Leveraging Software-Defined Networking for QoS in home networks

AH Taute

orcid.org/0000-0003-4769-1593

Dissertation submitted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor: Dr M Ferreira

Graduation ceremony: May 2019

Student number: 24250058

Declaration

I, Anton Taute hereby declare that the dissertation entitled "Leveraging Software-Defined Networking for QoS in home networks" is my own original work and has not already been submitted to any other university or institution for examination.

AH Taute

Student number: 24250058

Signed on the 20th day of November 2018 at Potchefstroom.

Acknowledgements

"*And whatever you do, whether in word or deed, do it all in the name of the Lord Jesus, giving thanks to God the Father through Him.*" ∼ Colossians 3:17 (NIV)

First and foremost, I want to thank my God and Saviour, Jesus Christ. My life would be meaningless without Your grace and love. I am immensely grateful for all the blessings You give me.

To my supervisor, Dr. Melvin Ferreira, thank you for your guidance, patience and eagerness to help. I not only see you as my study leader, but also as a mentor and role model.

I would not be where I am today without the support of my family, and especially my parents, who are my biggest fans. Thank you for your encouragement and sympathetic ears during the tough times.

During all my years as a student, I was fortunate enough to always have my colleague, confidant and close friend, Hanco Buys, at my side. Thank you for your camaraderie and loyalty. I will miss our frequent tea and fingerboard breaks!

To the TeleNet research group and the other occupants of room 214 of the past two years, thank you for the chance to learn from you and keeping my spirits up; and especially G.J. Dyason, for enduring the late-night work sessions together.

My thanks and appreciation for John Lewis from OpenServe, for taking the time to provide me with valuable feedback and advice.

Lastly, thank you to the Telkom Centre of Excellence (CoE), for their financial support and giving me the opportunity to further my studies.

Abstract

Home networks refer to devices at the edge of the internet. These networks face unique provisioning challenges. As the home network is growing and more complex devices are added, the Internet Service Provider (ISP) may struggle to fulfil the specific requirements of each home user. This study investigates the advantages of Software-Defined Networking (SDN) and proposes a way the ISP can leverage it to improve the Quality of Service (QoS) and monitor the traffic in home networks. The SDN controller offers the ISP a centralised point of control that can potentially manage multiple home networks via SDN-enabled devices.

A comprehensive survey is conducted on existing work related to SDN implementations for home networks. It is found that most implementations require the home gateway router to be replaced with an SDN-enabled device. Scalability considerations are also rarely investigated. Based on these current shortcomings, an experimental setup that resembles a typical South African home network is designed and tested in the Mininet emulation environment. The Analytical Hierarchy Process (AHP) is used to identify the Ryu controller as the most suitable for the design, compared to six other open-source SDN controllers.

Experiments with different network traffic types are quantitatively compared using four QoS parameters, namely throughput, jitter, packet loss and round-trip time. Two different use cases are compared with each other, first where QoS is not implemented and second where specified traffic is prioritised by the SDN controller. The SDN switch (Open vSwitch) is configured using Ryu's REST Application Programming Interface (API) and the OpenFlow protocol. A Graphical User Interface (GUI) application is developed (using the `tkinter` package in Python) to offer a simple and easy way for the ISP to configure their clients' networks.

By using the statistics of queues installed on the SDN switch, the controller can monitor the network traffic of a home network. The scalability of this design is tested by emulating an increasing number of queues installed on OVS and measuring the amount of

overhead traffic between the controller and switch while the controller is monitoring the home network. Other controller performance metrics such as Central Processing Unit (CPU) use, memory use, flow installation time, fragmentation of the statistics reply packets and the delay times of the reply packets are also investigated.

The framework presented in this study forms a basis on which the ISP can build a platform to improve the QoS of their users' home networks.

Keywords: *Home networks, Internet Service Provider, Mininet, Monitoring, Network emulation, OpenFlow, Open vSwitch, Quality of Service, Ryu, Scalability, Software-Defined Networking*

Contents

List of Figures

List of Tables

List of Acronyms

ACS Auto-Configuration Server

ADSL Asymmetric Digital Subscriber Line

AHP Analytical Hierarchy Process

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

BGP Border Gateway Protocol

CI Consistency Index

CPE Customer Premises Equipment

CPU Central Processing Unit

CR Consistency Ratio

cURL Client Uniform Resource Locator

CWMP Customer Premises Equipment Wide Area Network Management Protocol

DSL Digital Subscriber Line

ECDF Empirical Cumulative Distribution Function

FIFO First-In, First-Out

FTTH Fibre To The Home

GUI Graphical User Interface

HTB Hierarchical Token Bucket

HTTP HyperText Transfer Protocol

IoT Internet of Things

IP Internet Protocol

ISP Internet Service Provider

JSON JavaScript Object Notation

MAC Media Access Control

MOS Mean Opinion Score

NAT Network Address Translation

NFV Network Function Virtualisation

NOS Network Operating System

NTT Nippon Telegraph and Telephone Corporation

ODL OpenDaylight

ONF Open Networking Foundation

OS Operating System

OSPF Open Shortest Path First

OVS Open vSwitch

PC Personal Computer

QoE Quality of Experience

QoS Quality of Service

RAM Random Access Memory

REST Representational State Transfer

RI Random Index

RSS Resident Set Size

RTP Real-time Transport Protocol

RTT Round-Trip Time

SDN Software-Defined Networking

SOHO Small-Office / Home-Office

SSL Secure Sockets Layer

TCAM Ternary Content-Addressable Memory

TCP Transmission Control Protocol

ToS Type of Service

UDP User Datagram Protocol

VLAN Virtual Local Area Network

VM Virtual Machine

VoIP Voice over Internet Protocol

VSZ Virtual Set Size

WSGI Web Server Gateway Interface

Chapter 1

Introduction

In this chapter, an overview of the study is given. It starts with an introduction to Software-Defined Networking (SDN) and home networks in section 1.1. The motivation to justify the need for this study is given in section 1.2. The research problem questions and objectives are outlined in section 1.3, with the method to answer and address these questions given in section 1.4. Lastly, section 1.5 provides an overview for the rest of the dissertation.

1.1 Background

As home networks grow and become more complex, it becomes increasingly complicated for the Internet Service Provider (ISP) to deliver Quality of Service (QoS) to their clients. Each home network is unique and has different requirements based on the needs of the users. The devices of some home users may require more bandwidth than others or a specific application may require priority over the others that utilise the internet connection of the home network. This could especially be a concern for real-time applications that are bandwidth-intensive, such as live video streaming and video conferencing.

One way in which an ISP can improve the QoS provisioning for home networks is to add more functionalities to their access networks. Software-Defined Networking (SDN) is a new approach to implement, operate and maintain networks. The concept of SDN was developed to make networks more configurable and flexible [2]. This is done by decoupling the data forwarding and logic control functionalities of network devices and adding an SDN controller that can configure several devices at once. This makes the network more programmable by providing a centralised point of control for the network.

Since its inception, various studies have tested SDN implementations, predominantly at core- and campus networks [3]. Large corporations such as Google have already utilised SDN to improve their data centres [4]. According to Haque & Abu-Ghazaleh [5], there are advantages to also integrate SDN technology with home networks. The centralised control plane can coordinate resource allocation to effectively and fairly utilise the available bandwidth. This allocation can take place across different home networks and also across applications within a single home network.

1.2 Motivation

There exist many advantages of an SDN approach over legacy networks, which an ISP may be able to leverage to improve the QoS of their clients' access networks. Service providers continually strive to deliver better QoS for their clients, to stay relevant and not be outperformed by their competitors.

A lot of research has already been done that utilises SDN at the home network [6]. However, there is a lack of research on scalability and robust software-enabled home networks [5]. Most solutions also require that all network elements be SDN-compatible, which is currently not a viable option for all home networks [3]. The integration of SDN and legacy network technologies are thus worthy to consider, both from the ISP's perspective (no new installations needed for all their clients) as well as for home users (no new technology to get used to).

1.3 Research Questions and Objectives

This study aims to answer the following research questions:

1. How can an ISP leverage SDN to improve the QoS for its home network clients who are still using their legacy network devices?
2. How scalable is an SDN design for the ISP to monitor their clients' home networks?

The following objectives are addressed to answer the research question:

- Research existing SDN implementations for home networks.
- Design a framework based on SDN that an ISP can use to implement QoS and monitor the traffic of its home network users while incorporating their legacy devices.
- Emulate the design framework to evaluate and verify its effectiveness.
- Quantitatively investigate QoS provisioning on the framework for different network traffic, and compare it with the case where SDN is not used.
- Quantitatively investigate the scalability of the framework while it is monitoring home networks.

1.4 Research Method

The method to complete this study is broken down in different steps, as shown in Figure 1.1. Firstly, a literature study is done on relevant subjects, with the focus on the different layers and interfaces present in SDN. The current shortcomings and needs of home networks are looked at, in the context of provisioning QoS. Different ways to

evaluate network implementations are discussed. The literature study concludes with a comprehensive survey done on other studies that either used SDN to monitor home networks or use SDN to provision QoS for home networks.

Based on the findings of the literature study, a system design is presented to address some of the identified shortcomings of the related work. The system design can be used as a framework for an ISP to monitor and provision QoS for home networks. The specific design choices are then considered and selected to be able to implement and evaluate the design. The Analytical Hierarchy Process (AHP) is used to decide between different SDN controllers and to identify the most suitable one for the system design. The interaction between the different design choices and technologies are explained as part of the design.

A suitable emulation platform is selected and used to evaluate the system design. Two different use cases are implemented and quantitatively compared with each other: one where QoS is not implemented and one where SDN is used to implement QoS for a home network. Different types of network traffic are also used to verify the system design. How the framework can be used for monitoring and so aid the ISP to improve QoS provisioning for home networks, are discussed and tested. The scalability of the design while the monitoring takes place, is evaluated by measuring different metrics of the SDN controller. Lastly, experiment validation is done by implementing a network traffic scenario and comparing the results with published research.

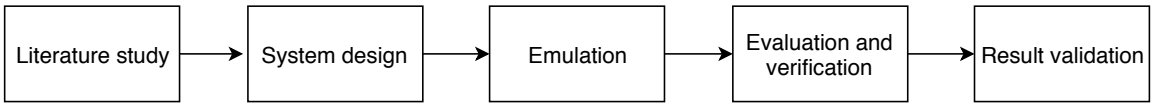


Figure 1.1: Research methodology

1.5 Dissertation Overview

The structure of the dissertation is as follows: Chapter 2 contains the information reviewed as part of the literature study. Chapter 3 then presents the system design, with each component, interaction between them and the specific design choices given. The design is emulated and evaluated in terms of its QoS provisioning capabilities, with the results presented in Chapter 4. Verification and validation are also performed in this chapter. The scalability results of the design while monitoring network traffic are given in Chapter 5. The dissertation is concluded in chapter 6 with a discussion about the key findings of the study and recommendations for future work to be done.

Chapter 2

Literature Study

In this chapter, aspects relevant to the study are researched. The concept of a home network is explained in section 2.1. An overview of SDN is given in section 2.2, including the various protocols, SDN-enabled switches and controllers. Several topics regarding QoS are discussed in section 2.3. An overview of how to evaluate network configurations is given in section 2.4. Finally, the survey of related work for this study is presented in section 2.5.

2.1 Home Networks

Home networks are located on the periphery of a centralised network. Usually, this refers to the devices at the edge of the internet and consists of a wide variety of wired and wireless end systems or hosts, such as desktop computers, servers, mobile computers (laptops, smartphones and tablets) and a wide range of Internet of Things (IoT)-enabled devices [7]. Home networks are also sometimes known as Small-Office / Home-Office (SOHO) networks or edge networks.

Access networks are the networks that physically connect home networks with their

first routers on the connectivity paths to other end systems. These end systems include home networks, mobile networks and enterprise networks [7]. The access network is also referred to as the demarcation point or the home gateway in the context of home networks [8]. Internet modems, Wi-Fi access points and home network routers are some of the common devices associated with the access network. These devices are also known as the Customer Premises Equipment (CPE).

Customers are connected to the core network through the access network. Core networks consist of links that interconnect different end systems with each other by providing a mesh of packet switches [7]. Common access network technologies to which core networks provide services (in particular internet access) to customers include Digital Subscriber Line (DSL), cable, dial-up, Fibre To The Home (FTTH) and Asymmetric Digital Subscriber Line (ADSL), which provides different upload and download rates [8]. An Internet Service Provider (ISP) uses the core network to connect their clients with the internet.

Figure 2.1 displays the differences and interfaces between the home, access and core network. An oversimplified representation of the ISP core network is given from a provisioning perspective that excludes, for example, the aggregation of metro layers.

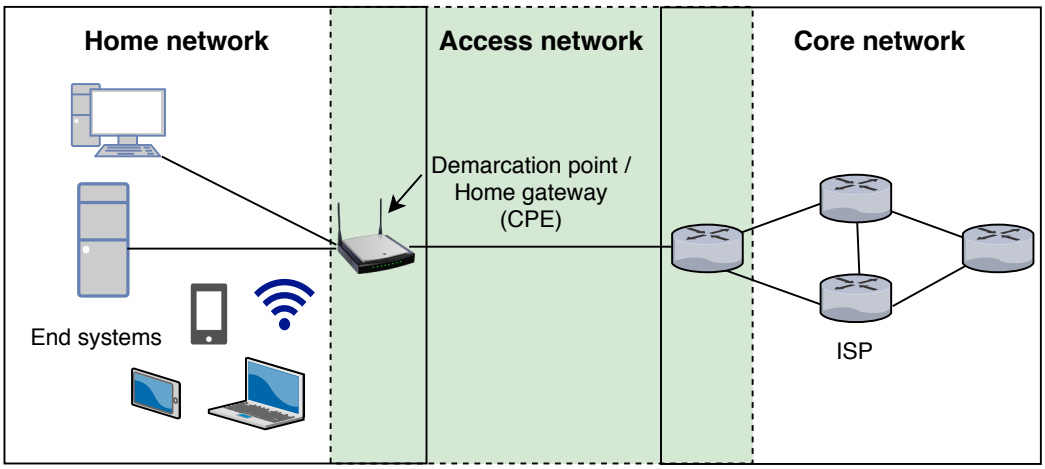


Figure 2.1: The home, access and core network

2.1.1 Problems in Home Networks

The management of home networks is faced with unique challenges. As more internet-enabled devices are produced and used, home networks become more complex and unmanageable [9]. It is therefore difficult to keep network access control and policies consistent, and it is common for a typical home network to be poorly managed, insecure and broken [10]. These networks are prone to failure with no measures in place to systematically improve services after deployment. Another significant issue when managing home networks is the low-level configuration required for different implementations, and the lack of technical knowledge among home users to accomplish this [6].

Monitoring of the different network traffic in home networks are not always accurate. A single test to measure the speed of an internet connection could likely report misleading results and have not much bearing on the network's long-term performance [8]. Better techniques are thus needed to monitor the different types of traffic on home networks, so that the ISP can provide the agreed-upon QoS to their clients.

Home networks often experience a bottleneck problem as the access network can become easily congested [11]. One way to address this problem is to prioritise the network traffic that is more important to the home user or the traffic that is sensitive to delay, e.g. Voice over Internet Protocol (VoIP) traffic, over the other traffic by implementing various QoS implementations (see section 2.3.3).

2.2 SDN

2.2.1 SDN vs Traditional Networks

Commercial switches and routers generally do not provide an open software platform or a way to virtualise their hardware or software. Thus the switch's internal features

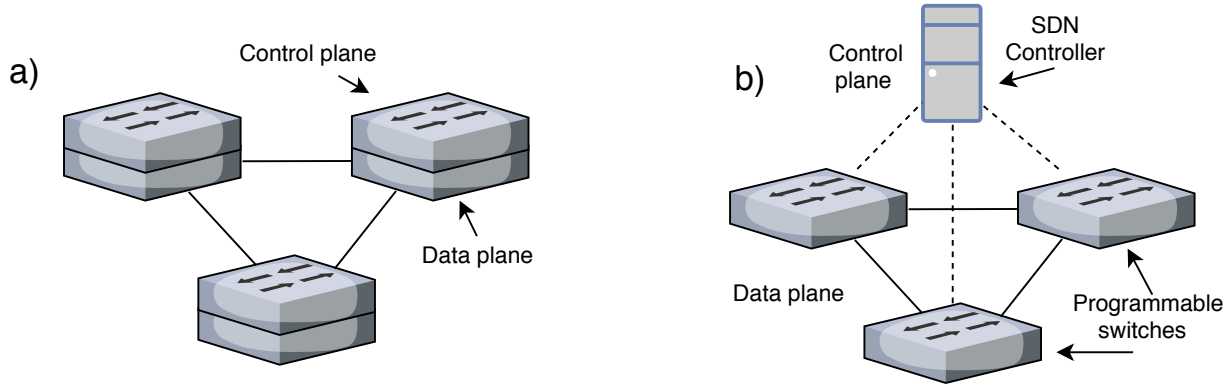


Figure 2.2: Comparison between a traditional (a) and SDN network (b)

are hidden, while the internals also differ between different vendors. Legacy network equipment and protocols also do not provide a practical way to experiment with and test new designs and network protocols in an adequate realistic configuration. This creates a barrier that prevents researchers from experimenting with new ideas and contribute to network innovation [12]. The SDN paradigm and, in particular, OpenFlow (see section 2.2.5), were developed to give a practical way to experiment with and test new network protocols and ideas.

Networks contain three functionality planes: the data plane that consists of the networking devices which forward traffic and execute policy; the control plane that consists of the protocols which handle the traffic and enforce policy; and the management plane that consists of software services which monitor control functions and define policy [13]. In traditional networks, the control and data plane logic are both present in the network elements (mainly network switches and routers). This means that each device must be configured separately.

SDN aims to make the network more reconfigurable and programmable by decoupling the data and control plane logic [2]. While the data plane logic remains within the network elements, the control plane logic is moved to an external controller. Thus, the fundamental difference between SDN and traditional (or legacy) networking, is the presence of an SDN controller which creates a centralised point of control. Figure 2.2 shows the difference between a legacy and SDN network with regards to the different

functionality planes [14].

The centralising of the network control function offers several benefits: modification of the network is less error-prone and easier to execute, high-level policies can be maintained by automatically reacting to changes in the network state, and the process of developing network servers, functions and applications are simplified [13].

The controller contains software, usually a Network Operating System (NOS) that allows it to create a virtualised abstraction of the entire network [13]. This makes it easier to program and modify the network, policies and applications with high-level languages.

Moving the control plane functionality from individual devices to a centralised device (controller) simplifies the network setup by removing the need for complex distributed control plane protocols such as Open Shortest Path First (OSPF), Border Gateway Protocol (BGP) and Spanning Tree. Instead, the controller determines the network topology and configures the table entries of the individual forwarding devices. The network traffic can also be split across multiple links, as the controller can discover multiple paths from the origin to the destination of a packet [15].

In the ONF White Paper [16], the need for SDN as a new network architecture is explained. The advantages of SDN over legacy networks is described as such:

> "In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralised, and the underlying network infrastructure is abstracted from the applications. As a result, enterprises and carriers gain unprecedented programmability, automation, and network control, enabling them to build highly scalable, flexible networks that readily adapt to changing business needs."

2.2.2 SDN vs NFV

Network Function Virtualisation (NFV) is the process of providing virtual abstractions for network services like load balancing and firewalls. These services are then executed

in software instead of in traditional hardware devices such as routers. While SDN focuses mainly on the centralised point of control and orchestration of network traffic through network automation (programmability of the control plane), NFV focuses on the abstraction of network services (programmability of the data plane) [17]. NFV can be implemented in traditional networks using existing orchestration paradigms. It is also possible to implement SDN without utilising NFV; an SDN controller can be used as a broker with existing legacy network devices to orchestrate the network via interacting Application Programming Interfaces (APIs).

SDN and NFV are thus not dependent on each other, but rather complementary technologies. NFV can support SDN by exposing functions of network devices that may become components in services that are orchestrated by SDN [18]. Figure 2.3 illustrates the area where SDN and NFV overlap and where they are distinct from each other.

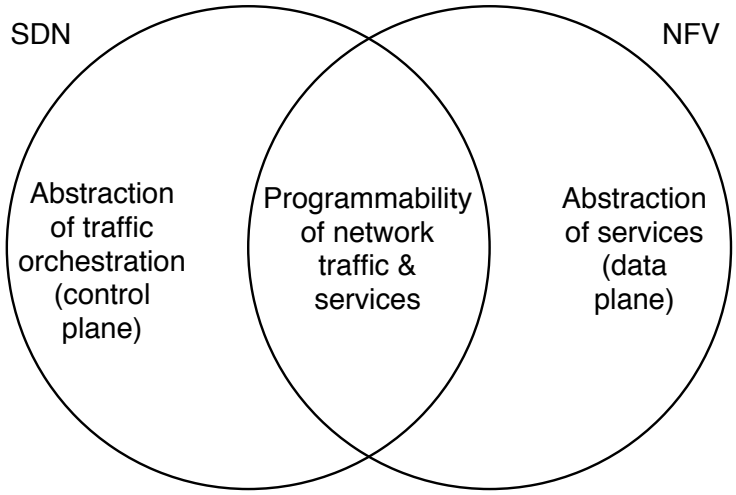


Figure 2.3: Interaction between SDN and NFV

2.2.3 SDN Layers

The architecture of software-defined networks is composed of different layers that are vertically connected, as shown in Figure 2.4 [13]. On the bottom is the hardware infrastructure, which consists of the forwarding devices and the data plane functionality. Any SDN-enabled device (e.g. switches) can be deployed in the network. Section 2.2.7 contains more information about SDN-enabled switches.

The southbound interface on top of the hardware layer forms the connection between the control and data plane elements. OpenFlow [12] is the most widely used standard for this interface and is a product of the Open Networking Foundation (ONF) [19]. Section 2.2.5 gives more information about OpenFlow.

The middle layer consists of the control plane functionality and the SDN controller, on which the NOS runs. The controller must coordinate the flow set-up as originated by the network applications and update each element to keep the network state consistent [13]. This provides logically centralised control for the network. An overview and comparison of different open-source SDN controllers are given in section 2.2.4. Network hypervisors can be used for virtualisation and enable different virtual machines to share the same hardware. Network slicing techniques are also implemented at this layer.

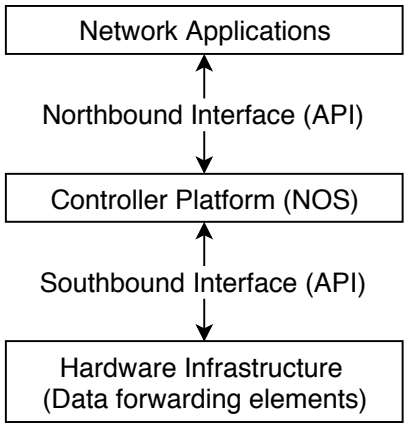


Figure 2.4: SDN layer architecture

On top of the controller is the northbound interface, which forms the connection between the control and management plane elements. There is no defined standard yet, and existing controllers usually propose or define their own APIs [3]. Language-based virtualisation is used to create abstractions of the network.

Above the virtualisation is network programming languages that are responsible for generating and installing lower level instructions at each network device. Network applications form the top layer and contain the management plane of the network. These applications can have features such as measuring and monitoring the network, providing security, performing traffic engineering to balance the load or minimising power consumption.

2.2.4 SDN Controllers

There is a wide range of SDN controllers that have been successfully implemented in different applications, including enterprise networks and research studies. An overview of some of the most used and popular open-source controllers is shown in Table 2.1, as adapted from [2], [13] and [20]. Some controllers were developed to aid in research and fast-prototyping implementations (Beacon, Maestro, NOX, POX and Ryu), while others are more suited for deployment in data centres or enterprise networks (Floodlight, MuL, OpenDaylight (ODL) and ONOS).

There have been several studies in which controllers are compared to each other [30] [31]. In [32], seven controllers are evaluated in terms of performance, reliability and security. The Beacon controller achieves the best average throughput with different number of threads. The MuL controller delivers the best latency result, but both MuL and Maestro had several failures during long-term testing when given a specified workload profile. The Ryu controller had the best security, as it passed four out of the five security tests.

The Analytical Hierarchy Process (AHP) is used in [33] to compare the POX, Ryu, Trema, Floodlight and ODL controllers against criteria such as interfaces, platform

Table 2.1: Overview of SDN controllers

Controllers	**Language**	**Developer**	**Northbound API**	**OpenFlow versions supported**	**Uses**
Beacon [21]	Java	Stanford University	Ad-hoc	1.0	Event-based & threaded operations, research
Floodlight [22]	Java	BigSwitch	REST	1.0, 1.1, 1.3	Enterprise networks, campus
Maestro [23]	Java	Rice University	Ad-hoc	1.0	Modular network control applications, research
MuL [24]	C	Kulcloud	Multi-level interface	1.0, 1.3, 1.4	Application development, data centres
NOX [25]	C++	Nicira	Ad-hoc	1.0	Campus networks, research
ONOS [26]	Java	Linux Foundation	REST	1.0, 1.3	Distributed data centres
ODL [27]	Java	Linux Foundation	REST, RESTCONF	1.0, 1.3, 1.4, various extensions	Data centres, enterprise networks
POX [28]	Python	Nicira	Ad-hoc	1.0	Fast prototyping, debugging, campus networks
Ryu [29]	Python	NTT OSRG group	Ad-hoc	1.0, 1.2, 1.3, 1.4, Nicira extensions	Fast prototyping, campus networks

support, productivity, documentation and modularity. Ryu scored the highest value at the end of the process.

Controllers can operate in three modes to add new flow entries to their connected switches [34]: reactive, proactive and hybrid mode. In proactive (or static) mode, flow entries are set up before new flows arrive at the switch. The controller is not involved in any new flow rule installation, as when a packet arrives at the switch, the processing action of that packet is already known to the switch.

In reactive mode, no flows are pre-programmed for the switches. When a new packet arrives, it is forwarded to the controller, which decides on the processing action for that type of packet according to network policy. The controller adds a new flow entry to the switch, to enable the switch to apply the corresponding actions for future packets that are matched to that flow. In hybrid (or proactive-dynamic) mode, flows are proactively and reactively installed.

Controller performance is a primary concern for the design of networks that are scal-

able [34]. The three main factors that contribute to scalability issues in SDN networks are the separation of the control and data plane (overhead traffic between the controller and network devices), the number of requests handled by the controller (can result in the controller becoming a bottleneck due to limited computation resources) and the delay of communication between the controller and switch (the round-trip time of the link that increases the flow setup latency).

2.2.5 OpenFlow

OpenFlow was developed at Stanford University as a new switch feature to extend the programmability of networks at college campuses [12]. The main research goals were to accomplish a high-performance switch with low-cost implementations and the ability to isolate experimental traffic from production traffic in an operational network.

OpenFlow uses flow-tables, generally built from the switch's TCAM, and utilises their functions that are common for most switches, such as implementing firewalls, Network Address Translation (NAT) and collecting statistics. A remote SDN controller manages the flow-tables by using the OpenFlow protocol via a secure channel. The controller can add and remove flow-entries from the switches' flow tables.

Each entry in the switch's flow-table has three fields (see Figure 2.5): a header that defines the flow, an action that defines the processing of the packet and statistics about the flow. The header has fields that are matched with the incoming packets. Header fields include the ingress port, source and destination Ethernet Media Access Control (MAC) addresses, source and destination Internet Protocol (IP) addresses, and source and destination Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) ports. Any of these fields can be a wildcard entry that will match with all packets on that particular field. Almost every new version of OpenFlow has added more matching fields to the header [35].

The action can be to forward the flow's packets to a given port, encapsulate and forward it to a controller or to drop it. The statistics keep track of the number of packets

Matching Rule									Action	Statistics
Switch port	MAC src	MAC dst	Eth type	VLAN ID	IP src	IP dst	TCP psrc	TCP pdst	1. Forward packet to port(s) 2. Encapsulate and forward to controller 3. Drop packet	Counters

Figure 2.5: Fields of an OpenFlow 1.0 flow-table entry

and bytes that each flow has matched on, as well as the time passed since the last packet matched [12].

Different types of OpenFlow protocol messages are exchanged between the controller and the switch, as described in the OpenFlow Switch Specification [36]. Handshake and switch configuration messages are used to establish the connection between the switch and controller. Multipart request and reply messages are used to retrieve various statistics from the switch. Symmetric messages, such as echo request and reply messages, are used to verify if the connection between the controller and switch is still established. Flow entries can be removed from the flow tables in two ways, either actively by the controller (using an asynchronous message) or automatically by the switch's flow expiry mechanism. When either the `idle_timeout` or `hard_timeout` values of a flow is exceeded, the flow entry is removed .

Various experiments have been done using OpenFlow to improve production networks, such as improving the management and QoS of networks, traffic monitoring of networks and implementing Virtual Local Area Networks (VLANs). In section 2.5, specific studies that use OpenFlow are discussed.

2.2.6 OpenFlow vs P4

P4 [37] is a programming language that is used to program switches. Where OpenFlow exposes the data plane (network devices) for the control plane to populate a set of well-known forwarding tables, P4 programs the switch directly. P4 can be imple-

mented either by an external controller or using the switch's operating system. Table 2.2 compares OpenFlow with P4 [38].

Table 2.2: Comparison between OpenFlow and P4

OpenFlow	**P4**
Exposes dataplane for control plane	Programs data plane directly
Populate forwarding tables in switch	Programs switch directly
Programmable and fixed-function switches	Only programmable switches
Requires external controller	Controlled by either switch OS or external controller

2.2.7 SDN-enabled Switches

There are several types of SDN switches available. Companies such as IBM, HP and NEC have released carrier-grade hardware switches for data centres or enterprise networks that are compatible with OpenFlow. Table 2.3 gives an overview of SDN-enabled switches, with the focus on open-source software implementations.

Open vSwitch (OVS) [39] is one of the most used virtual switches available. It supports standard management interfaces and enables the programming of the forwarding functions for network traffic. OVS can be ported onto Application-Specific Integrated Circuit (ASIC) switches and is available as a package [40] for the OpenWrt operating system [41].

OVS consists of three main components: ovsdb-server, ovs-vswitchd and the datapath kernel module (also knows as openvswitch_mod.ko). The ovsdb-server is a lightweight database server in user space that contains all switch configurations. The ovs-vswitchhd user space daemon implements the switch and queries the ovsdb-server for the configurations. It is responsible for flow lookup, port mirroring and VLAN implementation. The datapath kernel module is usually written specifically for the host operating system it is run on. It is responsible for packet lookup, flow modification and forwarding [42].

Table 2.3: Comparison of SDN switches, enhanced and adapted from [1]

Name	**Implementation Language**	**Developer**	**Description**
Open vSwitch [39]	C & Python	Open Community	Open-source switch platform implementation for virtualised servers. Can also be ported to multiple hardware platforms.
Indigo [43]	C	Big Switch Networks	Based on the Stanford reference that runs on hardware switches. Also supports hypervisor-based switches.
ofsoftswitch13 [44]	C & C++	Ericsson, CPqD	User space software switch, compatible with OpenFlow 1.3.
Zodiac [45]	C & C++	Northbound Networks	Claims to be the smallest OpenFlow hardware switch.

OpenFlow is used to control packet forwarding in OVS, by providing the protocol to allow communication between the ovs-vswitchd and an SDN controller. The controller can then add, remove and monitor the switch's flow tables, as well as reroute specified packets to the controller or sent packets from the controller to the switch.

When a packet enters OVS, it is received by the datapath module. If ovs-vswithd has already instructed the datapath on how to handle packets of this type, it follows the specified action. Actions are a list of physical ports or tunnels on which to transmit the packet and can also specify packet modifications, sampling or to drop the packet. If no actions are listed for the packet, it is sent to ovs-switchd. The ovs-vswitchd module matches the packet received from the datapath module against the flow tables received from the controller, accumulates the actions applied and finally caches the result in the kernel datapath, for future packets of that type [42].

OVS has three utilities that can be used to configure the different components: ovs-vsctl is used for querying and updating the configuration of ovs-vswitchd, ovs-ofctl can query and control the OpenFlow and controller configurations on the switch, and ovs-dpctl configures the datapath kernel module.

OpenFlow cannot manage ports, configure QoS queues or associate the controller and switches. In OVS, this functionality is managed with the configuration database, which

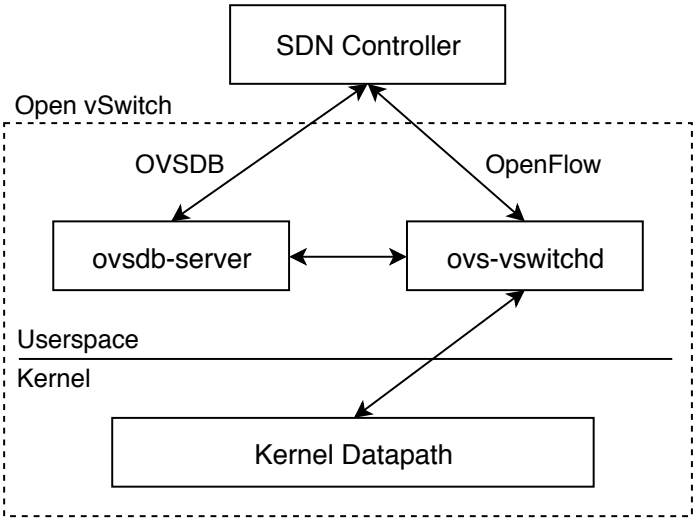


Figure 2.6: Components of Open vSwitch

is part of the ovsdb-server. The controller can access the ovsdb-server with the OVSDB protocol [46]. The interactions between the SDN controller and the different components of OVS are shown in Figure 2.6 [42].

2.3 Quality of Service

2.3.1 Metrics

Quality of Service (QoS) refers to performance guarantees of a network [47]. QoS metrics describe network characteristics or the behaviour of the network. The most common characteristics that describe the quality of network traffic are listed below:

- Reliability: an indication of how likely a packet is to reach its final destination. The packet loss percentage metric describes the reliability of a network.
- Delay: the time a packet takes to flow from its source to destination. The Round-Trip Time (RTT) metric describes the delay of a network.

- Jitter: the variation of delay in which packets arrive at their destination. The average difference in delay times of packets is taken to calculate the jitter of a network.
- Bandwidth: The maximum number of packets that can be received per unit time. The throughput is the actual amount of network traffic that is received at a host and is measured in bits per second (bps).

2.3.2 Sensitivity of Applications

Every type of application requires different network characteristics to execute properly. Table 2.4 lists the sensitivity of common applications to the above-mentioned network characteristics [47].

Table 2.4: Sensitivity of applications to QoS metrics

Application	**Reliability**	**Delay**	**Jitter**	**Bandwidth**
FTP	High	Low	Low	Medium
HTTP	High	Medium	Low	Medium
Audio-on-demand	Low	Low	High	Medium
Video-on-demand	Low	Low	High	High
Voice over IP	Low	High	High	Low
Video over IP	Low	High	High	High
Online gaming	High	High	High	Medium

2.3.3 Flow Scheduling

There are different scheduling techniques that network devices can use to process the flow of packets. The default scheduling technique in most interfaces, known as First-In, First-Out (FIFO) queuing, processes the packets in the order they arrive and does

not differentiate between different types of traffic [47]. The queueing delay in FIFO often has a big negative impact on the QoS of real-time network traffic (e.g. VoIP).

Priority (or class-based) queueing assigns a priority class to each packet, based on a specific field in the packet header, e.g. the Type of Service (ToS) field of an IPv4 header. Packets with higher priority are processed before the packets with lower priority. The QoS of time-sensitive traffic can thus be improved by giving them priority over non-time sensitive traffic. However, if there is a continuous flow in the higher-priority queues, the packets in the lower-priority queues will never be processed and may be dropped after enough time has passed. This is known as starvation [47].

In weighted fair queuing, packets are also assigned to different priority classes with each priority queue given a weight based on the priority of that queue. Higher priority queues are given higher weight values. The number of packets for each queue that is processed, is in proportion to the corresponding weight value of that queue. More packets of the queues with higher weight values are processed, while packets in queues with lower weight values are not ignored and also given a share of processing time [47].

2.3.4 QoS vs QoE

Although the term 'QoS' can be used for a broad range of meanings, it is considered to be an objective, quantitative measure to describe the performance of a network. The service provider and user agree upon certain network characteristics that must be fulfilled and can be measured by QoS metrics.

The Quality of Experience (QoE) is a subjective, qualitative measure to describe the user's perceived network performance [48]. QoE is the impact of the network behaviour on the end user and is also known as the user-level QoS. One metric to measure QoE is known as the Mean Opinion Score (MOS), which is a score given by users on a scale that usually ranges from 1 (worst experience) to 5 (best experience).

The QoS and QoE of a network have an impact on each other, but their relationship

is not necessarily consistent. Some imperfections in the network can impact the QoS but may go unnoticed by the user. A small delay that is not captured by network measurements may render an application useless and drastically impact the QoE.

2.4 Network Evaluation

There are several ways to evaluate network configurations and policies without implementing it in an active, production network. Network simulators, such as ns-3 [49], offer a high degree of control, repeatability, manageability and isolation. The focus of simulators is on modelling network elements, which is not necessarily the same as the code deployed in real networks. Therefore, simulators lack realism and are also known to be lagging behind the newest available technologies [50].

Networks can also be evaluated by implementing a prototype on hardware in a controlled environment. A testbed consisting of various network elements can be used to evaluate policies on a large scale. Some examples of SDN testbeds include BeHop [51], that can test SDN implementations for dense WiFi networks, and the large OFELIA testbed [52] that spans over multiple countries in Europe. These options offer an advantage over simulators, as actual devices and layer interactions are tested to run real applications [17]. However, prototypes and testbeds are more expensive than network simulators and are not accessible to all researchers.

Emulators provide a middle ground between simulation and practical implementations. The primary goal of emulation is to substitute elements with an actual representation of that element, without resorting to modelling it. The repeatability, isolation and manageability of simulators are combined with the realism of testbeds [50]. Experiments can be performed, tested and refined in emulation before implementing it on hardware where it is more difficult to debug and correct errors.

2.4.1 Mininet

Mininet [53] is an open-source network emulator, initially developed at Stanford. It creates and runs virtual networks (consisting of virtual hosts, switches, controllers and links) running real switch and application code on a single Linux kernel. As such, Mininet is described as a network emulation orchestration system [54].

Mininet was implemented to be flexible, deployable, interactive, scalable, realistic and share-able. A rapid SDN prototype can be created by using lightweight virtualisation and an extensible command-line interface and API. Mininet can create custom topologies, customise packet forwarding and run actual programs like web servers and monitoring tools.

The performance fidelity of the emulations is restricted by the resource limits of the single machine on which Mininet is running. At this stage, only wired links are emulated. Mininet can also not handle different Operating System (OS) kernels at the same time.

Mininet runs on Linux systems and utilises the Linux OS virtualisation mechanisms by running processes in network namespaces and using virtual Ethernet pairs. The Mininet hosts run Linux network software, and the switches support OpenFlow. Each switch can be connected to a remote SDN controller. A Python API is also included for customisation of the networks.

2.5 Related Work

A number of protocols that do not rely on SDN technologies are used by ISPs to interact with their clients' gateway devices (CPE). The most popular standard, TR-069 [55], describes the communication between home gateways and an Auto-Configuration Server (ACS). It is also known as CWMP (Customer Premises Equipment Wide Area Network Management Protocol). With TR-069 an ISP can perform remote management

and monitor performance. There are, however, security risks involved with the use of the standard, such as the use of self-signed certificates over the Secure Sockets Layer (SSL) [56].

Several studies aim to improve the network management and QoS provisioning for home networks without using SDN and TR-069. In [57] an operating system for home devices is implemented by creating an abstraction of a Personal Computer (PC), called HomeOS. Network devices appear as peripherals and act like applications on the PC. High-level abstractions are used to develop applications to configure connected devices in a specified way. This simplifies the task of the user to manage and extend the technology at the home network.

In [58] a framework is designed that allows users to formulate sophisticated 'comic-strip' policies using an application. These comic-strip policies are passed to a policy engine running on a custom home network router designed to facilitate a variety of management tasks. The policy engine translates the policies to specifications that becomes the instructions for the home network router. These instructions can include to implement various QoS provisioning (such as prioritising certain network traffic) for devices.

Without SDN, the implementations of the above-mentioned studies lack a centralised point of control and the programmability of network devices. Six features that SDN implementations provide that can potentially benefit the QoS of a network have been identified in [59]:

- Flow-based forwarding: route the flows of different applications with different priorities.
- Dynamic flow rule update: update the installed flow rules in real-time based on network link characteristics.
- Flow and packet analysis: acquiring and classifying the header fields.
- Flow path analysis: use the global network view to maintain the states of flow

paths.

- Traffic monitoring: tracking of various levels (e.g. per-flow, per-port or per-device) of network statistics.
- Queue configuration: queue management can be done by using southbound interfaces such as the OVSDB protocol.

In [5] eighteen different studies that present software enabled home network architectures are compared by Haque and Abu-Ghazaleh, with the focus on whether virtualisation and user involvement are required. A comprehensive survey of forty-two home network solutions that use SDN is presented in [6] by Alshnta et al. The studies are classified depending on their specific target application, such as home network management, home network QoE, home network security and internet use management among others. The authors contrast two requirements that home networks must solve to integrate different devices: ease of use for the home users, and tight control of their network traffic information and setup preferences to enforce privacy.

For the literature studied in this dissertation, the studies considered use SDN to either provide a form of QoS or QoE to home networks or monitor the network traffic of home networks in some way. Except for [60] and [61], all the studies are also surveyed by either Haque and Abu-Ghazaleh or Alshnta et al. A study that is a work-in-progress, where the detailed implementation and results are not yet ready to be published, is excluded from this survey, e.g. [62], [63] and [64].

2.5.1 Home Network QoS Provisioning with SDN

Several studies propose frameworks that aim to improve the QoS and QoE of users in home networks by leveraging SDN. A comprehensive survey was done on seventeen studies to evaluate their main contributions, use cases and testing methods (if results are presented). A summarised review of the survey result is given in Table 2.5.

Table 2.5: Related work - home network QoS / QoE provisioning

Author	Summary of contribution	Use Cases	Evaluation
Yiakoumis et al. [10]	Isolate resources and network traffic with network slicing	Customised slices for applications	Prototype
Yiakoumis et al. [65]	Users select priority of applications with user-agents	VoIP, video streaming	Prototype
Fratczak et al. [66]	Home network slicing from which the control can be outsourced	Remote troubleshooting, network configuration	Prototype
Georgopoulus et al. [67]	Ensures QoE fairness of adaptive bitrate video streaming for all users	Video streaming	Testbed
Kumar et al. [68]	Users control QoS for different devices and applications	Video streaming, web browsing, large downloads	Prototype
Georgopoulus et al. [60]	OpenCache: Improves video streaming with in-network cache close to end-user	Video streaming	Testbed (OFELIA)
Gharakheili et al. [69]	Users can limit bandwidth or restrict network access for devices	Video streaming & conferencing, parental control	Prototype
Ramakrishnan & Zhu [70]	Optimised approach to improve QoE of adaptive bitrate video streaming	Video streaming	Simulation
Wang et al. [71]	Compares different network slicing strategies	Slicing based on application, location and bandwidth usage	Mininet
Eghbali & Wong [72]	Pricing scheme for ISPs based on a Stackelberg game model	Bandwidth slicing	Numerical analysis
Gharakheili et al. [73]	Outsourcing new services to customise the internet use of devices	Video conferencing, filter content	Prototype
Seddiki et al. [61]	Specify priorities for different classified application flows	Video streaming, VoIP	Prototype
Abuteir et al. [74]	Network assisted video streaming by dynamic traffic shaping	Video streaming, bandwidth slicing	Simulation (ns3)
Abuteir et al. [75]	Dynamic traffic shaping based on statistics to allocate bandwidth to clients in real time	Video streaming, bandwidth slicing	Simulation (ns3)
Bakhshi & Ghita [76]	Dynamic queue-based traffic optimisation based on user traffic profiles	User-defined profiles	Mininet
Bozkurt & Benson [77]	Router on which users can specifiy priorities for different applications	Video streaming	Prototype
Jang et al. [78]	Optimise bandwidth allocation for IoT enabled smart homes	Bandwidth allocation	Not tested

Most of the studies try to solve the bottleneck problem in the access network by giving specified high-sensitivity network traffic (such as online video streaming) priority over other traffic, mostly with proactive flow rules that the controller installs on the home gateway device. In the studies that evaluated their design, the throughput achieved at an end device in the home network is mostly measured. When video streaming is con-

sidered, factors such as the buffering time, bitrate received and average video quality are measured. Other QoS metrics such as packet loss and RTT are measured in [76], while Seddiki et al. [61] performs numerous tests including measuring throughput, RTT, jitter and the CPU and memory usage of the controller.

2.5.2 Home Network Monitoring with SDN

Some studies focus on accurately monitoring the traffic of home networks by exploiting the centralised point of control that SDN provides. A comprehensive survey was done on five studies to evaluate their main contributions, use cases and testing methods (if results are presented). A summarised review of the survey result is given in Table 2.6. In all of the studies, the focus is mainly on monitoring the internet usage of the home networks; thus the throughput and number of packets captured are mostly measured and presented.

Table 2.6: Related work - home network monitoring

Author	Summary of contribution	Use Cases	Evaluation
Calvert et al. [79]	Home network data recorder for troubleshooting	Measure internet performance, security, auto-configuration	Prototype
Chetty & Feamster [80]	Improve visibility of home networks to allow ISP monitoring	Outsourcing measurement applications	Not tested
Mortier et al. [9]	Home router that enables traffic isolation, measurements and user interfaces	Device management	Prototype
Chetty et al. [81]	Real-time home network internet usage, can limit specified devices' usage	Control internet data caps	Testbed (21 homes)
Xu et al. [82]	Integrates heterogeneous devices of an IoT-enabled smart home	Home automation	Not tested

2.5.3 Analysis

A few observations can be made of the related work studies surveyed. Different evaluation methods are used to test the proposed design in each study, as shown in Figure 2.7. Mininet [54] and ns-3 [49] are used to perform emulation and simulation respec-

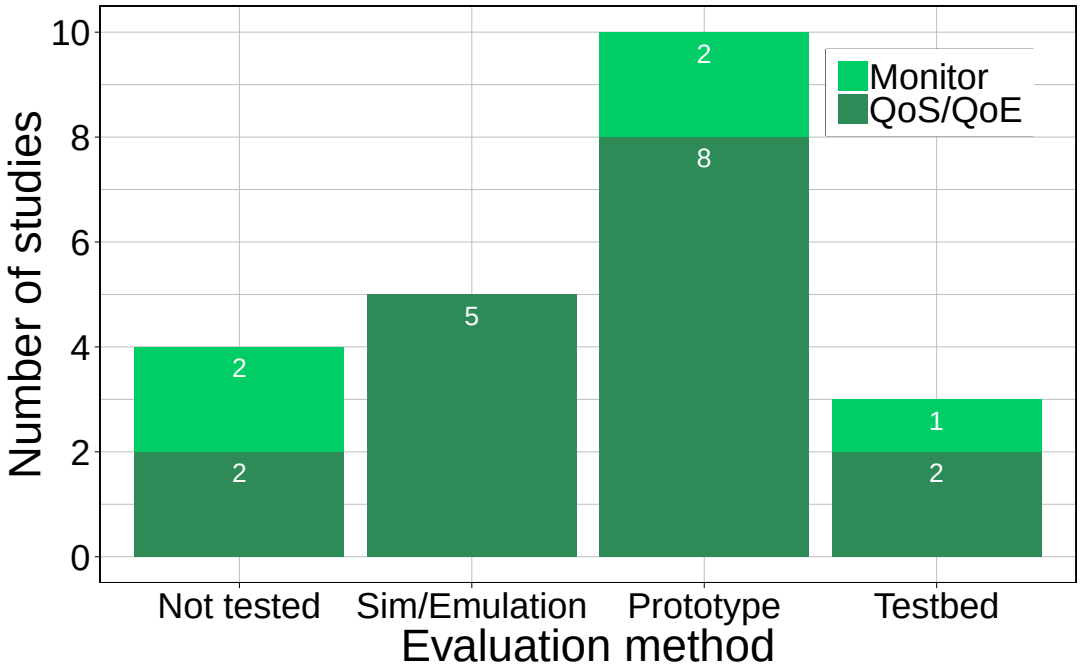


Figure 2.7: Frequency of evaluation methods used

tively. Most implementations involve a prototype that evaluates the design on home network hardware in a controlled testing environment. Large testbeds of devices are also used in some cases to evaluate the design.

Open vSwitch is mostly used as the SDN-enabled network element, and in eight of the studies it is installed on an SDN-enabled commercial router (mostly the TP-Link WR1043ND router) that is running the OpenWrt operating system. The Floodlight controller is a popular choice to implement, while NOX, POX and Ryu are also used. In all implementations, OpenFlow is used as the southbound interface between the network devices and the controller (except in [70], [74] and [75] in which the interface is not specified).

The scalability of the solutions that the studies present are rarely considered. In [79], [82] and [78] the scalability of the designs are discussed, but no quantitative results are given. In [10] the prototype is implemented for seven clients at once, while it is speculated that it will be able to handle many more. In [70] up to 35 clients are considered in simulation. The throughput and per-packet switching of Open vSwitch are considered

for up to 500 000 flows in [9].

Of all the practical implementations that are tested, all except [68] and [69] advocates that the home gateway should be replaced with an SDN-enabled device. This means that users should replace their current legacy routers with ones that are SDN-enabled.

2.6 Concluding Remarks

As discussed in section 2.1.1, home networks experience various management issues. SDN offers a centralised point of control and simplifies the programmability of network devices, which an ISP can leverage to improve the QoS of a home network. Based on the problems experienced in home networks, the advantages that SDN can provide and the current limitations in the related work, an experimental setup is designed in the next chapter that aims to improve the QoS in home networks.

Chapter 3

System Design

In this chapter, the design of the study is presented. In section 3.1, the considerations and broad framework of the design are discussed. The AHP selection method is used in section 3.2 to identify the best controller for the proposed design. Sections 3.3 and 3.4 show the interactions between the Ryu controller, OpenFlow and Open vSwitch, with the focus on QoS provisioning and network monitoring. The functional analysis of the design is shown in section 3.5. Lastly, the design of a Graphical User Interface (GUI) to simplify the process of configuring the controller and switches are given in section 3.6.

3.1 Proposed Framework

The programmability and centralised point of control that SDN provide can be leveraged by an ISP. Tasks such as QoS provisioning and monitoring of the home network traffic can potentially be simplified and improved by adding an SDN controller at the ISP premise or home network.

In most of the related work presented in the previous chapter, a framework similar

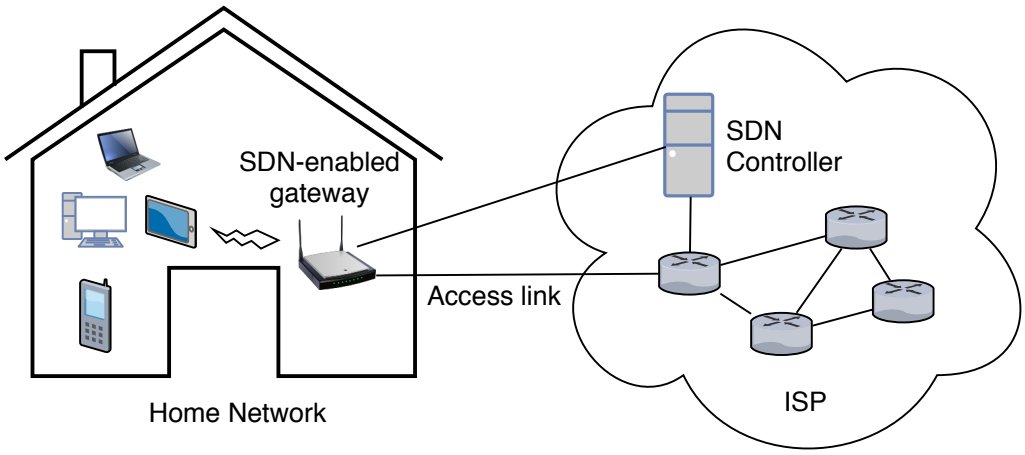


Figure 3.1: Design of related work where home gateway is SDN-enabled

to Figure 3.1 is shown. The controller is connected to the home gateways as well as the ISP routers that provide the access links to the home networks. This means that both the ISP routers and home gateways must be SDN-enabled and compatible with the particular southbound interface (such as OpenFlow). Additional overhead traffic going to the controller is also generated by connecting more devices to it.

If the home gateways are SDN-enabled, there are certain advantages for the home users. Customised feedback can be sent to the controller to give preference to specified devices and applications of the user's choice. Users that are tech-savvy also have the option to implement their own custom controller at home to have full control and perform accurate measurements of their network traffic as delivered by the access link.

However, to implement this setup, the ISP devices providing the home gateways, as well as the home gateways, must be replaced by new SDN-enabled equipment. This can lead to significant expenses for the ISP if they want to use SDN in their provisioning networks. As mentioned in [6], there is also a lack of technical knowledge among the average network home user. Introducing an SDN-enabled device in the home network could cause unnecessary confusion for users if the ISP requests more user interaction for network customisation purposes.

This study proposes a design where only the ISP devices are replaced with SDN-enabled devices while the legacy gateways in the home networks can still be used and not be replaced, as shown in Figure 3.2. Traffic and device differentiation is done on the access link to provide QoS for the home networks. One requirement for this setup is that all the devices (with their IP addresses) on the home network should be known by the SDN-enabled ISP device. The only overhead traffic that is introduced as a result of the SDN controller, is the signalling traffic between the ISP access network device and the controller.

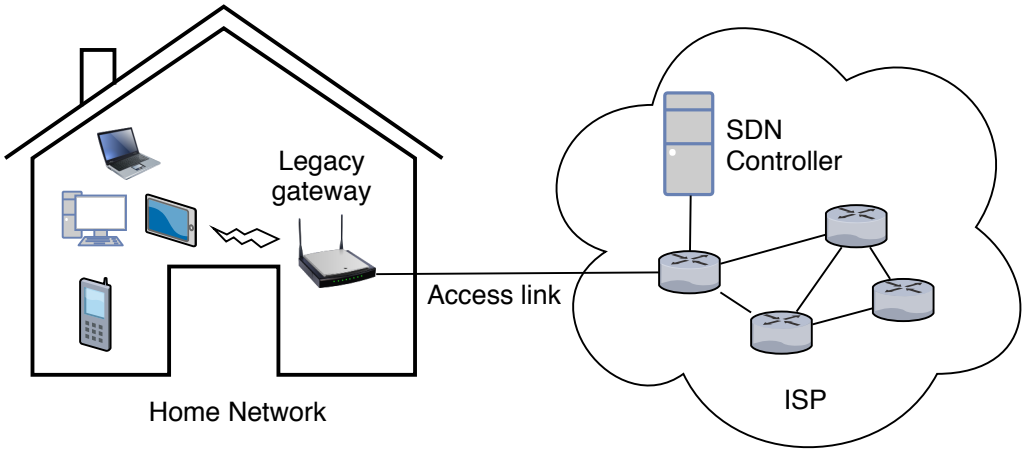


Figure 3.2: Design where home network keeps legacy gateway

3.2 Controller Choice

A suitable SDN controller must be selected to implement the design from the previous section. For this study, the choice of controller is limited to open-source and OpenFlow-compatible controllers. Seven viable controllers are identified: Floodlight, Maestro, MuL, ONOS, OpenDaylight (ODL), POX and Ryu. As in [33], the AHP is used to decide the controller that will fulfil the experimental requirements the best, as it uses pairwise prioritisation and intrinsically verifies the results by performing consistency checks. More information about the AHP is given in Appendix A.1.

Four criteria are identified: ease of use, documentation, performance and reliability. The ease of use includes the programming language in which the controller is written and the amount of time it takes to develop a prototype. This is an essential factor to consider for the experiment and thus is given the highest weight. Documentation represents how accessible the documentation and implementation tutorials are of the controller. This also includes the amount of other related work studies that have been done with that controller. The performance factor relates to the throughput, latency and multi-threading capabilities of the controller. Reliability includes the security of the controller. As the experiment are not run for a long time and this study do not consider network security, the reliability of the controller is less important than the other factors.

All the pairwise comparisons, judgement matrices and consistency results of the AHP are given in Appendix A.2. Table 3.1 displays the final results: the normalised criteria weights and each controller's priority value to each criterion. The last row gives the final rating for each controller. The Ryu controller scored the highest with a score of 0.214, which will then be used to implement the proposed design.

Ryu applications (or scripts) are written in Python and executed by invoking the ryu-manager command. Several applications are available to use as part of the Ryu source code, including a simple learning switch (the 'simple_switch.py' script), implementing QoS ('rest_qos.py') and creating a firewall ('rest_firewall.py').

The Representational State Transfer (REST) API is used as the northbound interface for

Table 3.1: The AHP results for controllers' comparison

Criteria	**Weight**	**Floodlight**	**Maestro**	**MuL**	**ONOS**	**ODL**	**POX**	**Ryu**
Ease of use	0.558	0.165	0.086	0.127	0.039	0.043	0.270	0.270
Documentation	0.263	0.137	0.028	0.052	0.298	0.298	0.063	0.125
Performance	0.122	0.227	0.036	0.393	0.154	0.108	0.021	0.061
Reliability	0.057	0.066	0.031	0.025	0.185	0.136	0.144	0.413
Total	1.000	0.160	0.062	0.134	0.129	0.123	0.178	0.214

Ryu applications. A HyperText Transfer Protocol (HTTP) server is hosted on the machine running the controller, which is accessed with Client Uniform Resource Locator (cURL) commands via the Web Server Gateway Interface (WSGI). The cURL commands use the high-level JavaScript Object Notation (JSON) [83] data-interchange format to communicate with the controller's server. The commands are translated to lower-level commands to configure the running application itself or the network elements connected to Ryu through southbound protocols.

3.3 QoS & Monitoring in OpenFlow

The match and action fields in OpenFlow tables can be used to implement QoS. In the earliest OpenFlow version (1.0), a flow action could be associated with a minimum-rate guaranteed queue installed on the SDN-enabled network element connected to the controller. In version 1.2, this functionality was extended by adding a maximum-rate limit attribute. More recently, version 1.3 introduced flow meters to perform rate-limiting actions.

There are a few differences between queues and meters, as shown in Table 3.2. In this study, only queues will be used to perform QoS, as the minimum-rate functionality is an essential feature needed to perform QoS for home networks. As OpenFlow does not handle queue management (such as the creation and deletion of queues), the OVSDB protocol will be used (see section 3.4).

Table 3.2: Comparison between queues and meters

Queues	**Meters**
Managed outbound with other protocols (OVSDB)	Managed inband with OpenFlow
Switch-specific installations	Meter table contains rows of meters
Maximum and minimum rates	Only maximum rate limiting

Ryu uses OpenFlow as southbound interface. The latest version supported by Ryu applications (included in the source code) to configure QoS is OpenFlow 1.3; thus the information presented in this section applies to the OpenFlow 1.3 standard as described in its specification document [36].

To associate certain network traffic with a minimum-guaranteed or maximum-limit rate queue, a flow rule is installed in the switch's flow table. Flows are modified with `ofpt_flow_mod` messages, which is constructed with the OFPFlowMod class in Ryu. Figure 3.3 shows the structure of the OFPFlowMod class and how the other classes interact with it to create a flow modification message. The 'command' field can be either set to add, remove or modify a flow entry.

The network traffic that matches the properties as specified in the OFPMatch class is associated with a queue ID installed on the switch. Match fields that can be set include the source and destination MAC addresses ('nw_src' and 'nw_dst'), the packet transport protocol such as TCP, UDP or ICMP ('nw_proto') and the source and destination port numbers ('tp_src' and 'tp_dst'). The Ryu application in the 'rest_qos.py' script can be used to construct a cURL message that sends a flow modification message to match the specified traffic with a queue ID.

The statistics field of flow entries in OpenFlow tables can be used to monitor network traffic. Ryu uses the OFPQueueStatsRequest class (with a Queue ID field) to construct an OpenFlow `ofpt_multipart_request` message of the `ofpmp_queue` type. The switch

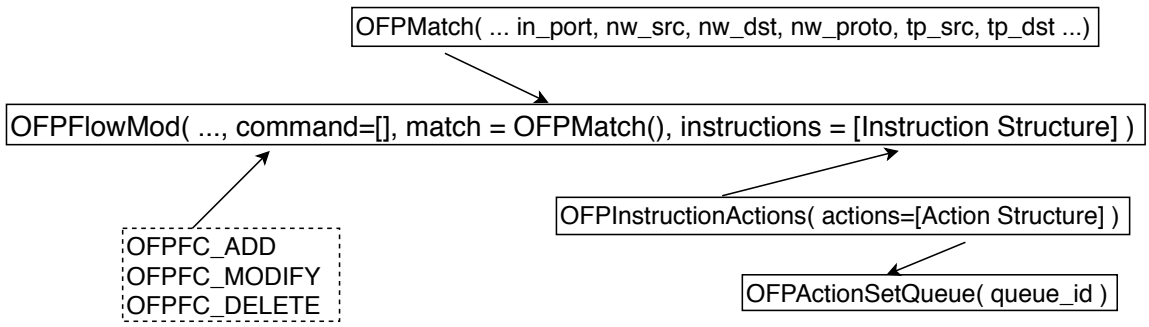


Figure 3.3: OpenFlow flow modification class structure in Ryu

responds with one or more `ofpt_multipart_reply` messages of the `ofpmp_queue` type that Ryu interprets with the OFPQueueStatsReply class. These messages include information about all the queues installed on the switch.

The following fields with their data are included in the reply message: the queue ID ('queue_id'), bytes transmitted on the queue ('tx_bytes'), amount of packet transmitted on the queue ('tx_packets'), amount of errors transmitted on the queue ('tx_errors') and the amount of time that the queue has been installed ('duration_sec'). The Ryu application in the 'rest_qos.py' script can be used to construct a cURL message that sends a statistics request to the switch, either for a specified queue or for all the queues installed.

3.4 OVS Queues

The Open vSwitch (OVS) software is chosen for the SDN-enabled ISP router, as it is the most popular open-source option and is closely integrated with the OpenFlow protocol and the Ryu controller. Queue management (creation and deletion) can be done with Ryu's REST API via the OVSDB protocol. The 'rest_conf_switch.py' Ryu application is used to connect to the ovsdb-server in the OVS userspace. The 'rest_qos.py' application is used to add or delete the queues on OVS. The VSCtl and OVSBridge libraries in Ryu are used to communicate via the OVSDB protocol. Queues can be created with a cURL command that contains the outgoing port name of OVS, the maximum rate limit of all queues and then a list of all required queues. Each queue can have a maximum rate limit and a guaranteed minimum rate.

OVS applies the Hierarchical Token Bucket (HTB) [84] technique to guarantee the minimum and limit the maximum network traffic processed in queues. HTB is a faster replacement for the class-based queueing technique and is part of the traffic control (`tc`) kernel function in Linux operating systems [85].

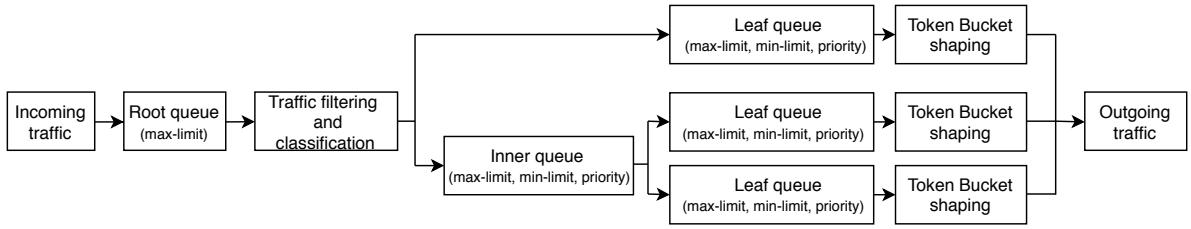


Figure 3.4: HTB queueing algorithm

An HTB class consists of a root (or parent) queue and can contain several children queues. If a child queue also has at least one child queue, it becomes an inner queue. Leaf queues are all the queues without children. Inner queues are only responsible for traffic distribution, while the leaf queues consume network traffic. Each inner and leaf queue can be assigned a guaranteed minimum rate, maximum limit rate and priority value, while the root queue only has a maximum limit that is enforced for all incoming traffic.

Figure 3.4 displays a flow diagram of the HTB technique. After the incoming traffic is shaped by the maximum limit of the root queue, it is classified and processed on its associated inner queue, satisfying the maximum and minimum constraints, until it reaches the leaf queue. The processing and traffic shaping of network packets on each queue is done with the token-bucket algorithm (shown in Figure 3.5).

In the token-bucket algorithm, the bucket has a fixed maximum capacity, with tokens added at a fixed rate to the bucket until the maximum capacity is reached. For every packet processed in the queue, one token is removed from the bucket. A packet can only be processed if there is a token in the bucket to remove. This allows network traffic to be bursty, but not to exceed a regulated maximum rate [47].

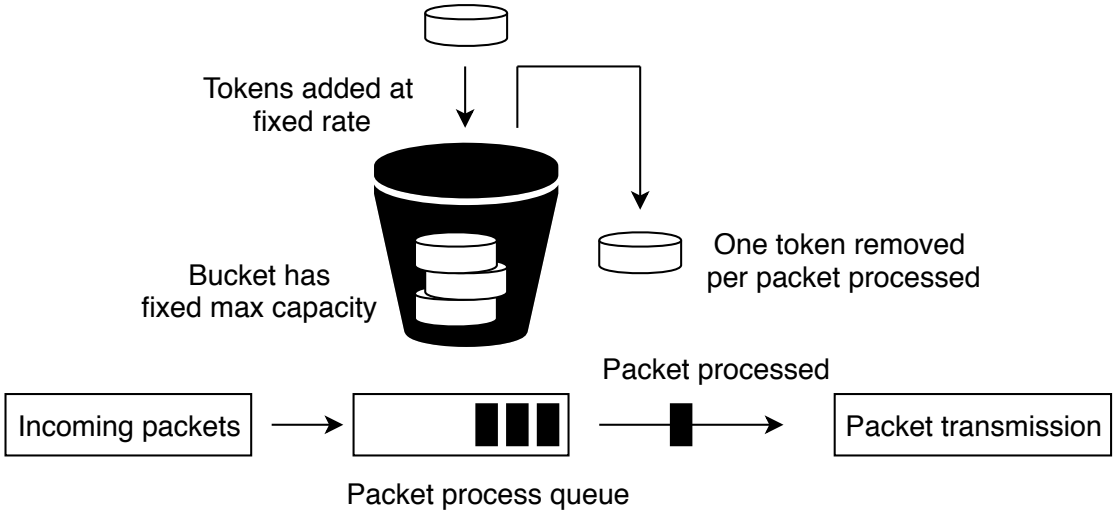


Figure 3.5: Token bucket algorithm

3.5 Functional Analysis of Design

A functional analysis (shown in Figure 3.6) can be constructed based on the design framework in Figure 3.2 and the interactions between Ryu, OpenFlow and OVS, as given in the previous sections. The ISP routers that provide the access links to home networks, contain the queues that can guarantee or limit the rate of specified network traffic. An SDN controller is hosted at the ISP premise and is connected to the ISP router. The queues and flow rules that are associated with specified queues are managed by the controller, via the southbound interfaces.

The controller can query the queue statistics on the ISP router to monitor the traffic of the home networks. These results can be shown to an ISP operator via a controller interface (see section 3.6). The operator can then make decisions based on the monitor results to install queues and flow rules to improve the QoS for the home networks, via the northbound interface. Alternatively, the monitor results can be fed to a solver running alongside the controller to dynamically make decisions and installing the calculated queues and flow rules on the ISP router. This solver can be any algorithm that aims to improve the QoS of home networks based on the statistical results it receives,

including heuristic algorithms, discrete optimisation techniques and machine-learning algorithms that tries to predict when users require certain traffic to be prioritised.

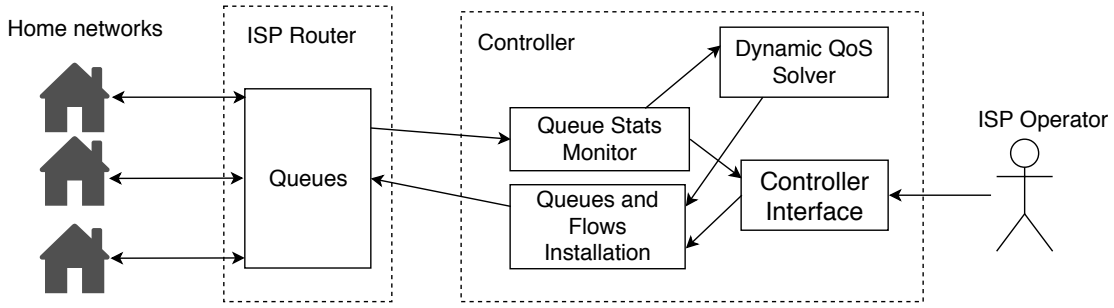


Figure 3.6: Functional interactions of proposed system design

3.6 User Interface

To make it easier for an ISP operator to configure and interact with Ryu and OVS, a Graphical User Interface (GUI) is designed and implemented using the `tkinter` library [86] in Python. This application must run on the same machine as Ryu. The QoS configuration for multiple homes that are connected to the one ISP SDN-enabled access switch can be done with the application. Figure 3.7 shows a screenshot of the application interface.

Figure 3.8 shows the initialisation procedure of the application. The ISP operator must first enter the subnet of the home network that must be configured. The application then populates the list of destination IP addresses from the 'ip_addresses.txt' file that are within the specified subnet. The file contains all the destination and source IP addresses of packets that the controller has received. Extra functionality is added to the Ryu controller to export all the IP addresses received to the text file. This list of IP addresses shown in the application represents the different devices in the home network.

Before the operator can install new queues or flow rules, the application queries the

switch for queues and flow rules that may already be installed. If found, the queues and rules lists are populated from the 'current_queues.txt' and 'rules.txt' files respectively. If no installed queues are found, a new 'current_queues.txt' file is created. QoS rules cannot exist without any queues present on the switch, thus new 'rules.txt' and 'rule_counter.txt' files are also created. If no flow rules are found, new 'rules.txt' and 'rule_counter.txt' (which keeps track of the QoS Rule ID number of the next rule that will be installed) files are created.

Figure 3.9 shows the general flow of the application. If queues have not already been added to the switch, the operator can create them by entering a maximum, minimum or both parameters for each queue. The application performs an error check to verify that the entered data are valid numerical values. If the error check is not passed, a descriptive error message is shown. If the check is passed, the entered queue is added to 'queues.txt'. When all the required queues are successfully entered and added to

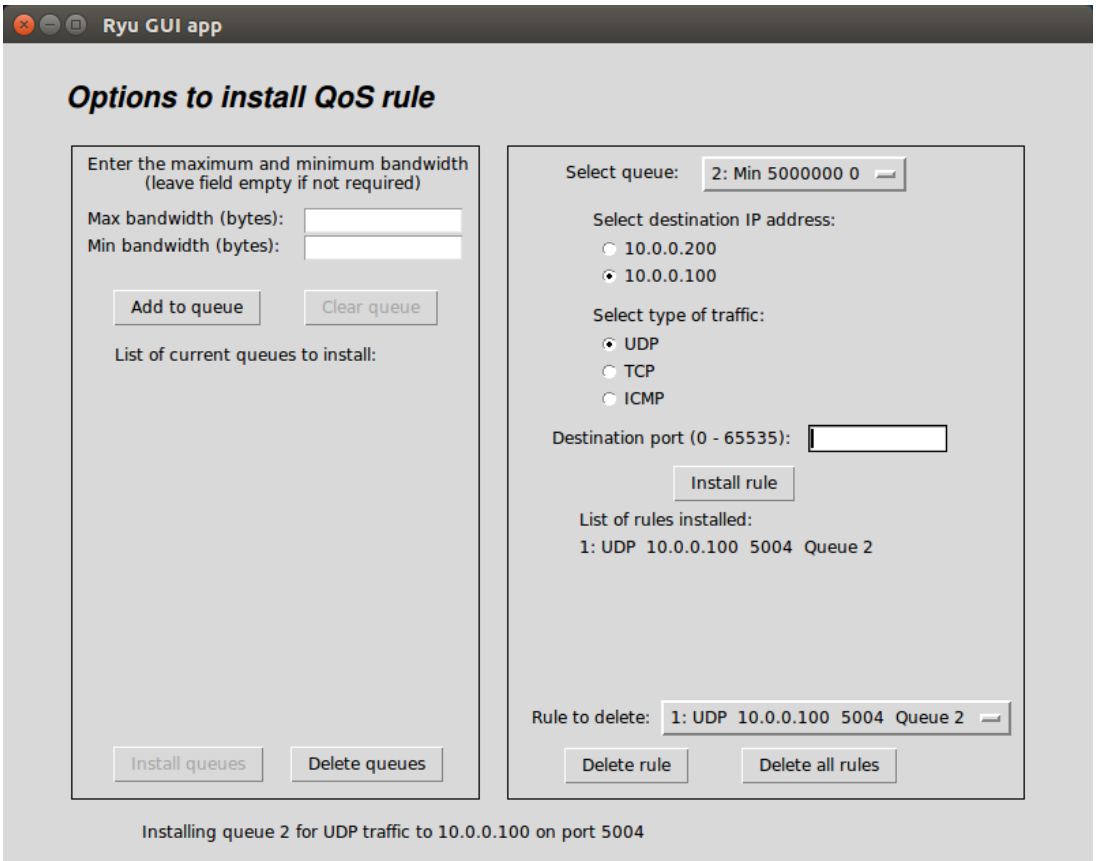


Figure 3.7: Screenshot of GUI application

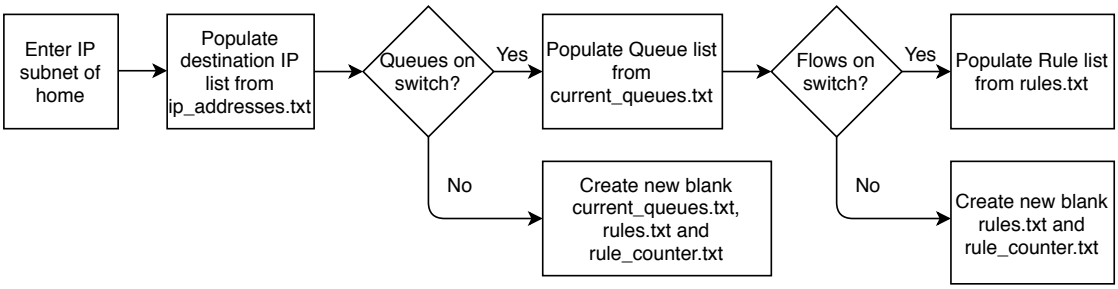


Figure 3.8: Initialisation of GUI application

the file, the operator can add them all to the switch. The application uses the 'install_queues.sh' script (which contains a Ryu REST API cURL command) to add the entered queues and add them to the 'current_queues.txt' file. The 'queues.txt' is then cleared. Due to the queue management of OVS (that uses the Linux `tc` class), the queues cannot be individually added to or deleted from the switch.

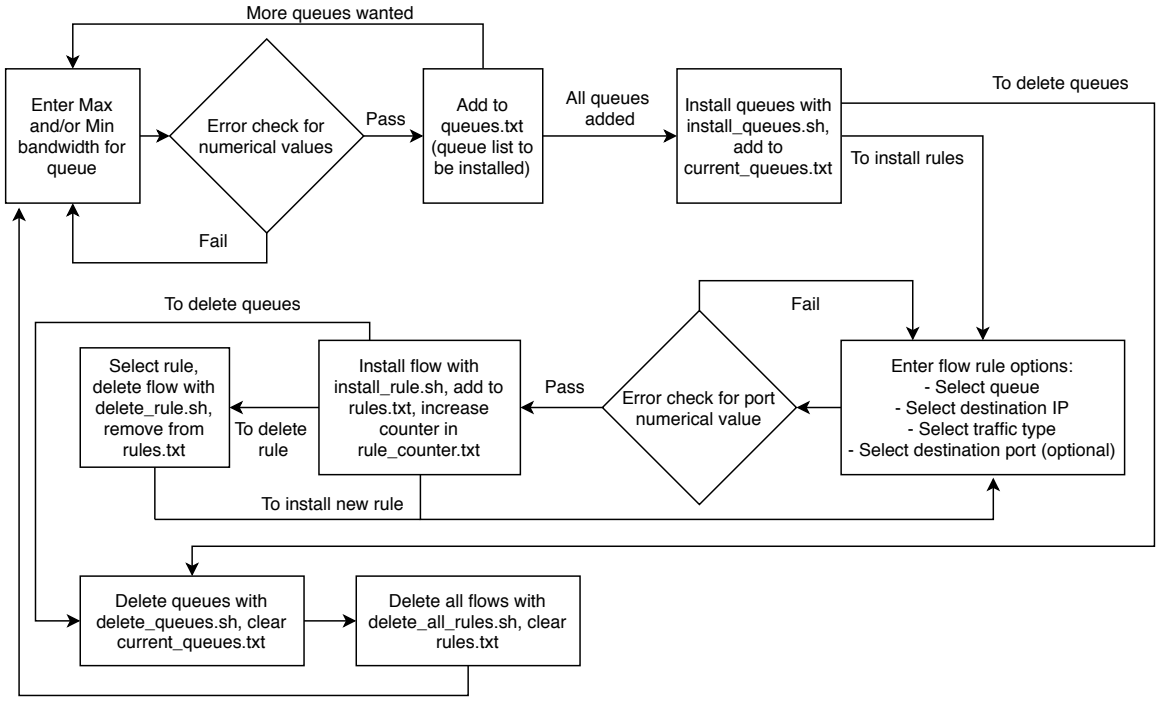


Figure 3.9: Using GUI application to manage queues and flows

The operator can then install flow rules that are associated with an added queue. A drop-down menu is populated by all the added queues (from 'current_queues.txt'). The required queue, destination IP address of the device that requires priority traffic and the network protocol (TCP, UDP or ICMP) must be selected to install a flow rule. The destination port number can also optionally be entered, for which an error check is done to verify that it is a valid numerical value. The 'install_rule.sh' script is used to install the flow rule on the switch (again with a Ryu REST API cURL command). The rule is added to the 'rules.txt' file, and the value in 'rule_counter.txt' is increased.

All the installed flow rules are added to a drop-down menu. The operator can select one to delete, which calls the 'delete_rule.sh' script to remove the flow from the switch and the 'rules.txt' file. There is also an option to delete all the flow rules installed. The contents of the 'rules.txt' file are removed and the 'delete_all_rules.sh' script deletes all flow rules from the switch. The operator can also decide to delete all added queues. This clears the 'current_queues.txt' file and uses the 'delete_queues.sh' script to remove the queues from the switch. After all the queues are deleted, all the remaining installed flow rules that may exist are also removed, as the flow rules are worthless without their associated queue.

3.7 Concluding Remarks

In this chapter, the design of a framework that an ISP can use to implement QoS for their clients by leveraging SDN, was discussed. The Ryu controller will be used to install QoS policies on an OVS device to verify if the proposed design would work. A GUI application that works with Ryu, was developed to simplify the process of creating queues and flow rules on OVS.

Chapter 4

QoS Provisioning Results

In this chapter, the results of provisioning QoS for home networks are presented. In section 4.1, the implementation of two cases that are compared to each other in the experiment is discussed. The different network traffic that is tested, with their results, are presented in section 4.2. The verification of the design is discussed in 4.3. Finally, the process of validating the experiment is given in section 4.4.

4.1 QoS Experiment Setup

To evaluate if it is possible for the ISP to use the SDN controller to implement QoS in a home network, two use cases are compared with each other:

- Case 1: The SDN controller sets no QoS priority queues. This case is comparable to a traditional non-SDN network.
- Case 2: The SDN controller installs a static QoS flow at a specified port of a device's interface to prioritise the throughput for all applications on that port.

Table 4.1: QoS metrics measured in experiment

Metric	Unit	Measurement utility	Version
Throughput	Megabits per second (Mbps)	bwm-ng [87]	v0.6
Packet loss	Percentage (%)	iperf [88]	2.0.5
Jitter	Milliseconds (ms)	iperf	2.0.5
RTT	Seconds (s)	ping	Linux iputils-s20121221

The static QoS flow in case 2 is installed for only one host in the home network. The host's traffic on the specified port will thus have priority over the rest of the access link traffic. This can be especially useful when the host is streaming a video while the rest of the home devices are busy with non-time sensitive tasks (like downloading files from the internet). The video traffic is given priority and gives the best viewing experience possible for the user. Four QoS metrics are measured in each use case, as shown in Table 4.1, to quantitatively compare the cases with each other.

Figure 4.1 displays the experimental network setup that is tested. Host 1 and host 2 are two computers forming part of the emulated home network. Both are connected to the home router within the same subnet. This represents one of the home networks shown in Figure 3.6. The home router functions as a gateway to the network of the ISP,

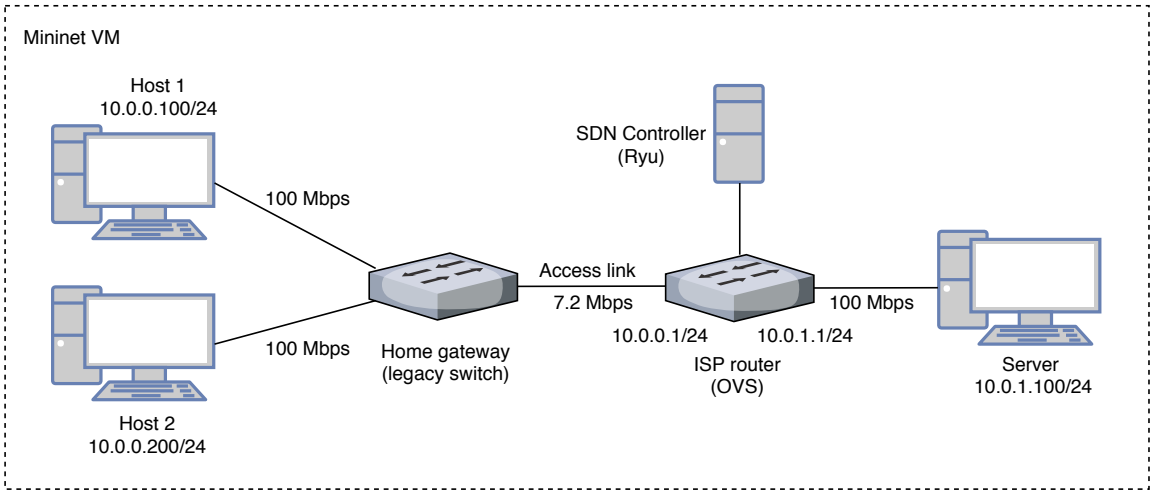


Figure 4.1: Experimental setup with two hosts, a server, two routers and SDN controller

which is represented by another router and the server. The ISP router is connected to the SDN controller, as the queues and flow rules are installed on this router to shape the access link traffic going to the home router. The server is used to emulate traffic for hosts 1 and 2.

The bandwidth of the access links between the home router and ISP router is limited to 7.2 Mbps, as this is the average internet download speed in South Africa [89]. The links between the hosts and home router, as well as between the server and ISP router, are limited to 100 Mbps.

Table 4.2 lists all the software and their versions that are used in the experiment. All the switches and hosts are emulated using the Mininet software. Both the Ryu controller and Mininet topology are within the same virtual machine (Virtualbox). The following three Ryu scripts are used in the experiments: 'rest_conf_switch.py' (connects to the switch using the OVSDB protocol), 'rest_router.py' (allows IP addresses to be added to the switch's interfaces, so that it acts like a router) and 'rest_qos.py' (allows the installation of queues and QoS rules).

Table 4.2: Software used for emulation experiments

Category	**Software used**	**Version**
Virtual machine	Oracle Virtualbox	5.2.20
Host operating system	Windows	10 (64-bit)
Guest operating system	Ubuntu	16.04 (64-bit)
Emulator	Mininet	2.3.0d4
Controller	Ryu	4.28
Switch	Open vSwitch	2.5.4
Southbound communication	OpenFlow	1.3

4.2 QoS Provisioning Results

4.2.1 First Experiment - iperf Traffic

The first experiment uses solely iperf traffic sent from the server to host 1 and 2. The server sends a constant stream of traffic (5 Mbps) to host 1 on port 5004 for one minute. After 30 seconds, the server sends a constant stream of traffic (also 5 Mbps) to host 2 on the default port 5001.

In the first case, no QoS is implemented by the controller and the installed queues are only used to limit the access link with a maximum rate of 7.2 Mbps. In the second case, two flow rules are installed: one that uses a minimum-rate queue of 5 Mbps to prioritise UDP traffic going to host 1 on port 5004 and one that uses a minimum-rate queue of 100 bps to prioritise ICMP traffic going to host 1. The latter flow rule is used to prioritise the RTT traffic (as used by ping) of host 1 over host 2. Figure 4.2 shows the flow diagram of the experiment.

The results of the first experiment are shown in Figures 4.3 to 4.6. In each graph, the red lines with square markers represent the first case (without QoS), while the blue lines with triangle markers represent case 2 (with QoS). For the first 30 seconds, the QoS metrics at host 1 is similar for both cases. In the first case, the throughput decreases while the jitter, packet loss and RTT increase, while host 2 starts to utilise the link as

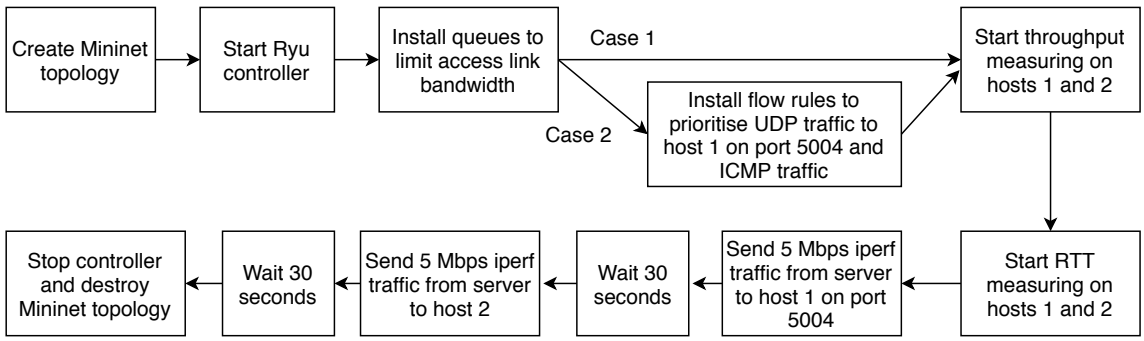
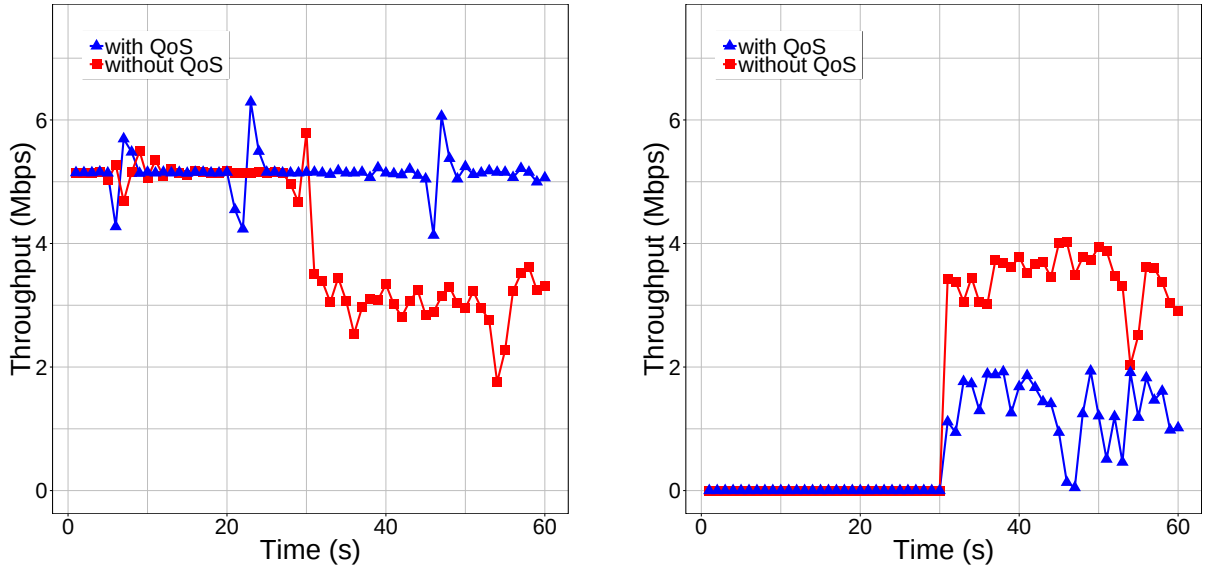


Figure 4.2: Flow of the first experiment

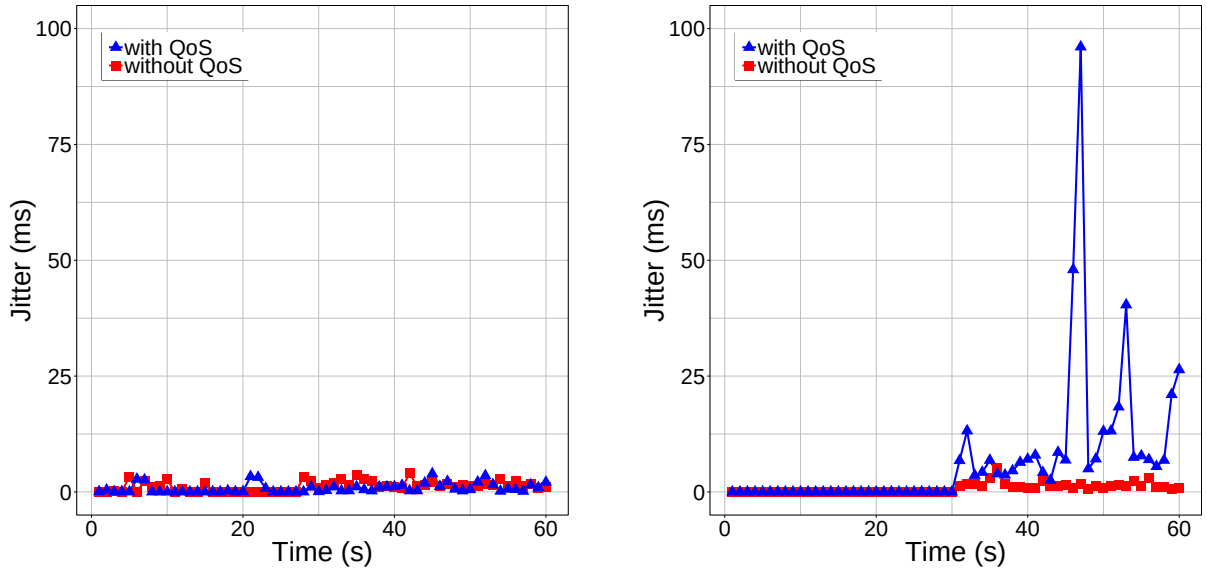
well. In the second case (with QoS), the throughput at host 1 is overall higher, while the jitter, packet loss and RTT are less, as in the first case.

The experiment is repeated ten times. The average and standard deviation for each



(a) Host 1 (b) Host 2

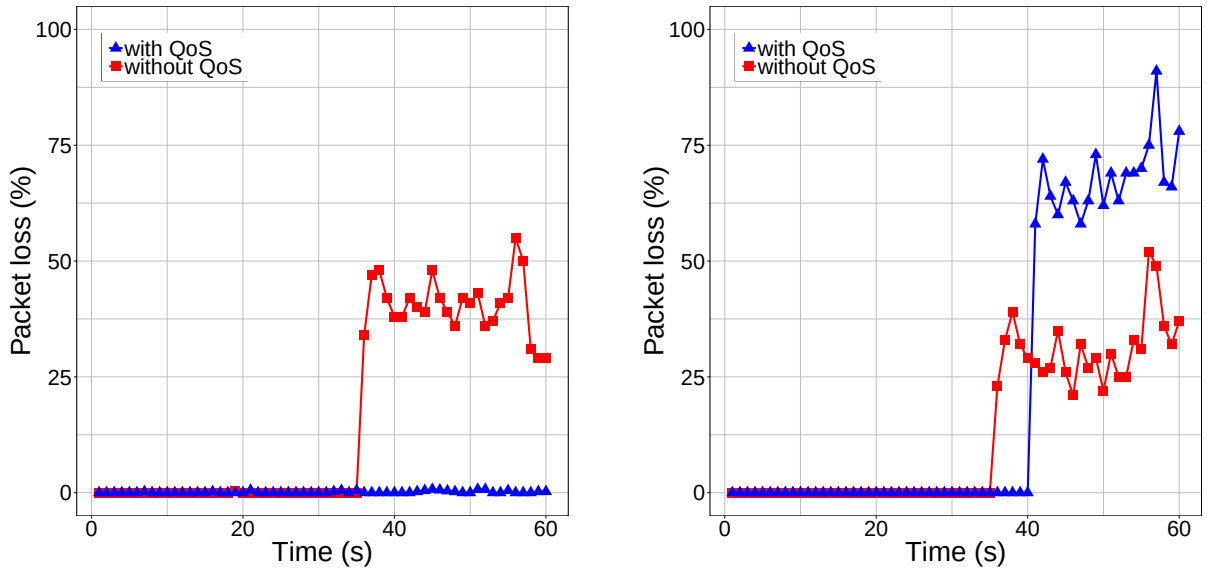
Figure 4.3: Throughput results of experiment 1



(a) Host 1 (b) Host 2

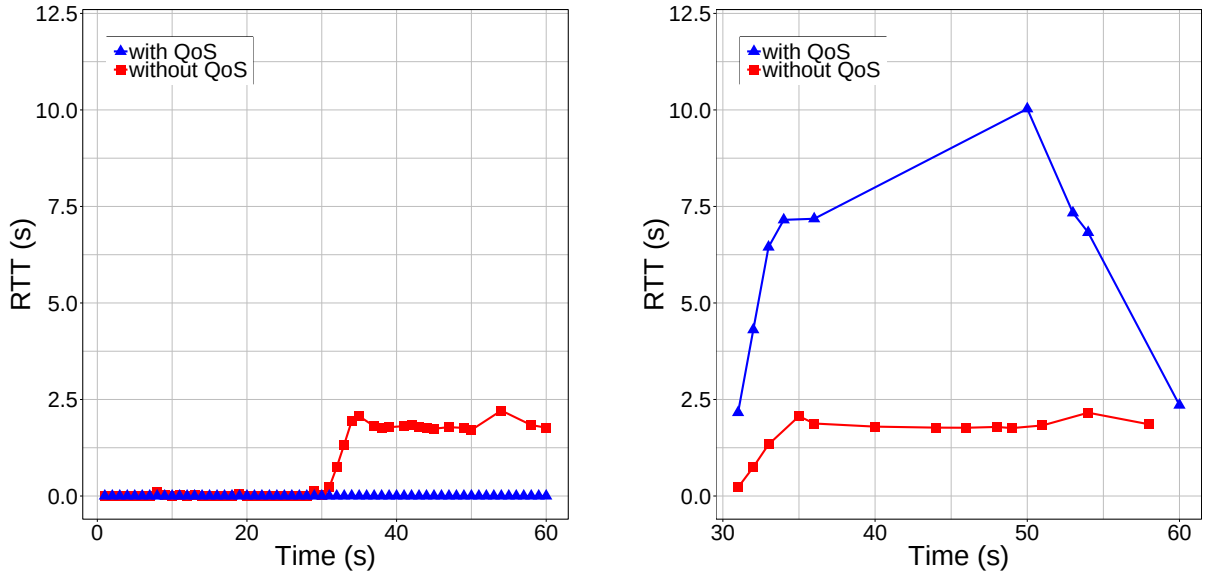
Figure 4.4: Jitter results of experiment 1

QoS metric and each case are shown in Table 4.3, host 1 for the whole minute and host 2 for the last 30 seconds. The averages of the second case's QoS metrics at host 1 improve from the first case's and have a lower standard deviation, indicating fewer



(a) Host 1 (b) Host 2

Figure 4.5: Packet loss results of experiment 1



(a) Host 1 (b) Host 2

Figure 4.6: RTT results of experiment 1

Table 4.3: QoS statistics for experiment 1

Host	Use case	Throughput		Jitter		Packet loss		RTT	
		Avg.	SD.	Avg.	SD.	Avg.	SD.	Avg.	SD.
1	1	4.129	1.175	1.717	1.948	16.188	19.614	0.773	0.868
1	2	5.123	0.571	1.141	1.630	0.117	0.233	0.003	0.019
2	1	3.206	0.554	2.191	2.271	30.777	15.684	1.688	0.551
2	2	1.309	0.649	27.243	51.725	41.931	33.665	5.645	3.424

variability. The opposite occurs at host 2, where the averages of the second case's QoS metrics are worse than the first case's with higher variability, except for the throughput.

4.2.2 Second Experiment - Video Traffic

The second experiment uses actual video traffic as well as iperf traffic. Host 1 streams a video (the 480p version of the widely used Big Buck Bunny video [90] hosted by the server for one minute. The video streaming is done using vlc media player (2.2.2) [91] over the Real-time Transport Protocol (RTP) on port 5004. After 30 seconds, the server sends a constant stream of traffic (5 Mbps) to host 2 using iperf.

As in the first experiment, no QoS is implemented in the first case, while in the second case a flow that uses a minimum-rate queue of 5 Mbps to prioritise UDP traffic going to host 1 on port 5004 is installed on the switch. Figure 4.7 shows the flow diagram of the experiment.

The results of the second experiment are shown in Figures 4.8 and 4.9, with a box plot illustrating the spread of each use case's data. Only the throughput data of host 1 and 2 are compared. The jitter and packet loss data is not available for host 1, as video traffic is used instead of iperf traffic.

A similar trend is seen as in the first experiment. In the second case (with QoS), the video streaming data is not reduced for host 1, as in the first case (without QoS), where

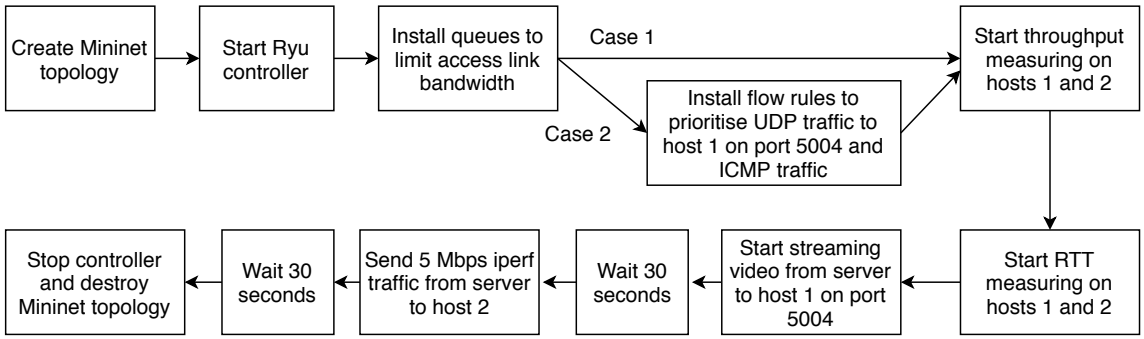
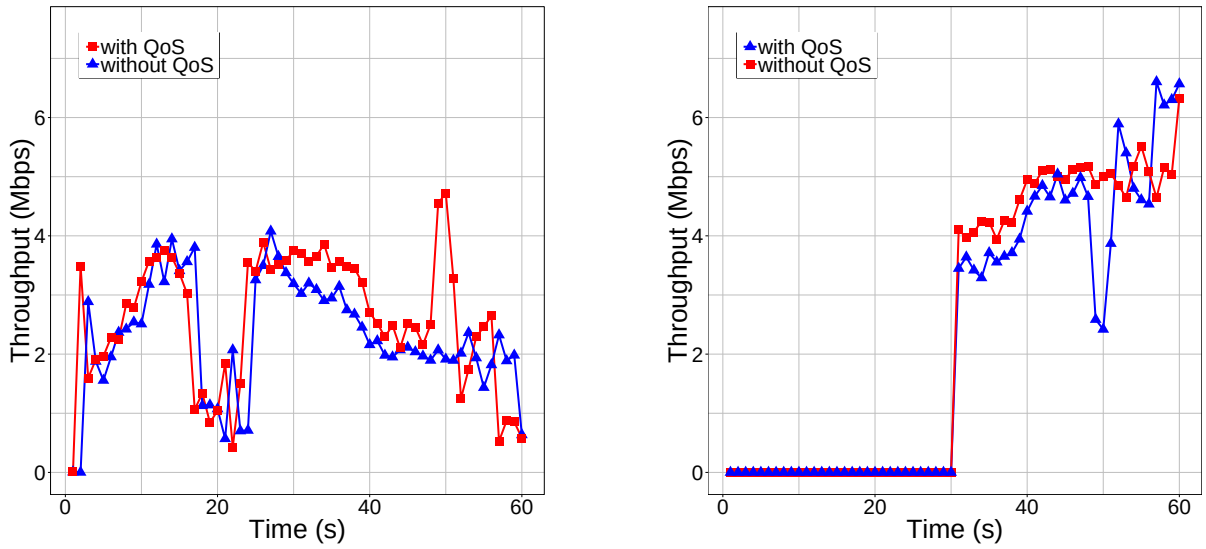


Figure 4.7: Flow of the second experiment

host 2 also utilises the available bandwidth.

The experiment is also repeated ten times. The average and standard deviation for each QoS metric and each case are shown in Table 4.4. The metrics are calculated, again, for the whole minute at host 1 and the last 30 seconds at host 2. The average and variation increase at host 1 from the first case to the second, as the host is free to utilise more bandwidth of the access link. At host 2 the average decreases while the variation increases, as the video traffic of host 1 is given priority.



(a) Host 1

(b) Host 2

Figure 4.8: Throughput results for experiment 2

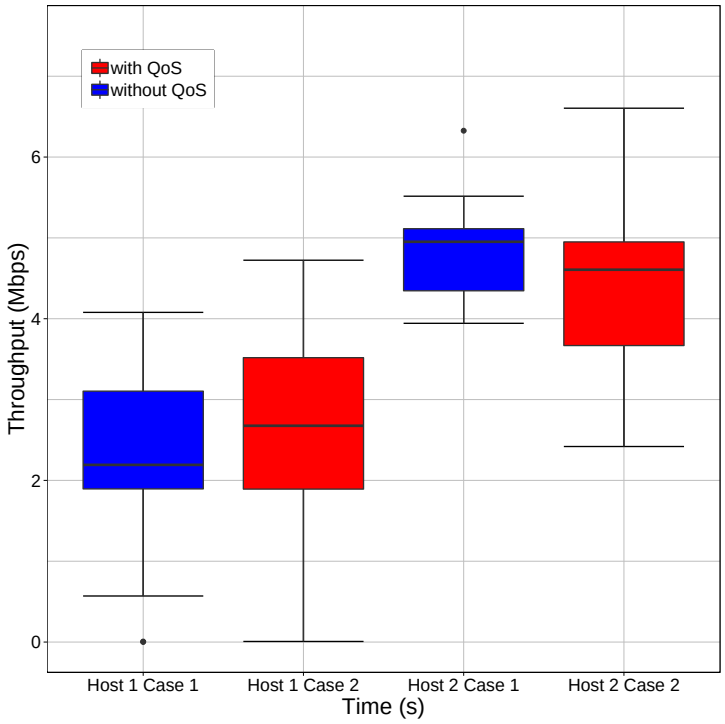


Figure 4.9: Throughput boxplot for experiment 2

Table 4.4: QoS statistics for experiment 2

Host	**Use Case**	**Throughput**	
		Avg.	SD.
1	1	2.357	0.930
1	2	2.600	1.111
2	1	4.840	0.542
2	2	4.490	1.081

4.3 Design Verification

Certain results are to be expected for the experiments performed in the previous section. The design is verified by comparing the two use cases implemented and determining whether the design behaves as expected. Data verification is thus performed by comparison using graphical displays, as described in [92].

In the experiments performed in the previous section, a baseline is established in the first case where QoS is not implemented. All traffic going to host 1 and 2 are given equal best-effort priority and each achieves a similar throughput when the same amount of traffic is sent to both. This can be seen on the graphs in Figure 4.3 (the red, square data points) where both hosts receive a throughput of around 3.5 Mbps during the last 30 seconds of the experiment when 5 Mbps of iperf traffic are sent to both. The jitter, packet loss and RTT are also similar for both hosts during this time, as roughly the same amount of network resources are allocated to both traffic streams.

When QoS is implemented in the second case, specified traffic going to host 1 is processed on a 5 Mbps minimum-rate queue. OVS prioritises the traffic on the minimum-rate queue over the rest of the traffic that are not associated with queues. In the graphs of Figure 4.3, it can be seen that the throughput of host 1 stays about 5 Mbps during the second case (the blue, triangle data points). The traffic going to host 2 is allocated the excess bandwidth and fluctuates around 1.5 Mbps. As more network resources are given to the traffic going to host 1, the jitter, packet loss and RTT of host 1 are also significantly better than that experienced by host 2.

The results coincide with the expectations for each use case implemented, and thus the design is verified as working correctly.

4.4 Validation Experiment

In their thesis [93], Krishna implements and evaluates an end-to-end bandwidth guaranteeing model, based on OpenFlow and OVS. Queues, priority flows and meters are used to guarantee bandwidth for traffic going to specified hosts, while maximising the best-effort traffic when the guaranteed queues are fulfilled. This setup is not specific to home networks, but the experiment performed in their 'Traffic Prioritization' section can serve as baseline to compare results and validate the experiments performed in this study. Their experimental parameters are thus reproduced using the proposed setup of this study. The similarities and differences between the setup as implemented

by Krishna and this study are shown in Table 4.5.

The topology of the experiment is shown in Figure 4.10. Four hosts act as traffic senders

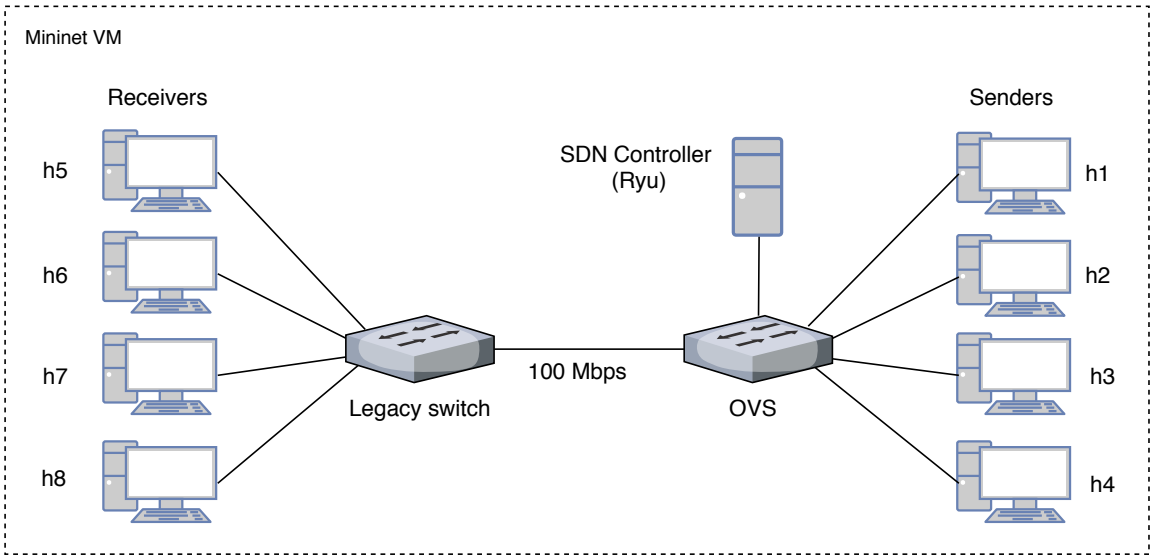


Figure 4.10: Topology of validation experiment

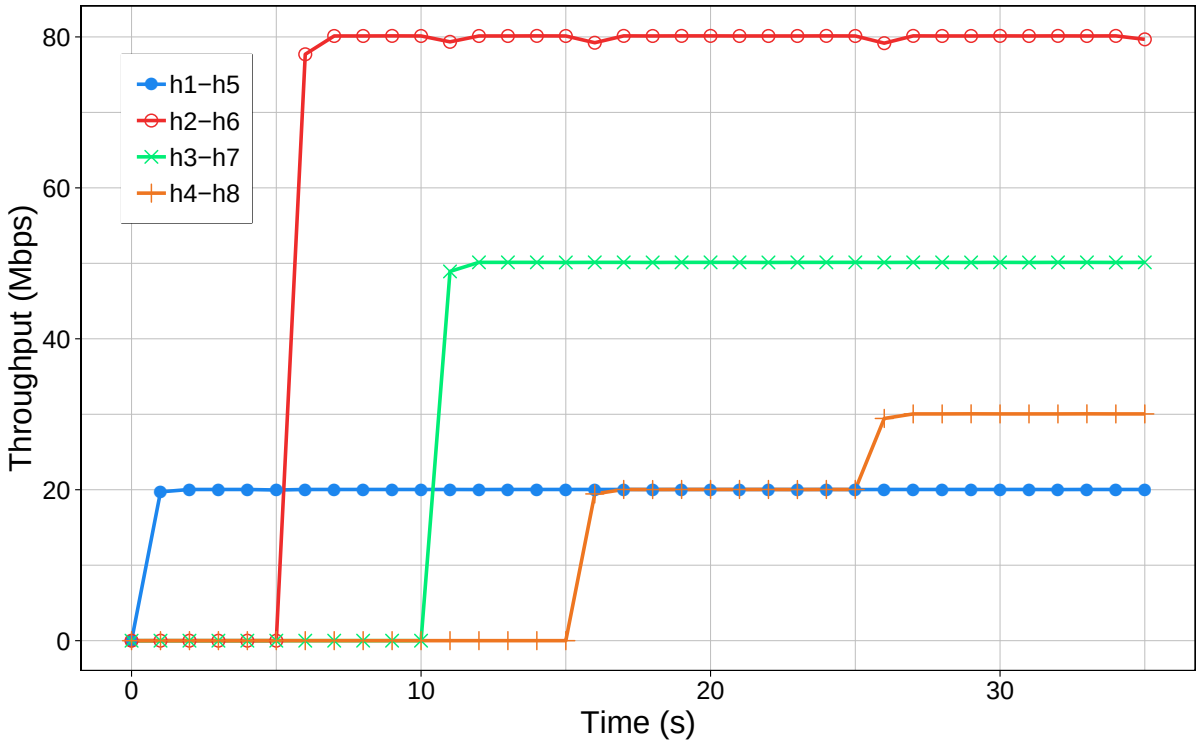


Figure 4.11: Data rate generated at senders for validation experiment

Table 4.5: Comparison of validation experiment implementations

Category	Implementation by Krishna	Implementation of this study
Number of sender hosts	4	4
Number of receiver hosts	4	4
SDN Switch	OVS 2.3.2	OVS 2.5.4
Number of switches	3	2
Southbound interface	OpenFlow 1.3	OpenFlow 1.3
Controller	Ryu	Ryu
Switches connected to controller	All	One
Evaluation framework	Testbed of Intel Xeon servers	Mininet
QoS provisioning	Queues and priority values	Queues
Measurement location	Sender and receiver hosts	Sender and receiver hosts, and controller

(h1 to h4) while the other four are receivers (h5 to h8), where each sender generates network traffic for one receiver. The senders are connected to the OVS switch (acting as the ISP router and connected to the Ryu controller), while the receivers are connected to a legacy switch (acting as the home gateway). All the links are limited to 100 Mbps.

The experiment is performed for 35 seconds, with all the traffic generated using iperf. Host h1 sends 20 Mbps of traffic to host h5 for the whole duration of the experiment. After 5 seconds, host h2 sends 80 Mbps of traffic to host h6 for the rest of the experiment duration. At the 10th second, host 3 sends 50 Mbps of traffic to host h7, also until the experiment ends. At the 15th second, host h4 sends 20 Mbps of traffic to host h8. This is increased to 30 Mbps after 10 seconds have passed, and remains 30 Mbps until the experiments ends. Figure 4.11 shows the sent data rate as generated at hosts h1 to h4, and measured at these hosts using the bwm-ng monitor tool. The traffic sent is visually verified as being correct.

A flow is installed for each of the host traffic pairs. The first pair (h1-h5) is given a minimum-rate queue of 1 Mbps, and is seen as best effort network traffic. The other three pairs are given a minimum-rate queue of 30 Mbps each. In [93], the network traffic between h1 and h5 is given a higher priority than the other network traffic; thus all excess traffic after the minimum-rate queues have been satisfied, are allocated to them.

The bandwidth allocation of excess bandwidth after priority queues have been fulfilled are not considered and are outside the scope of this study. Based solely on the HTB algorithm, the excess bandwidth are distributed according to the weight that it is needed; thus the flows that receives more traffic would be allocated more bandwidth than flows that receive less traffic.

The experiment is reproduced in Mininet, firstly without any QoS provisioning (Figure 4.12) and then with the same QoS rules applied as in [93] (presented as lighter colours in Figure 4.13). The throughput results measured at the receivers, as published in [93], were recreated and are also shown as the reference values (denoted by the 'ref' suffix) in Figure 4.13, presented with darker colours.

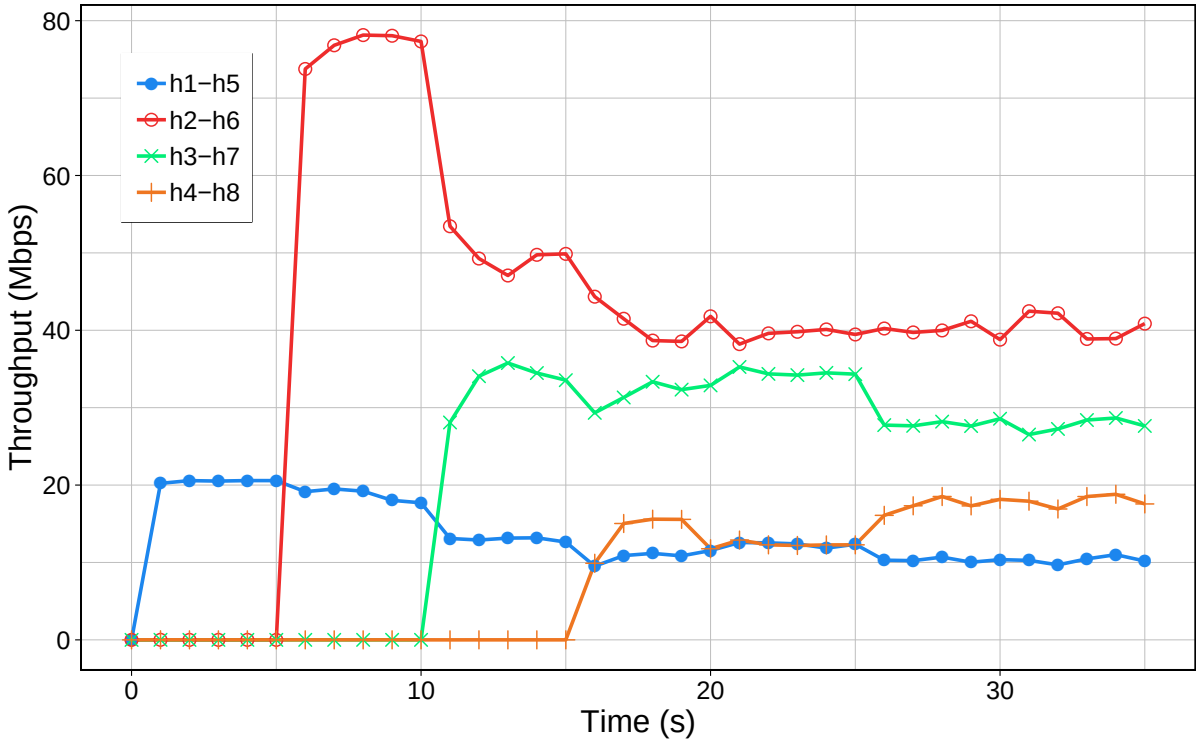


Figure 4.12: Throughput results at receivers without QoS

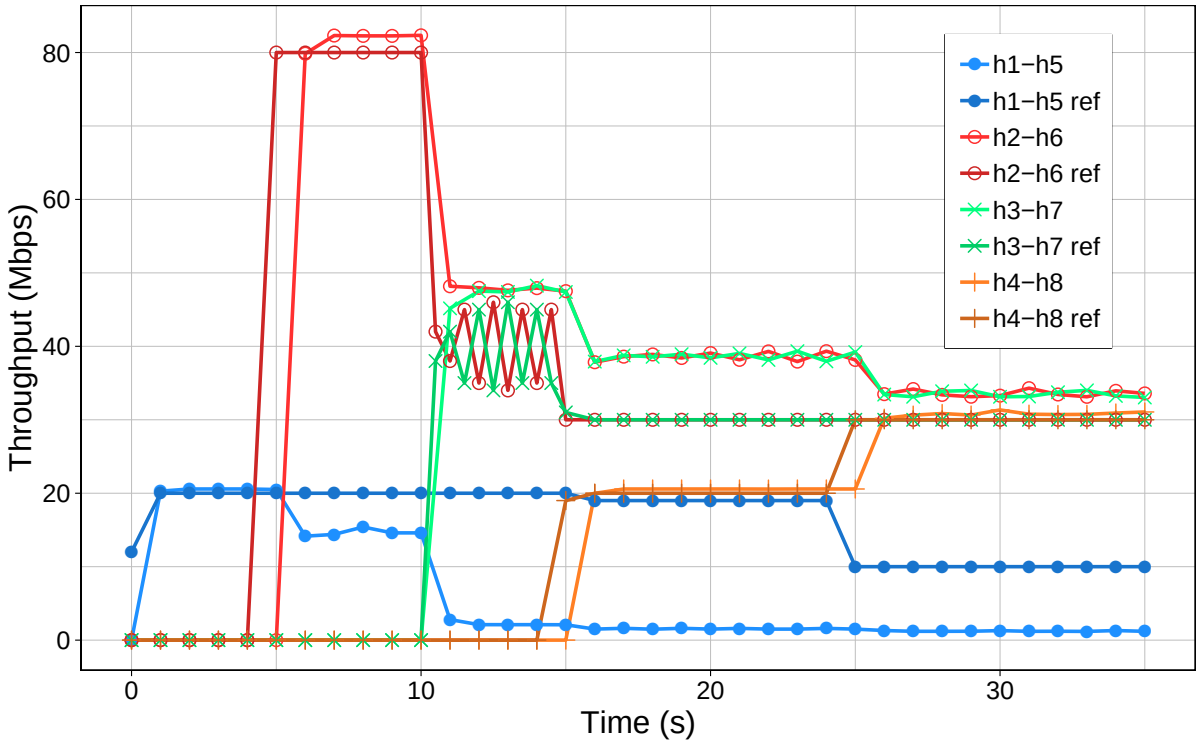


Figure 4.13: Comparison between reference experiment of throughput results at receivers

The results where QoS is applied follow the same trend as in the reference results (as compared in Figure 4.13), the only significance difference being the throughput received at host h5. In [93], the excess bandwidth after all queues are satisfied are allocated to the traffic going to host h5, while in the design of this study, the excess bandwidth is given to the pairs that have more traffic on them (hosts h6 and h7) as per the HTB algorithm.

When the reproducing the experiment, all the network traffic is monitored by the controller, by requesting the queue statistics every second. These results are shown in Figure 4.14 and follow a similar trend to the measured results in Figure 4.13, with a slight delay when new network traffic starts to flow in the network. More details about the controller monitoring the network are given in the next chapter.

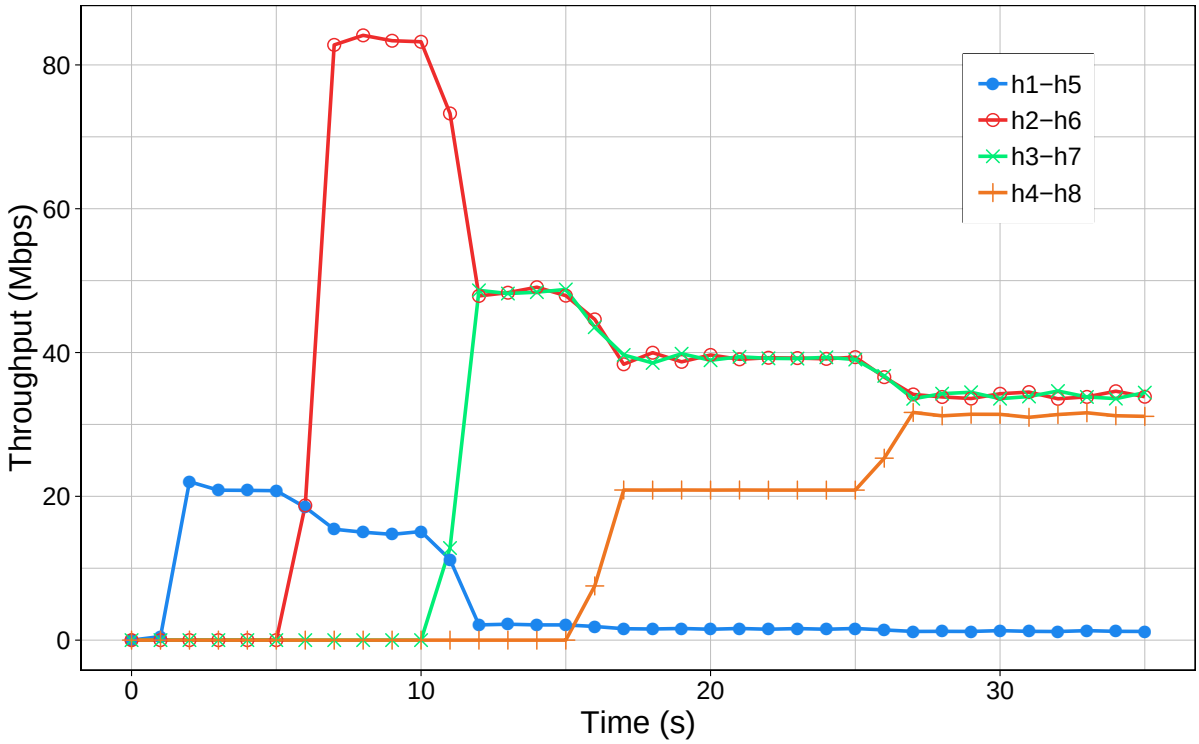


Figure 4.14: Throughput results as measured by controller

4.5 Concluding Remarks

In this chapter, experiments with different types of traffic were performed to prove that static QoS can be provided to users connected to an SDN-enabled router. Design verification by graphical comparisons is done, while experiment validation is performed by comparing results performed on the proposed design with published results. The next step is to consider the provisioning dynamic QoS, which can only be achieved by monitoring the traffic of the home network, and the scalability of such a design where monitoring takes place.

Chapter 5

Monitoring and Scalability Results

In this chapter the monitoring of home networks are discussed and evaluated, based on the proposed design of chapter 3. The implementation of dynamic QoS based on the available measurements at the controller is outlined in section 5.1. An experiment to test the control plane scalability of such monitoring is designed and implemented in section 5.2, with its results given in section 5.3.

5.1 Monitoring

The SDN controller can monitor network traffic by retrieving the statistics of the queues installed on the SDN-enabled switch. This is done by using the OpenFlow protocol (`ofp_queue_multipart_request` and `ofp_queue_stats` messages). The controller can install flows on the switch where each flow's action corresponds to a different queue; thus each queue can be set to monitor different types of traffic (e.g. TCP or UDP) on different ports going to different network devices. Figure 5.1 shows an example of how the controller can measure UDP and TCP traffic going to the hosts.

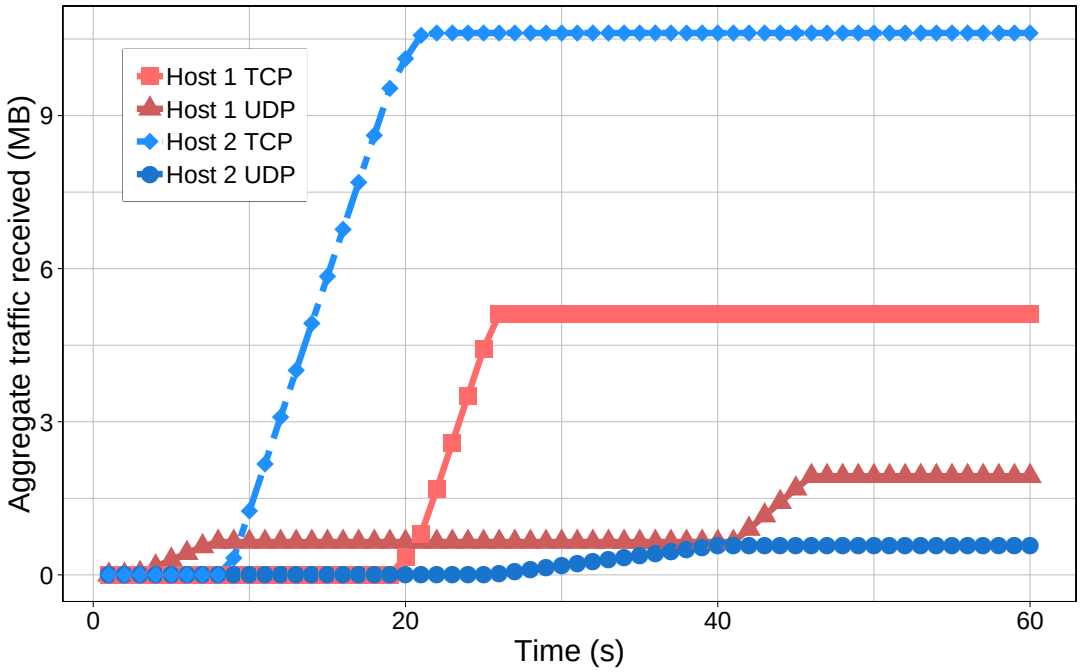


Figure 5.1: Traffic monitoring of two hosts

The signalling overhead in SDN networks still requires more research to understand its implications fully [59]. It is often overlooked by other SDN home network proposals, as highlighted in the Related Work section in chapter 2 (section 2.5). Control plane scalability issues can arise for the SDN controller if the trade-off between measurement accuracy, timeliness and the amount of overhead traffic is not considered. In this study, the signalling overhead and scalability of the proposed design are investigated. The only overhead present that SDN introduces to the network in the proposed design is the messages between the ISP switch and the controller.

5.2 Scalability Experiment Setup

To test the control plane scalability of a design where the traffic of several home networks are monitored by a controller (Figure 5.2), a different number of queues are installed on the ISP router. This OVS is connected to the Ryu controller which is hosted in its own Virtual Machine (VM). In this study, as in [94], the overhead traffic consid-

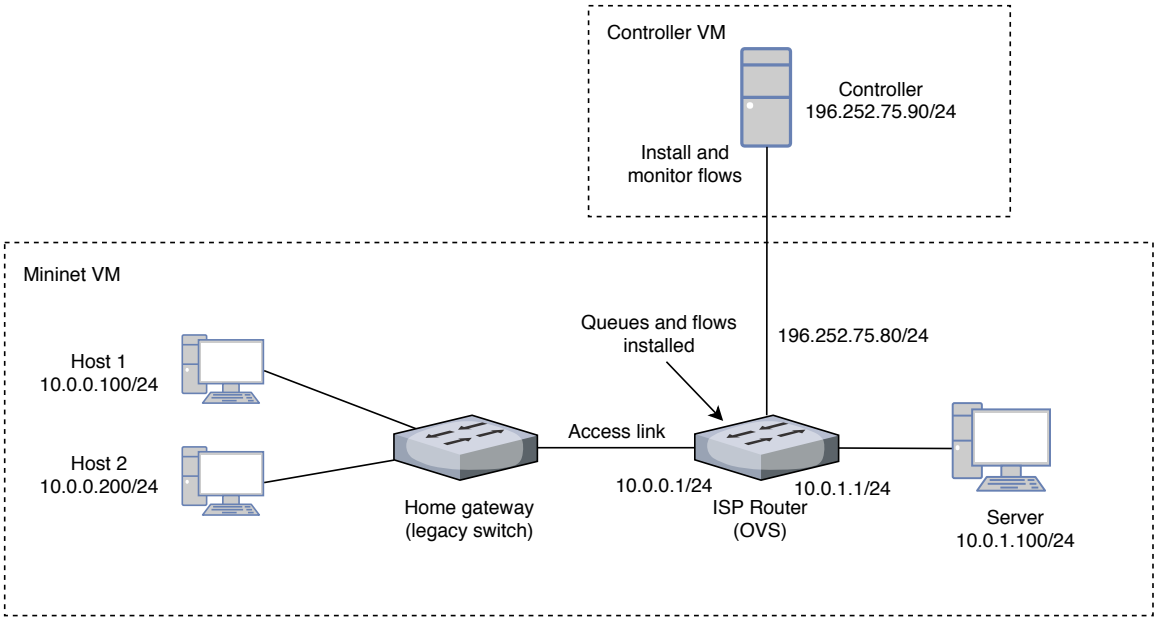


Figure 5.2: Scalability experiments setup

ered is the statistic reply messages sent from the SDN-enabled switch (ISP router) to the controller. As this monitoring traffic between the OVS and controller is only dependent on the number of queues added, only one home network consisting of a legacy switch and two hosts (host 1 and host 2) and a host (server) acting as traffic generator are added to the network (all emulated in Mininet in their own VM).

The controller monitors the switch by polling the queue statistics every second. This is done to establish a baseline of scalability for the proposed design and to simplify the experiment. Many monitoring systems use an adaptive polling system (such as [98] and [99]) to find a trade-off between accuracy and less overhead.

Table 5.1: Scalability metrics measured in experiment

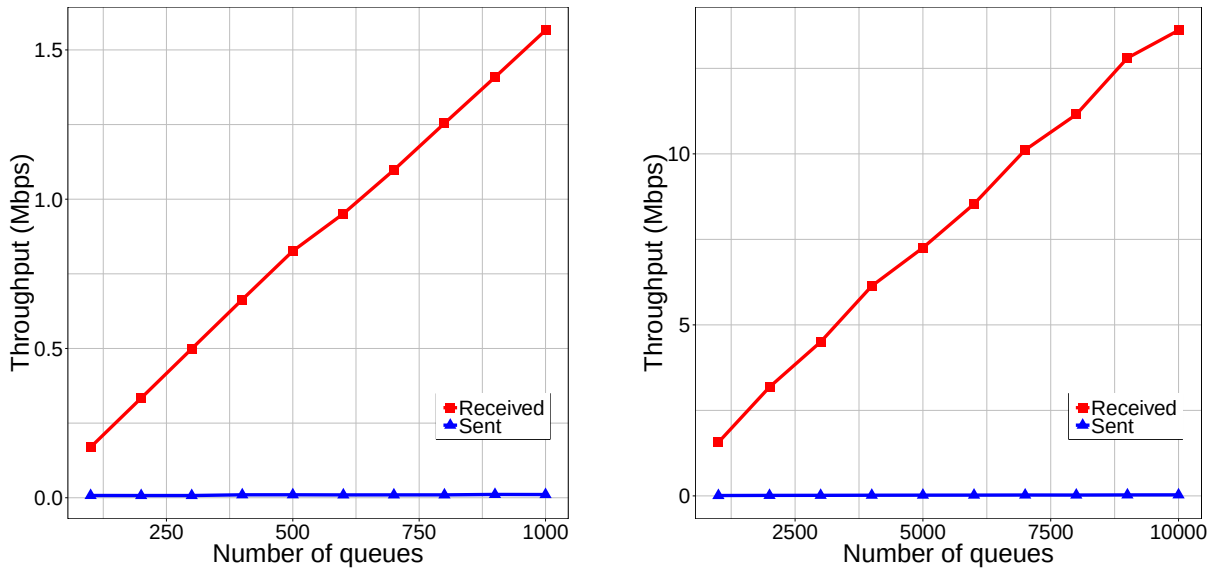
Metric	Unit	Measurement utility	Version
Throughput	Megabits per second (Mbps)	bwm-ng	v0.6
CPU use	Percentage (%)	pidstat [95]	11.2.0
RAM use	Kilobytes (KB)	pidstat	11.2.0
Flow installation time	Seconds (s)	Linux date function [96]	8.25
Packet length	Kilobytes (KB)	tcpdump [97]	4.9.2

The experiment runs for one minute for each number of queues added to the switch. For each queue value, the experiment is repeated 5 times. The metrics measured, with their units and utilities used to measure it, are given in Table 5.1. Two sets of experiments are done: firstly the number of queues ranges from 100 to 1000 with an increment for every 100 and secondly where the queues range from 1000 to 10 000 with an increment for every 1000 queues.

5.3 Results

5.3.1 Bandwidth

Figure 5.3 shows the average throughput measured, as sent and received by the controller, on the controller-switch link. This traffic is not present on the access link (between the home legacy switch and the ISP router) and thus the client will not be billed for the bandwidth used on this link. The amount of traffic sent by the controller stays consistent for a different number of queues, as the amount of statistic request messages



(a) 100 to 1000 queues

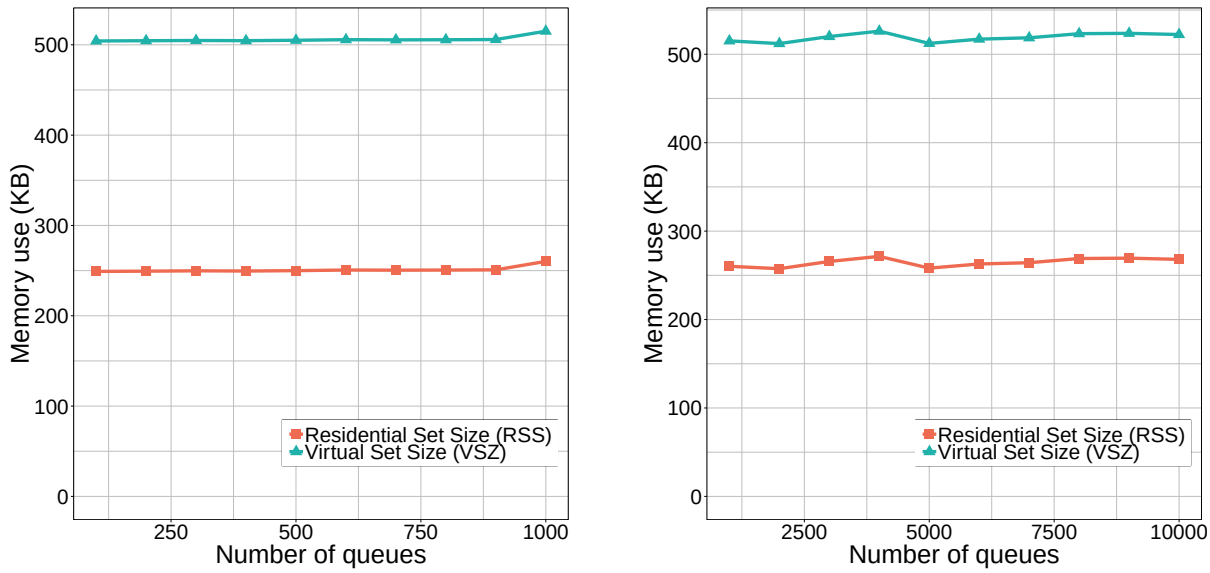
(b) 1000 to 10 000 queues

Figure 5.3: Bandwidth results

stays the same. The traffic received increases linearly as the queues increases.

5.3.2 RAM Use

Figure 5.4 shows the Random Access Memory (RAM) used by the controller (the ryu-manager process). Both Resident Set Size (RSS) and Virtual Set Size (VSZ) memory use are shown. RSS is the non-swapped physical memory that is currently being used by a process, while VSZ is the total memory allocated to that process [100]. The memory used by the controller stays relatively consistent as the number of queues increase. This means that the controller is scalable in terms of memory use and that the amount of memory the controller needs is not a physical constraint for monitoring larger networks.



(a) 100 to 1000 queues

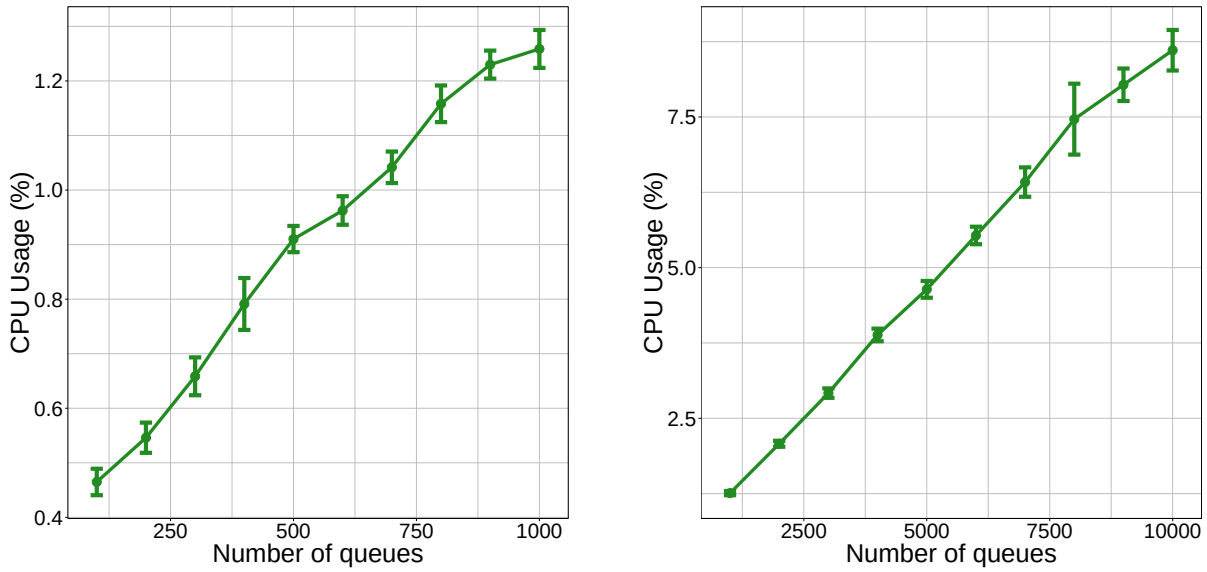
(b) 1000 to 10 000 queues

Figure 5.4: RAM use results

5.3.3 CPU Use

Figure 5.5 shows the average percentage of the Central Processing Unit (CPU) used by the controller (the ryu-manager process). It increases fairly linearly as the amount of

queues increases. For some number of queue values (especially at 8000), the standard deviation is quite large, as seen by the size of the error bars. This means that the CPU use by the controller varies and is not necessarily a constant value for a given number of queues that are monitored. The highest average percentage measured (about 8.5%) is not significantly high, but can have an impact on a machine that also runs other critical tasks, and care should be taken to not impose a too large workload on it.



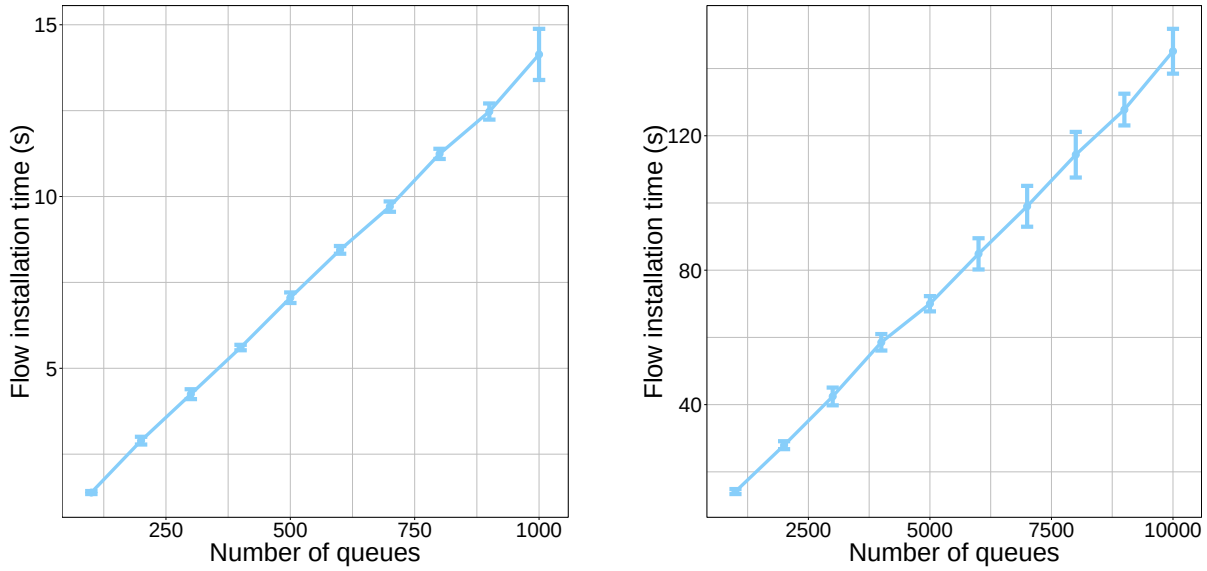
(a) 100 to 1000 queues

(b) 1000 to 10 000 queues

Figure 5.5: CPU use results

5.3.4 Flow Installation Time

Figure 5.6 shows the average time it takes for the controller to install flows on Open vSwitch, where each flow corresponds to one queue. This can be a factor to consider when the controller must be reset and reconfigured to monitor all the traffic again. It increases fairly linearly as the amount of queues increases. As the number of queues increases, the variability of the time also increases, as shown by the larger error bars.



(a) 100 to 1000 queues

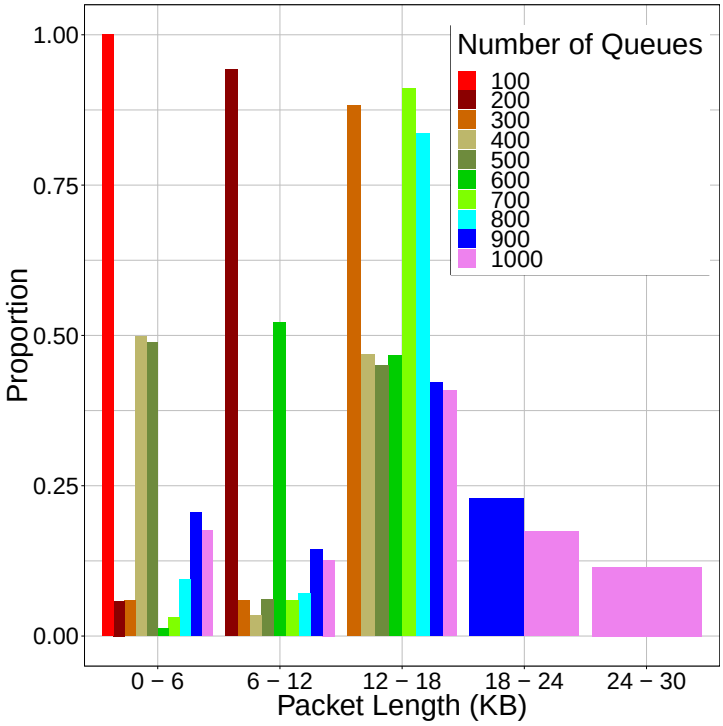
(b) 1000 to 10 000 queues

Figure 5.6: Flow installation time results

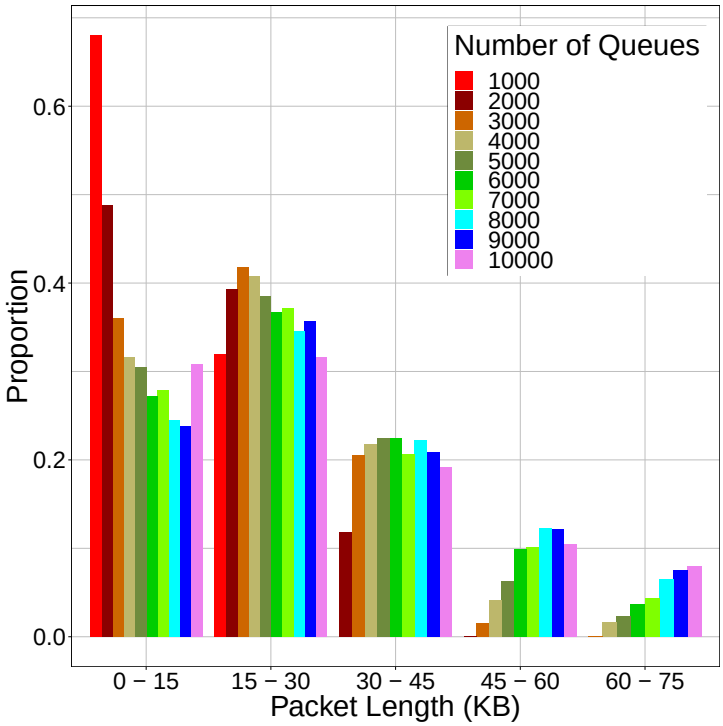
5.3.5 Statistic Reply Packet Lengths

As more queues are monitored, the payload length of the statistic reply packet also increases. Packet fragmentation occurs if the length is too large. Figure 5.7 shows the distribution of the packet lengths where each colour represents a different number of queues. All the statistic reply packets received for all five iterations of the experiment were aggregated, and the proportion that a length will occur in the specified bins were calculated. Packet fragmentation occurs already when 200 queues are used. As OpenFlow messages are transported in a TCP container and the maximum length of a TCP packet is 65 535 bytes, packets larger than 60 000 bytes are only present when the number of queues is 3000 or more.

Figure 5.8 displays the Empirical Cumulative Distribution Function (ECDF) plots of the packet lengths. When 100 queues are installed, all the packets have a length of 4016 bytes with no fragmentation taking place. As the number of queues increases, the packet fragmentation frequency also generally increases, with are visualised with less straight vertical lines and acute step-wise increments in the figures.

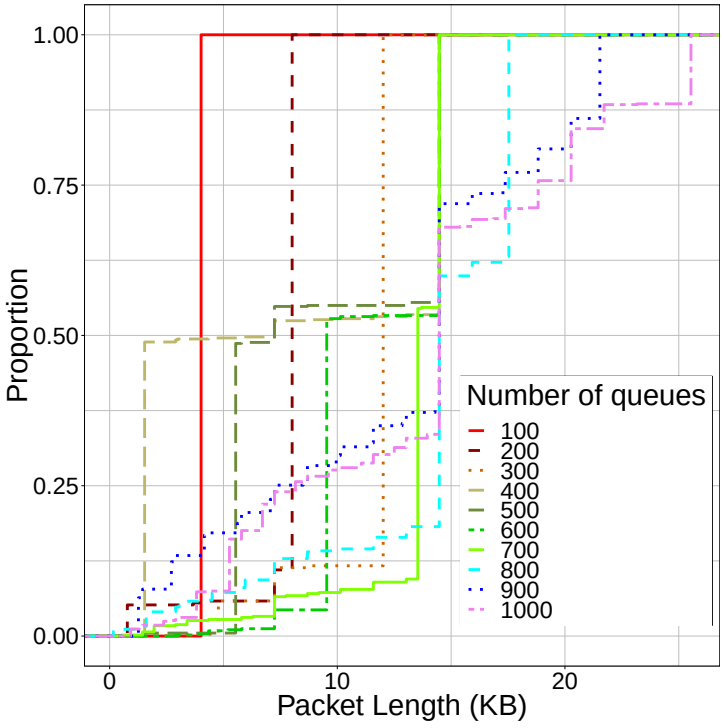


(a) 100 to 1000 queues

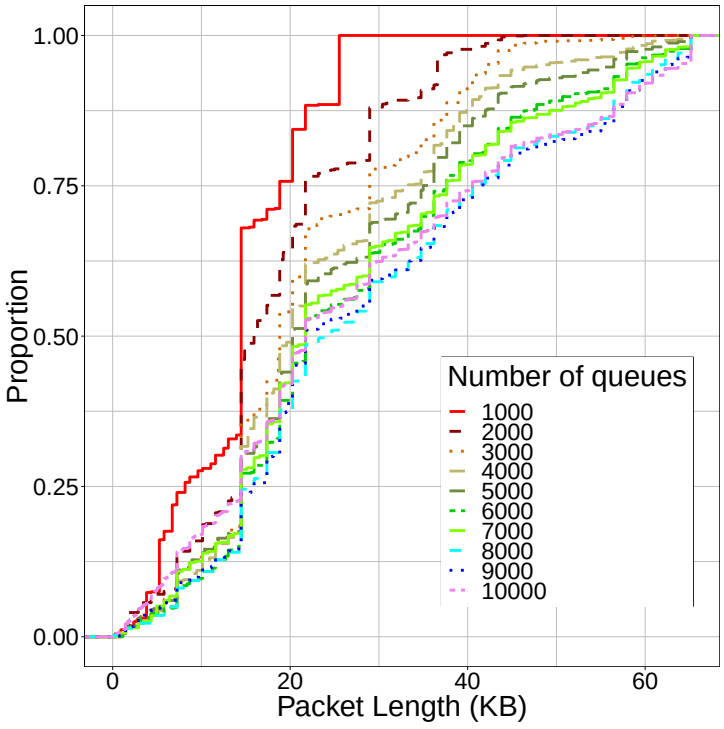


(b) 1000 to 10 000 queues

Figure 5.7: Length of statistic reply packet probability histogram results



(a) 100 to 1000 queues

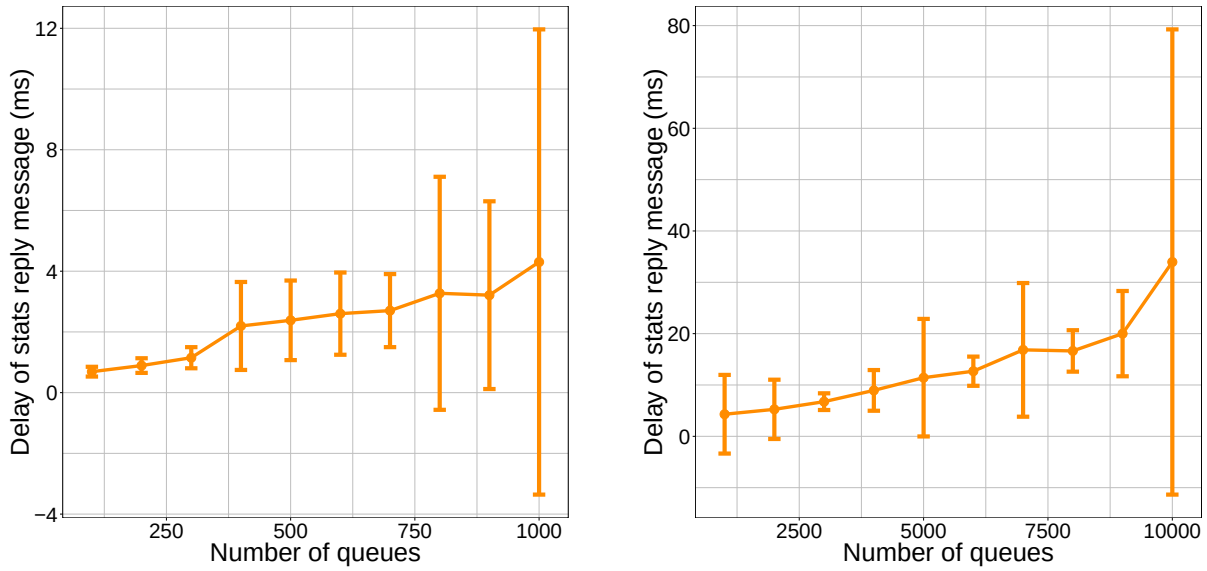


(b) 1000 to 10 000 queues

Figure 5.8: Length of statistic reply packet probability ECDF results

5.3.6 Statistic Reply Packets Delay

Because of the packet fragmentation, the time it takes for all the statistic reply packets to arrive at the controller increases. The delay is determined by calculating the time difference between when the statistic request message is sent and the arrival of the final packet of the reply message. Figure 5.9 displays the results of the average delay for different number of queues. The large error bars indicate that the delay time varies a lot, especially as the number of queues increases. The time that it will take for all the reply packets to arrive is thus not a value that can easily be estimated based just on the number of queues that are being monitored.



(a) 100 to 1000 queues

(b) 1000 to 10 000 queues

Figure 5.9: Average delay of the statistic reply packets

5.4 Concluding Remarks

Various monitoring mechanisms are possible with the proposed design. The scalability of a design where the controller monitors the queue statistics on the SDN-enabled switch every second was investigated. The bandwidth consumption, CPU used and

flow installation time scales linearly as the number of queues increases, while the RAM used stays consistent. Packet fragmentation occurs when a large number of queues are used and can have an impact on the processing time and data integrity of the controller.

Chapter 6

Conclusion

In this chapter the dissertation is concluded. An overview of the whole dissertation is given in section 6.1. The key research findings and contributions are also highlighted. Recommendations for possible future work that can follow this study are discussed in section 6.2, with some final closing remarks in section 6.3

6.1 Dissertation Overview

This dissertation started with an introduction to SDN and the advantages it provides over current legacy networks. Two research problems were formulated to be answered in this study: firstly how an ISP can leverage SDN to improve QoS for its home network clients, and secondly how scalable such a design would be if it monitored the home networks.

A comprehensive literature study was conducted, where an overview of home networks, SDN and QoS concepts were given. Related work studies were surveyed, and the shortcomings of the literature were identified. Based on these limitations, a de-

sign framework was presented that incorporates the existing legacy devices present in today's home networks. Design choices were made (using the AHP to choose the most suitable SDN controller) and implemented in emulation. A GUI was designed to interact with the network devices through the SDN controller.

The design was evaluated by emulating different types of network traffic and comparing the results for two cases where the controller provided no QoS and provided bandwidth priority for certain traffic respectively. As the design setup generated the expected results, the design was verified. The experiment was validated by recreating a published experiment [93] and comparing the output with the published result.

The controller can monitor the network traffic going to the home networks by installing queues on the SDN-enabled access switch hosted at the ISP and polling the queue statistics. It was shown how the controller could monitor the traffic going to different home network devices over different protocols (TCP and UDP packets) by using the installed queues. The scalability of the design was evaluated by measuring various performance metrics of the machine in which the controller was running.

6.1.1 Research Findings and Contributions

The following research objectives (as listed in section 1.3) were achieved during the course of this study:

- Existing SDN implementations for home networks were researched by conducting an extensive survey on related work. It was seen that the designs presented rarely incorporate existing legacy network devices together with SDN-enabled devices. The scalability of the designs was also almost never considered.
- A system framework was designed that connects the switches that provide the access links to home networks (hosted by an ISP) to an SDN controller. The design allows that the legacy routers currently used at home networks to not be replaced.

- The design was implemented and verified by emulation in Mininet, using Open vSwitch, OpenFlow and the Ryu controller. An interface was developed to simplify the interaction between an ISP operator and the controller.
- The QoS provisioning capabilities of the design were quantitatively investigated, by emulating different types of network traffic (iperf and video streaming data) and measuring various QoS metrics (throughput, jitter, packet loss and RTT). Two use cases, where QoS is implemented and not implemented by the controller, were compared with each other. The experiment was validated by performing a network traffic scenario and comparing the results to published, peer-reviewed research.
- The scalability of the design was investigated while it monitored the queue statistics of network traffic going to home networks. The effect of a different number of queues installed at the ISP access network switch on several metrics related to the controller (throughput between controller and switch, memory use, CPU use, flow installation time, packet fragmentation and packet delay) were studied and analysed.

A significant contribution of this study is the comprehensive literature survey that was conducted on existing published research about improving home networks using SDN. A similar survey was done and presented in [6], where the focus was on whether the studies surveyed contains cloud-based implementations and make use of third-party developers. In this study, only research involving improving the QoS and monitoring home networks were considered, with the emphasis on their specific design implementations and evaluation procedures.

There are several open-source SDN controllers available that could have been used to implement the design. After all the available options were surveyed and several studies that compare the controllers were researched, the AHP was conducted to identify the most suitable controller for the experiments performed in this study. All the detailed comparison matrices and consistency ratio results, where the seven controllers are compared with each other, are given in Appendix A.2.

One notable advantage that SDN provides over legacy networks is the programmable interface that the controllers provide for developers. Not only does it provide a centralised point of control for potentially multiple connected network devices, but all the configuration could be done by using a high-level abstraction programming language such as Python, without using the lower-level language of the devices itself. The GUI that was developed in Python (using the tkinter library) to interact with the controller serve as proof for this concept, as it provides a simplified way to program the network devices.

As mentioned, the scalability of proposed SDN designs is rarely considered. As far as the authors are aware, this dissertation presents the first published results of the packet fragmentation and the delays of the statistic reply messages that occur while the controller monitors the queue statistics of an Open vSwitch device. The other scalability results can also be used by developers to consider the amount of overhead between the controller and switch, and adjust their implementations accordingly.

6.2 Recommendations for Future Work

The next step after emulating a design is to evaluate the experiment on actual hardware, either by implementing a prototype, or using a testbed of devices. At this stage, commercial switches with access network proviosining capabilities that are SDN- and OpenFlow-compatible are relatively expensive and not always a viable option for researches. The Zodiac GX device by Northbound Networks [101] offers a cheaper alternative to implement OpenFlow experiments and offers five 1 Gigabit ethernet ports. It runs OVS as ported on the OpenWrt (LEDE version 17) operating system. In this study, an effort was made to implement the experiments using the Zodiac GX device, but it was found that OVS queues were not configured correctly due to the underlying tc (traffic control) architecture of the switch.

Instead of using priority queues to provision QoS for a home network, other techniques such as network slicing can be used. The ISP router can be divided into different slices

for each home network connected or based on different applications used. Network slicing can also be used together with priority queues to improve the QoS for the home users.

In this study, only static QoS provisioning was implemented. It was shown in section 5.1 how the design could monitor the network traffic of home networks and provide measurements to the controller of network traffic going to different home devices over different protocols. This can be the input for a model that calculates dynamic QoS policies for the home network. Some initial designs to improve SDN networks (outside the context of solely home networks) with external 'solvers' that are given measured network data have been proposed already. For example, in [102] an Artificial Neural Network executed in Matlab is proposed to predict the performance of SDN devices, based on metrics such as the throughput and RTT.

Traffic classification implementations (such as Snort [103] and nDPI [104]) can also be used to distinguish between more types of network traffic and provide more focused QoS provisioning for home users. Such a classification platform for enterprise networks is described in [105].

6.3 Closure

The possibilities of SDN implementations to improve home networks in the future are plentiful, as SDN technologies provide a simple way for developers to configure and program network elements. In this dissertation, it was investigated whether SDN can be used to specifically improve the QoS of home networks. A framework was designed that an ISP can use to improve the QoS provisioning for the home networks of their clients. Using this framework, the network traffic can also be monitored. The studies in this dissertation show that there indeed exists a great potential for an ISP to leverage the power of SDN to improve the home networks they serve.

Bibliography

[1] T. Bakhshi, "State of the art and recent research advances in software defined networking," *Wireless Communications and Mobile Computing*, vol. 2017, 2017.

[2] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[3] P. Goransson and C. Black, *Software Defined Networks: A Comprehensive Approach*. Waltham, Massachusetts: Morgan Kaufmann, 2014.

[4] B. Salisbury. (2013) Inside Google's Software-Defined Network. [Online]. Available: https://www.networkcomputing.com/networking/inside-googles-software-defined-network/512240144

[5] I. T. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2713–2737, 2016.

[6] A. M. Alshnta, M. F. Abdollah, and A. Al-Haiqi, "SDN in the home: A survey of home network solutions using Software Defined Networking," *Cogent Engineering*, vol. 5, no. 1, pp. 1–40, 2018.

[7] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Pearson, 2013.

[8] S. Sundaresan, R. Teixeira, G. Tech, N. Feamster, A. Pescapè, and S. Crawford, "Broadband Internet Performance : A View From the Gateway," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 134–145, 2011.

[9] R. Mortier, T. Rodden, T. Lodge, D. Mcauley, C. Rotsos, A. W. Moore, A. Koliousis, and J. Sventek, "Control and Understanding: Owning Your Home Network," in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference*, 2012, pp. 1–10.

[10] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, 2011, pp. 1–6.

[11] Internet Society, "Bandwidth Management: Internet Society Technology Roundtable Series," 2012. [Online]. Available: http://www.internetsociety.org/doc/bandwidth-management-internet-society-technology-roundtable-series

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[13] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[14] M. Casado, N. Foster, and A. Guha, "Abstractions for Software-Defined Networks," *Communications of the ACM*, vol. 57, pp. 86–95, 2014.

[15] Ashton Metzler & Associates, "Ten Things to Look for in an SDN Controller," 2013. [Online]. Available: http://www.ashtonmetzler.com/HowtoEvaluateSDNControllers.pdf

[16] Open Networking Foundation, "Software-defined networking: The new norm for networks," 2012.

[17] H. Farhady, H. Lee, and A. Nakao, "Software-Defined Networking: A survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[18] S. Schaller and D. Hood, "Software defined networking architecture standardization," *Computer Standards and Interfaces*, vol. 54, pp. 197–202, 2017.

[19] OpenFlow - Open Networking Foundation (ONF). [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[20] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *Proceedings of the 18th Mediterranean Electrotechnical Conference (MELECON)*, 2016, pp. 1–6.

[21] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop*, 2013, pp. 13–18.

[22] BigSwitch. FloodLight. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[23] Z. Cai, A. Cox, and E. T. S. Ng, "Maestro: A System for Scalable OpenFlow Control," 2011. [Online]. Available: http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf

[24] MuL - High Performance SDN. [Online]. Available: http://www.openmul.org/

[25] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[26] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, and B. Lantz, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[27] OpenDaylight. [Online]. Available: https://www.opendaylight.org/

[28] Nicira. POX. [Online]. Available: https://github.com/noxrepo/pox

[29] Nippon Telegraph and Telephone Corporation. Ryu SDN Framework. [Online]. Available: http://osrg.github.io/ryul

[30] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of SDN controllers," *2013 IEEE International Conference on Communications*, pp. 3504–3508, 2013.

[31] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A reality check," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, pp. 79–85.

[32] A. Shalimov, D. Zimarina, and V. Pashkov, "Advanced Study of SDN / OpenFlow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, 2013, pp. 1–6.

[33] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, 2014, pp. 1–7.

[34] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[35] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using open flow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[36] Open Networking Foundation, "OpenFlow Switch Specification 1.3.0," 2012.

[37] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[38] N. McKeown and J. Rexford. (2016) Clarifying the differences between P4 and OpenFlow. [Online]. Available: https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.html

[39] Open vSwitch. [Online]. Available: http://openvswitch.org/

[40] Open vSwitch OpenWrt package. [Online]. Available: https://github.com/openwrt/packages/tree/master/net/openvswitch

[41] The OpenWrt Project - Wireless Freedom. [Online]. Available: https://openwrt.org/

[42] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, A. Networks, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 117–130.

[43] Indigo Virtual Switch. [Online]. Available: http://www.projectfloodlight.org/indigo-virtual-switch/

[44] ofsoftswitch13. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[45] S. Wang, K. G. Chavez, S. Kandeepan, and P. Zanna, "The smallest software defined network testbed in the world: Performance and security," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2018, pp. 1–2.

[46] B. Pfaff and B. Davie. (2013) The Open vSwitch Database Management Protocol - RFC 7047. [Online]. Available: https://tools.ietf.org/html/rfc7047

[47] B. A. Forouzan, *Data Communications and Networking*, 5th ed. McGraw-Hill Education, 2013.

[48] Y. Chen, K. Wu, and Q. Zhang, "From QoS to QoE: A tutorial on video quality assessment," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 1126–1165, 2015.

[49] ns-3 Network Simulator. [Online]. Available: https://www.nsnam.org/

[50] G. Judd and P. Steenkiste, "Using Emulation to Understand and Improve Wireless Networks and Applications," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005, pp. 203–216.

[51] Y. Yiakoumis, M. Bansal, A. Covington, J. Van, R. Sachin, and K. N. Mckeown, "BeHop: A Testbed for Dense WiFi Networks," 2014.

[52] OFELIA Testbed. [Online]. Available: http://www.fibre-ict.eu/index.php/testbeds/ofelia

[53] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[54] B. Lantz, N. Handigol, B. Heller, and V. Jeyakumar. (2017) Introduction to Mininet. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[55] K. Wich and J. Blackford, "TR-069 CPE WAN Management Protocol: Issue 1 Amendment 6," The Broadband Forum, 2018.

[56] Sericon Technology. (2014) Major Problems with TR-069. [Online]. Available: https://www.routercheck.com/2014/08/14/major-problems-tr-069/

[57] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An Operating System for the Home," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 337—-352.

[58] D. Pediaditakis, A. Gopalan, N. Dulay, M. Sloman, and T. Lodge, "Home network management policies: Putting the user in the loop," in *2012 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, 2012, pp. 9–16.

[59] M. Karakus and A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A survey," *Journal of Network and Computer Applications*, vol. 80, no. May 2016, pp. 200–218, 2017.

[60] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service: Leveraging SDN to efficiently and transparently support video-on-demand on the last mile," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2014.

[61] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song, "FlowQoS: Per-Flow Quality of Service for Broadband Access Networks," Georgia Institute of Technology, 2015.

[62] G. Agapiou, I. Papafili, and S. Agapiou, "The role of SDN and NFV for dynamic bandwidth allocation and QoE adaptation of video applications in home networks," in *2014 Euro Med Telco Conference - From Network Infrastructures to Network Fabric: Revolution at the Edges, EMTC 2014*, 2014, pp. 1–4.

[63] I. Trajkovska, P. Aeschlimann, C. Marti, T. M. Bohnert, and J. Salvachua, "SDN enabled QoS provision for online streaming services in residential ISP networks," in *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, 2014, pp. 33–34.

[64] H. Yang, X. Wang, C.-T. Nguyen, and S. Lu, "Conan: Content-aware access network flow scheduling to improve QoE of home users," in *2016 ACM Conference on Special Interest Group on Data Communication*, 2016, pp. 575–576.

[65] Y. Yiakoumis, S. Katti, T.-Y. Huang, N. McKeown, K.-K. Yap, and R. Johari, "Putting home users in charge of their network," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 2012, p. 1114.

[66] T. Fratczak, M. Broadbent, P. Georgopoulos, and N. Race, "HomeVisor: Adapting home network environments," in *Proceedings of the 2nd European Workshop on Software Defined Networks (EWSDN)*, 2013, pp. 32–37.

[67] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, no. November 2015, 2013, p. 15.

[68] H. Kumar, H. H. Gharakheili, and V. Sivaraman, "User control of quality of experience in home networks using SDN," in *2013 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, 2013, pp. 1–6.

[69] H. H. Gharakheili, J. Bass, L. Exton, and V. Sivaraman, "Personalizing the home network experience using cloud-based SDN," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014,*, 2014.

[70] S. Ramakrishnan and X. Zhu, "An SDN Based Approach to Measuring and Optimizing ABR Video Quality of Experience," Cisco Systems, 2014.

[71] S. Wang, X. Wu, H. Chen, Y. Wang, and D. Li, "An optimal slicing strategy for SDN based smart home network," in *Proceedings of 2014 International Conference on Smart Computing,*, 2014, pp. 118–122.

[72] H. Eghbali and V. W. Wong, "Bandwidth allocation and pricing for SDN-enabled home networks," in *IEEE International Conference on Communications*, vol. 2015-Septe, 2015, pp. 5342–5347.

[73] H. H. Gharakheili, L. Exton, V. Sivaraman, J. Matthews, and C. Russell, "Third-party customization of residential Internet sharing using SDN," in *25th International Telecommunication Networks and Applications Conference*, 2015, pp. 214–219.

[74] R. M. Abuteir, A. Fladenmuller, and O. Fourmaux, "SDN based architecture to improve video streaming in home networks," in *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)*, vol. 2016-May, 2016, pp. 220–226.

[75] ——, "An SDN approach to adaptive Video Streaming in Wireless home networks," in *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2016, pp. 321–326.

[76] T. Bakhshi and B. Ghita, "User-centric traffic optimization in residential software defined networks," in *23rd International Conference on Telecommunications*, 2016, pp. 1–6.

[77] I. N. Bozkurt and T. Benson, "Contextual Router: Advancing Experience Oriented Networking to the Home," in *Proceedings of the ACM SOSR '16 (poster/demo session)*, 2016, pp. 1–7.

[78] H. C. Jang, C. W. Huang, and F. K. Yeh, "Design a bandwidth allocation framework for SDN based smart home," in *7th IEEE Annual Information Technology, Electronics and Mobile Communication Conference*, 2016.

[79] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou, "Instrumenting home networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, p. 84, 2011.

[80] M. Chetty and N. Feamster, "Refactoring network infrastructure to improve manageability," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 3, p. 54, 2012.

[81] M. Chetty, H. Kim, S. Sundaresan, S. Burnett, N. Feamster, and W. K. Edwards, "uCap: An Internet Data Management Tool For The Home," in *Proceedings of the ACM CHI'15 Conference on Human Factors in Computing Systems*, vol. 1, 2015, pp. 3093–3102.

[82] K. Xu, X. Wang, W. Wei, H. Song, and B. Mao, "Toward software defined smart home," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 116–122, 2016.

[83] Introducing JSON. [Online]. Available: https://www.json.org/

[84] M. Devera. (2002) HTB Linux queuing discipline manual - user guide. [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

[85] tc-htb - Linux man page. [Online]. Available: https://linux.die.net/man/8/tc-htb

[86] Python Software Foundation. Graphical User Interfaces with Tk. [Online]. Available: https://docs.python.org/2/library/tk.html

[87] Volker-Gropp. (2015) Bandwidth Monitor NG. [Online]. Available: https://www.gropp.org/?id=projects&sub=bwm-ng

[88] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu. iPerf. [Online]. Available: https://iperf.fr

[89] MyBroadband. (2017) Average Wi-Fi speeds in South Africa. [Online]. Available: https://mybroadband.co.za/news/wireless/213598-average-wi-fi-speeds-in-south-africa.html

[90] Big Buck Bunny. [Online]. Available: https://peach.blender.org/

[91] VideoLAN Organization. (2018) VLC media player. [Online]. Available: https://www.videolan.org

[92] R. G. Sargent, "Verification and validation of simulation models," in *Proceedings of the 2009 Winter Simulation Conference*, 2010, pp. 162–176.

[93] H. Krishna, "Providing End-to-end Bandwidth Guarantees with OpenFlow," Master's thesis, Delft University of Technology, 2016.

[94] Q. He, X. Wang, and M. Huang, "OpenFlow-based low-overhead and high-accuracy SDN measurement framework," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 2, 2018.

[95] pidstat - Linux man page. [Online]. Available: https://linux.die.net/man/1/pidstat

[96] date - Linux man page. [Online]. Available: https://linux.die.net/man/1/date

[97] tcpdump - Linux man page. [Online]. Available: https://linux.die.net/man/8/tcpdump

[98] N. V. Adrichem and C. Doerr, "OpenNetMon: Network Monitoring in OpenFlow SDNs," in *IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–8.

[99] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014.

[100] ps Linux Man-page. [Online]. Available: http://man7.org/linux/man-pages/man1/ps.1.html

[101] Zodiac GX. [Online]. Available: https://northboundnetworks.com/products/zodiac-gx

[102] A. Sabbeh, Y. Al-Dunainawi, H. S. Al-Raweshidy, and M. F. Abbod, "Performance prediction of software defined network using an artificial neural network," in *2016 SAI Computing Conference*, 2016, pp. 80–84.

[103] Snort Home Page. [Online]. Available: https://www.snort.org/

[104] nDPI. [Online]. Available: https://www.ntop.org/products/deep-packet-inspection/ndpi/

[105] B. Ng, M. Hayes, and W. K. Seah, "Developing a traffic classification platform for enterprise networks with SDN: Experiences & lessons learned," in *Proceedings of 2015 14th IFIP Networking Conference*, 2015, pp. 1–9.

[106] T. L. Saaty, "How to make a decision: The Analytic Hierarchy Process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, 1990.

[107] eigen - RDocumentation. [Online]. Available: https://www.rdocumentation.org/packages/base/versions/3.5.1/topics/eigen

Appendix A

Analytic Hierarchy Process

A.1 Background

The analytic hierarchy process (AHP) is a formalised and systematic approach to decide between different options. The simplicity and power of the AHP make it a popular decision-making tool.

The first step of the AHP is to deconstruct the decision-making problem into a hiearchy of goals, criteria and alternatives. Figure A.1 shows the decomposition of choosing the SDN controller.

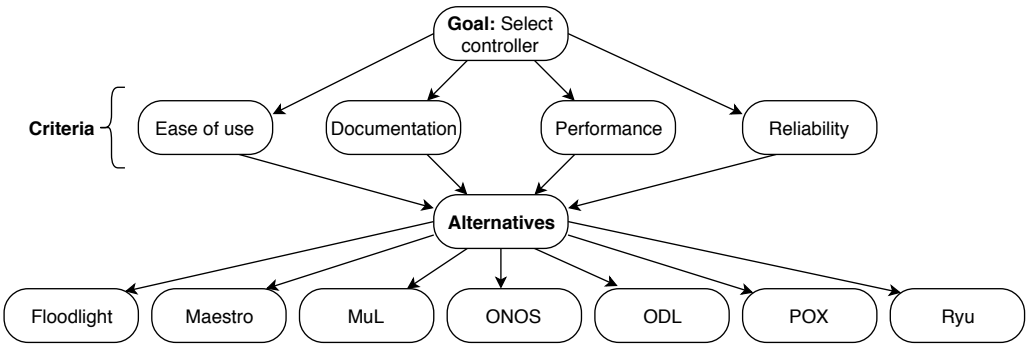


Figure A.1: AHP hierarchy visualisation

Pairwise comparisons are made for the different criteria with each other on a qualitative scale. A value between 1 and 9 is assigned to the criterion that has a higher preference for each pair. The meaning of the preference values are given in Table A.1 [106].

Table A.1: Pairwise comparison scale

Scale	**Degree of preference**
1	Equal importance
3	Moderate importance of one factor over another
5	Strong or essential importance
7	Very strong importance
9	Extreme importance
2,4,6,8	Intermediate values

The pairwise comparisons are organised into a square matrix, with the diagonals equal to 1 and all the other values being the reciprocal of their diagonally opposite value. This means that the value at position (i, j) is the reciprocal of the value at (j, i). Pairwise comparisons are also made between all the alternatives for each criterion of which the results are also organised into a similarly structured square matrix. Each of these matrices is known as a judgement matrix.

Each value of the judgement matrix is then divided by the total of that column to calculate the normalised average of each row. These averages are known as the priority vector for the specific criterion. Each priority vector is then multiplied with the weight of the corresponding criteria and aggregated to calculate the global rating. The alternative with the highest global rating is deemed the best choice.

As the AHP is the subjective comparison of different alternatives, some inconsistency is to be expected. The consistency of the comparisons must, therefore, be verified to ensure that the results are trustworthy. The Consistency Index (CI) of a judgement matrix of order n is calculated with equation A.1, where λ_{max} is the maximum eigenvalue of the matrix.

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{A.1}$$

The calculated CI can be compared with the average CI of several random-like matrices, called the Random Index (RI). The RI values (calculated in [106]) for different orders of matrices are given in Table A.2. The ratio of the CI and RI is defined as the Consistency Ratio (CR), as shown in equation A.2. In [106] it is suggested that a CR of 0.1 or less (thus less than 10% inconsistency) is acceptable for the AHP. The pairwise comparisons in the judgement matrix should be reconsidered if the CR is greater than 0.1.

$$CR = \frac{CI}{RI} < 0.1 \tag{A.2}$$

Table A.2: Average RI value for the different orders of a matrix

n	1	2	3	4	5	6	7	8	9	10
RI	0	0	0.52	0.89	1.11	1.25	1.35	1.40	1.45	1.49

A.2 Results

A.2.1 Criteria Comparison

Table A.3 displays the pairwise comparison matrix for the different criteria. Each value is then divided by the total of that column to calculate the normalised matrix, shown in Table A.4. The average of each row is given in the last column, which represents the normalised weight for each criterion.

The largest eigenvalue (λ_{max}) of the judgement matrix in Table A.3 is calculated using the `eigen` function in the `R Statistical Software` [107]. Table A.5 shows the CR of

the criteria pairwise comparisons, calculated with equations A.1 and A.2. As the CR is less than 0.1, the matrix is seen as consistent.

Table A.3: Criteria pairwise comparison

Criteria	Ease of use	Documentation	Performance	Reliability
Ease of use	1	3	5	7
Documentation	0.333	1	3	5
Performance	0.2	0.333	1	3
Reliability	0.143	0.2	0.333	1
Total	1.676	4.533	9.333	16

Table A.4: Criteria normalised average

	Ease of use	Documentation	Performance	Reliability	Average
Ease of use	0.597	0.662	0.536	0.438	0.558
Documentation	0.199	0.221	0.321	0.313	0.263
Performance	0.119	0.074	0.107	0.188	0.122
Reliability	0.085	0.044	0.036	0.063	0.057

Table A.5: Criteria consistency ratio

λ_{max}	CI	RI (n=4)	CR
4.117	0.039	0.89	0.044 <0.1

A.2.2 Alternatives Comparison

Tables A.6, A.9, A.12 and A.15 display the pairwise comparison matrix between the different controllers for each of the criteria. The normalised results are shown in Tables A.7, A.10, A.13 and A.16, with the last column in each representing the priority vector of that particular criterion. The CI and CR of each judgement matrix are shown in Tables A.8, A.11, A.14 and A.17. Each CR is less than 0.1, which indicates consistent pairwise comparisons.

Table A.6: Ease of use pairwise comparison

Ease of use	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu
Floodlight	1	3	3	5	3	0.333	0.333
Maestro	0.333	1	0.333	3	3	0.333	0.333
MuL	0.333	3	1	5	3	0.333	0.333
ONOS	0.2	0.333	0.2	1	1	0.2	0.2
ODL	0.333	0.333	0.333	1	1	0.2	0.2
POX	3	3	3	5	5	1	1
Ryu	3	3	3	5	5	1	1
Total	8.2	13.667	10.867	25	21	3.4	3.4

Table A.7: Ease of use normalised average

Ease of use	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu	Average
Floodlight	0.122	0.220	0.276	0.200	0.143	0.098	0.098	0.165
Maestro	0.041	0.073	0.031	0.120	0.143	0.098	0.098	0.086
MuL	0.041	0.220	0.092	0.200	0.143	0.098	0.098	0.127
ONOS	0.024	0.024	0.018	0.040	0.048	0.059	0.059	0.039
ODL	0.041	0.024	0.031	0.040	0.048	0.059	0.059	0.043
POX	0.366	0.220	0.276	0.200	0.238	0.294	0.294	0.270
Ryu	0.366	0.220	0.276	0.200	0.238	0.294	0.294	0.270

Table A.8: Ease of use consistency ratio

λ_{max}	CI	RI (n=4)	CR
7.468	0.078	1.35	0.058 <0.1

Table A.9: Documentation pairwise comparison

Documentation	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu
Floodlight	1	5	3	0.333	0.333	5	1
Maestro	0.2	1	0.333	0.143	0.143	0.333	0.2
MuL	0.333	3	1	0.2	0.2	0.333	0.333
ONOS	3	7	5	1	1	7	3
ODL	3	7	5	1	1	7	3
POX	0.2	3	3	0.143	0.143	1	0.333
Ryu	1	5	3	0.333	0.333	3	1
Total	8.733	31	20.333	3.152	3.152	23.667	8.867

Table A.10: Documentation normalised average

Documentation	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu	Average
Floodlight	0.115	0.161	0.148	0.106	0.106	0.211	0.113	0.137
Maestro	0.023	0.032	0.016	0.045	0.045	0.014	0.023	0.028
MuL	0.038	0.097	0.049	0.063	0.063	0.014	0.038	0.052
ONOS	0.344	0.226	0.246	0.317	0.317	0.296	0.338	0.298
ODL	0.344	0.226	0.246	0.317	0.317	0.296	0.338	0.298
POX	0.023	0.097	0.148	0.045	0.045	0.042	0.038	0.063
Ryu	0.115	0.161	0.148	0.106	0.106	0.127	0.113	0.125

Table A.11: Documentation consistency ratio

λ_{max}	CI	RI (n=4)	CR
7.436	0.073	1.32	0.054 <0.1

Table A.12: Performance pairwise comparison

Performance	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu
Floodlight	1	7	0.333	3	3	9	5
Maestro	0.143	1	0.143	0.2	0.2	3	0.333
MuL	3	7	1	5	5	9	7
ONOS	0.333	5	0.2	1	3	7	5
ODL	0.333	5	0.2	0.333	1	7	3
POX	0.111	0.333	0.111	0.143	0.143	1	0.2
Ryu	0.2	3	0.143	0.2	0.333	5	1
Total	5.121	28.333	2.130	9.876	12.676	41	21.533

Table A.13: Performance normalised average

Performance	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu	Average
Floodlight	0.195	0.247	0.156	0.304	0.237	0.220	0.232	0.227
Maestro	0.028	0.035	0.067	0.020	0.016	0.073	0.015	0.036
MuL	0.586	0.247	0.469	0.506	0.394	0.220	0.325	0.393
ONOS	0.065	0.176	0.094	0.101	0.237	0.171	0.232	0.154
ODL	0.065	0.176	0.094	0.034	0.079	0.171	0.139	0.108
POX	0.022	0.012	0.052	0.014	0.011	0.024	0.009	0.021
Ryu	0.039	0.106	0.067	0.020	0.026	0.122	0.046	0.061

Table A.14: Performance consistency ratio

λ_{max}	CI	RI (n=4)	CR
7.710	0.118	1.32	0.088 <0.1

Table A.15: Reliability pairwise comparison

Reliability	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu
Floodlight	1	3	3	0.333	0.333	0.333	0.143
Maestro	0.333	1	1	0.2	0.2	0.2	0.111
MuL	0.333	1	1	0.2	0.2	0.2	0.111
ONOS	3	5	5	1	3	1	0.333
ODL	3	5	5	0.333	1	1	0.333
POX	3	5	5	1	1	1	0.2
Ryu	7	9	9	3	3	5	1
Total	17	29	29	6.067	8.733	8.733	2.232

Table A.16: Reliability normalised average

Reliability	Floodlight	Maestro	MuL	ONOS	ODL	POX	Ryu	Average
Floodlight	0.059	0.103	0.103	0.055	0.038	0.038	0.064	0.066
Maestro	0.020	0.034	0.034	0.033	0.023	0.023	0.050	0.031
MuL	0.020	0.034	0.034	0.033	0.023	0.023	0.050	0.025
ONOS	0.176	0.172	0.172	0.165	0.344	0.115	0.149	0.185
ODL	0.176	0.172	0.172	0.055	0.115	0.115	0.149	0.136
POX	0.176	0.172	0.172	0.165	0.115	0.115	0.090	0.144
Ryu	0.412	0.310	0.310	0.495	0.344	0.573	0.448	0.413

Table A.17: Reliability consistency ratio

λ_{max}	CI	RI (n=4)	CR
7.292	0.049	1.32	0.036 <0.1

Appendix B

Conference Article Contribution

The following article was published in the proceedings of the Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2018. The conference is hosted annually by Telkom and was held from 2 to 5 September 2018 at the Arabella Hotel, near Hermanus, South Africa. The article was presented as part of the Access Network Technologies sessions. The article can be referenced as follows:

A. Taute and M. Ferreira, "Leveraging SDN for QoS in Home Networks," in *Proceedings of Southern Africa Telecommunication Networks and Application Conference (SATNAC) 2018*, Hermanus, South Africa, Sept. 2018, pp. 20-25.

Leveraging Software-Defined Networking for QoS in Home Networks

Anton Taute, Melvin Ferreira
School of Electrical, Electronic and Computer Engineering
North-West University, Potchefstroom Campus, South Africa
[1]24250058@student.g.nwu.ac.za
[2]melvin.ferreira@nwu.ac.za

***Abstract*—Home networks today face unique provisioning challenges. As the network is growing and more complex devices are added, the Internet Service Provider (ISP) may struggle to fulfil the specific requirements of each home user. This paper investigates the advantages of Software-Defined Networking (SDN) and proposes a way the ISP can leverage it to improve the Quality of Service (QoS) in home networks. An experimental setup that resembles a typical South African home network is tested in the Mininet emulation environment. Two use cases, where the capabilities of the SDN controller are utilised and not utilised, are quantitatively compared using four QoS parameters, namely bandwidth, jitter, datagram loss and round-trip time. For each use case, two different scenarios (one with simulated traffic and one with video streaming traffic) are investigated. It is found that by implementing the SDN controller, the QoS of a host can indeed be improved. The framework presented in this paper forms a basis on which the ISP can build a platform to improve the QoS of their users' home networks. This paper also serves as a foundation for future practical implementations and is documented to be fully reproducible.**

***Index Terms*—Home networks, Internet Service Provider, Mininet, Quality of Service, Software-Defined Networking**

I. Introduction

Software-Defined Networking (SDN) is a new approach to implement, operate and maintain networks. The concept of SDN was developed to make networks more configurable and flexible [1]. Since its inception, various studies have tested SDN predominantly at core- and campus networks. Large corporations such as Google have already utilised SDN to improve their data centres [2]. More recently, studies have been done to incorporate SDN in home networks as well.

The fundamental difference between SDN and traditional networking is the presence of an SDN controller which creates an external, centralised point of control. The centralising of the network control offers several benefits: modification of the network is less error-prone and more straightforward to execute, high-level policies can be maintained by automatically reacting to changes in the network state, and the process of developing network servers, functions and applications are simplified [3].

An Internet Server Provider (ISP) may be able to leverage these advantages of SDN to improve the Quality of Service (QoS) of their clients' access networks. As home networks grow and become more complex, it becomes increasingly complicated for the ISP to guarantee a specified QoS for a particular service. This is especially a concern for real-time applications, such as live video streaming and conferencing.

In this paper, we aim to exploit the advantages of SDN in home networks by emulating a scenario as two different use cases: with and without utilising the capabilities of SDN. The allocation of the available bandwidth across applications within a single home network is tested. Four QoS parameters (bandwidth, jitter, datagram loss and round-trip time) are measured and compared to find the quantitative difference between the use cases.

Two questions form the rationale of the experiment:

- Does it make a difference in the QoS of a home network when using SDN compared to a traditional non-SDN network?
- Can an ISP leverage SDN to improve the QoS to its home network client-base?

The rest of this paper is organised as follows: in section II literature background on home networks and SDN networks are given; section III gives an overview of the work similar to that done in this study; section IV outlines the overall design and context of the experiment while section V gives the detailed methodology that was followed to perform the experiment; in section VI the results of the experiment are presented, and lastly, section VII discusses the outcome of the experiment.

II. Background

A. Home networks

Home networks are located on the periphery of a centralised network. Usually, this refers to the devices at the 'edge' of the internet and consist of a wide variety of wired and wireless end systems or hosts, such as desktop computers, servers, mobile computers (laptops, smartphones and tablets) and a wide range of IoT-enabled (Internet of Things) devices [4]. Home networks are also sometimes known as SOHO (Small-office / home-office) networks.

The management of home networks is faced with unique challenges. As more internet-enabled devices are produced and used, home networks become more complex and unmanageable [5]. It is therefore difficult to keep network access control and policies consistent, and it is common for a typical home network to be poorly managed, insecure and broken [6].

These networks are prone to failure with no measures in place to systematically improve services after deployment.

According to Gharakheili et al. [7], home networks do not provide scalable solutions for user customisation, as the user needs higher levels of technological expertise, and the customisations vary across different vendors with no uniform solutions. Broadband networks are also relatively expensive to deploy, and there are no clear guidelines on how the cost can be shared among several service providers [6]. Home networks can experience a bottleneck problem as the access network can become easily congested [8].

Dixon et al. [9] describe three challenges of home networks, namely management (including security and access control), application development (because of heterogeneity of topology, devices and coordination) and incremental growth (where the compatibility of new technology and its integration with the existing technology can create problems).

B. SDN

Commercial switches and routers generally do not provide an open software platform or a way to virtualise their hardware or software. Thus the switch's internal flexibility is hidden, while the internals also differs between different vendors. This created a high barrier that prevented researchers from experimenting with new ideas and contributing to network innovation [10]. The SDN paradigm and, in particular, OpenFlow, was developed to give a practical way to experiment with and test new network protocols and ideas. This is accomplished by making the network more reconfigurable and 'programmable' through the centralised point of control.

The architecture of software-defined networks is composed of different layers that are vertically connected, as shown in Fig. 1. On the bottom is the hardware infrastructure, which consists of the forwarding devices. Any SDN-enabled device (e.g. switches) can be deployed in the network. Open vSwitch [11] is one of the most used, open-source, virtual switches available.

The southbound interface on top of the hardware forms the connection between the control and data plane elements. OpenFlow [10] is the most widely used standard for this interface and is a product of the Open Networking Foundation (ONF) [12]. The middle layer consists of the SDN Controller, on which the Network Operating System (NOS) runs. The controller must coordinate the flow set-up as originated by the network applications and update each element to keep the network state consistent [3]. This provides logically centralised control for the network.

On top of the controller platform is the northbound interface, which forms the connection between the control and management plane elements. There is no defined standard yet, and existing controllers usually propose or define their own APIs [1]. Network applications form the top layer of the SDN architecture. Typical functions of these applications include to measure and monitor the network, provide security, perform traffic engineering to balance the load and to minimise power consumption.

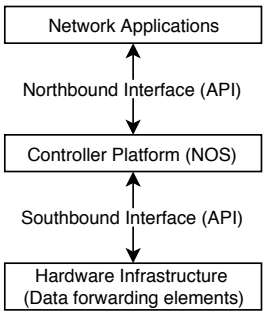


Fig. 1. SDN Layer Architecture [3]

III. Related Work

To integrate SDN technology with home networks can bring advantages for the users [13]. The centralised control plane can coordinate resource allocation to efficiently and fairly utilise the available bandwidth. This allocation can take place across multiple home networks and also across applications within a single home network. In this paper, the latter is implemented.

An overview of SDN studies we surveyed in the context of home networks is given in Table I. Several studies focused on improving the QoS and QoE of video streaming in home networks using SDN-based strategies. In terms of evaluation, three studies used simulation or emulation to test their design, another three implemented their own experimental prototypes and two used large testbeds. Of all the studies, only one [14] is documented in a way to accurately reproduce the experiments.

All except one study [7] advocated that the home router (or residential gateway) should be replaced with an SDN-enabled device. In this paper an alternative approach is followed where the ISP hosts the SDN controller which is only connected to the SDN-enabled routers at the ISP. This setup enables users to keep their current (legacy) home routers and allow the ISP to control the access link by leveraging SDN.

Haque & Abu-Ghazaleh [13] identify three open challenges for SDN-based home networks. Except for [15], scalable and robust software enabled home networks are not yet addressed. Task distribution and hierarchical designs are also lacking in literature of home network designs. This can result in further variation and longer response time in network performance. To have ease of use and network architectures that can operate even with different network providers, multi-home interference mitigation needs to be further investigated.

This paper aims to build on the existing work by presenting SDN for home networks from the perspective of the ISP. It also serves as a foundation for future practical implementations by being fully reproducible.

IV. Experimental Design

In order to make it easier for the ISP to configure home networks, the centralised point of control that the SDN controller offers can be exploited. The controller can be connected to ISP routers that are connected to home gateways (providing the access link for the homes) as shown in Fig. 2.

TABLE I
OVERVIEW OF RELATED WORK

Author	Contributions	Use Cases	Evaluation
Ramakrishnan et al. [16]	Optimised approach to improve QoE for users	Video streaming	Custom Simulation
Georgopoulos et al. [17]	Improve video-on-demand with in-network cache	Video streaming	OFELIA Testbed
Gharakheili et al. [7]	User network customisation by user and service differentiation	Video streaming and conferencing	Prototype
Zinner et al. [18]	Dynamic resource allocation on per-flow basis	Video streaming	Prototype
Lee et al. [19]	Cloud-based home connection and management	Device registration	Mininet emulation
Seddiki et al. [14]	Specify priorities for different application flows	Video streaming	Prototype
Abuteir et al. [20]	Network Assisted Video Streaming by dynamic traffic shaping	Video streaming	NS3 simulation
Moyano et al. [21]	Residential network management applications	Network monitoring; video streaming	Virtualised testbed

The design assumes that the ISP routers providing the access links are SDN compatible, but the legacy gateways in the home networks can be used and do not have to be replaced. In a non-SDN environment, QoS provisioning must be done on each individual home gateway. By leveraging SDN, the ISP can use the centralised controller to potentially provide QoS for multiple homes. The controller also typically allows the user to program the routers with a simpler high-level language (such as Python or Java; see Table III for specific examples) than legacy router programming languages.

To test if it is possible for the ISP to use the SDN controller to implement QoS in a home network, two use cases are compared with each other:

- Case 1: The SDN controller sets no QoS priority queues. This case is equivalent to a traditional non-SDN network.
- Case 2: The SDN controller installs a static QoS flow at a specified port of a device's interface to guarantee a minimum amount of bandwidth for all applications on that port.

The static QoS flow in case 2 is installed for only one host in the home network. The host's traffic on the specified port will thus have priority over the rest of the access link traffic. This can be especially useful when the host is streaming a video while the rest of the home devices are busy with non-time sensitive tasks (like downloading files from the internet). The video traffic is given priority and gives the best viewing experience possible for the user. Four QoS parameters are measured in each use case, as shown in Table II, to quantitatively compare the cases with each other.

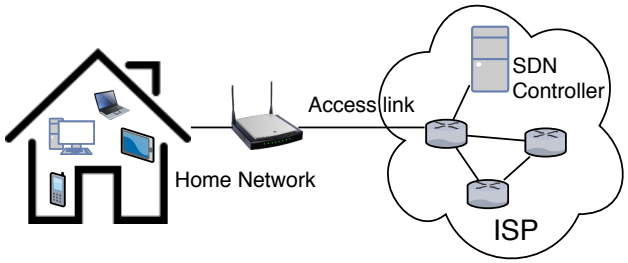


Fig. 2. SDN implemented for home networks from the ISP's perspective.

V. METHODOLOGY

The first step to compare the two use cases as explained in the previous section is to emulate a typical home network and implement each case. The Mininet software [24] (v2.3.0d1) is chosen to perform this. Mininet creates and runs virtual networks (consisting of virtual hosts, switches, controllers and links) running real switch and application code on a single Linux kernel. As such, Mininet is described as a network emulation orchestration system [25]. Mininet was implemented to be flexible, deployable, interactive, scalable, realistic and share-able. A Python API (running Python v2.7) is also included for customisation of the networks.

Fig. 3 displays the experimental network setup that is tested. Host 1 and host 2 are two computers forming part of the emulated home network. Both are connected to the home router within the same subnet. This represents one of the home networks shown in Fig. 2. The home router functions as a gateway to the network of the ISP, which is represented by another router and the server. The ISP router is connected to the SDN controller, as the QoS rules are installed on this router to shape the access link traffic going to the home router. The server is used to emulate traffic for hosts 1 and 2. The static QoS in the second use case is implemented for the traffic going to host 1.

The bandwidth of the access links between the home router and ISP router is limited to 7.2 Mbps, as this is the average internet download speed in South Africa [26]. The links between the hosts and home router, as well as between the server and ISP router, are limited to 100 Mbps.

Open vSwitch (v2.5.2) is chosen as the virtual switch software to make the hardware devices SDN compatible and is installed on the ISP router. OpenFlow is chosen as the southbound interface between the router and controller, as this is the most widely used open-source standard available [3].

TABLE II
EXPERIMENT PARAMETERS MEASURED

Parameter	Unit	Measurement Utility
Bandwidth	Megabits per second (Mbps)	bwm-ng [22]
Datagram loss	Percentage (%)	iperf [23] (UDP mode)
Jitter	Milliseconds (ms)	iperf (UDP mode)
Round-trip time (RTT)	Milliseconds (s)	ping

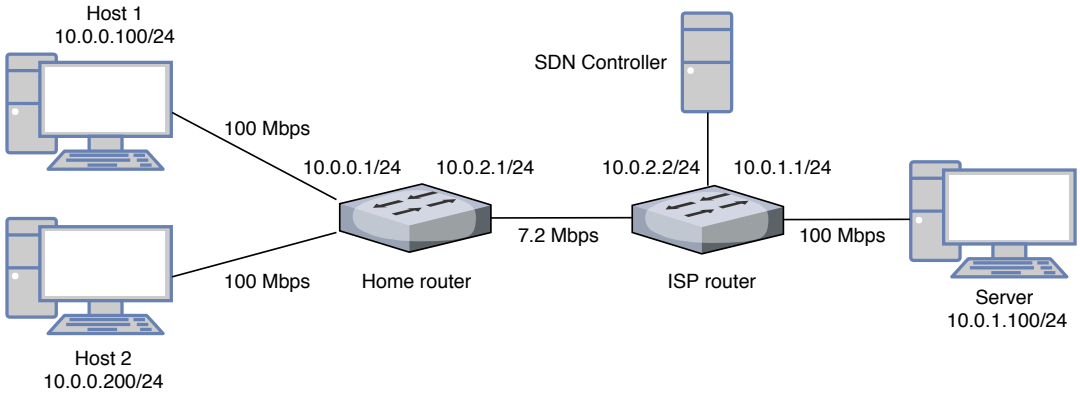


Fig. 3. Experimental setup with two hosts, a server, two routers and SDN controller

Version 1.3 of OpenFlow is used, as it has the capabilities to set priority queues and flow meters with maximum and minimum rates [27].

There exists a variety of open-source SDN controllers. An overview is given in Table III. In [28] a feature-based comparison is made for most of the controllers listed in Table III. The analytic hierarchy process (AHP) is used to determine a score for each controller. Features such as interfaces, platform support, documentation, modularity and OpenFlow support are considered. At the end of the process, the Ryu controller scored the highest. Similar considerations were taken for this study, and after performing an AHP of our own, the Ryu controller (v4.22) was chosen for the experiment.

The Ryu controller uses the REST API interface via curl commands to interact with. The static QoS rule is installed by sending an `OFPT_FLOW_MOD` message from the controller to the Openflow-enabled ISP router. The traffic on the link is shaped using the Hierarchy Token Bucket (HTB) algorithm, as implemented by the Linux-htb packet scheduler [29].

The experiment is emulated in Virtualbox (v5.2.8) with the Ubuntu 16.04 (64-bit) operating system as guest. Four Processors (with VT-x acceleration) and 8 GB of RAM are allocated to the guest machine. Virtualbox is installed on a Windows 10 Pro (64-bit) host machine, with an Intel i7-7700 3.6 GHz CPU, four physical cores (eight logical processors) and 16 GB RAM.

VI. Results

Two scenarios with different network traffic are used to implement each use case with the experiment setup:

- Scenario 1: The server sends a constant stream of traffic (5 Mbps) to host 1 using iperf (on port 5004) for one minute. After 30 seconds, the server sends a constant stream of traffic (5 Mbps) to host 2 using iperf (on port 5001).
- Scenario 2: Host 1 is streaming a video (the 480p version of the widely used Big Buck Bunny video[1]) hosted by the server for one minute. The video streaming is done using vlc media player (2.2.2) [37] over the Real-Time Transport Protocol (RTP) on port 5004. After 30 seconds, the server sends a constant stream of traffic (5 Mbps) to host 2 using iperf.

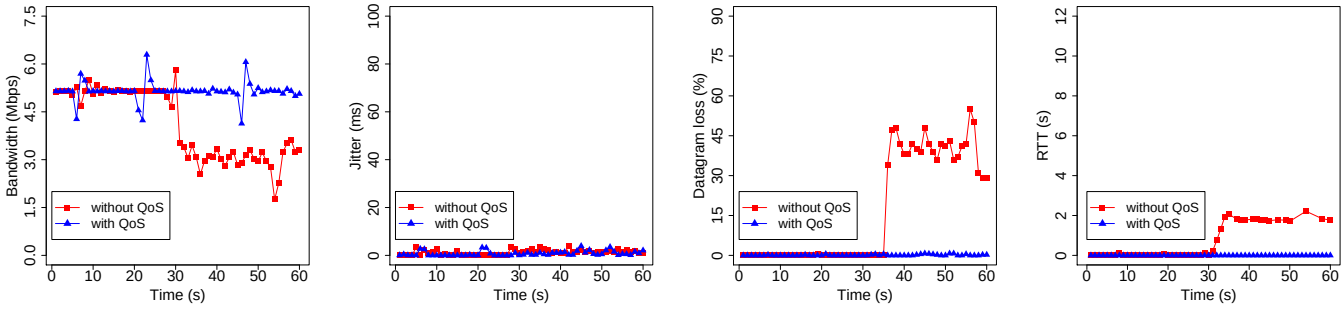
[1] www.bigbuckbunny.org

TABLE III
Overview of Open-source SDN controllers

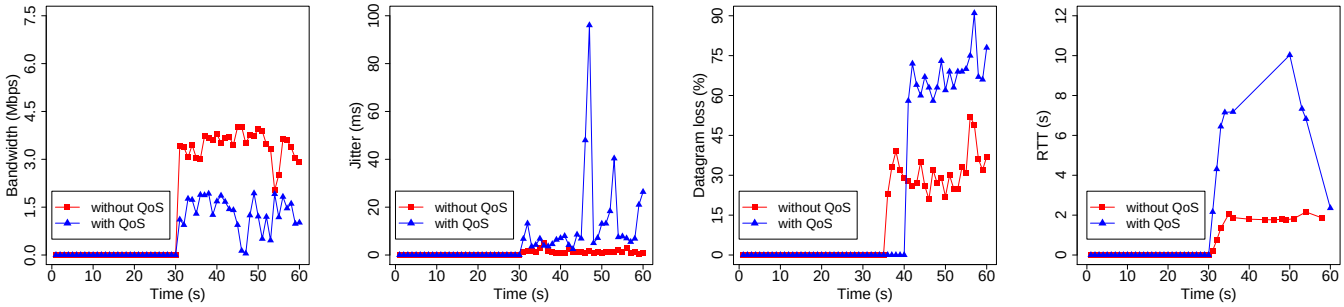
Controller	Language	Developer	Primary Use	AHP Score as per [28]
POX [30]	Python	Nicira	Fast prototyping	0.146
Beacon [31]	Java	Stanford University	Threading	N/A
Floodlight [32]	Java	BigSwitch	Campus networks	0.275
MuL [33]	C	Kulcloud	Data centres	N/A
Open-Daylight [34]	Java	Linux Foundation	Data centres	0.268
Ryu [35]	Python	NTT OSRG group	Fast prototyping	0.287
ONOS [36]	Java	Linux Foundation	Data centres	N/A

In the first case, no QoS is implemented by the SDN controller. In the second case, a static QoS rule for host 1 is installed on the ISP router. This rule guarantees 5 Mbps of bandwidth for UDP traffic on port 5004. In scenario 1, the four QoS parameters as shown in Table II are measured in both cases for hosts 1 and 2. In scenario 2, only the bandwidth is measured on the hosts. To calculate the average and standard deviation values, the emulation is performed ten times for each case and scenario.

The results of the first scenario are shown in Fig. 4. In each graph, the red lines with square markers represent the first case (without QoS), while the blue lines with triangle markers represent case 2 (with QoS). For the first 30 seconds, the QoS parameters at host 1 is similar for both cases. In the first case, the bandwidth decreases while the jitter, datagram loss and RTT increase, while host 2 starts to utilise the link as well. In the second case (with QoS), the bandwidth at host 1 is overall higher, while the jitter, datagram loss and RTT are less, as in the first case. The average and standard deviation for each QoS parameter and each case are shown in Table

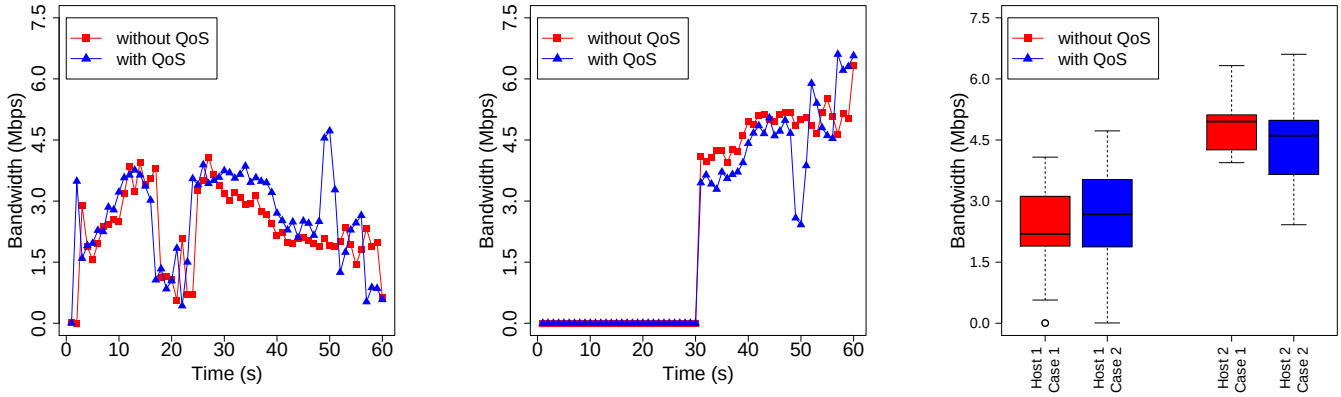


(a) Bandwidth of host 1 (b) Jitter of host 1 (c) Datagram loss of host 1 (d) Round-trip time of host 1



(e) Bandwidth of host 2 (f) Jitter of host 2 (g) Datagram loss of host 2 (h) Round-trip time of host 2

Fig. 4. Results of scenario 1 with case 1 (without QoS) in red with square markers and case 2 (with static QoS) in blue with triangle markers.



(a) Bandwidth of host 1 (b) Bandwidth of host 2 (c) Box plot of scenario 2

Fig. 5. Results for scenario 2 of case 1 (without QoS) in red with square markers and case 2 (with static QoS) in blue with triangle markers.

IV, host 1 for the whole minute and host 2 for the last 30 seconds. The averages of the second case's QoS parameters at host 1 improve from the first case's and have a lower standard deviation, indicating fewer variability. The opposite occurs at host 2, where the averages of the second case's QoS parameters are worse than the first case's with higher variability, except at the bandwidth.

The bandwidth results of the second scenario are shown in Fig. 5, with a box plot illustrating the spread of each use case's data. A similar trend is seen as in the first scenario. In the second case (with QoS), the video streaming data is not reduced for host 1, as in the first case (without QoS), where host 2 also utilises the available bandwidth. The average and standard deviation for each QoS parameter and each case are shown in Table V. The parameters are calculated, again, for the whole minute at host 1 and the last 30 seconds at host 2. The average and variation increase at host 1 from the first case to the second, as the host is free to utilise more bandwidth of the access link. At host 2 the average decreases while the

TABLE IV
QoS Statistics for Scenario 1

Host	Use Case	Bandwidth		Jitter		Datagram loss		RTT	
		Avg.	SD.	Avg.	SD.	Avg.	SD.	Avg.	SD.
1	1	4.13	1.18	1.72	1.95	16.19	19.61	0.77	0.87
1	2	5.12	0.57	1.14	1.63	0.12	0.23	0.00	0.02
2	1	3.21	0.55	2.19	2.27	30.78	15.68	1.69	0.55
2	2	1.31	0.65	27.24	51.73	41.93	33.67	5.64	3.42

variation increases, as the video traffic of host 1 is given priority.

TABLE V
QoS Statistics for Scenario 2

Host	Use Case	Bandwidth	
		Avg.	SD.
1	1	2.36	0.93
1	2	2.60	1.11
2	1	4.84	0.54
2	2	4.49	1.08

VII. Conclusion and Future Work

In this paper, an experiment was designed to test if SDN can improve the QoS of a home network. Two use cases were emulated with two different network traffic scenarios. The SDN controller was able to guarantee a particular QoS for one of the hosts and performed consistently better than the cases where no QoS was implemented. The framework presented in this paper can be used by the ISP to leverage the advantages of SDN to improve the QoS of home networks. The ISP can host SDN controllers with each managing a number of home networks. The controllers can install different QoS policies on each home router, based on the needs of those users.

This experiment also forms the foundation for future work on implementing SDN to improve home networks. The next use case that will be investigated is the installation of dynamic QoS rules. The SDN controller must monitor the traffic in the access link and then determine whether to install or remove QoS policies on certain ports for specific hosts and traffic. After the emulation process, the experiments will be repeated on a testbed with real hardware.

Acknowledgement

We acknowledge and thank the Telkom Centre of Excellence (CoE) for their financial support.

References

[1] B. A. A. Nunes *et al.*, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] B. Salisbury. (2013) Inside Google's Software-Defined Network. [Online]. Available: https://www.networkcomputing.com/networking/inside-googles-software-defined-network/512240144

[3] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Pearson, 2013.

[5] R. Mortier *et al.*, "Control and Understanding: Owning Your Home Network," *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference*, pp. 1–10, 2012.

[6] Y. Yiakoumis *et al.*, "Slicing home networks," *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks - HomeNets '11*, p. 1, 2011.

[7] H. H. Gharakheili *et al.*, "Personalizing the home network experience using cloud-based SDN," *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, WoWMoM 2014*, 2014.

[8] Internet Society, "Bandwidth Management: Internet Society Technology Roundtable Series," p. 8, 2012.

[9] C. Dixon *et al.*, "An Operating System for the Home," *Nsdi*, pp. 337—-352, 2012.

[10] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[11] Open vSwitch. [Online]. Available: http://openvswitch.org/

[12] (2017) OpenFlow - Open Networking Foundation (ONF). [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[13] I. T. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2713–2737, 2016.

[14] M. S. Seddiki *et al.*, "FlowQoS: Per-Flow Quality of Service for Broadband Access Networks," Georgia Institute of Technology, 2015.

[15] J. Rückert, R. Bifulco, and M. Rizwan-Ul-Haq, "Flexible traffic management in broadband access networks using Software Defined Networking," *2014 IEEE Network*, 2014.

[16] S. Ramakrishnan and X. Zhu, "An SDN Based Approach to Measuring and Optimizing ABR Video Quality of Experience," Cisco Systems, 2014.

[17] P. Georgopoulos *et al.*, "Cache as a service: Leveraging SDN to efficiently and transparently support video-on-demand on the last mile," *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 2014.

[18] T. Zinner *et al.*, "Dynamic application-aware resource management using Software-Defined Networking: Implementation prospects and challenges," *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014.

[19] M. Lee, Y. Kim, and Y. Lee, "A home cloud-based home network auto-configuration using SDN," *ICNSC 2015 - 2015 IEEE 12th International Conference on Networking, Sensing and Control*, pp. 444–449, 2015.

[20] R. M. Abuteir, A. Fladenmuller, and O. Fourmaux, "SDN based architecture to improve video streaming in home networks," *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, vol. 2016-May, pp. 220–226, 2016.

[21] R. Flores Moyano, D. Fernández Cambronero, and L. Bellido Triana, "A user-centric SDN management architecture for NFV-based residential networks," *Computer Standards and Interfaces*, vol. 54, no. April 2016, pp. 279–292, 2017.

[22] Volker-Gropp. (2015) Bandwidth Monitor NG. [Online]. Available: https://www.gropp.org/?id=projects&sub=bwm-ng

[23] J. Dugan *et al.* iPerf. [Online]. Available: https://iperf.fr

[24] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," *Workshop on Hot Topics in Networks*, pp. 1–6, 2010.

[25] B. Lantz *et al.* (2017) Introduction to Mininet. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[26] MyBroadband. (2017) Average Wi-Fi speeds in South Africa. [Online]. Available: https://mybroadband.co.za/news/wireless/213598-average-wi-fi-speeds-in-south-africa.html

[27] "OpenFlow Switch Specification 1.3.0," Open Networking Foundation, 2009.

[28] R. Khondoker *et al.*, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, 2014, pp. 1–7.

[29] (2018) tc-htb - Linux man page. [Online]. Available: https://linux.die.net/man/8/tc-htb

[30] Nicira. POX. [Online]. Available: https://github.com/noxrepo/pox

[31] D. Erickson, "The beacon openflow controller," *Proceedings of the second ACM SIGCOMM workshop*, pp. 13–18, 2013.

[32] BigSwitch. FloodLight. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[33] (2018) MuL - High Performance SDN. [Online]. Available: http://www.openmul.org/

[34] OpenDaylight. [Online]. Available: https://www.opendaylight.org/

[35] Nippon Telegraph and Telephone Corporation. Ryu SDN Framework. [Online]. Available: http://osrg.github.io/ryul

[36] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, pp. 1–6, 2014.

[37] VideoLAN Organization. (2018) VLC media player. [Online]. Available: https://www.videolan.org

Anton Taute obtained his B.Eng degree in Computer and Electronic Engineering in 2016. He is a Telkom CoE student pursuing his M.Eng degree at the North-West University. His research interests include Software-Defined Networking and network emulation.