enabled regional ISP network

ment (CPE) at the edge of the customer network, which allows the subscriber traffic to access the copper telephone lines through a modem. The Digital Subscriber Line Access Multiplexer (DSLAM) aggregates the analog signals carried across the individual subscriber DSL circuits. The analog data is subsequently transported in ATM packets to the ISP network and into the BNG where it is facilitated for authorisation and authentication accordingly. The CPE authenticates with the ISP using PPP over ATM (PPPoA)[1] to encapsulate its associated subscriber session establishment information and the BNG will either terminate each PPPoA session itself or facilitate doing so through a Network Access Server (NAS) or one of the other connected Authentication, Authorisation and Accounting (AAA) servers. Depending on whether the PPPoA session credentials were correct and what policy may exist for this subscriber, the BNG either then forwards subsequent subscriber traffic through the ISP network to the appropriate destination or rejects the authentication attempt outright. Appropriate destinations for subsequent subscriber packets to be forwarded to include typical endpoints, such as a Layer 3 IP router for external network access (e.g. the Internet), a L2TS (to handle additional PPP encapsulation) or another Layer 2 switch.

There are many different ISP architectures which may involve a BNG appliance. One issue with the architectural model presented in Figure 2.2 is that

[1]Applicable to this example only. CPEs may authenticate using a number of protocols, such as PPP over Ethernet (PPPoE) or Internet Protocol over Ethernet (IPoE)

the BNG is a SPOF for subscriber access to the Internet or other service providers. Although many ISP network configurations contain more than a single BNG, this single centralised BNG configuration is still common, especially among smaller ISPs. Even though most BNG hardware itself is classified as a carrier-grade appliance (which carries 99.999% availability expectations) this dependency is not just problematic from a fault or failure point of view. Full BNG appliances regularly extends beyond simply the price of the physical hardware (i.e., support, licensing, and add-on card costs) so scaling to the requirements of the network by acquiring multiple or redundant BNGs can result in substantial costs to the ISP itself.

This project looks at addressing and removing the dependency on a centralised BNG appliance in modern-day ISP network architectures by moving away from a centralised model to a distributed, decentralised one using SDN concepts and technologies. This will offer an improved architecture not just to those network configurations where a sole BNG facilitates all subscriber aggregation, but also to other network configurations where distributing and bringing the BNG functionality closer to the subscribers themselves may have performance advantages. Using SDN and OpenFlow, it is hoped this project introduces advantages such as increases to speed, versatility and robustness of the ISP network model, as well as subsequent reduction of costs associated with the construction and ongoing support of that network.

2.3 Implementation Tools

2.3.1 Open vSwitch

OVS is an open source multilayer virtual switch. It has been designed as both a switching stack for hardware virtualisation and a production-ready multilayer switch, allowing for both the creation of complete virtual Layer 2 networks on a single host and control stack functionality for actual hardware switches. OVS has native support for OpenFlow versions 1.0-1.3, and since the release of OVS version 2.3.0 also mostly supports later OpenFlow versions (1.4, 1.5). Due to the obvious limitations of building and emulating a sufficiently sized real-world ISP network configuration for this project, OVS is an excellent tool to realistically design and simulate OpenFlow-enabled network architectures.

2.3.2 Ryu

There are many open source OpenFlow controllers currently available, including Beacon [5], Floodlight [6], Trema [44], OpenDaylight [29] and Ryu [10]. Among these different frameworks, only Ryu includes support for all of the OpenFlow versions discussed in Section 2.1, whereas the other noted frameworks mostly only support OpenFlow version 1.0 in an official capacity. Ryu is a Python-based framework and is highly suitable for rapid prototyping. Ryu offers a component-based SDN framework which provides software components backed by a well-defined API. Ryu makes it easy for developers to create and build new network management and control applications, and supports various network device management protocols. In this project Ryu has been used to design, build and implement various OpenFlow controllers to achieve the outlined goals of the projects.

Chapter 3

Investigation

3.1 Software Defined Networking and OpenFlow Technologies

This section is focused on investigating concepts and discussing research in the area of SDN and the OpenFlow protocol which may influence the direction of the project. It will address some of the concepts and research being carried out in the area of high availability OpenFlow controllers, as well as work investigating the necessary supported network communication protocols. The practical realisations, implementation details and limitations of these areas will also be explored in the context of achieving the project aims.

3.1.1 High Availability

One of the primary motivations in the context of reducing dependency on specific system components is high availability of the system itself, i.e. operational availability of the system that is able to be maintained continuously for long periods of time. This is often achieved with failover mechanisms and system redundancy, which allow the system to continue to function even if some operational components of that system were to fail. In order to sufficiently reduce the singular dependence on a centralised BNG in existing ISP architectures, any developed distributed BNG architecture must maintain this property of high availability. Given that one of the primary aims of this project is to reduce singular dependence on the BNG appliance, any architectural model that is designed to replace it should not suffer from the same singular dependence vulnerability. Some of the mechanisms investigated to introduce high availability into this project are discussed below.

Distributed OpenFlow Controllers

OpenFlow controllers capable of operating concurrently in a distributed fashion are an important step towards achieving high availability in this project. Relying on a single OpenFlow controller fails to introduce a reduction in dependence from a single BNG, and therefore the introduction of additional controllers would be advisable. Additionally, the Ryu OpenFlow controller framework does not support multi-threading and will subsequently fail to scale successfully to cater for the large throughput values that a BNG would encounter [33]. Therefore, one solution to this problem would be to introduce multiple OpenFlow controllers which are able to distribute the governance and load of the network amongst themselves equally, removing both the singular dependency and throughput scale issues. This contributes towards high availability by distributing the total network load across multiple points-of-presence meaning that no single controller is expected to facilitate the establishment of all required subscriber flows. A controller framework structured this way could also continue to function in the absence of all participating controllers, introducing redundancy to the architecture. Furthermore, OpenFlow controllers implemented in this fashion gain the additional advantage of being able to be deployed closer to their target switches, introducing speed advantages to the network by virtue of reducing the latency of the behavioural control and traffic forwarding interactions between the controller and switches.

One of the major pitfalls of running a distributed series of OpenFlow controllers is the requirement for controllers to freely exchange state information about the network switches they govern. While there have been a number of studies conducted on distributed controller frameworks which appear to achieve this property [14, 35, 43], no mechanism yet exists for this functionality to be achieved using Ryu. Given that an investigation into achieving OpenFlow controller distributivity using Ryu is a research question in its own right and is too large in scope for this context of this project, the decision to pursue a distributed controller framework for this project was therefore investigated but not implemented.

Redundant OpenFlow Controllers

As noted in Section 2.1, the OpenFlow switch specification from version 1.2 onwards describes mechanisms for OpenFlow controllers to exchange role information between themselves. The roles the controller may take vary between ‘equal’, ‘master’, and ‘slave’. The ‘equal’ role (OFPCR_ROLE_EQUAL) is the default role that the controller will take unless otherwise specified. This gives the controller full access to a switch and its associated asynchronous messages used for governance, as well as full flow table read/write capabilities. The disadvantage of the ‘equal’ role is that multiple controllers may share this role equally and contest governing a given switch individually instead of collectively. The switch does not facilitate any arbitration or resource sharing between the controllers [26], so it is wasteful to have multiple controllers vying for control of the same switch without first synchronising state information to establish between those controllers any prior behavioural traffic forwarding decisions made on the switch’s behalf.

The remaining ‘master’ and ‘slave’ roles (OFPCR_ROLE_MASTER and OFPCR_ROLE_SLAVE respectively) allow the controllers to be configured in a redundant relationship. The ‘master’ controller role is similar to the ‘equal’ controller role, except in any controller configuration it is expected there will be only a single master controller. This master controller retains the full read/write capabilities for each governed switch’s flow table and other control messaging mechanisms. Any remaining controllers in this master/slave configuration are marked as slaves and therefore are only permitted read-only access to the switch. Slave controllers are denied the ability to execute any controller-to-switch commands. Additionally, the roles can freely be exchanged between controllers in the configuration. If a slave controller were to decidedly rise to the role of master, the intermediary switches immediately notify and forcibly change the existing master controller to the ’slave’ role. A role request mechanism (OFPT_ROLE_REQUEST) is used to facilitate this change.

The role request mechanism enables administrators to easily tie active controller states to external watchdog applications as a way of monitoring the health of the controllers in a common configuration. Such a mechanism allows

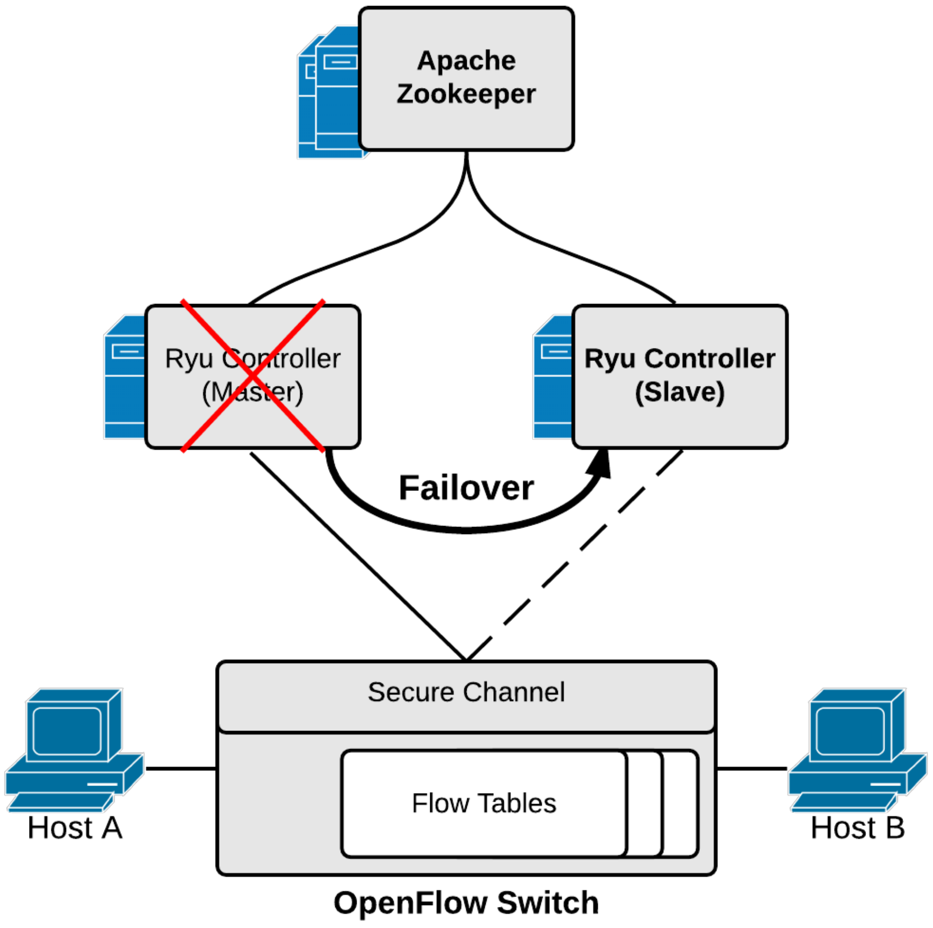

Figure 3.1: Apache Zookeeper operation with multiple OpenFlow controllers

the election of a new master controller in the case that the existing master controller fails or an event determining that the controller roles should change occurs. One example of an open source application capable of achieving this with the Ryu controller framework is Apache Zookeeper [38]. Apache Zookeeper can be used to achieve high availability with a series of OpenFlow controllers by maintaining an awareness of the current status or role of any controllers in the configuration. If the master controller in the configuration were to match a predetermined failure condition or stop responding, Zookeeper is able to trigger an event and notify the remaining slave controllers, subsequently causing a new master controller election process among the slave controllers. A basic example of this relationship is visually represented in Figure 3.1. Zookeeper helps ensure high availability of the controllers by facilitating an elegant failover mechanism. However, Apache Zookeeper does not achieve high availability on its own. Administrative attention is still required to remedy any ‘failed’ controllers. Fortunately, Zookeeper can likely lend itself to a comprehensive stateful monitoring system by notifying administrators of changes to the controller configuration. The disadvantage of this approach however is that all flow creation still takes place on a single controller at a time.

3.1.2 Protocol Support

An important consideration to achieve a viable BNG replacement architecture for this project was determining which required communications protocols were supported and able to be handled or parsed by the Ryu OpenFlow controller framework. As evidenced by [40] and [41], a standardised BNG is required to understand and handle a substantial number of different protocols with specific configuration requirements in order to both support end users on an access network and connectivity to upstream NSPs or ASPs[1]. Any developed OpenFlow controller needs to facilitate the handling of these protocols in an agnostic manner and have the ability to forward any related packets to the correct destinations, for instance from a Layer 2 Tunnelling Protocol (L2TP) Access Concentrator (LAC) through to an L2TP Network Server (LNS) for L2TP traffic, or a DHCP Server for DHCP IP lease establishment traffic. Therefore, the controller must be able to read, analyse, and categorise different packets based on values extracted from various Internet Protocol Version 4 (IPv4) and Internet Protocol version 6 (IPv6) packets, Ethernet frame header fields, or encapsulated packet data payloads.

Ryu contains a moderately sized API and packet library with support for a number of common protocols [32]. Despite the absence of existing API calls for several necessary protocols such as L2TP and PPP over Ethernet (PPPoE), the full contents of layer 3 IP packets are exposed to the controller. This gives developers the ability to extract IP layer headers and payloads individually from each packet the controller is forwarded. Subsequently, developers are able to write protocol handling functionality into their controller, or alter-

[1]The exception to this are those protocols which do not get encapsulated inside Ethernet frames and do not present as Ethernet to a layer 2 Ethernet switch, such as ATM-based DSL networks which still use ATM upstream of the DSLAM. In order to accommodate ATM-based DSL networks, the assumption has been made that an interworking function (IWF) will be present in modern day DSL-based access networks in order to allow the DSLAM to present aggregated subscriber sessions encapsulated inside Ethernet frames.

natively use a third-party packet library such as libpcap [15] with scapy [2]. While the ability to expose packet data and process it with custom-written protocol handlers is a useful capability, it would have been preferential if the Ryu framework supported a wider range of protocols by default. One likely reason for the lack of supported protocols for carrier-grade ISP networking using Ryu is the lack of support for these protocols in OpenFlow itself.

3.2 Broadband Remote Access Server

This section focuses on investigating concepts and discussing existing works on the existing BNG architecture which may influence the direction the project ultimately takes. Existing practical realisations and implementations will be discussed as well as any limitations of these approaches with respect to this project.

3.2.1 Current Dependency Reduction Strategies

Carrier network architectures employ a number of approaches to distribute the necessary functionality of the BNG appliance out to other devices in order to reduce any inherent SPOF vulnerabilities present in the basic model depicted in Figure 2.2. These approaches do not replace the BNG appliance itself but rather simply distribute its functionality among multiple appliances. These approaches typically employ specifically designed carrier network architectures and are commonly seen in larger ISPs. Some of these architectures use multiple BNG appliances distributed among a series of variably-sized regional networks such that a SPOF may still remain for a given access network but may only affect a smaller geographically-locked subset of subscribers. Other approaches cluster multiple BNG appliances together to ensure high availability of the session aggregation capabilities by failing over to secondary or tertiary appliances in the event of a fault or failure [36]. These approaches help remove the SPOF vulnerability associated with using a single centralised BNG appliance, but carry a high cost. The cost can be restrictive for smaller ISPs and can present a barrier to entry in a competitive market place, therefore lending merit to investigating alternative BNG approaches. Those approaches which look to resolve this dependency but do not rely on dedicated BNG appliances are few in number, but those which have been documented aim to distribute the embedded functionality of the BNG among other network appliances to remove specific dependencies.

One such approach to distributing functionality typically handled by the BNG appliance was presented in [17]. The presenter described a Wireless ISP (WISP) where a series of PPPoE termination servers or NASs were load balanced behind the primary router at the access network edge. A load balanced solution between multiple PPPoE termination points allowed users connecting and authenticating through this wireless access network to be equally distributed among a series of servers. Such an approach not only ensured high availability for the authentication service itself through the use of multiple end points, but also enabled a solution that scales to the requirements of the network. Using such an approach would easily facilitate the addition of more NASs, enabling the network administrators to respond to subscriber growth easily with additional hardware. The physical topology of such an approach could additionally be designed in order to provide authentication server services much closer to the subscriber base than modern architectures do. The speed at which session negotiation is facilitated could therefore be improved. Additionally, this approach could be improved by distributing the function of the access network edge router among multiple edge routers such that any SPOF is not simply moved closer to the customer by having all subscriber sessions route through a single common appliance.

3.2.2 Software Defined Networking and OpenFlow Solutions

There have been a number of projects looking to improve carrier networks using SDN and OpenFlow. Many of these projects have revolved around defining and developing mechanisms to support subscriber access convergence and aggregation using these technologies. One of the major projects investigating this area is the SPARC Consortium's FP7 project [42], which is extending existing carrier networks by introducing SDN-facilitated split architectures in order to introduce more dynamic network control and connectivity into the carrier network. The SPARC Consortium define their project as the investigation of splitting the traditionally monolithic IP router architecture into separate forwarding and control components using OpenFlow. One of SPARC's specific target areas of research is improving the OpenFlow protocol for use in carrier networks and researching how it may be deployed to achieve IP router functionality similar to that of a BNG appliance. In some of their resulting publications [13, 34] they suggest that the OpenFlow version 1.0 standard lacks in support for carrier grade network operation in areas such as supported communications protocols, inter-domain capabilities, and failure recovery. Their investigation into different facets of redesigning the carrier network architecture using OpenFlow have resulted a number of relevant publications, some of which are discussed below.

In [13], a SDN use case involving distributed BNG functionality is presented as a realised component of a carrier network split architecture. The use case is an OpenFlow-enabled carrier network and investigates how the former BNG's role in terminating PPPoE traffic is able to take place without a dedicated BNG appliance. The authors theorise that the actual termination of any PPPoE sessions will need to be abstracted to a separate NAS architecture (termed 'PPPoE application'), which connects to an OpenFlow controller. The controller is used to create and manage any initial flows between the PPPoE end points and intermediary network devices, as well as interfacing with the NAS for PPPoE control and access concentration. The flow creation process will need to create flow table entries in intermediary devices that allow the PPPoE application to receive both PPPoE discovery packets and PPP session packets. The forwarding of those packets will be able to be facilitated by those OpenFlow-enabled intermediary devices based on these flow table entries. This model however has only been outlined as an initial design, with the authors conceding that any processing performed by the PPPoE application will require extensions to the current OpenFlow protocol. However, this particular distributed BNG model would have the disadvantage of being very slow at forwarding PPP-based traffic if implemented. The PPPoE application in this model is described as interfacing directly with the controller, rather than with the common layer 2 network, which is also known as the 'OpenFlow fast path'. This would mean that any PPPoE/PPP traffic required to reach the PPPoE application would have to be forwarded directly through the controller, rather than through the common network. This would substantially contribute towards a high load on the controller itself and in a production network would not scale to a great number of users. A visual representation of this model is depicted in Figure 3.2.

An additional paper that investigates an OpenFlow based BNG architecture is presented in [45]. It presents a prototype OpenFlow BNG and, similarly to other papers discussed above, investigates an OpenFlow-based architecture capable of terminating the PPPoE protocol. In this paper, the developed OpenFlow implementation was extended to include the Revised OpenFlow Library (ROFL) as a series of vendor extensions to accommodate PPPoE support. Using ROFL enabled the authors to support the encapsulation/decapsulation of PPPoE and PPP on the data plane, as these protocols are not yet defined in the OpenFlow standard. The outlined approach for handling these network protocols showed similarities to the design described in [13], whereby the PPPoE discovery frames (identified by a unique EtherType value of '0x8863') are sent to the OpenFlow controller for session establishment. PPP session frames (identified by a unique EtherType value of '0x8864') are subsequently handled solely on the data plane. The authors also outlined that

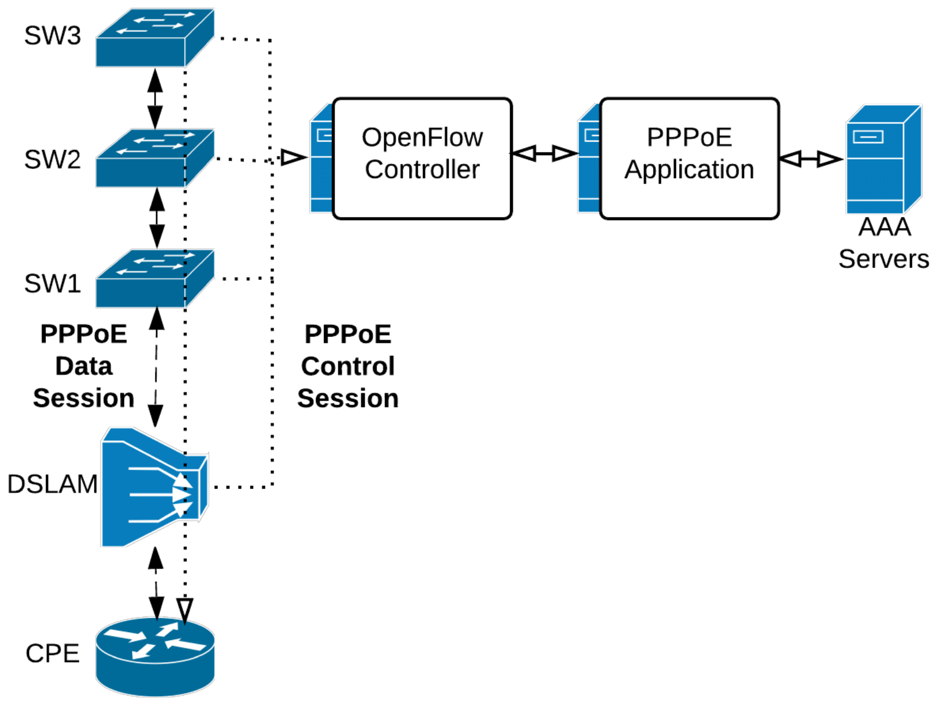

Figure 3.2: PPPoE session creation with split architecture BNG model

an important consideration for a model in which PPPoE is supported by an OpenFlow-enabled network is the ability to track the present states of those PPPoE sessions which are currently active. They stipulated that although initial designs handled session state in the controller, the use of virtual ports which encapsulate session state was under consideration as a more efficient alternative. This paper shows that if OpenFlow-based technologies are to be presented as a viable approach for improving upon existing carrier network models and architectures, the OpenFlow standard must include more support for typical carrier network protocols. Although it is promising that some functionality for this family of protocols is obtainable through the use of vendor extensions, it is worth noting that vendor extensions are typically bound to vendor specific hardware and therefore make widespread adoption in carrier networks difficult. Fortunately, a precedent has been set by the OpenFlow development team. Multiprotocol Label Switching (MPLS) protocol support for OpenFlow was in strong demand and began as a third-party documented extension to the OpenFlow 1.0 standard. MPLS later became one of the first new protocols to be supported in the OpenFlow 1.1 standard, so it is likely that there will be more protocol support as the standard continues to mature.

Chapter 4

Implementation

4.1 Broadband Network Gateway Session Aggregation

As mentioned in Section 2.2, this project examines using SDN and more specifically the OpenFlow protocol to explore solutions for reducing dependency on the centralised BNG carrier network model. The focus on addressing this aspect of ISP networks was chosen as it represented a common network configuration which could be potentially improved with the use of SDN technologies. This project started by addressing the implementation of mechanisms to handle the following basic session establishment protocols and standards for reasons outlined:

- *DHCP*

  DHCP is an IP-based protocol which provides a mechanism for hosts to be assigned addresses using a common IP address scheme. This allows hosts to intercommunicate with one another on the same IP network. It is commonly used on both metropolitan-area and large scale Ethernet based networks to allow subscriber connections to gain network access through the BNG appliance.

- *VLAN*

  VLAN is a mechanism that allows network administrators to create subsets of network domains such that hosts behave as though they are on the same physical wire as one another, even though they may be on entirely different network segments. VLANs use a tagging system to uniquely identify these domains and forward Ethernet frames to them accordingly. It is commonly used on both metropolitan-area and large scale Ethernet based networks to divide up layer 2 access networks into specific domains and can be used to help apply and manage policy for different network segments. In an ISP, 802.1ad VLAN stacking can be used to transparently encapsulate point-to-point customer VLAN information, allowing hosts at different network endpoints to communicate on the same layer 2 network as facilitated through the ISP network.

These protocols have been implemented first with the assumption that modern day BNG's primarily serve subscribers from Ethernet-based access networks. The goal of implemented support for these protocols is to facilitate the connecting of subscribers through the distributed BNG switching fabric and authorisation mechanisms. Once a subscriber is successfully authorised on to the network, their active session is provided internet access by forwarding their traffic through internet-accessible flows. This enables end-to-end connectivity between the two end points with the distributed BNG in the middle.

Basic support for these protocols represented the introductory stage of this project, as together these protocols represent the most basic session establishment mechanism to implement: IPoE. Implementing IPoE first would allow for a background and familiarity with the problem domain and implementation tools to be developed, before additional protocols were investigated or implemented.

4.1.1 Dynamic Host Configuration Protocol

In order to implement support for DHCP in a software controller, some basic principles about the protocol operation needed to be fully realised. The DHCP protocol requires a server on a network which listens for incoming DHCP packets broadcast over that network. A client who wishes to lease an IP address to communicate on the network broadcasts a DHCP Discover request out to the whole network, attempting to discover whether there are any DHCP servers on the network listening. If a DHCP server receives a discovery broadcast packet, it will reserve an IP address for the client (either randomly or one that a client has requested) from a pre-determined pool of addresses, before sending a broadcast DHCP Offer packet back into the network in reply. Although this packet is broadcast to the entire network, it contains the clients unique MAC address in order to differentiate this offer between any other DHCP offer packets being actively exchanged to other hosts. Once this DHCP Offer packet is received by the original client, the client then makes an official request for that address and broadcasts it onto the network once again this time as a DHCP Request packet. The request packet is handled in this fashion so that any other DHCP servers which may also have responded to the initial discovery packet with their own offer are notified of which lease the client is accepting and can subsequently withdraw any unwanted respective DHCP offers and return the reserved IP addresses back into their pool. Finally, the DHCP server completes the establishment phase by replying to the DHCP Request packet with a broadcasted DHCP Acknowledge packet, which contains lease information and completes the IP-lease negotiation once received by the requesting client. A client which reaches this state now applies its leased IP address and is subsequently free to participate in the network. This basic negotiation process can be referred to as the DHCP DORA (Discover, Offer, Request, Acknowledge) handshake.

The DHCP protocol also supports features other than the basic IP address lease negotiation sequence which need to be considered. These features include variable-length packet embedded DHCP options and also the basic IP lease termination or rejection control phases. To offer a regular level of support for the DHCP protocol, these would needed be implemented in the controller also.

The DHCP implementation approach uses specific flows to facilitate the subscriber IP lease negotiation process. The created flows are crafted to know how to direct broadcast traffic matching specific user datagram protocol (UDP) source and destination port numbers to the DHCP server itself, and the controller knows how to return those responses back to the subscriber. Additionally, the controller is able to extract specific options from the DHCP packets themselves. Most importantly, among the different options is DHCP option 53, used to facilitate IP lease negotiation message passing. DHCP option 53 (titled ‘DHCP Message type’) is a one byte option appended to a DHCP packet which indicates the purpose of that DHCP packet, e.g. a DHCP Discover or Offer packet. By reading the DHCP packets option 53 value and the associated UDP source and destination port numbers, the controller is able to determine what the purpose of the packet is and can handle it accordingly.

The first thing the developed controller does is establish connectivity with the DHCP server (using the mechanism outlined in Section 4.3) and maintains this awareness of how to reach it. The developed controller operates reactively and individually handles each subscriber's IP negotiation process as the controller receives packets forwarded to it from any switches for the purposes of flow matching. Packets which are sent to the controller matching the criteria of a DHCP Discover (DHCPD) packet initiate a DHCP handling process. This process is depicted in Figure 4.1. First, the controller polls or queries any connected AAA services (discussed in Section 4.2) in order to establish whether the source MAC address of the received packet belongs to a valid subscriber with a specified connection type (i.e. DHCP or VLAN). The AAA service is expected to return a true or false value. If the MAC address is unknown to the service, it is blacklisted and future packets from that MAC address are subsequently dropped. If the subscriber ends up being valid, the AAA service also returns the specified service type of that subscriber to the controller for processing. Using this data, the controller is then able to create a flow entry to facilitate that connection type. A corresponding DHCP flow entry is specified in Table 4.1, which would be used to facilitate broadcast-based communication between the subscriber and DHCP server. Of note is the UDP source/destination port combination, which are used by the requesting client to deliberately target DHCP servers listening on those ports as per the protocol specification. Once this control flow is established, the controller then forwards the originally received DHCP Discover packet on to the DHCP server.

Ethernet source MAC	Ethernet dest. MAC	EtherType	IP Protocol	UDP source port	UDP dest. port
[*subscriber source MAC address*]	ff:ff:ff:ff:ff:ff	0x0800 [*IP*]	17 [*UDP*]	68	67

Table 4.1: Packet characteristics for matching DHCP control flow entry

Once the controller has facilitated broadcast-based communication between

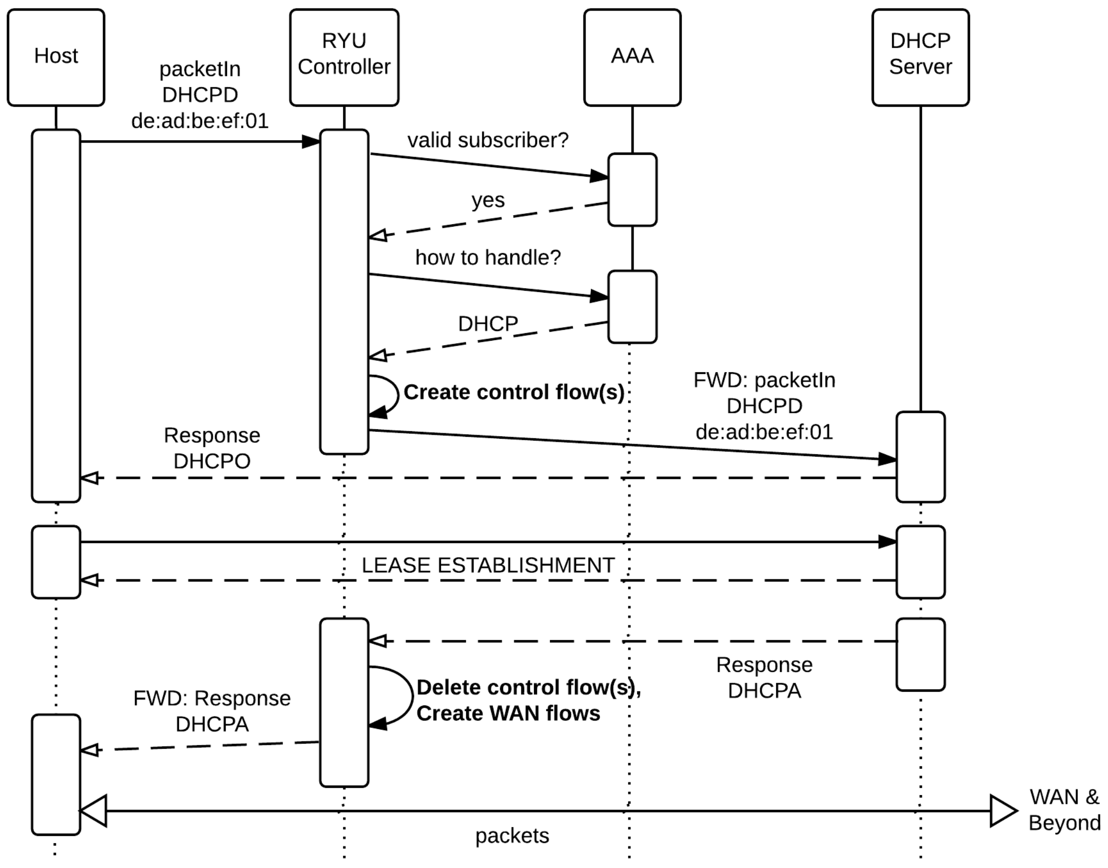

Figure 4.1: Controlled DHCP session establishment process

the subscriber and the DHCP server using these control flows and forwarded the intercepted DHCP Discover (DHCPD) packet to the server, the server then continues with next stage of the IP lease negotiation process. With the authenticity of the subscriber confirmed, the DHCP server is now free to reply with a DHCP Offer (DHCPO) to the subscriber. The subscriber's subsequent DHCP Request (DHCPR) packet which is sent in response is automatically sent through the existing control flow to the DHCP server. If something has gone wrong, the very same control flow can be used to forward the DHCP server a DHCP Decline packet (in the event of an IP address conflict) starting the process again, or alternatively the DHCP server can send a DHCP Nak to the client through the controller. However, assuming that everything has gone correctly, the DHCP server should respond with a DHCP Acknowledge (DHCPA) packet. A flow to automatically facilitate this DHCP Acknowledge has been deliberately withheld at this point, so that the switch is forced to forward the packet to the controller. A DHCP Acknowledge packet forwarded to the controller indicates that the DHCP lease negotiation process is finalised and the controller is able to create new flows that enable subscriber traffic to be forwarded through to a edge IP router or wide area network (WAN). An example of these WAN accessible flows are presented in Table 4.2. Additionally, the control flows which initially facilitated the IP address lease are removed from the flow tables as they are no longer required. Finally, the controller then forwards the final intercepted DHCP Acknowledge packet to the subscriber to finish the negotiation.

Flow	Match fields	Priority	Actions
client-to-WAN	in_port=[*subscriber accessible switch port*], eth_src=[*subscriber source MAC address*]	100	output:[*WAN accessible switch port*]
WAN-to-client	eth_dst=[*subscriber source MAC address*]	101	output:[*subscriber accessible switch port*]

Table 4.2: Example flow entry pair for subscriber WAN access

The controller deliberately does not retain the DHCP control flows that were originally created to aid in the IP lease negotiation phase. If these control flows were to be retained by any switches, the DHCP control flow count would grow at the rate of ($n*2$) per subscriber, which is unsustainable for limited size flow tables. However, the controller retains the ability to individually facilitate and forward additional DHCP control packets by way of specific handlers constructed to handle certain DHCP packets on an isolated basis. An example case of a packet which would have otherwise used an existing control flow is the DHCP Release packet, which is a subscriber initiated mechanism for surrendering a leased IP address. During the development of these mechanisms, regular network operation was an assumption that was made. If the assumption is correct and the bulk of the DHCP server's activity is facilitating new IP leases using control flows, then the remaining DHCP edge case packets being forced to traverse through the controller (albeit at a much slower than line-speed rate) is an acceptable compromise.

The installed WAN accessible flows are responsible for facilitating two way communication between the WAN edge and the subscriber. They function as 'catch all' default flows for each subscriber in order to capture all subscriber non-authentication/authorisation traffic and are subsequently assigned moderately low priority values. As mentioned in Section 2.1, priorities can be used to determine an order of precedence for a flow over another similar or partially matched flow. A moderately low flow priority value means these WAN flows are used when no more specific flows are installed. The WAN edge device end point in this instance functions as the handover to a different network, and is able to resolve the correct subscriber for any traffic destined to it by using Address Resolution Protocol (ARP) to resolve the destination IP address to the appropriate subscriber MAC address. A subscriber which reaches the state of having associated installed WAN flows is considered to belong to a basic WAN-accessible tunnel and therefore has internet or WAN access.

4.1.2 Virtual Local Area Network

By default, Ethernet frames which do not have deliberately tagged VLAN IDs typically are tagged on VLAN ID 1, the standard default VLAN. The use of multiple VLANs enables the division of the network into subsets of layer 2 broadcast domains, allowing administrators to reduce the total broadcast traffic occurring on the network by limiting the areas in which broadcast traffic is able to propagate. Administrators are able to append VLAN tags to Ethernet frames by configuring specific switch ports to tag egress (outgoing) frames received from a connected host with specific VLAN IDs. These same tags are able to be stripped by the switch from ingress (incoming) frames destined for the host in the opposite direction on the very same switch port. Ports with this VLAN functionality are known as ‘untagged’ or ‘access’ ports. For instance, an untagged frame coming from a host to a VLAN 10 access port will be subsequently tagged with the VLAN 10 tag as it passes through the port. If a VLAN 10 tagged frame attempts to reach the originating host from the other side of that same switch port, the VLAN 10 tag is stripped from the frame in order to be received by the host on the default VLAN (VLAN 1). Other VLAN port configurations include the following:

- *Tagged port*

  Tagged VLAN ports are ports which are able to pass Ethernet frames with existing VLAN tags already appended to them intact without manipulating the tags themselves. These ports need to be configured to determine which VLAN ID tagged frames may pass through them. Tagged ports are often referred to as trunk ports, as often they often facilitate switch-to-switch communication via uplink ports which permit the transport of many different VLANs through the same port.

- *Hybrid port*

  Hybrid VLAN ports are a combination of both access and tagged VLAN ports. Hybrid ports allow the untagged port functionality for a specified default VLAN (i.e., VLAN ID 1) and also the tagged port functionality for other defined VLANs. For instance, a hybrid port may be configured such that default untagged frames entering the port are tagged onto VLAN 10, while at the same time the port allows free passage to those frames which may already have VLAN ID tags 20 or 30. Commonly, Voice over Internet Protocol (VoIP) VLANs are configured as hybrid ports to support the ability to connect additional network hosts off of a VoIP handset while retaining VLAN isolation.

Figure 4.2 gives a basic visual depiction of how the untagged port and tagged port basic concepts can be applied to an actual switch with hosts. In this example, ports 1-3 on both switch A and B are configured as 'untagged' for

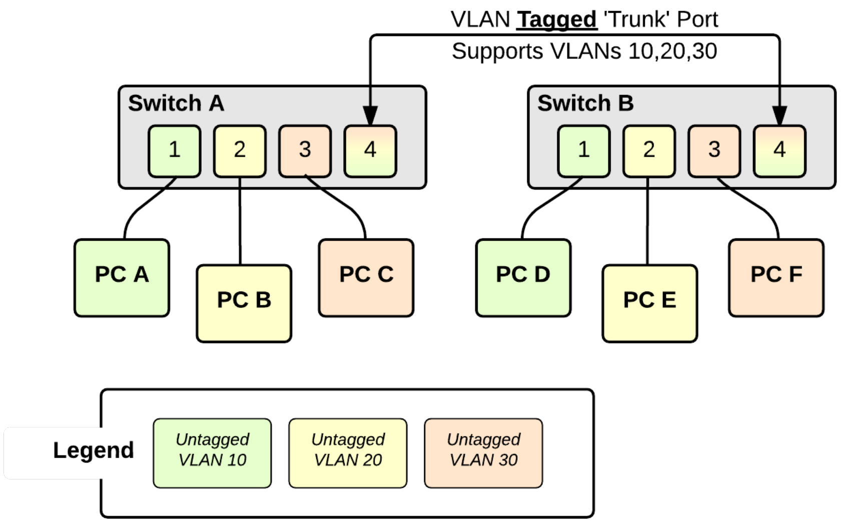

Figure 4.2: Example of basic VLAN port configurations

their own colour-coded respective VLANs. Port 4 of each switch is configured as a ‘tagged’ port and allows frames tagged with VLAN ID 10, 20 or 30 to traverse the link between switches. In this example, assuming each PC is passing untagged frames to the switch, PC A and PC D would be able to communicate with each other in the isolated VLAN 10 broadcast domain network subset, PC B and PC E in the VLAN 20 network, and finally PC C and PC F in the VLAN 30 network.

In order to facilitate the configuration of switch ports in this project, the controller has been supplemented by a VLAN configuration file which administrators are able to configure to specify the VLAN configuration for each of the switch ports governed by that controller. The port configurations are extracted from a local .YAML configuration file and are parsed and applied by the controller to their associated ports when the controller is first initialised during start up. Using this configuration file, the controller is also able to apply blanket catch all default VLAN configurations, as well port specific VLAN-exclusions. This introduces a similar level of configuration capability as is present in enterprise switches.

The current state of VLAN support in the controller is the ability for managed switches to facilitate and pass VLAN tagged frames through specific ports. The frames are intercepted and forwarded to the controller in exactly the same way as regular frames. The controller possesses the ability to strip (or pop) the VLAN tags from the frames and make decisions based on the tag values accordingly. Using a subscriber service type awareness mechanism (described in Section 4.2), the controller is able to subsequently create flow rules to send traffic matching specific VLAN tag values to specific locations or alternatively forward specific subscriber traffic based on the extracted VLAN tags. Additional work is still required to fully implement support for packets which are required to be forwarded on by the controller, in accordance with the VLAN restrictions associated with each port. Additionally, the ability to accommodate more complex VLAN configurations such as VLAN stacking using this controller is another feature that has not been able to be implemented, and is discussed in Section 5.3.1.

4.2 Dynamic Flow Creation

In carrier networks, the typical role of AAA servers is to act as a security architecture responsible for controlling and managing subscriber access to specific services (authentication, authorisation) and actively tracking the resources that those subscribers are using (accounting). Two of the most common implementations of AAA in carrier networks are RADIUS [20] and Diameter [21]. In this project, a centralised subscriber information database was implemented to interface with the developed controller in order to perform basic AAA functions. Using this subscriber database allows the controller to extract accurate information about known subscribers, their unique identifying connection information, and how their incoming session establishment requests should be handled by the controller and associated switching fabric. The controller is able to use this information to structure the creation of flows to meet the requirements of each subscriber, in a mechanism termed 'dynamic flow creation'. Dynamic flow creation enables the controller to query for the appropriate way to handle a given subscriber and then create flows to forward the traffic of those subscribers according to their purchased or selected service. For instance, the source MAC address from an intercepted packet can allow dynamic flow creation to facilitate the creation of flows to lease the subscriber an IP address through DHCP or alternatively forward future layer 2 traffic from that subscriber through a tunnel to a remote endpoint such as a satellite office.

The subscriber database was implemented as an Apache Cassandra [37] database. Apache Cassandra is a NoSQL database, with practical advantages such as support for high availability, scalability and real-time input/output. NoSQL databases are a good fit for interfacing with network controllers given the advantages gained from simpler data structure designs, scale-out architectures and operational agility when compared to traditional relational databases. Apache Cassandra also supports a number of Python database drivers and APIs, allowing Apache Cassandra to interface seamlessly with any controller developed using Ryu. Apache Cassandra was specifically chosen for a proof of concept subscriber database given it can easily scale to run across a cluster of database instances on multiple physical hosts. In order to avoid reintroducing a SPOF into a developed SDN-enabled architecture this was an important consideration and ensures high availability of the database, allowing Apache

Cassandra to scale to the production network requirements as necessary.

Different protocol header and data values can be extracted from unmatched packets sent to the controller for governance. Using this data, controllers are able to make decisions about how to accommodate or cater for future traffic matching these packet criteria using flows, in accordance with what rules have been explicitly defined inside the controller. By interfacing with a subscriber database, the controller gains an extended awareness of how to handle specific subscribers whose packets may not contain any special indicators which may be used to individually determine the flows required for the subscriber. This is especially important for an ISP, as subscribers may require a variety of different services with varying levels of support. Introducing a subscriber database allows network administrators not only to authoritatively control the ways in which subscribers are handled from a centralised point, but also provides a place to store relevant information about subscribers which may not have a bearing on how their packets are managed, such as active connection information. Furthermore, since flows are created based on the information extracted from the database at flow creation time, administrators gain the additional advantage of being able to update subscriber information without needing to restart the controller to begin applying their changes. However, this carries the caveat of not applying to any existing installed flows, only those which are created after the change is made. Additionally, a centralised subscriber database facilitates the construction of a database front-end portal which can be used to make self-service or service desk changes easily.

4.3 Anonymous Unicast Host Detection

In order to accommodate limitations imposed on the project by the simulation of a network in the lab environment, connectivity extensions were written into the Ryu controller to allow the controller to maintain connectivity with staticly connected hosts, i.e. a DHCP server or an IP router on the WAN edge. These changes were necessitated by the presence of OVS in the network topology rather than an actual real-world commodity switch. With a commodity switch in a real world carrier network, network administrators are able to maintain an awareness of how their networks are physically laid out, including configurations for the ports which carry traffic to and from specific appliances, servers or services. Using OVS and a virtualised host lab, this spatial awareness was not always easy to achieve. Connected virtual hosts were not consistently connected to persistent OVS virtual switch ports and in the development environment often appeared to change port bindings dynamically. Any flows which required the forwarding of packets to specifically targeted appliances needed an awareness of which virtual ports were associated with those appliances. Subsequently, a reliable mechanism to bind these hosts to those ports in order to facilitate those flows was required in the controller.

As briefly mentioned in Section 4.1.1 ARP acts as a layer 2 MAC-to-IP address lookup, where a requesting host floods the layer 2 network to request the identity or location of a host matching a specific IP address. The ARP request propagates across the entire layer 2 network. If the requested host receives the ARP request, they reply to the requesting host with the required information in a specifically targeted response. This enables the creation of a binding of the responding hosts MAC address to the requested IP address in the original host's ARP table. Additionally, since the response message traverses the layer 2 network in order to reach the original host, any intermediary layer 2 switches between the hosts learn forwarding information about which switch port best reaches which host (the newly identified host in addition to both the requesting and responding hosts). ARP forms the basis of the host detection mechanism in the controller developed for this project.

The resulting mechanism to bind dynamic ports to specific appliances functions similarly to an anonymous unicast transmission. A unicast transmission in this context is the sending of one-to-one messages to a specific network destination as identified by a unique address. Using a generated unicast transmission allows the controller to maintain port binding information about appliances which the controller and the switches are required to interface with. The controller does this by binding switch ports in software to corresponding hosts that generate specific protocol responses to an input. This approach relies on sending an ARP packet from the controller across the layer 2 network to target specific known appliances. Since the Ryu software controller will not apply controller-specified rules to any packets destined for the controller itself (as traffic targeting the controller specifically already matches an

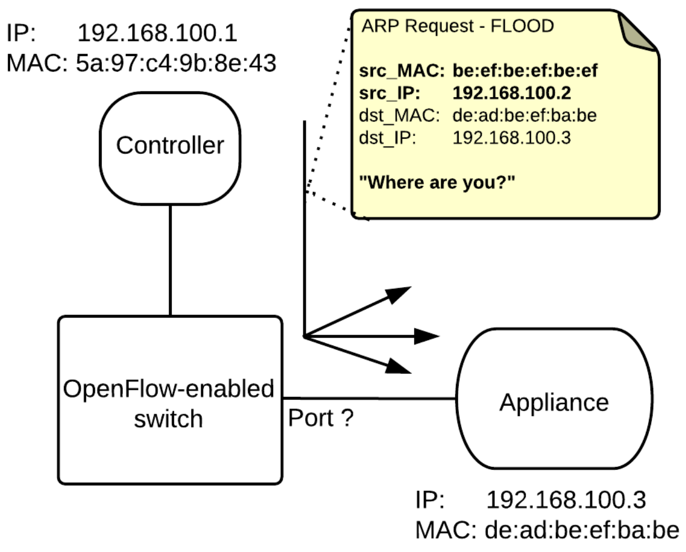

Figure 4.3: Anonymous unicast flooded ARP request example

existing hidden control flow), the controller requires a mechanism to intercept controller-destined traffic and handle it accordingly in software. The ARP header of the flooded packet contains fictitious source MAC and IP addresses in order to force the target host into responding with an ARP response to a host who does not actually exist. Since the specified host does not exist, these packets are forwarded to the controller instead. This allows the controller to receive any ARP response packets generated by the targeted host upon receipt of the original flooded ARP packet, as they are sent to fictitious addresses with no corresponding flows to handle them. To the controller, these packets are destined for a host for which a flow does not already exist and therefore are sent into the software controller for a flow lookup. Subsequently, the controller is able to determine which virtual port the targeted appliance has replied from, and can save this information for later use. A depiction of this process is presented in Figures 4.3 and 4.4.

In Figure 4.3, the controller is flooding the original ARP request from the directly connected switch and onto the layer 2 network. The crafted ARP request contains information the network administrator already knows, such as the destination IP and MAC address of the appliance that the controller is targeting. Since the controller does not however know where this host is,

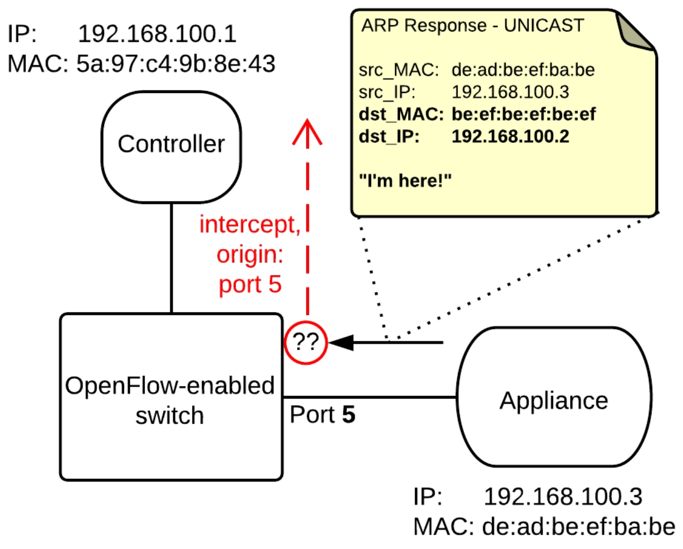

Figure 4.4: Anonymous unicast ARP response example

it spoofs the source MAC and IP address inside its ARP request to elicit an ARP response to these addresses. This way, the controller can coerce the targeted appliance to reply to the fictitious address, allowing the controller to intercept the response. The controller recognises the spoofed destination MAC and IP addresses and saves the port information associated with the incoming packet. The generated ARP response process is depicted in Figure 4.4.

The anonymous unicast mechanism would not be necessary in a real-world environment. However, its development was required to accommodate the dynamic port allocation that was experienced with simulating layer 2 switches and interconnected hosts. The anonymous unicast mechanism enables the controller to deliberately and forcibly learn which virtual port specific appliances can be accessed through. Without this mechanism, the controller would only be able to learn about this port association if the appliance were to send packets through a connected OpenFlow-enabled layer 2 switch for the first time. Given that this project attempted to simulate a carrier network, only having an awareness of where appliances lay in the network once they started generating traffic on their own accord was considered an unnecessary session establishment time delay.

It should be noted that there were a number of other OVS virtual port persistence mechanisms discovered later in the development cycle of this project, such as the ability to persistently bind virtual ports to virtual host interfaces in the OVS underlying database. However, the implemented anonymous unicast host detection mechanism still lends itself well to other circumstances. In a complex OpenFlow-enabled network topology, a given controller is not able to determine the port binding information for a virtual switch being governed by a different controller. In that situation, any connection information for that switch remains solely accessible to that controller. The anonymous unicast host detection mechanism allows an administrator to expose the necessary path for a given controller to take to reach a desired target host, irrespective of the virtual switch that the host remains connected to.

Chapter 5

Evaluation and Discussion

5.1 Broadband Network Gateway Session Aggregation

5.1.1 Dynamic Host Configuration Protocol

Evaluation

One problem with the implemented DHCP session establishment mechanism is the lack of controller support for easily parsing extended DHCP options. In order to facilitate the most basic IP lease negotiations, the only option that was able to be supported in the controller was DHCP option 53, the DHCP Message Type option. While the ability to parse other one byte options may be easily implementable, any other appended DHCP options of a greater length than one byte would require a new controller mechanism for that option to be read or extracted by the controller individually. As a result, the ability to support many important DHCP options which are used to establish IP leases with specific, non-default properties and make subsequent forwarding decisions based on that information is diminished. For instance, DHCP option 51 titled 'IP address Lease Time' is a four byte option which enables a DHCP server to indicate inside a DHCP packet the time (in seconds) in which the client is permitted a leased IP address before that lease expires and an IP address renegotiation will need to take place. If this option were to be supported in the controller, it could be used to populate subscriber flow expiration time frames and allow flows to expire alongside the IP address lease they were originally associated with. A renewal of the IP address lease would renew the flow tied to it, which would automate the expiration of unused flows. In the meantime, such flows need to be manually retired upon receipt of a DHCP Release packet. Unfortunately, a number of hosts do not release their IP leases this way and simply disconnect from the network, meaning that without an automated flow expiration system a significant number of subscriber DHCP flows may exist until manually removed.

The existing mechanism designed to handle DHCP traffic in this controller lends itself well to the maintenance of authoritative subscriber information. All DHCP server responses to the DHCP client pass through the controller and, although this introduces some latency, it enables the controller to extract IP lease negotiation data from the packets themselves. Session data obtained in this way can subsequently be written into the database as a mechanism of maintaining current up-to-date information on a subscribers active internet session. The immediately implementable examples of client, server and gateway IP addresses come directly from the DHCP packet themselves. Each can be helpful for administrators to monitor a particular subscriber session and can be added to tools to aid with ISP operation, administration and management (OAM). In the instance of multiple network end points or multiple DHCP servers, this extracted network information can allow an administrator to gain an immediate view of which IP address is being used by a subscriber, which server leased the IP address, and which network gateway the subscriber traffic is being forwarded through. Additional subscriber session capabilities and information can be further extracted if more flexible DHCP option handling is implemented in the controller.

Discussion

There are improvements that could be made to the supported DHCP functionality in order to reduce latency associated with IP address lease negotiation. Currently, each DHCP control flow which is used to forward subscriber DHCP packets to the server are installed on an individual basis, pending a successful outcome from a subscriber service type handler query made on the subscriber database. A substantial speed improvement to flow creation could be realised if there was single default catch-all DHCP control flow used to facilitate all IP lease negotiation with a DHCP server. Using a single control flow would save controller processing power, throughput and time on flow creation rates (which as discussed in Section 3.1.1 is an inherent limitation of using the Ryu controller). A single DHCP control flow would be able to forgo the creation and deletion of DHCP control flows in quick succession for each DHCP client as they attempt to lease an IP address through the controller. However, a new mechanism which is able to differentiate between IPoE service subscribers who are authorised to obtain an IP address through DHCP and those who authenticate though a different mechanism (i.e. PPPoE) would need to be implemented in the controller. This is an important security capability as it uses OpenFlow flow creation to isolate subscriber traffic into specific flows and allows administrators to ensure subscriber traffic is only permitted access to those services or appliances which are required for them to gain WAN access. Additionally, such an approach would see the loss of the controller's ability to manually expire flows based on the receipt of a DHCP Release packet. If the DHCP Release packet is no longer received by the controller and instead is forwarded immediately through to the DHCP server, deliberate IP lease expiration will happen without the knowledge of the controller. Subsequently, leases may expire independent of expiring the associated flows, which over time may lead to flow tables filled with inactive, unused flows. Given the advantages of the installation of a single DHCP control flow, this avenue should be investigated further. However, it would require the re-engineering of this controller further to accommodate for the problems that it would introduce.

5.1.2 Virtual Local Area Network

Evaluation

One weakness of using a configuration file to determine individual VLAN port configurations is the reduced ability to make dynamic changes to VLANs in an OpenFlow-enabled network. This is because VLAN configurations are only applied to specified ports at the time that the controller is originally initialised. Subsequently, any changes to port specific VLAN configurations (either major or minor) will require the entire controller to be re-initialised or restarted. While this is not a substantial undertaking and only carries the disadvantage of delaying flow creation by the time it takes the controller to restart, it can make network administration problematic. Network administrators are currently in the position to determine port configurations on any commodity switches independent of other switches, and can immediately have those changes applied in a production network environment. Although an authoritative controller architecture introduces the advantage of centrally managing switches, the ability to easily deploy any changes immediately is lost. In order to preserve both characteristics using the developed controller, a mechanism would need to be implemented to allow the dynamic VLAN modification of switch port configurations.

Unfortunately, full VLAN support that accommodates the preservation of VLAN information when packets are intercepted and forwarded on by the controller was not implemented. This feature would require the controller to determine which ports or hosts specific received packets were permitted to be forwarded to, based on the VLAN configuration of the port or host the packet was received from and the port or host the packet was otherwise destined for. This could be accomplished with some work by using the VLAN configuration file and a controller forwarding table to determine whether a packets identifying characteristics would be permitted en route to its destination, as well as what outgoing port VLAN configuration must be applied the Ethernet frame encapsulating the packet. Additionally, greater support for more complex functionality (i.e. 802.1ad VLAN stacking) could be implemented once the VLAN support in both Ryu and OpenFlow is improved.

Discussion

Attempting to dynamically modify switch port VLAN configurations is more difficult than it seemingly appears. It is not simply a challenge of discovering the best way to deploy any port VLAN configuration changes to an active controlled switch in a production network. The other important consideration requiring investigation is how any VLAN configuration changes alter or impact existing installed flows in the switch flow tables. Modification or removal of VLANs from ports may prevent previously existing flows from successfully forwarding packets through to specific end points. One way to accommodate this situation is a controller function which compares the existing flow tables and determines where VLAN configuration changes will impact those flows, triggering handler or modification events in response. Such events may include immediately expiring previously matching flows by overwriting these existing flows with identical flows with an immediately expiring flow timeout.

This would force the flows to be subsequently re-established via the controller, which would be aware of the updated VLAN configurations. If any former flows were still actively forwarding packets when they were forcibly expired, they would be automatically re-established upon receipt of the first packet not already matching a remaining flow. At this point the controller could receive these packets and create new flows in accordance with the updated VLAN configuration. Upon successful flow creation, regular operation would resume.

5.2 Dynamic Flow Creation

Evaluation

The subscriber database for dynamic flow creation is currently accessed directly through the controller. Depending on the flow-creation rate, this could introduce a bottleneck in the developed architecture, especially since the controller platform is currently not distributed. The controller would be required to sequentially query the database a substantial number of times for each unhandled subscriber packet that reaches it in order to accommodate flow creation for each associated subscriber. Given the poor throughput performance of the Ryu controller in high rate flow creation tests [33], placing additional operational strain on the controller by forcing it to interface with a subscriber database on a flow-by-flow basis may further compound the performance problems. A potential solution is rewriting the controller logic to dynamically create flows for all known subscribers during controller initialisation instead of during operation. This in itself is not a perfect solution however as there is potential for the installation of flows to accommodate subscribers who may not be currently active but none-the-less remain in the database. This additionally results in increasing the number of flows being installed in flow tables. It may be possible to use multiple fallback flow tables to accommodate these high numbers of installed flows, but this requires further investigation. Furthermore, a mechanism would need to be developed to update the controller with any new or modified subscriber handling conditions which may be inserted into the database after the controller has already been initialised to ensure flow tables and the database subscriber flows are synchronised.

Discussion

In a real world carrier network, there may not always be an authoritative subscriber database from which to draw information about how to handle specific subscribers. More commonly, this information can be sourced from existing AAA services. The implemented subscriber database presented in this project should be considered as a supplemental extra that accommodates the exchange of information for large numbers of subscribers on a wide range of different services. Depending on how subscribers are handled by their respective service AAA server, a central subscriber database may help by introducing a central point to manage subscribers across all services. Such an extension would allow a subscriber database to function as a modular add-on to existing AAA services, while supporting subscriber management through a single authoritative source. For instance, a subscriber database could interface with a DHCP server to facilitate the leasing of static IP addresses that have been deliberately specified in the database for a given subscriber. This configuration would be applied to the DHCP server each time a change was made in the database and would apply to subscribers once the subscriber made an IP address lease request.

5.3 Project and Technology Limitations or Difficulties

There were a number of limitations and difficulties discovered during the implementation of the project. These manifested as issues with the protocol support of the simulation environment, technical limitations of the software controller framework, and common developmental issues with documentation and bugs. This section details these limitations and difficulties and their overall effect on the project outcomes.

5.3.1 802.1ad VLAN Support

The OVS virtual switch is an ongoing work in progress which aims to enable network automation in a programmatic way, facilitating the innovation of smarter ways for networks to operate by network operators and researchers alike. Much like OpenFlow, new features and supported protocols are continually being introduced regularly into OVS with subsequent version updates. However, one notable omission to the OVS framework as of the writing of this report is support for the 802.1ad Ethernet standard [30], also known as provider bridging, stacked VLANs, or Q-in-Q. The 802.1ad standard introduces the ability to append multiple VLAN tags onto a single Ethernet frame in a standardised way, allowing for more complex network topologies. Carrier networks that support 802.1ad enable a service provider more flexible forwarding capabilities such as the ability to separate, encapsulate and transport customer VLANs across a service provider network without compromising the ISPs network operation. Stacked VLANs is commonly used in carrier networks to support metropolitan Ethernet configurations, much like the New Zealand UFB Fibre network. Stacked VLANs also gives network operators the ability to receive multiple VLAN-tagged Ethernet frames which can be used to distinguish the upstream provider, handover port or service that a given Ethernet frame was received on. Subsequently, sequences of identifying VLAN tags can be used to determine specific subscribers and the network path that their traffic has taken to arrive at the ISP. A visual representation of this topology from a UFB Retail Service Provider (RSP)'s perspective is provided in Figure 5.1.

Figure 5.1 depicts a point-to-point connection occurring over an optical fibre access network. A CPE sits at the edge of the access network and acts as the point-of-presence for the subscriber in the access network topology. Traffic that leaves the CPE may be encapsulated with a customer VLAN identifier (CVID) tag, meant to be handled at another remote location. An optical network unit (ONU) switch aggregates the optical connections from each subscriber's Op-

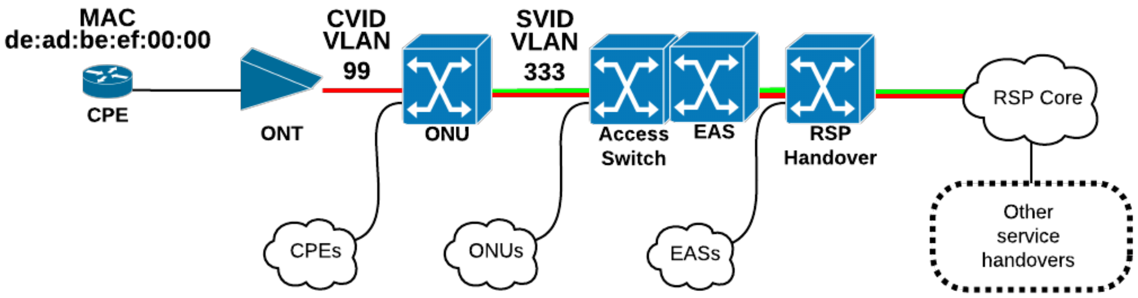

Figure 5.1: 802.1ad operation in an ISP network

tical Network Terminator (ONT) within in a common geographic area. Once a session hits the ingress port of the ONU it is affixed with a Service VLAN Identifier (SVID) or SVLAN tag. These sessions are then transported through the local fibre carrier (LFC) network where decisions about where to forward the traffic can be made at each of the intermediary access switches or Ethernet aggregation switches (EAS) (based on the VLAN tags). Finally, frames from the originating session are received at the RSP handover as doubly-tagged SVID:CVID frames. These frames appear to the RSP at this point in the configuration depicted in Table 5.1.

(8 Bytes) Preamble	(6 Bytes) Destination MAC	(6 Bytes) Source MAC
...	ff:ff:ff:ff:ff:ff	de:ad:be:ef:00:00

(4 Bytes) 802.1q SVID Header	(4 Bytes) 802.1q CVID Header
0x88a8 0x014d [333]	0x8100 0x0063 [99]

(2 Bytes) EtherType	(n Bytes) Payload	(4 Bytes) CRC/FCS
0x0806	...	...

Table 5.1: Basic 802.1ad VLAN-stacked ARP packet

At this point, the traffic can be forwarded across the RSPs network to the intended recipient. Using a series of tags in this fashion enables a BNG to pop each tag individually and make subsequent decisions based on their values. For instance, the BNG may have a series of rules to say that if the original CVID tag is VLAN ID 1, then to treat it as regular traffic. Otherwise, if the CVID tag matches a specific value, such as 99, to forward the frames to a specific location. The ability for a developed system to accommodate 802.1ad for this purpose is critical for service providers that provide point-to-point or WAN solutions for subscribers.

Although OVS does support the ability to accommodate double-tagged frames in a non-standardised pre-802.1ad way, this can lead to issues in some ISP networks. Many LFC network handovers present egress frames to RSPs as 802.1ad frames, and due to 802.1ad compatibility issues, RSPs then modify the 802.1ad VLAN header to convert it into a regular 802.1q VLAN header, in order for that frame to be supported by the RSP network. Before a 2011 amendment, the 802.1q standard determined that in the tag control information (TCI) portion of the VLAN header (between the EtherType and VLAN ID), the bit immediately preceding the VLAN ID (called the Canonical Format Indicator bit, or CFI) was used for indicating the compatibility of the tagged frame for older legacy networks. In an Ethernet network, this bit would always be set to 0. In 802.1ad, this same bit (now named the Drop Eligible Indicator bit, or DEI) is used for a different purpose, namely determining whether any QoS policy (in the preceding bits of the TCI) should be used to ascertain whether frames are eligible to be dropped by a switch in the presence of network congestion. In an ISP or RSP network with equipment conforming to the pre-2011 802.1q VLAN standard, dynamically changing an 802.1ad header to a 802.1q header in the aforementioned way can cause problems. If a network administrator is not careful, changing an 802.1ad DEI bit into a 802.1q CFI bit can cause any frame specific QoS policy to behave unpredictably. Given it is common to use VLAN CLI bits to determine network segment QoS policy, by conforming to the 802.1ad (or post 2011 802.1q) standard, OVS is able to support this standardisation for network interaction between handovers.

We assume that real world switches will not be subject to the same inherent VLAN restrictions that OVS currently is. While there are workarounds to implement a VLAN configuration that supports multiple VLAN tags in a similar mechanism to 802.1ad in OVS, these will not be necessary in a real carrier network assuming the existence of modern layer 2 switching hardware. The lack of support for the 802.1ad standard is an inherent limitation of using OVS as the simulating layer 2 switching fabric for this project. Subsequently, realistic and standardised ISP carrier network topologies using 802.1ad were not able to be simulated for this project.

5.3.2 Controller Scalability

The Ryu controller framework does not currently support multi-threading. This is an inherent limitation to the project as without being able to scale across multiple CPU cores, the controller may be unable to keep up with the throughput demands of a moderately sized carrier network. [33] demonstrated that the Ryu controller framework was less suitable for enterprise deployment after using throughput and latency metrics to determine whether Ryu was able to scale to meet high network demands. These tests were performed across 32 switches with up to 100,000 hosts. The results indicated that the failure of Ryu to accommodate these high numbers (when compared to other controller frameworks) was a lack of multi-threaded capability. The best controllers sampled in the same research were multi-threaded and outperformed Ryu due to their incoming message thread dispatcher algorithms. These controllers saw a higher degree of throughput performance as their CPU cores scaled to the number of connected switches. Such a positive correlation is impossible to achieve with the current iteration of Ryu as the number of CPU cores utilised will always be one, no matter how many switches are being actively managed. Subsequently, the use of Ryu to create a carrier network distributed BNG architecture is not feasible due to Ryu's inability to handle large numbers of hosts and switches.

5.3.3 Controller Distributivity

As briefly discussed in Section 3.1.1, there does not currently exist any mechanism using the Ryu controller framework to achieve a truly distributed series of OpenFlow controllers. Although there is some consolation in the ability to introduce redundancy into the controlling framework by use of a master/slave configuration and Zookeeper, a distributed controller framework would be preferable to implement in order to address the controller scalability issues as discussed in Section 5.3.2. A single Ryu controller possesses the ability to access a single CPU core at any given time, but extending that capability with a distributed series of Ryu controllers presents an opportunity to address the challenges associated with using a single Ryu controller and typical carrier network throughput. This is one reason why the lack of controller distributivity is a significant project limitation, as it continues to restrict the capability to introduce a system which will scale appropriately to carrier network requirements. Additionally, the ability to introduce distributivity into the Ryu framework offers not only high availability, but also efficiency and speed advantages. While a prototype of a SDN-enabled distributed BNG architecture is possible without controller distributivity, its omission is a notable limitation of the capabilities of a new BNG architecture. The basic differences in capabilities between a distributed controller model and a redundant controller model are shown in Table 5.2.

	High Availability	Scalability	Speed	Monitored	Implementation Complexity
Distributed Controllers	x	x	x	?	Very Complex
Redundant Controllers	x			x	Simple

Table 5.2: Distributed vs. redundant controller comparison

Table 5.2 indicates that distributed controllers introduce a greater feature set than simple redundant controllers. Specific disadvantages of the Zookeeper-facilitated redundant model are failures to address scalability of the system (by having only a single controller handling all active subscriber flow creation at any given time) and the inability to leverage speed increases by deploying closer to the physical locations of the subscriber base, subsequently reducing flow creation and operational latency. These serve as limitations as the developed controller will be much slower than the BNG appliance that it is attempting to improve upon. The Zookeeper model does however offer the ability for monitoring systems to gauge the health of a redundant controller system through external event alerting which is helpful for system administration.

Creating a distributed controller framework using Ryu would be a large research undertaking in its own right. Fortunately, there have been indications that such research is currently under investigation. In the meantime, using Ryu with Zookeeper appears to be the only current approach to high availability, leaving remaining limitations unaddressed.

5.3.4 API Documentation

As is common when working with some APIs, the lack of extensive documentation for various components and API calls of the Ryu controller was a source of difficulty. The Ryu controller framework and codebase is documented well and there is a lot of support available through various forums and mailing lists. However, difficulties did arise when developing using software functionality that was not extensively documented and was only briefly mentioned in official resources. This led to a trial and error, guess and check development methodology for a period of time, which was not effective or productive. Specifically, these difficulties affected the ability to easily de-capsulate the contents of a packet such as both the headers and payloads of encapsulated datagrams, as well as dynamically extract variably sized DHCP options from DHCP payloads as noted in Section 5.1.1. Additional difficulties were experienced when attempting to follow documentation that was not correct, leading to delays when specific software controller functions or methods did not work as documented or expected. In some instances, the Ryu documentation was too vague about specific implementation details, which necessitated referencing back to the OpenFlow switch specification to understand why Ryu functioned the way it did.

5.3.5 Bugs and Software Regression

Ryu and OVS are open source projects and attract much attention from a wide audience. Contributors to these projects range from researchers and academics to enterprise network architects and industry experts. While most interest in the project is of substantial benefit to each respective development community, it can lead to software bugs and regression. The basic definition of software regression is a software bug which stops a feature of a system working after some system event, such as an upgrade or patch. With such a large base of contributors, it is easy for unintended behaviours to be programmed into software by accident. This is despite the use of revision control systems and patch acceptance policies associated with both the OVS and Ryu projects. Software regression did unfortunately occur with the use of OVS in this project. The apparent reintroduction of a bug in OVS caused difficulties with unpacking full IP packet payloads. The problem manifested as the truncation of unbuffered packets received by the Ryu controller framework which occurred when the priority of the flow corresponding with the delivery of those packets was 0 (a priority value typically reserved for packets which do not match any other flows, known as a 'table miss'). Unbuffered packets in this instance are those packets which are deliberately sent fully intact to the controller for processing, as opposed to buffered packets which only contain a limited portion of the overall packet in order to increase controller processing efficiency. For specific protocol packets where there is required information after the observed 128 byte truncation point, this presented substantial development problems. DHCP packets were particularly affected, as the DHCP header is variably sized and a number of valuable variable length DHCP options are located in the tail of that header. This was fixed by changing the priority value of the corresponding DHCP traffic flows from 0 to any other priority. This bug was subsequently re-reported to an OVS developer.

Chapter 6

Conclusions

6.1 Contributions Made

This project has investigated the ways in which OpenFlow may be used to improve upon and address existing weaknesses and vulnerabilities in the existing centralised BNG model. A basic design was implemented and presented that offers software controller support for two basic subscriber session establishment protocols. Additionally, a conceptual subscriber database was introduced to improve the way in which session establishment protocols interface with AAA services in order to dynamically forward subscriber traffic using OpenFlow. Along with these proposed OpenFlow controller designs which address some basic distributed BNG concepts, this project has also investigated a number of approaches and limitations which may be used to realise a fully operational OpenFlow-enabled distributed BNG solution in the future, as the technologies involved mature.

6.2 Future Work

Given the scope of the implemented components of this project, there is a large amount of subsequent work that can be carried out to develop a controller framework capable of functioning as a distributed BNG and operating a basic production carrier network. Many of these facets were not simply overlooked during development, but remain as important considerations for future work. Given the scope of a fully realised and implemented project capable of replacing a network model as inherently complex and well-iterated as the

BNG, much work remains to develop a fully working prototype. The discussion of several of these remaining required components for developing a OpenFlow-enabled distributed BNG are detailed below. Many of them may be able to form the basis of projects such as this in their own right.

6.2.1 Additional Protocol Support

The OpenFlow protocol is a constantly evolving entity, which can be attributed to the operational flexibility OpenFlow affords to network administrators and the increased interest in network virtualisation. Given the interest in the protocol and SDN concepts, new features and supported protocols are being introduced into the standard on a regular basis. Subsequently, it is difficult to predict whether any introduced future features may benefit the approaches discussed in this project and, if they do, what the timespan to introduce those features may be. At the time of writing this report, the OpenFlow version 1.5 specification was still being written, and version 1.4 is still not yet well implemented. Open Networking Foundation (ONF) members have suggested that this version of the specification will contain greater levels of support for carrier network SDN, including support for carrier network OAM and the underlying technologies that enable it [16]. Once version 1.5 of the standard is supported in Ryu in full, it is possible that some of the underlying necessary BNG supported protocols which are absent in existing OpenFlow specifications (as discussed in Section 3.1.2) will be introduced in some capacity. This could make an OpenFlow-enabled distributed BNG model for ISP carrier networks more feasible by offering more complete support for the wide range of protocols required to support media-agnostic modern-day ISP carrier networks. Other alternative options include the investigation or extension of the ROFL library and bringing its capabilities to OpenFlow, OVS and Ryu. Access to the protocols supported in ROFL could allow this project to further evolve into a working prototype.

6.2.2 Flow Management and Operational Robustness

Networks have traditionally been able to operate dynamically and autonomously. This is due to the fundamental concept of connectionless networking, which determines how the transmission of data is achieved between two network end points for which no dedicated paths exist. It is an important concept to consider when designing traffic flows as depending on the nature of the flows, it is possible that specific network hosts are required to be traversed for end-to-end connectivity. Additional work is required to retain this important connectionless networking property and the overall operational robustness of the developed system. Many of the flows created by a distributed BNG controller will assume full end-to-end connectivity between network end points and may be installed in flow tables of any intermediary devices as such. The problem with this approach is the assumption that those hosts will remain up and operational, given that if any of the intermediary devices between two end points were to change or be lost, some end-to-end flows may fail to continue to forward traffic correctly. A dynamic flow-based system will be necessary to ensure the loss of any intermediary hosts in an end-to-end path does not affect the ability for the system to function correctly and forward traffic accordingly.

In the context of subscriber sessions and any corresponding installed flows, control mechanisms such as keep alives and timeout mechanisms still need to be investigated. Each OpenFlow enabled device in a network topology will have inherent restrictions on the maximum number of flows it is able to store. Assuming that there is a positive correlation between the number of active subscribers being aggregated through a distributed BNG and the number of flows required to manage them, scaling to accommodate every single subscriber may eventually be difficult. Subsequently, dynamic methods to manage sessions or flows that are not currently active should be implemented. These methods will ensure that subscribers who are not currently active are expired from the system, using mechanisms such as DHCP lease expiration discussed in Section **??**. A subscriber re-establishing a connection through the BNG system can be facilitated each time from scratch with minimal extra effort, so there is no problem with liberally expiring sessions.

6.2.3 Additional Authentication, Authorisation and Accounting Support

AAA servers in an ISP carrier network often operate as a function of the BNG appliance itself or as an external appliance which the BNG connects to and interfaces with. Since the BNG functions as a common aggregation point for subscriber traffic, this is a sensible architecture. AAA servers are required to authenticate and authorise subscribers with respect to permitted or accessible network services, but also must account for the usage of those services by each subscriber. The proof of concept AAA approach presented in Section 4.2 has been designed as an authoritative subscriber database. While useful for making dynamic traffic forwarding decisions, this database does not function as an extension to existing AAA services, but rather as a supplemental service. Substantial work is required to extend the developed controller AAA function endpoints into a modular API-based design that allows for multiple protocols to be supported, much like a BNG appliance would. Such an approach would allow for the extensible support of different carrier network configurations with a number of different AAA services. This subsequently would help accommodate varying kinds of production carrier networks present in modern day ISPs.

6.2.4 Complex Network Topologies

Aside from the technical limitations of this project that were discussed in Section 5.3, there are a number of additional project components remaining which would require development to fully realise a distributed BNG implementation. Due to the developmental time constraints, expanded network topologies of multiple layer 2 switches and upstream network endpoints were not tested or evaluated during the development cycle. To support these topologies intermediary devices would need a way to determine the logical layout of the network, as they would allow each OpenFlow-enabled switch to dynamically understand how best to forward Ethernet frames to the correct end points. Additionally, a large network of layer 2 switches more accurately represents an ISP carrier network than the simple single switch topology used in the initial stages of the development cycle. This single switch topology does not require OpenFlow-enabled switch to switch communication, or introduce multiple upstream endpoints. Once the project demonstrates a capability to forward Ethernet traffic within a larger topology such as this, it would be closer to developing a solution which could be used in a basic production carrier network.

6.2.5 Real World Testing and Evaluation

An important component of developing a distributed BNG implementation is the testing and evaluation of any developed solution on real-world hardware with realistic production traffic. However, the use of different real-world OpenFlow-enabled switches or production traffic flows may have the possibility of introducing unexpected or unhandled behaviours to the controller. A well-structured and thorough evaluation of a variety of BNG use cases is a necessary approach to evaluate the performance and practical feasibility of any developed solution. Fortunately, OpenFlow is an enabler of such evaluations, permitting the execution of experimental protocols and implementations on real-world production networks alongside regular network operation [18]. Using such an abstract approach to evaluate experimental network modifications enables network researchers and administrators alike to design comprehensive unit-based tests for their experiments. These unit-based tests can be based on any facet of the experiment design and, in the context of this project running on a carrier network, could be used to individually assess and evaluate specific protocols, processes or functions of the developed implementation. Unfortunately, due to developmental time constraints, testing of this nature was not carried out for this project. To measure the success of this project or any of the approaches outlined in this report, real world testing would be necessary.

6.3 Conclusions

This project has focused on the operational flexibility provided to network administrators by the use of OpenFlow, via its centralised software controller architecture. Centralised software controllers provide the ability to dynamically reconfigure switches in order to govern how traffic is forwarded, which can be leveraged to explore and investigate potential solutions to shortcomings or vulnerabilities in existing network protocols and models. Given the limitations discussed, it is not currently possible to introduce a comprehensive, scalable and complete distributed BNG solution using the OpenFlow controller software

Ryu and OpenFlow version 1.3. However, some of the smaller controller designs and concepts introduced in this project are suitable for governing session establishment mechanisms in smaller network environments, and can act as a starting point for future development of carrier-grade SDN components. The shortcomings of developing a complete distributed BNG solution are due to the lack of scalability and carrier-grade high availability as a result of the inability to distribute the functionality of the Ryu software controller among multiple controllers, as well as necessary session establishment and network communication protocols being not fully implemented in either OpenFlow or Ryu yet. However, if these problems were to be addressed (or a different, more suitable controller software framework chosen), the developed OpenFlow-enabled distributed BNG architecture would be viable.

References

[1] ATM End-to-End Working Group. Core Network Architecture Recommendations for Access to Legacy Data Networks over ADSL (TR-025). Technical report, Broadband Forum, September 1999.

[2] Biondi, P. Scapy. http://www.secdev.org/projects/scapy/, n.d. [Accessed 1 October, 2014].

[3] M. Davy, G. Parulkar, J. van Reijendam, D. Schmiedt, R. Clark, C. Tengi, I. Sekar, P. Christian, I. Cote, and G. China. A Case for Expanding OpenFlow/SDN Deployments On University Campuses. White paper, Global Environment for Network Innovations, June 2011.

[4] J. Dix. Open Networking Foundation (ONF) Executive Director on the group's achievements, goals. http://www.networkworld.com/article/2683312/opensource-subnet/open-networking-foundation-onf-executive-director-on-the-groups-achievements-goals.html, September 2014. [Accessed 30 September, 2014].

[5] D. Erickson. The beacon openflow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM.

[6] Floodlight Contributors. Floodlight SDN OpenFlow Controller. https://github.com/floodlight/floodlight, December 2011. [Accessed 28 September, 2014].

[7] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, April 2012.

[8] Google Inc. Inter-Datacenter WAN with centralized TE using SDN and OpenFlow. https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-googlesdn.pdf, July 2012. [Ac-

cessed 1 June, 2014].

[9] P. Goransson and C. Black. *Software Defined Networks - A Comprehensive Approach*, volume 1. Morgan Kaufmann, first edition, 2014.

[10] Open Source Software Computing Group. Ryu, component-based software defined networking framework. http://osrg.github.io/ryu/, 2013. [Accessed 20 March, 2014].

[11] R. Groves and B. Benetti. Microsoft's Demon - Datacenter Scale Distributed Ethernet Monitoring Appliance. Presented during Sharkfest '12 - Wireshark Developer and User Conference, Berkeley, California, 2012.

[12] C. Hellberg, D. Greene, and T. Boyes. *Broadband Network Architectures - Designing and Deploying Triple-Play Services*, volume 1. Prentice Hall, first edition, 2007.

[13] M. Kind, F. Westphal, A Gladisch, and S. Topp. Splitarchitecture: Applying the software defined networking concept to carrier networks. In *World Telecommunications Congress (WTC), 2012*, pages 1–6, March 2012.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[15] libpcap Contributors. the LIBpcap interface to various kernel packet capture mechanism. https://github.com/the-tcpdump-group/libpcap, October 1999. [Accessed 1 October, 2014].

[16] B. Mack-Crane. OpenFlow Extensions. Presented during the 2013 ONF and US-Ignite SDN Workshop, Sunnyvale, California, October 2013.

[17] C. Matta. Users Equal Distribution on Multi-PPPoe Servers - Using RouterBoard and RouterOS. Presented during the 2012 Mikrotik User Meeting, Dubai, UAE, August 2012.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. White paper, Open Networking Foundation, March

2008.

[19] NEC Corporation. Case Study - Kanazawa University Hospital. https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-nec.pdf, 2012. [Accessed 28 September, 2014].

[20] Network Working Group. Remote Authentication Dial In User Service (RADIUS). Request for comments, Internet Engineering Task Force, June 2000.

[21] Network Working Group. Diameter Base Protocol. Request for comments, Internet Engineering Task Force, September 2003.

[22] Open Networking Foundation. OpenFlow Switch Specification — Version 1.0.0 (Wire Protocol 0x01). Specification, Open Networking Foundation, December 2009.

[23] Open Networking Foundation. OpenFlow Switch Specification — Version 1.1.0 (Wire Protocol 0x02). Specification, Open Networking Foundation, February 2011.

[24] Open Networking Foundation. OpenFlow Switch Specification — Version 1.2.0 (Wire Protocol 0x03). Specification, Open Networking Foundation, December 2011.

[25] Open Networking Foundation. OpenFlow Switch Specification — Version 1.3.0 (Wire Protocol 0x04). Specification, Open Networking Foundation, June 2012.

[26] Open Networking Foundation. OpenFlow Switch Specification — Version 1.3.2 (Wire Protocol 0x04). Specification, Open Networking Foundation, April 2013.

[27] Open Networking Foundation. OpenFlow Switch Specification — Version 1.4.0 (Wire Protocol 0x05). Specification, Open Networking Foundation, October 2013.

[28] Open Networking Foundation. Extensibility. https://www.opennetworking.org/working-groups/extensibility, 2014. [Accessed 30 September, 2014].

[29] OpenDaylight Contributors. OpenDaylight — A Linux Foundation Collaborative Project. http://www.opendaylight.org/, 2014. [Accessed 28 September, 2014].

[30] B. Pfaff. Open vSwitch Manual. Manual, Open vSwitch, n.d. [Accessed 5 October, 2014].

[31] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII). New York City, NY (October 2009)*.

[32] RYU project team. *RYU SDN Framework - Using OpenFlow 1.3*. Open Source Software Computing Group, 2014. [Accessed 1 October, 2014].

[33] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, pages 1:1–1:6, New York, NY, USA, 2013. ACM.

[34] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester. Software defined networking: Meeting carrier grade requirements. In *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, pages 1–6, Oct 2011.

[35] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[36] Agilent Technologies. Understanding DSLAM and BRAS Access Devices. White paper, Agilent Technologies, February 2006.

[37] The Apache Software Foundation. The Apache Cassandra Project. http://cassandra.apache.org/, 2009. [Accessed 6 August, 2014].

[38] The Apache Software Foundation. What is Zookeeper? http://zookeeper.apache.org, 2010. [Accessed 1 October, 2014].

[39] The Architecture and Transport Working Group. DSL Evolution - Architecture Requirements for the Support of QoS-Enabled IP Services (TR-059). Technical report, Broadband Forum, September 2003.

[40] The Architecture and Transport Working Group. Broadband Remote Access Server (BRAS) Requirements Document (TR-092). Technical report, Broadband Forum, August 2004.

[41] The Architecture and Transport Working Group. Migration to Ethernet-

Based DSL Aggregation (TR-101). Technical report, Broadband Forum, April 2006.

[42] The SPARC Consortium. SPARC - Split Architecture Carrier Grade Networks. http://www.fp7-sparc.eu/, n.d. [Accessed 2 October, 2014].

[43] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[44] Trema Contributors. Full-Stack OpenFlow Framework in Ruby and C. https://github.com/trema/trema, April 2011. [Accessed 28 September, 2014].

[45] H. Woesner and D. Fritzsche. Sdn and openflow for converged access/aggregation networks. In *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013*, pages 1–3, March 2013.