Towards the Development of an Optimal SDN Controller Placement Framework to Expedite SDN Deployment in Emerging Markets

Presented by:
Lusani Mamushiane

Supervisor:
Dr Joyce Bertha Mwangama
Dept. of Electrical and Electronics Engineering
University of Cape Town

Co-Supervisor:
Dr Albert A. Lysko
NextGen Enterprises and Institutions
Council for Scientific and Industrial Research

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Masters of Science in Electrical Engineering (Electrical Engineering)

July 8, 2019

Dedication

In memory of my mother Takalani Victoria Ravhuhali. You left fingerprints of grace in our lives. Your words will forever regulate my life.

Vha edele nga mulalo.

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This thesis is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signed by candidate

Lusani Mamushiane
July 8, 2019

Acknowledgments

I would like to thank my Lord and Saviour Jesus Christ for being my key pillar throughout this journey.

To my supervisor Dr Joyce Mwangama. I can't describe how grateful I am to have had you as my supervisor. Thanks so much for your willingness to assist me whenever I cried for help and for your constant faith in me. May God richly bless you in all your endeavors.

Similar profound gratitude to my mentor and supervisor Dr Albert Lysko. Thank you so much for your motivation, patience, enthusiasm and willingness to share your immense knowlege in networking . Your passion and dedication to develop young people is trully humbling. I am forever indebted to you.

I am also hugley appreciative to my manager Sabelo Dlamini. Thank you so much for granting me the time I needed to complete this work. Most importantly, I sincerely appreciate your financial support and words of encouragement. You truly are a great leader and I will forever be grateful.

To my friends (Mpho Nkosi and Dr Hlabishi Kobo) and my brother Azwianzi Mukhubu your support and love kept me going and I am truly grateful.

Most importantly, I would like to express my profound gratitude to my dear husband Masabela Mamushiane for his continous words of encouragement and unfailing support. This work would not have been possible without your selflessness and reviews. I Love You So Much.

Abstract

The challenge of poor broadband penetration in emerging markets is generally attributed to the high cost of deployment and operations for broadband infrastructure. Operators are more comfortable to rollout infrastructure in urban areas than in rural (i.e. remote, sparsely populated and low income) areas, due to the attractive profit margins they present. The repercussion of this is a wide "digital divide" between urban and rural areas, resulting in social and economic exclusion. The exclusion of rural areas stifles economic growth. In order to bridge this divide, a more cost effective telecommunication infrastructure is indispensable. This means adopting an architecture that minimizes both network deployment costs (CapEx) and operational costs (OpEx), while maintaining a high service quality level and ensuring business agility. There is a general consensus that a large portion of OpEx comes from the costs associated with the configuration and management of the telecommunication infrastructure.

Software Defined Networking (SDN) has emerged as a promising solution to revolutionize network deployment, operations and economic growth. This paradigm aims to address management and configuration complexities in legacy networks so as to reduce the total cost associated with deploying and running telecommunication infrastructures. Conventionally, network control and data planes are tightly coupled and deployed within the same proprietary network device. SDN presents a shift in paradigm by decoupling the control plane from the data plane, abstracting lower level functionality of underlying hardware and enabling network programmability through a centralized controller.

As the "brain" of the network, the controller must be able to process and respond to requests from the data plane promptly and proficiently. In order to optimize a controller's operational efficiency, factors such as the number of controllers deployed, type of controller and controller placement are considered. During the network planning stage of an SDN deployment, the important questions that must be answered are: given a wide area network (WAN) topology, how many controllers are needed and where should they be placed to optimize SDN performance? Henceforth, this is referred to as the controller placement problem. This problem constitutes competing objectives such as load balancing, latency, reliability and CapEx, thus no single best placement solution is available.

This study aims to address the controller placement problem by leveraging machine learning algorithms. Moreover, this study carries out a comparative performance evaluation of the most popular SDN controllers namely, Ryu, Floodlight, ONOS and

OpenDayLight. The results from the performance evaluation are used to study the controller placement problem on an emulation orchestration platform. In order to contextualize the problem to emerging markets and maintain realism, a local national research and education wide area network called SANReN is used to test the proposed algorithms. This study can potentially be used by network operators as a guideline to start integrating SDN or plan a new SDN deployment, by helping them make quick automatic decisions regarding optimal controller placement.

Contents

List of Figures

List of Tables

List of Acronyms

ICT	Information and Communication Technologies
GDP	Gross Domestic Product
CapEx	Capital Expenditure
SDN	Software Defined Networking
NFV	Network Function Virtualization
EC	Edge Computing
COTS	Commercial off-the-shelf
SFC	Service Function Chaining
ETSI	European Telecommunications Standards Institute
ISG	Industry Specification Group
API	Application Programming Interface
SLA	Service Level Agreement
IoT	Internet of Things
NFVi	Network Function Virtualization infrastructure
DDoS	Distributed Denial-of-Service Attack
WAN	Wide Area Network
SANReN	South African National Research Network
PAM	Partition Around Medoids
CLARA	Clustering for Large Applications
2G	Second Generation of Mobile Communications
5G	Fifth Generation of Mobile Communications
WAN	IP Multi-media Subsystem
EPC	Evolved Packet Core
COM	Commercial Network
NREN	National Research and Education Network
TCP	Transmission Control Protocol
BGP-LS	Border Gateway Protocol Link State
LISP	Locator ID Separation Protocol
I2RS	Interface to Routing System
OVSDB	Open vSwitch Database Management
NETCONF	Network Configuration
OF-CONFIG	OpenFlow Configuration
PCEP	Path Computational Element Protocol
TLS	Transport Layer Security
CORD	Central Office re-architected as Datacenter

ONOS	Open Network Operating System
MDSE	Model Driven Software Engineering Principle
MD-SAL	Model Driven Service Abstraction Layer
LSO	Lifecycle Service Orchestration
MEF	Metro Ethernet Forum
S3P	Security, Scalability, Stability and performance
SAL	Service Abstraction Layer
ONAP	Open Networking Automation Platform
GML	Geography Markup Language
PoP	Point of Presence

Chapter 1

Introduction

1.1 Background

Over the past decade, the use of information and communication technology (particularly in emerging markets) has reached the upper bounds of internet penetration [1]. This strong appetite for internet access is causing a high demand for bandwidth and putting excessive pressure on the telecommunication infrastructure. According to a Cisco White paper [2], internet usage is anticipated to continue on an upward trajectory in the foreseeable future. There is a consensus that the current infrastructure will not suffice to cater for these exploding demands [3]. This is primarily attributed to the rigidity of the legacy infrastructure caused by vendor-lockin (the use of proprietary silicon hardware) which stifles innovation and makes it difficult to scale the network on the fly. As a result of vendor lock-in, the cost associated with upgrading the infrastructure to cater for the changing traffic patterns is very high, meaning adding new features ad-hoc is virtually impossible [4]. Therefore, network operators desiring new features to address their market needs end up beholden to vendor's upgrade timelines and costs. To cater for the increase in internet demand, the infrastructure has to evolve from its current monolithic nature to a vendor-agnostic, programmable, cost-effective (in terms of deployment (CapEx) and operational costs (OpEx)) and flexible infrastructure.

Although internet penetration is high in emerging markets, there still exists a significantly wide digital divide between rural areas and urban areas. A digital divide is a social, educational and economic inequality to the access to, use of, or impact of modern ICT. According to the World Economic Forum [5], about 31% (on a global scale) of the rural population do not have 3G coverage. Most rural areas have 2G

coverage which only allows slow connection to bare minimum services such as slow web browsing. Poor rural network coverage is a function of the cost associated with rolling out rural broadband infrastructure and the return on investment (ROI) thereof. Given the low population density of rural areas, most operators are reluctant to extend quality broadband services to rural areas as it would not be commercially viable. However, operators in most emerging markets are often subjected to geographic coverage obligations (by their state government) that mandate them to cover these commercially unattractive areas. This has left operators in a quest for cost effective means to achieve their obligations while remaining profitable. Without competition from smaller ICT players due to them not affording infrastructure, incumbent operators ultimately monopolize the telecommunication sector. In an attempt to recover their capital investment, these monopolies inflate the cost of broadband services to all internet users (both urban and rural users) [6]. Since rural areas are typically low income areas due to the high unemployment rate [7], the increase in broadband cost perpetuates the digital divide even more, causing both social and economic exclusion. The repercussion for this is stagnation in economic growth [8].

In short, emerging markets are subjected to the following challenges with regards to broadband penetration and access:

- Lack of a vendor-agnostic, programmable telecommunication infrastructure to cater for the ever increasing bandwidth demands
- High infrastructure deployment costs causing a wide digital divide between rural and urban areas
- Lack of competition in the telecommunication industry causing monopolization by established operators and unfair broadband charges.

In order to address the aforementioned challenges, a stronger (scalable, resilient, vendor-agnostic) telecommunication infrastructure is needed. To date, Software Defined Networking (SDN), Network Function Virtualization (NFV) and Edge Computing technologies have emerged as promising candidates to revolutionize future telecommunication landscapes. Contrary to the traditional network architecture where the control and data plane of packet processing devices are tightly coupled, SDN presents a paradigm shift in networking by decoupling the control plane logic from the underlying physical infrastructure [9]. The control plane is then logically centralized in an external entity called a controller. By decoupling the control logic from the physical hardware, operators can programme new traffic engineering policies

(such as bandwidth management, security, protection and restoration policies) without worrying about the constraints of closed proprietary hardware and firmware. Moreover, the abstraction of lower level functionality provided by SDN enables convergence of heterogeneous hardware thereby fostering a vendor-neutral ecosystem. In addition to enabling centralized network provisioning and holistic network management, SDN promises benefits such as security granularity (by providing a central point of control to holistically and consistently disseminate security information), savings in operational costs (by automating network administrative tasks), savings in capital expenditures (by capitalizing on commodity hardware) and cloud abstraction (which is critical to consolidate and facilitate management of massive data centers) [10]. According to [11] a huge portion of operational expenditure is from costs related to the management and configuration of the telecommunication infrastructure. Therefore, leveraging SDN to automate management and configuration tasks is likely to improve return on investment (ROI) and stimulate network rollout in rural areas.

A key companion to SDN is Network Function Virtualization (NFV). In order to bypass the legacy monolithic hardware, NFV was proposed by the ETSI NFV standards developing organization [12]. NFV decouples network functions such as firewall, network address translator and caching from dedicated hardware appliances and implement them as a software running on high volume commercial off-the-shelf servers (COTS) [13]. Leveraging service function chaining (SFC) [14] and NFV's management and orchestration (MANO) engine [15], the entire classes of virtualized network node functions can be dynamically chained to create communication services. This includes scaling the network capacity up and down to meet current network demands. By migrating to virtual network functions, the overall cost invested on equipment is likely to be reduced, whilst the time-to-market new services and innovation are likely to be significantly improved. NFV opens unprecedented opportunities such as virtualization of the Evolved Packet Core (EPC), virtualization of the IP Multi-media Subsystem (IMS) and most importantly, network slicing. Network slicing allows multiple logical networks to be instantiated on top of a common shared physical infrastructure [16]. Therefore, multiple tenants (such as MVNOs, over the top (OTT) service providers and vertical markets (such as education, automotive, healthcare and manufacturing)), each having different requirements and constrains can coexist on the same physical infrastructure. SDN is an integral part to realising network slicing. While NFV handles the virtualization and chaining of network functions into end-to-end slices, SDN acts as a middleman between the tenants and the owner of the network infrastructure and ensures, through authorization and authentication, secure network service capability exposure to each tenant via its northbound APIs [17]. The primary function of SDN in this particular case is to ensure resource allocation dynamism (between slices) based on service level

agreements (SLAs) [18]. Moreover, SDN is useful to enforce strong isolation and thus differentiation between network slices. Network slicing is a promising technology to help bridge the digital divide in emerging markets [19]. This is provided that a wholesale access model is implemented to ensure unbiased leasing of network capacity by the infrastructure owner. By adopting network slicing, smaller ICT players no longer have to worry about deploying their own infrastructure to reach rural areas. They can simply plug into a shared infrastructure and deliver services to their customers. This paradigm promotes service-level competition, stimulates innovation which are key ingredients to affordable good quality broadband access in rural areas. To realize the full potential of network slicing, SDN and NFV performance optimization is inevitable.

Edge Computing (EC) like SDN and NFV is a current technological trend in the telecommunication arena. EC pushes cloud-computing capabilities closer to the end user thereby enabling analytics and data gathering to occur near the source of the data. This paradigm is mainly driven by the widespread adoption of internet of things (IoT), and the monumental increase in the number of mobile devices which puts enormous pressure on the core network. Therefore, moving cloud resources closer to the edge is important to alleviate congestion and contention on the core network as it reduces network latency and bandwidth consumption. This is critical to cater for latency-sensitive applications such as autonomous vehicles and healthcare. EC capitalizes on NFV and SDN. NFV is used to host a wide range of applications on a common network function virtualized infrastructure (NFVi) [20]. SDN is necessary to lower the complexity barriers in EC, such as automatic switching between edge and the traditional cloud computing environments as well as orchestration of the virtualized functionalities [21]. In the context of emerging markets, mission-critical applications such as healthcare, more especially those geographically located in rural areas, can benefit from EC. For instance, instead of transmitting mission-critical data through the core network, local compute applications can be used to ensure lower communication overhead [22]. EC is anticipated to reduce operational costs (through its use of COTS servers) and to improve network security (by deploying security applications closer to the user). Therefore with EC, people in rural areas can also enjoy a better quality of service just like those in urban areas. With EC operators will potentially reap better return on investment as they can sell more and better quality services, whilst incuring reasonable operational costs [23] [24].

As mentioned above, SDN, NFV and EC have been earmarked as the key stepping stones to ICT equality in the context of emerging markets. Each of these technologies has matured considerably such that there is little to no skepticism with regards to their viability. At this juncture, there has been a plethora of technology demonstrators ranging from virtualized functions (vEPCs and vIMs)[25] [26] [27],

orchestration platforms (such as OpenBaton and ONAP) [28] [29], SDN carrier grade and interoperability controllers (such as ONOS and OpenDayLight) [30] [31] and smart wearables (for edge computing appearing in the telemedicine domain) [32]. This work focuses strictly on SDN as a key enabler for bridging the digital divide in emerging markets while ensuring a fair ROI for the network operator.

1.2 Problem Statement

Although local area networks (LANs) like data center networks (DCNs) have already benefited from SDN, deploying SDN in real wide area networks (WANs) still poses several design challenges. As the centralized brain of the network, an SDN controller must be able to respond to control requests promptly. Moreover, control tasks such as dataplane monitoring, must be performed as efficiently as possible to maintain up-to-date state information. This requires optimization on the southbound interface. Due to the significant influence of propagation latency (switch-to-controller latency) on WAN performance, controller placement has emerged as a crucial design problem that influences SDN's southbound performance. Controller placement defines the location of SDN controllers relative to the dataplane elements, that yields better network perfomance.

Another aspect to controller placement has to do with the number of controllers deployed in a given WAN. Deploying a certain number of controllers has an impact on several objectives such as propagation latency and reliability. Even though the number of controllers may be known in advance, the location of these controllers still needs to be optimized to meet user requirements and contraints. Last but certainly not least, the type of controller implementation also has a huge effect on network performance. Given a plethora of SDN controllers to choose from, there is a need to benchmark controllers against a set of user-defined metrics so as to make an informed decision on which controller to deploy.

Therefore, the overall problem that must be addressed is: given a real SDN-enabled WAN, which SDN controller amongst the popular choices is ideal to use; how many SDN controllers are needed and where should they go to optimize user-defined requirements and constraints while maintaining acceptable runtime and accuracy. This is a multi-objective optimization problem and constitutes competing objectives. It is necessary to address this problem during the early stages of SDN planning.

1.3 Research Gap

The SDN controller placement problem has been studied in the past. To the best of the authors' knowledge, existing approaches propose the use of heuristic algorithms which present a trade-off between accuracy and algorithm runtime. Moreover, from the state of art review, there is no work that explores solutions to quantifying the number of controllers to implement given a WAN. There is also no recent study on SDN controller southbound benchmarking. Perhaps more importantly, there is currently no analysis of the controller placement problem purely using an emulation platform to mimic a real SDN deployment. Most studies relied on mathematical modeling to address the controller placement problem, making it difficult to verify validity and reliability of the results.

This study proposes several approaches that can be used to expedite network planning for efficient migration to the SDN era. As there is currently no study that features controller placement for the emerging market use case, this study has been tailored towards facilitating transition to SDN in emerging markets. Contrary to existing works that rely on mathematical modelling, this study leverages both mathematical based models and emulation to address the controller placement problem. Instead of fixing the number of controllers to deploy, several machine learning algorithms are proposed to determine the ideal number of controllers to use for a given WAN. Controller placement is a network planning problem, and normally not time sensitive. Consequently, this study proposes the use of exhaustive algorithms to optimize solution accuracy. To facilitate decision making regarding the ideal controller to deploy, this study carries out a comparative performance evaluation of the most popular SDN controllers. Finally, a mechanism to manage control plane overhead is proposed.

The key performance indicators used to gauge network performance are: (i) network latency (propagation + queuing + processing latency), (ii) reliability (in the event of link and/or node failure), (iii) control traffic load (number of packets on the southbound interface) and (iv) throughput (number of responses per second).

1.4 Research Questions

This study aims to answer the following research questions:

- How does controller placement impact performance of SDN-enabled WANs?
- How does the number of controllers affect the performance of SDN-enabled WANs?
- What optimization algorithms can be used to address controller placement and what are their relative performances?
- What are the topmost used open source SDN controllers?
- Which open source SDN controller is the most feature-rich?
- Which SDN controller exhibits the best throughput and latency performances?
- What mechanisms can be used to manage control plane overhead?
- What is the best decision methodology for multi-objective SDN network optimization?

1.5 Research Objectives

The objectives of this study are as follows:

- To investigate the impact of controller placement on network performance using mathematical modelling;
- To investigate the impact of controller placement on network performance using emulation;
- To carry out a quantitative and qualitative comparison of prominent open source SDN controllers against various metrics; and
- To investigate a mechanism to reduce control plane overhead.

The preceding objectives can be broken down into the following sub objectives:

- To review and compare the performances of different existing controller placement algorithms;
- To determine the ideal number of controllers to use given a network;
- To determine the optimal location of controllers in an SDN-enabled WAN;
- To determine a feature-rich controller amongst popular open source controllers;
- To determine the most efficient (in terms of performance) open source controller currently available;
- To investigate tunable control parameter for control load management; and
- To develop a decision methodology/practical guidelines for multi-objective SDN network optimization

1.6 Research Approach

To optimize controller placement in WANs, classical partitioning algorithms (such as Partition Around Medoids (PAM), k-means, Clustering for Large Applications (CLARA), Silhouette and Gap Statistics) from machine learning are explored. The idea is to leverage on network graph modeling and mathematical formulation to address the controller placement problem.

In order to verify the validity of the results obtained from the mathematical model and assess applicability on a real WAN deployment, the controller placement problem is analyzed using an emulation orchestration platform and a real open source SDN controller. In order to inform the decision regarding an ideal open source controller, a performance comparison of the most prominent SDN controllers is carried out prior to the emulation experiment. This evaluation constitutes competing objectives such as throughput, latency and resiliency.

Lastly, two control plane overhead management techniques are studied. The first technique (*technique one*) involves tuning several control parameters, such as polling frequency (which specifies how frequent the controller sends requests to the data plane for monitoring purposes) and flow timeouts (which specifies how long routing instructions can remain idle before deletion), to reduce control plane signaling overhead. The second technique (*technique two*) involves balancing the switch-to-controller mastership placement. To achieve this, a mastership-balance module is implemented. It is important

to note that *technique two* leverages results generated from implementing *technique one* and the outcome of the controller placement experiments to achieve multi-objective optimization.

1.7 Research Contribution

The major contributions of this study are as follows:

- This study provides a thorough state of the art review of SDN controller placement research and various solution implementations. This includes reviewing the merits and faults of various algorithms proposed to address the controller placement. This is necessary to identify improvement opportunities within this research space.
- Although SDN is anticipated to revolutionize network operations and economic growth, the idea of centralizing network control has been received with a lot of skepticism, especially in emerging markets where SDN adoption is still in its infancy. Most operators are not convinced that a centralized controller will, at the very least, match up the performance of legacy networks. This study aims to mitigate these concerns by qualitatively demonstrating the performance capabilities of centralizing network control. The idea is to demonstrate how software defined network planning can be improved through proper SDN controller placement. Therefore operators can apply some of the techniques proposed in this study as guidelines to facilitate transition to SDN.
- This study attempts to create a practical guideline on controller placement optimization by approaching the SDN controller placement problem from its different facets. Controller placement features various competing objectives (depending on user-defined requirements and constraints) that must be considered for realistic deployments. This study takes into account objectives namely, latency (controller-to-node latency), reliability and control plane overhead to address the controller placement problem.
- This study is tailored for emerging markets (particularly African economies) faced with the challenge of ICT inequality. Therefore this study features existing research and education networks to demonstrate the viability of migrating to SDN.

In a nutshell, this study involves the extension of four algorithms for solving the controller placement problem to expedite SDN adoption in emerging markets.

1.8 Research Outputs

The following publications led to developing the research topic:

1. L. Mamushiane and S. Dlamini, "Leveraging SDN/NFV as key stepping stones to the 5G era in emerging markets," in 2017 IEEE Global Wireless Summit (IEEE GWS 2017), Cape Town, 15-18 October 2017.

2. L. Mamushiane, A. A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers" . In 2018 IEEE Wireless Days (IEEE WD 2018),(pp. 54-59), UAE, Dubai, 2-5 April 2018.

3. L. Mamushiane, A. A. Lysko, and S. Dlamini, "SDN-enabled Infrastructure Sharing in Emerging Markets: CapEx/OpEx Savings Overview and Quatification," in IST Africa 2018, Botswana, 9-11 May 2018.

The following publications were developed during the registered period of Masters work:

1. L. Mamushiane, J. Mwangama and A. A. Lysko, "Optimum Placement of SDN Controllers in African Backbones: SANREN and ZAMREN as a Case Study," in 2018 Southern Africa Telecommunication Networks and Applications Conference (SATNAC 2018), South Africa, Cape Town, 2-5 September 2018.

2. L. Mamushiane, J. Mwangama and A. A. Lysko, "Given a SDN Topology, How Many Controllers are Needed and Where Should They Go?," Accepted for 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV/SDN 2018), Verona, Italy, 27-29 November 2018.

3. L. Mamushiane, J. Mwangama and A. A. Lysko,"Resilient SDN Controller Placement Optimization applied to and emulated on South African National Research Network (SANReN)" ,Submitted for 2019 IEEE Wireless Communications and Networking Conference (IEEE WCNC 2019), Morocco, Marrakech, 15-18 April 2019.

1.9 Scope and Limitations

One key limitation to this work is not having the neccessary equipment for testbed evaluation of our proposed approach(es). Ideally, we would like to deploy physical Ubuntu machines in different geographic locations to mimic a real WAN topology, utilizing a tool such as Cbench [33] to perform complex analysis of the controller placement problem. However, due to budgetary constraints, this work relies on simulation and emulation to address the controller placement problem. Prototypes built on top of an emulated environment can easily be moved to a real network environment with minimal changes on the codebase.

Another challenge involves limited options in terms of dataset regarding emerging markets' topologies (particularly African). This study makes use of a repository called Internet Topology Zoo [34] to obtain datasets of different national research and education networks (NRENs) and commercial networks (COMs). Unfortunately, the only datasets recorded for African backbones are SANREN (a South African NREN) and ZAMREN (a Zambian NREN). Nevertheless, the proposed approaches are generic and can be used to optimize other topologies of different sizes and configurations.

1.10 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 offers a literature review of traditional networks, SDN technology as well current research work on the controller placement problem. Chapter 3 offers algorithms for optimizing controller placement using mathematical formulations. This chapter also includes results obtained from the mathematical model. Chapter 4 offers a comparative evaluation framework for popular open source SDN controllers (both feature-based and performance comparison). The results and discussions from this evaluation are included in the same chapter. Chapter 5 feeds on the results from Chapter 3 and Chapter 5 to evaluate controller placement using of an emulated environment to mimic a real SDN deployment. Chapter 5 also describes various techniques for reducing control plane overhead and presents results generated by implementing these techniques. Lastly, Chapter 6 concludes this study and highlights lessons learned and future research directions.

Chapter 2

Literature Review

2.1 Introduction to Software Defined Networking

A vast majority of legacy networks are complex, closed and proprietary, and use integrated hardware and software to direct traffic across a series of routers and switches. This tight coupling between the hardware and software makes it difficult for operators to dynamically introduce new services to meet changing market needs. This is primarily because it takes a significant investment to build custom hardware, and several processes are required to ensure equipment vendors get the most out of each upgrade and new iteration [35]. Therefore adding new features on–demand is virtually impossible as operators are bound to vendors' timelines. Moreover, the numerous multi–vendor systems lack support for efficient and accurate remote troubleshooting and fault management. Currently, whenever there is a network fault, a specially trained Network Engineer is dispatched to the location of the failed network node for troubleshooting and fault resolution [36]. As vendor–specific commands are used, this is both time consuming and error prone thereby significantly impacting business agility and quality of service.

Software Defined Networking (SDN) has appeared as a promising solution to these challenges. The main goal of SDN is for the network to be open and programmable. SDN virtualizes the network by separating the control plane that manages the network from the data plane where traffic flows [9]. The control plane is then logically centralized in an entity called a controller and manages all network traffic. By separating control and data planes, SDN creates high–level abstractions of lower level functionality thereby enabling efficient orchestration and automation of network services across heterogeneous systems. For instance, an operator requiring a specific type of network behavior can

simply install relevant applications on the centralized controller using exposed APIs. These applications may be for common networking functions such as traffic engineering, security, load balancing, fault management, virtualization and quality of service [37].

Some of the compelling benefits of SDN are as follows:

- **Innovation Enablement**: SDN hides the complexity of the underlying forwarding functions thereby enabling organizations to develop richer applications, new services and business models;
- **Savings in Capital Expenditure (CapEx)** : by enabling organizations to use "white box" switches and routers and to re–purpose the legacy hardware for SDN compatibility;
- **Savings in Operational Expenditure (OpEx)**: through supporting automation of network management and configuration through increased programmability of the data plane;
- **Advanced Security**: via enabling a consistent dissemination of security policies from a single management console;
- **Service Agility**: by accelerating deployment of new applications and services to accommodate changing traffic patterns.

The overall comparison between traditional networks and SDN is presented in Table 2.1.

Table 2.1: Comparison between traditional networks and SDN [38]

Criteria	Traditional Networks	SDN
Network management and configuration	Requires the use of vendor-specific commands making it difficult to program changes and requires specialized training	Enables network programmability by exposing vendor-agnostic interfaces
Global network state awareness	Difficult due to vertical integration between control and data plane	Simplified via decoupled logically centralized controller
Maintenance cost	Higher	Less
Time required for upgrades/error handling	Sometimes takes months	Can take as little as a few minutes due to centralized control logic
Control plane load balancing	Not important	Important
Control plane utilization	Not relevant	Important
Control plane availability	Not important	Critical
Resource utilization	Less	High
Flow table and state information integrity	Critical	Important
Control plane integrity, authenticity and consistency	Not relevant	Important

2.1.1 SDN Architecture

SDN is a multi–tiered architecture comprising three functional planes namely, data plane, control plane and the application plane (see Figure 2.1). The data plane constitutes heterogeneous network elements (such as switches, routers, firewall, etc.), which expose their capabilities to the control plane via the controller's southbound programmable interface. The control plane constitutes a logically centralized controller with a global perspective of the network elements. The application plane comprises a wide range of applications such as cloud orchestration, SDN and business applications [39]. Applications communicate their requirements (by defining high–level policies) to the controller via the northbound programmable interface. The controller then processes and translates applications' requirements to low–level flow instructions, used to configure the data plane. The controller uses its northbound interface to provide an abstracted view of resource utilization and state to the application layer. This is necessary to hide the details that are unimportant and only present relevant information to the application plane. To address scalability concerns caused by centralized control, distributed controllers are generally deployed [40].

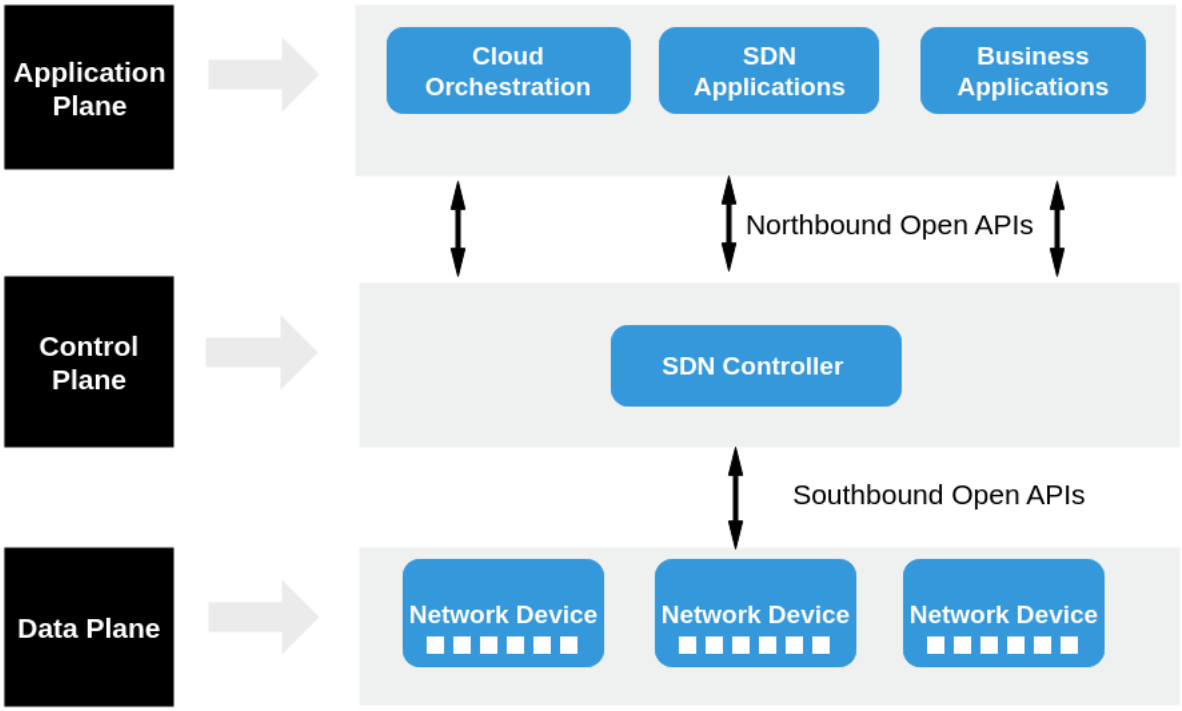


Figure 2.1: High level SDN reference architecture.

2.1.2 Protocols and Standards

SDN communication protocols are an integral part to achieve convergence of heterogeneous network devices, through high–level abstraction. Significant research effort has been exerted towards standardization of southbound and northbound protocols, especially on the southbound interface [35]. At this juncture, a plethora of control and management southbound protocols are available. Available options include management protocols such as OpenFlow Configuration (OF–CONFIG) [41], Network Configuration (NETCONF) [42], Open vSwitch Database Management (OVSDB) [43] and control protocols such as OpenFlow [44], Locator ID Separation Protocol (LISP) [45], Interface to Routing System (I2RS) [46], Path Computational Element Protocol (PCEP) [47] and Border Gateway Protocol Link State (BGP–LS) [48].

While control protocols are used to configure packet flow operations on the data plane, management protocols are used for data plane configurations (such as IP assignment, ports allocation and policy enforcement) [49]. Protocols such as PCEP, BGP–LS and I2RS (those whose application domain, as shown in Table 2.2, is "Hybrid SDN") have emerged as promising solutions to enable SDN support in traditional networks. This is commonly known as hybrid SDN. By adopting these protocols, operators can simply upgrade to SDN without rebuilding their existing infrastructure. By so doing, lower migration costs (especially CapEx) are incurred. On the other hand, protocols such as OpenFlow and OF–CONFIG lack compatibility with the legacy networks and require significant CapEx investments. Moreover protocols such as PCEP and BGP–LS have gained more traction in carrier grade SDN deployments due to their scalability attributes while OpenFlow has been a de facto protocol for data center environments [50] [51]. Table 2.2 presents a high level comparison of popular southbound SDN protocols.

Table 2.2: High level comparison of southbound protocols [9]

Protocol	Standardization Body	Purpose	Application Domain	Interface Name
OF–CONFIG	ONF	Management	SDN	Southbound
NETCONF	IETF	Management	SDN Hybrid SDN	Northbound, Southbound, East/Westbound
BGP–LS	IETF	Control	Hybrid SDN	Southbound East/Westbound
LISP	IETF	Control	Hybrid SDN	Southbound
OVSDB	ETSI	Management	SDN	Southbound
I2RS	IETF	Control	Hybrid SDN	Southbound
PCEP	IETF	Control	Hybrid SDN	Southbound
OpenFlow	ONF	Control	SDN	Southbound

Despite a rapid influx of SDN control protocols, OpenFlow still remains a prevalent choice for many SDN solutions. Recently, there has been a number of network equipment vendors (such as Cisco, Big Switch Networks, Dell, HP and Arista, to name a few) that have announced their support for OpenFlow. OpenFlow is a standardized protocol for instantiating flow instructions into the data plane over TCP. OpenFlow prescribes the Transport Layer Security (TLS) protocol to secure the southbound communication channel [52]. Figure 2.2 describes packet flow process within OpenFlow [53]. When a packet arrives to an OpenFlow kernel switch, the packet match fields (ingress port, metadata and packet headers) are matched against the switch's flow table entries. If a matching flow entry is found, the switch executes the instruction set associated with that flow entry and updates it's counters. This instruction set can be forwarding the packet to a specific egress port or dropping the packet. If no matching entry is found, this is a table miss. A table miss flow entry specifies how unmatched packets are processed. Options include forwarding unmatched packets to the controller (via a packet–In message) over the southbound interface, passing the packets to another table for further matching or dropping the packets. Upon receipt of packet–In messages, the controller make a decision and installs flow entries on the switch via a packet–Out message).

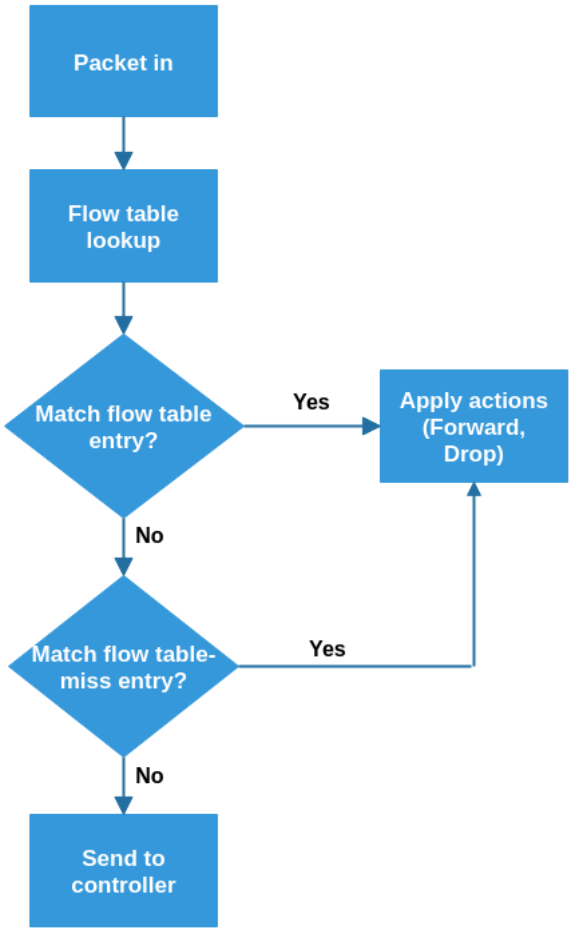


Figure 2.2: Packet flow process through an OpenFlow kernel switch [54].

As shown in Table 2.3, OpenFlow supports three message types, namely Symmetric, controller–to–switch and asynchronous [54]. Symmetric are unsolicited messages initiated by either the switch or the controller. These includes messages established during handshake sessions (such as Hello and Echo) and error reporting messages (such as link or node failure). Controller–to–Switch messages are initiated by the controller for monitoring and discovery purposes. Some of these messages are feature requests, switch state and role request messages. Last but certainly not least, asynchronous messages are sent by the switch to the controller. These include forwarding requests (Packet–In), and status messages (such as port–Status and Role Status).

Table 2.3: Summary of control messages supported by OpenFlow

Message Type	**Description**	**Examples**
Symmetric	Sent in either direction	Hello Echo Error
Controller–to–Switch	Initiated by the controller to the switch	Features Configuration Modify State Read State Packet–Out Role–Request
Asynchronous	Initiated by the switch	Packet–In Flow–Removed Port–status Role Status Flow–monitor

2.1.3 Open Source SDN Controllers

As described in section 2.1.1, a controller is the central brain of SDN networks. To date, there has been a lot of research efforts to develop open source SDN controllers to enable rapid prototyping of SDN solutions. Some of the most popular open source controllers are, ONOS, OpenDayLight and Ryu. These controllers support various features in terms of interoperability, scalability, security and complexity. This section overviews the aforementioned controllers and presents a summary with the feature–based comparison of the controllers in Table 2.4.

ONOS

ONOS is an open source SDN controller, pioneered by the ON.Lab and primarily designed to enable service providers to build real world SDN solutions. This controller is said to have been optimized (through its distributed core) to deliver features such as network scalability, reliability and high performance (latency and throughput) which are indispensable in production networks. ONOS has two abstraction frameworks in its northbound interface, namely the intent framework [55] and global network topology view. The intent framework is a subsystem that enables a network application to apply a service in the network in form of policy (i.e. what should be done) rather than mechanism (how it should be done). The global network view exposes the current status of the entire network (e.g. resource utilizations) to the application layer. The controller northbound abstracts the complexity of the underlying hardware from the application. Another abstraction is enabled by the southbound interface which represents the underlying hardware as objects, and allows convergence of disparate data plane devices through its support for different protocol plugins, e.g. NETCONF, OVSDB and OpenFlow. The east/westbound communication between distributed ONOS controller instances employs an extension of the BGP protocol, and enables distributed control instances to exchange state information of their respective SDN domains [9]. The core use case of ONOS is the Central Office re–architected as Datacenter (CORD) which capitalizes on SDN, NFV and cloud computing to transform the central office through hardware commoditization [56]. With this operators can achieve both the economies of scale (infrastructure build from generic servers) and business agility (rapid deployment and elastic scaling of network services to meet current demands) which are currently enjoyed by cloud service providers. ONOS has been deemed the controller for service providers due to its distributed core. To date there are a total of 9 releases of ONOS, namely, Avocet, Blackbird, Cardinal, Drake, Emu Falcon, Goldeneye, Hummingbird, Ibis, and Junco which highlights the extent of

community support.

OpenDayLight

OpenDayLight is a modular open source java–based SDN controller hosted by the Linux Foundation, used for customizing and orchestrating networks of any scale and size [30]. This controller leverages a model–driven software engineering (MDSE) principle known as Model–driven service abstraction layer (MD–SAL), which uses YANG as the data modeling language for abstraction lower–level functionality of data planes. In OpenDayLight, the underlying hardware is represented as objects whose inter–communication is managed by the SAL. The modular architecture of OpenDayLight offers users and solution providers free reigns to program customized traffic policies to satisfy their needs. The MD–SAL resides within the control layer and is the "brain" of the SDN network. Its northbound interface translates policies from the application layer to the data layer via its southbound interface. OpenDayLight supports a wide range of southbound protocols such as OpenFlow, BGP–LS, PCEP, LISP, NETCONF, OVSDB, etc. In terms of adoption coverage, OpenDayLight is at the core of open source management and orchestration frameworks such as Open Networking Automation Platform (ONAP) [28], OpenStack [57] and OPNFV [58], as well as standard development organizations such as Metro Ethernet Forum (MEF). For instance the UNI Manager [59] plugin release in OpenDayLight provides APIs for MEF's Lifecycle Service Orchestration (LSO) project [60] [61]. OpenDayLight is primarily focused on interoperability thereby making it a de facto standard for hybrid SDN deployments. To date there are a total of 9 OpenDayLight releases namely, Hydrogen, Helium, Lithium, Beryllium, Boron, Carbon, Nitrogen, Oxygen and Fluorine, in order of decreasing age. Each new release promises support of emerging use cases such as IoT, integrated NFV management and S3P (Security, Scalability, Stability and performance) achieved through clustering and federation.

FloodLight

Floodlight is an event–based SDN controller pioneered by Big Switch Networks. This controller is ideal for rapid prototyping in small scale environment as it exclusively supports OpenFlow on its southbound and lacks support for scalable protocols such as BGP–LS and PCEP. On it's northbound, Floodlight supports REST APIs making it easier for application developers to program different traffic engineering policies.

Moreover, Floodlight supports multi–threading, is highly modular and has a asynchronous framework [62].

Ryu

Ryu is a modular SDN controller typically used in cloud orchestration controller applications. The advantage of a modular controller is that other modules written in any language can be added to extend the functionality of the controller. Ryu supports REST API on the northbound abstraction interface and supports OF–CONFIG, NETCONF, OVSDB and OpenFlow for southbound communication [63]. Due to its lack of support for scalable protocols such as BGP–LS, Ryu is not suitable for large scale deployments making it ideal only for rapid prototyping and small scale deployments. Ryu is implemented in Python.

Summary of comparison

As shown in Table 2.4, OpenDayLight followed by Ryu support the most southbound interfaces compared to other controllers. Floodlight exclusively supports OpenFlow. This restricts the deployment of FloodLight to pure SDN deployments. The reputation, financial capacity and experience of contributors is pivotal to ensure active maintanance and sustainability of an SDN controller and stimulates trust and adoption [64] of that controller. From Table 2.4, it is clear that OpenDayLight followed by ONOS have the most support from vendors and open source communities. Floodlight and Ryu codebase development is monopolized by their founders. In terms of scalability, OpenDayLight and ONOS are more scalable since they both support distributed control making them suitable for both local and wide area network deployments. Both OpenDayLight and ONOS are highly modular making it easier to integrate new improvement features. One of the main shortcomings of ONOS is that it does not support cloud orchestration (e.g. OpenStack) which is imperative for virtual resource management.

The feature–based comparison above can be used by decision makers during the SDN network planning phase to benchmark the features of each controller against their requirements and constrains. For instance, to deploy a more scalable SDN network, ONOS would be an ideal choice due to its distributed core which improves availability and reliability. However, to ensure multi–vendor support, OpenDayLight would be the best choice because of its rich support for legacy southbound protocols. However, for

small scale campus networks, Ryu appears more attractive due to its centralized core which eliminates inter–controller latency and its use of Python which reduces deployment complexities. However, a feature–based comparison does not guarantee controller efficiency with respect to its performance. Therefore a decision on which controller to deploy is influenced both by a combination of feature support and performance of a controller as well as the intended application.

Table 2.4: Feature–based comparison of open source SDN controllers [65]

Feature	**ONOS**	**OpenDayLight**	**Floodlight**	**Ryu**
Southbound Interfaces	OF 1.0, 1.2, 1.3, 1.4, 1.5, NETCONF	OF 1.0, 1.2, 1.3, 1.4, 1.5 NETCONF/YANG, OVSDB, LISP, BGP–LS, SNMP PCEP	OF 1.0, 1.2, 1.3, 1.4, 1.5	OF 1.0, 1.2, 1.3, 1.4 NETCONF, OVSDB, OF–CONFIG
REST–API	Yes	Yes	Yes	Yes
GUI	Web–based	Web–based	Web–based	Yes(Initial phase)
Modularity	High	High	Medium	Medium
Orchestrator Support	No	Yes	Yes	Yes
OS Support	Linux, MAC, Windows	Linux, MAC, Windows	Linux, MAC, Windows	Most supported on Linux
Contributors	ON.LAB, Cisco, Huawei, Ericsson, Nec,Ciena Fujitsu, Sk Telecom	Linux Foundation with members covering over 40 companies such as Cisco, IBM, ect.	Big Switch Networks	Nippon Telegraph and Telephone Corporation
Documentation	Good	Very good	Medium	Good
Programming Language	Java	Java	Java	Python
Multi–threading Support	Yes	Yes	Yes	Yes
TLS Support	Yes	Yes	Yes	Yes
Virtualization	Mininet and OVS	Mininet and OVS	Mininet and OVS	Mininet and OVS
Application Domain	Data center and SD–WAN	Data center and SD–WAN	Campus	Campus
Distributed/Centralized	Distributed	Distributed	Centralized	Centralized

2.2 Related Work

The disassociation between the control and data plane offered by SDN brings about a new set of challenges. A fundamental problem that must be addressed during SDN network planning is the controller placement problem. The complexity of the controller placement problem increases with larger SDN networks, making it an NP–hard problem as it can not be solved in polynomial time [38]. Random placement of the controller(s) would likely increase the overhead delay of services. This would degrade the overall performance of the SDN network. The controller placement problem is three fold: (i) how to partition the network, (ii) which specific controller among the popular choices to deploy, and (iii) where to place the controller(s). Therefore, the overall objective of the controller placement problem is to determine the best locations to deploy the optimal number of specific controllers in a given network.

This problem constitutes various competing objectives such as load balancing, reliability and latency. Most studies assume that each switch incurs a fixed traffic load and measure load balancing by the number of switches supervised by each controller [38]. A few studies assume dynamic load on the switches and measure load balancing by the number of flow instantiations per second. Load balancing is only considered if control plane capacity is a constraint. This is commonly known as capacitated controller placement problem [66]. The capacitated controller placement problem assumes the controller has limited capacity while the uncapacitated counterpart assumes no capacity limitation. The controller placement can also be optimized considering the latency objective. Latency can be divided into two categories (i) switch–to–controller latency and (ii) inter–controller latency, where switch–to–controller is the round-trip latency on the southbound interface and the inter–controller latency is the round-trip latency on the east/westbound interface. There is a general consensus that propagation latency dominates in WANs [67]. As a result, most research works formulate their models with propagation latency as the main objective. In this work, latency also refers to propagation latency unless stated otherwise. Last but not least, the controller placement problem can be solved in consideration of reliability also known as fault-tolerance or resiliency. A fault-tolerant controller placement removes the single point of failure posed by the centralized control plane. There are two options to achieving reliability. One option is to deploy multiple identical controllers all managing the whole network in parallel, and select the correct results on the basis of a quorum [38]. Another option is to partition the network into several domains, with each domain supervised by a dedicated controller. With this option, reassignment of switches only occurs in the event of controller failure.

This section presents an analysis of state–of–the art controller placement solutions. To date there has been numerous research studies directed towards addressing the controller placement problem in SDN. These can be broadly classified into two categories: (i) studies that implemented exhaustive algorithms, as exampled by [68] –[69] and (ii) studies that implemented heuristic algorithms, as exampled by [72] –[85].

The controller placement problem was first introduced by Heller et al. [68] in 2012. The authors study the controller placement problem by investigating the impact of uncapacitated controller location on average and worst–case latency. The algorithm used in this study is k–center. To maintain realism, the authors tested their algorithm on the Internet2 OS3E topology [70]. Their results indicate that increasing the number of controllers decreases the overall network latency with a significant tradeoff between worst–case and average latency. The authors conclude that deploying one controller often suffices to meet existing latency requirements in campus networks. Expectantly, they also argue that one controller is not sufficient for large scale deployments with fault tolerant requirements.

Hu et al. [71] proposes the use of multiple controllers to ensure reliability in the control plane. To optimize controller placement, the authors carry out a comparative evaluation of optimization algorithms namely, random placement, l–w greedy and brute force. They focus their reliability metric on the "expected percentage of valid control paths", where a control path is defined as the interface between the switch and controller (southbound interface) as well as the connection between controllers (east/westbound interface). The algorithms were evaluated on Internet2 topology as well as various ISP topologies from the Rockefuel database [72]. From their simulations, random placement produced the least optimal results, while brute force produced optimal results after a significantly long runtime. As a result, the authors recommend the l–w greedy as the most optimal solution. This work is similar to Hu et al. [73] in that they both aim to optimize reliability in the event of node or link failure. However, latency (both switch–to–controller and inter–controller latency) and load balancing are not considered in these research works. Moreover, the number of controllers is assumed to be known in advance.

Tanha et al. [69] study the controller placement problem to optimize network resiliency in the event of controller failure while considering network deployment costs and satisfying switch–to–controller latency. In order to mimic a production scenario, the authors take into account the capacity of the controller and assume a varying switch load. To maintain realism, they assessed their algorithms on real tier–1 service provider topologies. The outcome of their experiments demonstrated that controller resiliency is topology dependent. The drawback of this solution is that it is resource intensive and

only ideal for small and medium network instances. The algorithm used in this study is the capacitated k–center algorithm.

The research work of Yao et al. [74] propose a heuristic algorithm for capacitated controller placement in consideration of switch–to–controller latency and traffic load of switches. The main objective of this work is to optimize controller load balancing under heterogeneous data plane load while minimizing switch–to–controller latency. Resiliency is handled by deploying additional controllers in the network. The main shortcoming of this solution is that it is less accurate in larger deployments and therefore applicable only for small–scale networks.

Jimenez et al. [75], also proposes a capacitated controller placement solution to optimize load balancing. Contrary to Yao et al., this work is not limited to the size of the network and propose a divide and conquer philosophy to achieve scalability and robustness. Moreover, authors assume homogeneous traffic load on the data plane. The solutions proposed by Jimenez et al. and Yao et al. optimize controller placement based on fixed traffic observed initially, but do not adapt to the changing traffic load. This shortcoming is addressed by Bari et al. [76] and Jourjon et al. [77] who propose a heuristic algorithm for dynamic controller placement i.e. controller placement based on current data plane load. The metrics considered are switch–to–controller latency and controller processing load. The solutions proposed rely on trial and error and are not as reliable. Sanner et al. [78] propose a genetic algorithm leveraging the NSGA II framework to optimize load balancing and inter–controller latency. Authors conclude that their solution consumes a lot of CPU resources and is only ideal for small–and medium–sized networks.

Rath et al. [79] propose a Non–Zero–Sum game theory approach to optimize controllers' utilization. In this solution, controllers can be added or removed dynamically and can also go to sleep mode occasionally based on the traffic load on the controllers. This solution is intended to reduce network deployment costs (by minimizing the number of controllers deployed) and operation costs (by optimizing energy consumption through on –demand controller deployment). This solution ignores controller placement in the network. Sallahi et al. [80] propose a mathematical formulation to find the optimum number of controllers to deploy. However, their approach suffers the same shortcoming as that proposed by Rath et al. in that it does not determine the optimal controller placement. Furthermore, both these research works are limited to small –scale networks.

Wendong et al. [81] study the tradeoff between reliability and latency using random placement, l–w greedy and simulated annealing. The results suggest that

simulated annealing yields the most optimal solution in comparison with the other approaches. The outcome of the tradeoff analysis indicate a significant tradeoff between reliability and latency. Authors argue that the number of controllers must be chosen carefully. They demonstrate that using too few controllers has an adverse effect on reliability while using too many controllers can result in a broadcast storm on the east/westbound interface.

Hock et al. [82] and Lange et al. [83] advocate for careful consideration of latency (controller–to–controller) and reliability (defined as resilience in the event of a node or link failure and control plane load balancing) during controller placement. This work proposes a resilient Pareto–based Optimal Controller placement framework to achieve optimal controller placement. The authors use load imbalance as the key metric, which is the difference between the controller having more switch assignments and the controller having fewer number of switches under its supervision. The results from this work indicate that the optimal solution is achieved when 20% of all network nodes are controllers. The downside of this solution is that, instead of partitioning the network into small administrative domains, the authors treat the network as a whole with controllers working collaboratively. This means the controllers frequently share their network state information with their peers to maintain an accurate global view. This increases the probability of incurring a network broadcast storm which increases inter –controller latency. Therefore, this solution is restricted to small–and medium–scale network instances. Furthermore, this solution ignores the average switch–to–controller latency which is a critical parameter in SDN.

Ksentini et al. [84] consider three objectives in optimizing controller placement: (i) switch–to–controller latency, (ii) inter–controller latency and (iii) control plane load balancing simultaneously. The authors propose a bargaining game–based algorithm to optimize controller placement. Authors claim that their results outperform other mono–objective–based controller placement results. However, their algorithm is only suitable for small–scale networks and is less accurate for larger network instances.

Last but certainly not least, He et al. [85] formulate controller placement model to optimize flow setup time, where flow setup time is the total amount of time taken by the controller to install a flow instruction on the switch's flow table. The authors argue that dynamic controller placement is necessary to help reduce flow setup time. The results from this work reveal that, for low flow densities, dynamic controller placement can reduce the flow setup time by up to 50% in comparison with static controller placement. However, for high flow densities, static controller placement produced better results.

Table 2.5 and Figure 2.3 classify the existing solutions on controller placement problem.

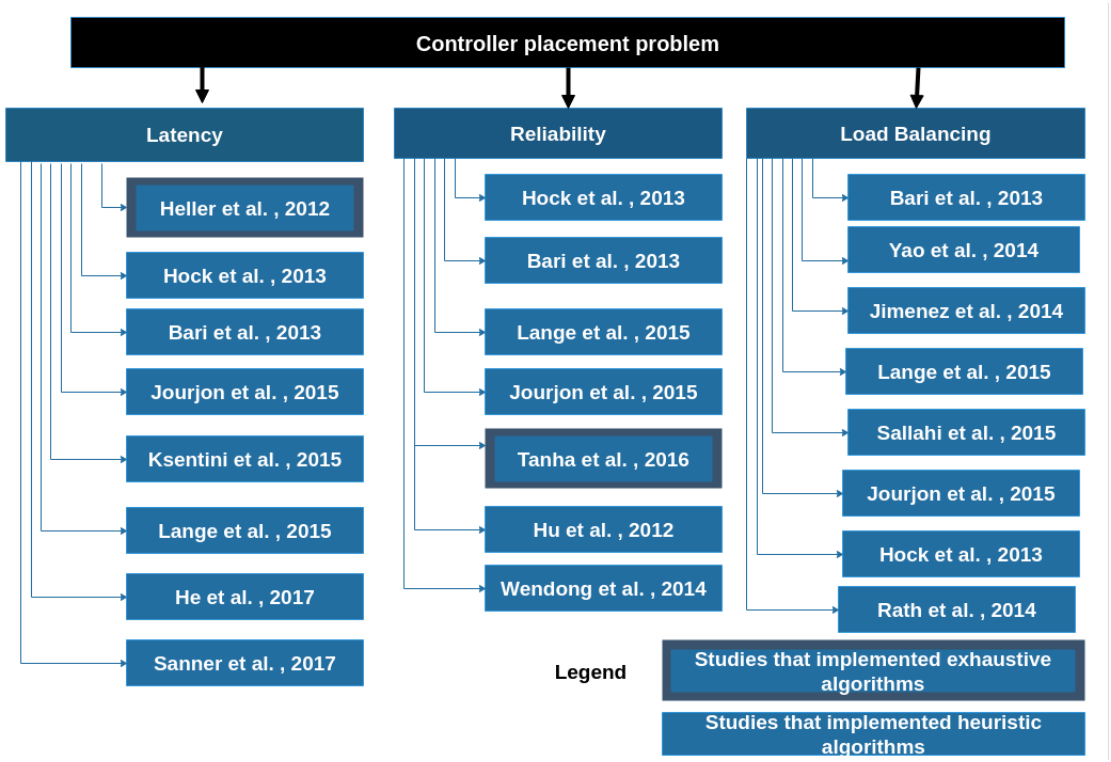


Figure 2.3: Taxonomy of related work

As demonstrated by Heller et al [68], Hock et al. [82] and Wendong et al. [81], there exists a significant tradeoff between load balancing, reliability (also known as resiliency) and latency. Therefore it is almost impossible to optimize one objective without sacrificing the other. This study attempts to address the controller placement problem in consideration of switch-to-controller latency metric. This metric has emerged as an important QoS determinant in SDN. This is primarily because the communication between the controller and data plane has to be seamless to ensure an accurate view of the network state and prompt data plane flow installations. From the state-of-the art review, it is apparent that most studies (with the exception of the work by Sallahi et al. [80]) assume the number of controllers to be known in advance. However, the model proposed by Sallahi et al. is ideal to plan a small–scale SDN and runs out of memory when solving larger instances. Moreover, most studies relied on heuristic algorithms to reduce algorithm runtime. However, this is achieved at the expense of solution accuracy. To the best of our knowledge, the only research studies that implement exhaustive algorithms are by Heller et al. [68] and Tanha et al. [69]. Both Heller et al. and Tanha et al. propose the use of k-center to solve the controller placement problem. However, k-center is sensitive to outliers and does not always consistently yield accurate results [86]. Perhaps more importantly, there is currently no analysis of the controller placement problem purely using an emulation platform to mimic a real SDN deployment. Most studies relied on mathematical modeling to address the controller placement problem, making it difficult to verify validity and reliability of the results.

Controller placement is a network planning effort, and is normally not time sensitive. Consequently, this study proposes exhaustive algorithms to optimize solution accuracy. To determine the optimal number of controllers to deploy given a wide area network, this study proposes the use of Silhouette [87] and Gap statistics [88] algorithms. To find the best locations to place SDN controllers, this study proposes the use of a classical machine learning algorithm called Partition Around Medoids (PAM) [89]. PAM is chosen because it is exhaustive and therefore optimizes solution accuracy. Moreover, in contrast to the k-center algorithm, PAM chooses data points as centers making it more robust to noise and outliers [89]. PAM has to the best of my knowledge never been used in the context of controller placement. This study considers the uncapacitated controller placement problem. For further SDN optimization, a comparative performance evaluation of prominent SDN controllers is presented. To mimic a real SDN deployment, the controller placement problem is studied using an emulation orchestration platform. This is something that to the best of our knowledge has not been done, and we consider it necessary to verify the outcome of the mathematical modeling. Finally, a mechanism to manage control plane overhead is proposed.

Table 2.5: Classification of existing controller placement solutions

Solution	Topology(s)	Network Size	Environment	Algorithm(s)	Placement Metric(s)	Network Partitioning
Heller et al. [68]	Internet2 OS3E	Large–scale	Static	k–center	average switch–to–controller latency worst–case latency	No
Hu et al.[71]	Internet2 OS3E	Small and medium-sized	Static	l–w greedy	Reliability	No
Tanha et al. [69]	Sprint ATT NA PSINET UUNET	Large-scale	Static	Capacitated k-center	switch–to–controller latency Reliability	No
Yao et al. [74]	Internet Zoo	Large–scale	Dynamic	Linear relaxation	switch–to–controller latency Load balancing	No
Jimenez et al. [75]	Sparse Medium Dense	Large–scale	Dynamic	k–critical	Load balancing	Yes
Bari et al. [76]	RF-I	Large–scale	Dynamic	DCP-GK	switch–to–controller latency Load balancing	Yes
Jourjon et al. [77]	Not discussed	Large–scale	Dynamic	LiDy+	switch–to–controller latency Load balancing	Yes
Sanner et al. [78]	Internet2 OS3E	Large–scale	Dynamic	NSGA	inter–controller latency load balancing	Yes
Rath et al. [79]	Random network with 28 switches	small-scale	Dynamic	Non–zero–Sum Game	Load balancing	No
Sallahi et al. [80]	Random network with 10, 20, 30, 40, 50, 75, 100, 150 switches	small-scale	Dynamic	CPLEX	Load balancing	No
Wendong et al. [81]	Internet2 OS3E	Large–scale	Static	l–w greedy	switch–to–controller latency Reliability	No
Hock et al. [82]	Internet2 OS3E	Small and medium–sized	Static	POCO	switch–to–controller latency Reliability Load balancing	No
Lange et al. [83]	Internet2 OS3E Internet Zoo	Large–scale	Dynamic	Simulated Annealing	switch–to–controller latency Reliability Load balancing	No
Ksentini et al.[84]	Ring Binary Tree	Large–scale	Static	No specific name	switch–to–controller latency Inter–controller latency Load balancing	Yes

Chapter 3

Controller Placement using Mathematical Modelling

3.1 Introduction

Wide area networks (WANs) are typically partitioned into smaller administrative domains to ensure failover and load distribution in the control plane [90]. This chapter describes the algorithms that were used to find the locations and number of controllers to deploy in SDN-enabled WANs in order to achieve a high quality of service (QoS). In order to compute the optimal number of controllers, we propose two "unsupervised" machine learning approaches namely, Silhouette and Gap Statistic. Unsupervised algorithms learn from input data that has no labeled responses [91]. These algorithms are classically used to analyze cluster quality through the metric of minimum distances between data points. In the context of controller placement, we leverage these algorithms to find the number of controllers that minimizes overall network propagation latency (i.e. switch-to-switch latency). To find the best locations for these controllers, we extend a facility location algorithm called Partition Around Medoids algorithm (PAM), with propagation latency (i.e. controller-to-switch latency) as our main objective function. For realism, we use the South African National Research Network (SANReN) as a case study. The choice of this topology was mainly motivated by the fact that it represents the emerging market case study which is the key use case of this study.

Since the links between SANReN's switches are known to be fiber where speed is approximately the speed of light in fiber (i.e. 2×10^8 m/s), we compute propagation latency by taking the ratio of average distance (between nodes) to speed of light in

fiber. The distances are calculated using the Harvesine approach. The results from our simulations and discussions are also presented in this chapter.

3.1.1 Assumptions

The following assumptions apply to the proposed algorithms:

- Switch-to-controller communication is assumed to happen out-of-band ;
- The bandwidth for all connection links is constant;
- Control path security and reliability has been perfectly solved;
- Controller and switches are co-located;
- Switches incur a fixed load.

3.2 Overview of Implemented Algorithm(s)

This study involves the extension of four algorithms for solving the controller placement problem to expedite SDN adoption in emerging markets. Subsection 3.2.1 introduces the algorithms used to find the optimal number of controllers to deploy given a wide area network, and subsection 3.2.2 describes the algorithms for finding the best locations to place SDN controllers.

3.2.1 Optimal number of Controllers

Silhouette Analysis

Silhouette Analysis is a method of interpretation within existing clusters, used to measure the quality of a cluster (how close each point in a cluster is to its adjacent clusters) for varying number of partitions [92]. In the context of the controller placement problem, we adopt and extend this algorithm to answer this question: given a wide area network topology, how many controllers are needed to achieve minimum intra-cluster propagation

latency variation? Equation 3.1 shows our objective function, where C_k is the k_{th} cluster and $L(C_k)$ is the intra-cluster propagation latency variation.

$$min \sum_{k=2}^{n} L(C_k) \tag{3.1}$$

Algorithm 1 outlines the Silhouette approach. The algorithm requires three input parameters namely, a clustering algorithm (clustAlg) to cluster network data plane nodes, distance function handle (disfun), network topology graph ((G(V,E,X)), where V denotes data plane nodes (switches), E denotes edges (links between switches), and X denotes the geographic locations (longitude, latitude) of nodes), and maximum number of controllers (maxNumControllers). The clustering algorithm used is called Partition Around Medoids (PAM) described in section 3.2.2 [92]. The harvesine distance approach was used to compute the great circle distances between pairs of switches [93]. The great-circle distance is the shortest distance between two locations on a sphere, measured along the surface of the sphere (as opposed to the ordinary Euclidean distance)[94] [95] . An alternative method to compute geographic distances is the Law of Cosines, which is optimal for shorter distances and is not as accurate for longer distances. To compute the great-circle distance, equation 3.2 which defines the harvesine approach is used, where φ_1 and φ_2 are the latitudes of point 1 and 2 respectively, λ_1 and λ_2 is the longitudes of point 1 and 2 respectively and r denotes the radius of the earth, a constant equal to 6371 km.

$$Distance = 2r \times arcsin(\sqrt{sin^2\frac{\varphi_2 - \varphi_1}{2} + cos(\varphi_1)cos(\varphi_2)sin^2\frac{\lambda_2 - \lambda_1}{2}}) \tag{3.2}$$

In this way we get the intraMean by calculating the average of the distances between each point in the dataset to its centroid (Instruction 4). The procedure to compute the optimal number of controllers using Silhouette (with steps/instructions enumerated from 1 to 12 in Algorithm 1) is as follows: First, a cluster model is created from input network data (Instruction 4). Next, the average propagation latency from each switch to its cluster centroid is calculated (Instruction 6), to find the intra-cluster propagation latency variation (intraClustVar).To this end, a model from the centroids is created (Instruction 7). Next, the average propagation latency between each centroid to the global center (Instruction 8-9) is calculated. In this way we obtain the inter-cluster propagation latency variation (interClustVar). The last step is to calculate the silhouette coefficient (Instruction 11). This procedure is repeated as specified by the maxNumControllers input parameter in order to calculate the silhouette coefficient for each number of controllers.

Moreover, for each number of controllers (Instruction 3), the number of iterations was set to 20 to maximize accuracy of the results.

The optimal number of controllers is one that yields the maximum silhouette coefficient. This coefficient has a range of [-1,1]. Therefore a value closer to +1 is preferred as it indicates better cluster configuration.

Algorithm 1 Silhouette Analysis

Require: $G(V, E, X)$, $maxNumControllers$, $disfun$, $clustAlg$

1: $totalNodes \leftarrow G(V, E, X).size()$
2: $k \leftarrow 2$
3: **for** $k \leftarrow 2$ *to* $maxNumControllers$ **do**
4: $clusters \leftarrow Cluster.train(\ G(V, E, X),\ k,\ disfun,\ clustAlg)$
5: **for** $j \in G(V, E, X)$ **do**
6: $intraClustVar \leftarrow clusters.computeCost(j)/totalNodes$
7: $centroids \leftarrow sc.parallelize(clusters.clusterCenters)$
8: $clusterCentroids \leftarrow Cluster.train(centroides)$
9: $interClustVar \leftarrow clusterCentroids.computeCost(centroides)/k$
10: **end for**
11: $Silhouette \leftarrow (interClustVar - intraClustVar)/max(intraClustVar,\ interClustVar)$
12: **end for**

Gap Statistics

Similar to Silhouette Analysis, Gap Statistic is a partitional algorithm typically used in neural networks, to measure the quality of clustering measure based on average intra-cluster variation [96] [88]. We adopt and enhance this algorithm to verify the results from our Silhouette Analysis. Therefore our goal is to determine the optimal number of SDN controllers to deploy given a network topology, and compare the outcome of the simulation the results from the Silhouette Analysis.

The Gap Statistic algorithm constitutes the following steps (enumerated by instructions from 1 to 12 in Algorithm 2): First the network topology is partitioned (using the PAM algorithm), by varying the number of controllers k (which corresponds to the number of clusters) from 2 to the maximum user-defined value (Instruction 3). This is followed by the computation of the average intra-cluster propagation latency variation (intraClusVar denoted by $L(C_k)$ in equation 3.3) between the switches (Instruction 4). Next a reference dataset ($rRef$ denoted by B in equation 3.4) of the network topology

is randomly generated (Instruction 6). The average intra-cluster latency variation of the reference dataset (intraClusVarRef denoted by $L^*(C_k)$ in equation 3.3) is computed (Instruction 7). The gap static is calculated using equation 3.3 and 3.4. Finally, the standard deviation of B Monte Carlo replicates is calculated [88]. The optimal number of controllers is one that meets the condition in equation 3.5, where s_{k+1} is denotes the standard deviation of B Monte Carlo replicates .

$$gap_n(k) = E_n^* log(L^*(C_k)) - log(L(C_k)) \tag{3.3}$$

where

$$E_n^* log(L^*(C_k)) = \frac{1}{B} \sum_b log(L^*(C_{kb})) \tag{3.4}$$

$$gap(k) \geq gap(k+1) - s_{k+1} \tag{3.5}$$

Algorithm 2 Gap Statistics

Require: $G(V, E, L)$, $maxNumControllers$, $disfun$, $clustAlg$, $nrefs$

1: $gaps \leftarrow [\,]$ {I}ntialize empty array
2: $k \leftarrow 2$
3: **for** $k \leftarrow 2$ *to* $maxNumControllers$ **do**
4: $intraClusVar \leftarrow clusAlg(G(V, E, L),\ maxNumControllers,\ disfun)$
5: **for** $i \in nrefs$ **do**
6: $rRef \leftarrow random(G(V, E, L)$
7: $intraClusVarRef \leftarrow clusAlg(\ rRef,\ disfun)$
8: $gap \leftarrow log(intraClusVarRef - intraClusVar)$
9: **end for**
10: $s_k \leftarrow standardDev(rRef, k, disfun)$
11: **return** $gap \leftarrow gap.argmax$ {Take maximum gap value}
12: **end for**

3.2.2 Optimal controller location

Johnson's Algorithm

In order to determine the best locations to place SDN controllers in a WAN, the shortest paths between each pair of switches must be known. Johnson's algorithm [97] provides a

means to find the shortest paths between node pairs and has become a popular method for addressing SDN optimization problems [98]. Therefore we used the results from this algorithm alongside the PAM algorithm to determine the best places to deploy controllers. A pseudocode of this algorithm is as shown in Algorithm 3 and consists of the following steps: First a new arbitrary switch (denoted by q) is added to the network graph, connected by zero-weight links to all other switches (denoted by v) in the network graph (Instructions 1-5). If this step detects a negative weight-cycle (i.e. a cycle whose weight sums to a negative number) , the algorithm is terminated (Instruction 6-7). Second, a single source shortest path algorithm called Bellman-Ford algorithm is evoked, to find the shortest path $h(v)$ from each switch v in the network to the new switch (Instructions 9-11). Next, the graph is reweighted to find new link weights w_{new} (Instruction 12-14). Finally, the new switch is removed, and Dijkstra's algorithm is used to compute the shortest paths $p(u, v)$ from each each node to every other node in the reweighted graph (Instructions 15-22).

Algorithm 3 Johnson's Algorithm

Require: $G(V, E)$ {undirected weighted network graph}

1: Compute G' *where* $V[G'] \leftarrow V[G] \cup q$ {G' is the new graph containing q}
2: **for** $v \in V[G]$ {for all switches (v) in the original graph} **do**
3: $E[G'] \leftarrow E[G] \cup (q, v) : v \in V[G]$
4: $z(q, v) \leftarrow 0$
5: **end for**
6: **if** $BELLMAN - FORD(G', w) == False$ **then**
7: **print** Error! Negative cycle detected.
8: **else**
9: **for** $v \in V[G]$ **do**
10: set $h(v) \leftarrow \delta(q, v)$ compute shortest path using Bellman-Ford
11: **end for**
12: **for** $(u, v) \in E[G']$ **do**
13: $w_{new} \leftarrow w(u, v) + h(u) - h(v)$
14: **end for**
15: **for** $u \in V[G]$ **do**
16: execute $Dijkstra(G, w_{new}, u)$ to compute $\delta_{new}(u, v)$ for all $v \in V[G]$
17: **for** $v \in V[G]$ **do**
18: $p(u, v) \leftarrow \delta_{new}(u, v) + h(u) - h(v)$
19: **end for**
20: **end for**
21: **end if**
22: **return** shortest path matrix

Partition Around Medoids (PAM)

After determining the optimal number of controllers to use given a WAN topology, the next step is to find the best locations to place the controllers such that the QoS is maximized. This can be achieved by leveraging "unsupervised" machine learning heuristic algorithms (such as Simulated Annealing [99] and Clustering LARge Applications (CLARA)[100]) or exhaustive algorithms (such as k-means [101] [102] and PAM [103] [104]. However, heuristic algorithms are suboptimal in the sense that they are primarily focused on optimizing runtime over solution accuracy. Therefore, heuristic algorithms are more ideal for scenarios requiring dynamic controller placement. However, this study explores static controller placement, where the controller placement problem is addressed during network planning. Therefore, the accuracy of the optimization algorithm is significantly more important than the speed of computation. From the available exhaustive algorithms, we opted for the PAM algorithm. This is mainly because the k-means algorithm is very sensitive to outliers which can lead to solution inaccuracy [105]. Unlike k-means, PAM is more stable, much faster and more accurate [106].

Algorithm 4 describes the steps we followed to compute the optimal locations of SDN controllers. Our approach assumes co-location of controllers and switches. First, k arbitrary switches (where k is the number of controllers to place)) are selected as the potential controller locations (Instruction 3). This is followed by the association of each switch to the closest controller (Instructions 4-6).While the cost of configuration (the overall propagation latency) decreases, the controller location R_i and switch S_o are swapped (Instructions 7-9), and each switch is reassigned to their closest controller location (Instructions 4-6). If an increase in configuration cost is detected , the swap is undone and the optimal controller locations that optimize QoS are found (Instructions 12-18). Two QoS parameters are considered in our solution, and that is the average propagation latency (which is the overall propagation latency) and the worst-case propagation latency (which is the maximum network latency). Equations 3.6 and 3.7 define how these parameters are defined, where $L_{avg}(Z')$ is the average latency, $L_{wc}(Z')$ is the worst-case latency, $d(v, z)$ is the shortest distance from the switch (node $v \in V$) to the controller (node $z \in Z$), $N = |V|$ denotes the number of nodes and $2X10^8$ is the speed of light in fiber.

$$L_{avg}(Z') = \frac{1}{(2X10^8)N} \sum_{v \in V} \min_{z \in Z'} d(v, z) \tag{3.6}$$

$$L_{wc}(Z') = \max_{v \in V} \min_{z \in Z'} d(v, z) \tag{3.7}$$

Algorithm 4 Partition Around Medoids (PAM)

Require: $G(V, E)$, $NumControllers$, $disfun$, $edgWeigts$

1: Compute shortest path matrix using Johnson's algorithm
2: $k \leftarrow NumControllers$
3: $R_{i\ (i\in[1..k])} \leftarrow$ randomly select k objects from $G(V, E,)$
4: **for** $S_o \in G(V, E)$ **do**
5: Compute similarity score of S_o with each $R_{i\ (i\in[1..k])}$ using $disfun$
6: Associate S_o to the most similar R_i
7: **end for**
8: **for** S_o *and* R_i **do**
9: $swapCost \leftarrow computeCost(S_o, R_i)$
10: **end for**
11: **if** $swapCost \leq 0$ **then**
12: $S_o \leftrightarrows R_i$
Go back to step 4
13: **else**
14: **for** $S_o \in G(V, E)$ **do**
15: Compute similarity score of S_o with each $R_{i\ (i\in[1..k])}$
16: Assign S_o to the most similar R_i
17: **end for**
18: **end if**
19: **return** cl

3.3 Experimental Work

3.3.1 Introduction

This section explains our implementation for solving the controller placement problem using the algorithms described in section 3.2. These algorithms are implemented in MATLAB 2018b. The primary objective is to establish the number of controllers for the achievement of minimum propagation latency and to determine the best locations to place these controllers in a WAN topology. The results from our simulation experiments are also presented in this section.

3.3.2 Topologies

To maintain realism, our proposed solution is applied on a real-world WAN called South African Research Network (SANReN), operated by CSIR'S Meraka Institute [107]. The reason for choosing the SANReN network was so that we could demonstrate our proposed solution on an emerging market use case. However, it may be noted that our solution is topology-agnostic and can easily be used to test other networks of different configurations and sizes.

The SANReN network (depicted in Figure 3.1) constitutes a core national backbone, with each Point of presence (PoP) connecting a metropolitan network. This work only focuses on the PoP-level instead of the router-level view of the SANReN network. This is because the router-level view is proprietary and not publicly available. Moreover, the PoP-level view has been deemed more useful [108] for several points: it provides a larger scale view of network links, which are most interesting for network optimization; it shows end users where they can connect to the network and it's the level where resiliency and redundancy are critical. The PoP-level geographical map of the SANReN topology is depicted in 3.2. It comprises 7 nodes and 7 fiber links configured in a ring topology. The data set of this topology was downloaded from a repository called The Internet Topology Zoo [34]. The format of the data set is in Geography Markup Language (GML) and includes geographic locations (longitude, latitude) of nodes and topological configuration of the SANReN network.

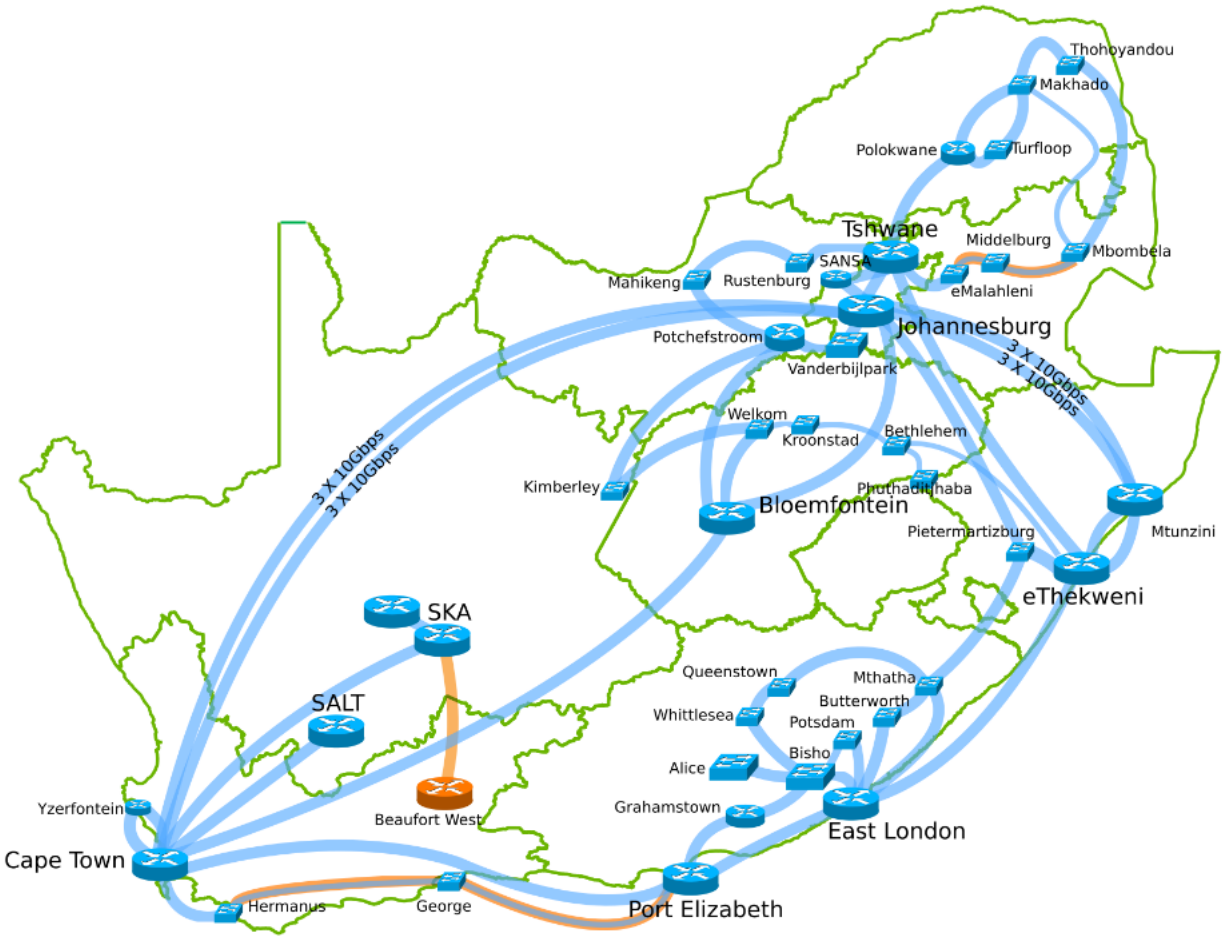


Figure 3.1: Geographical map of the SANReN network[107].

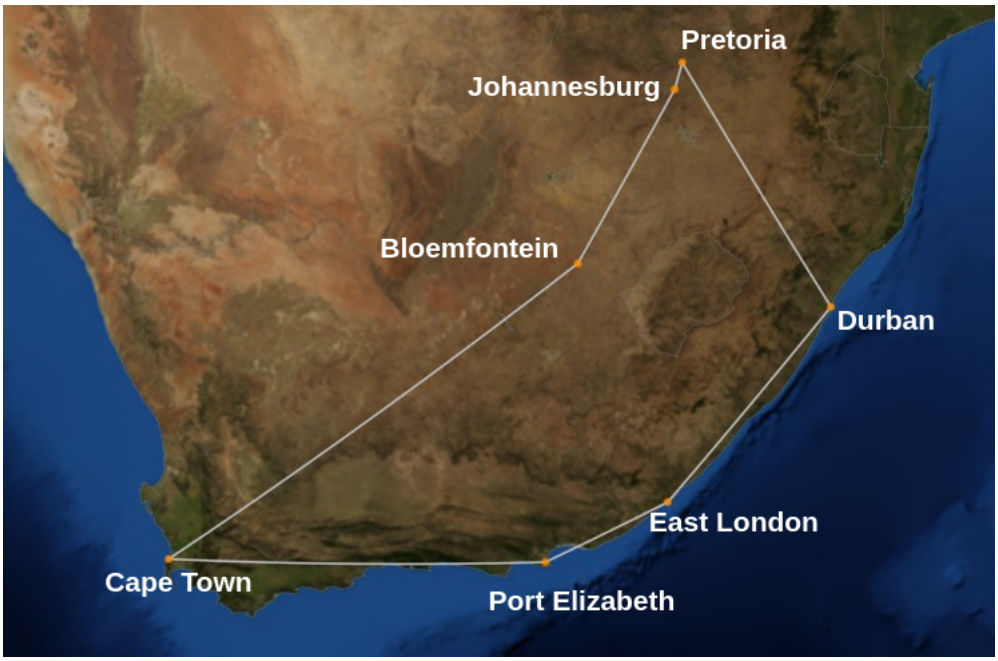


Figure 3.2: PoP-level geographical map of SANReN [34].

3.3.3 Hardware and software used for modelling

All the experiments have been executed under an Ubuntu Desktop 16.04 LTS-64 bit on a PC with the following specification: Intel(R) Core(TM) i7-5600U CPU, with 4 cores (8 threads), a clock speed of 2.60 GHz, RAM amount of 8 GB and a storage capacity of 250 GB.

3.3.4 Flowchart of proposed solution

The flowchart depicted in Figure 3.3 summarizes the steps in our proposed controller placement solution. First, network graph modeling is used to model the network topology as an undirected graph $G(V, E)$, where V denotes network switches and E represents fiber links (edges) connecting the switches. This is followed by the extraction of the geographic location data using the input dataset. Next, the Harvesine approach is applied on the location data to generate the distance matrix. To determine edge weights, an adjacency matrix is implemented between all connected switches. Then, computation of the number of controllers that minimize intra-cluster latency is carried out using Silhouette algorithm as described in section 3.2.1, Algorithm 1. To verify the results from Silhouette, Gap Statistic is implemented as described in section 3.2.1, Algorithm 2. This is followed by computation of the shortest path matrix by applying Johnson's algorithm outlined in Algorithm 3. The results from Silhouette, Gap Statistic and Johnson's algorithm, are used as inputs to the PAM algorithm discussed in section 3.2.2, Algorthm 4, which is used to find the best locations that minimizes propagation latencies, namely the average latency and worst-case latency defined in section 3.2.2 (equation 3.6 and 3.7). The key factor in our mathematical formulation is the distance under the assumption of constant bandwidth across all fiber links. Therefore under constant bandwidth, propagation latency is directly proportional to distance.

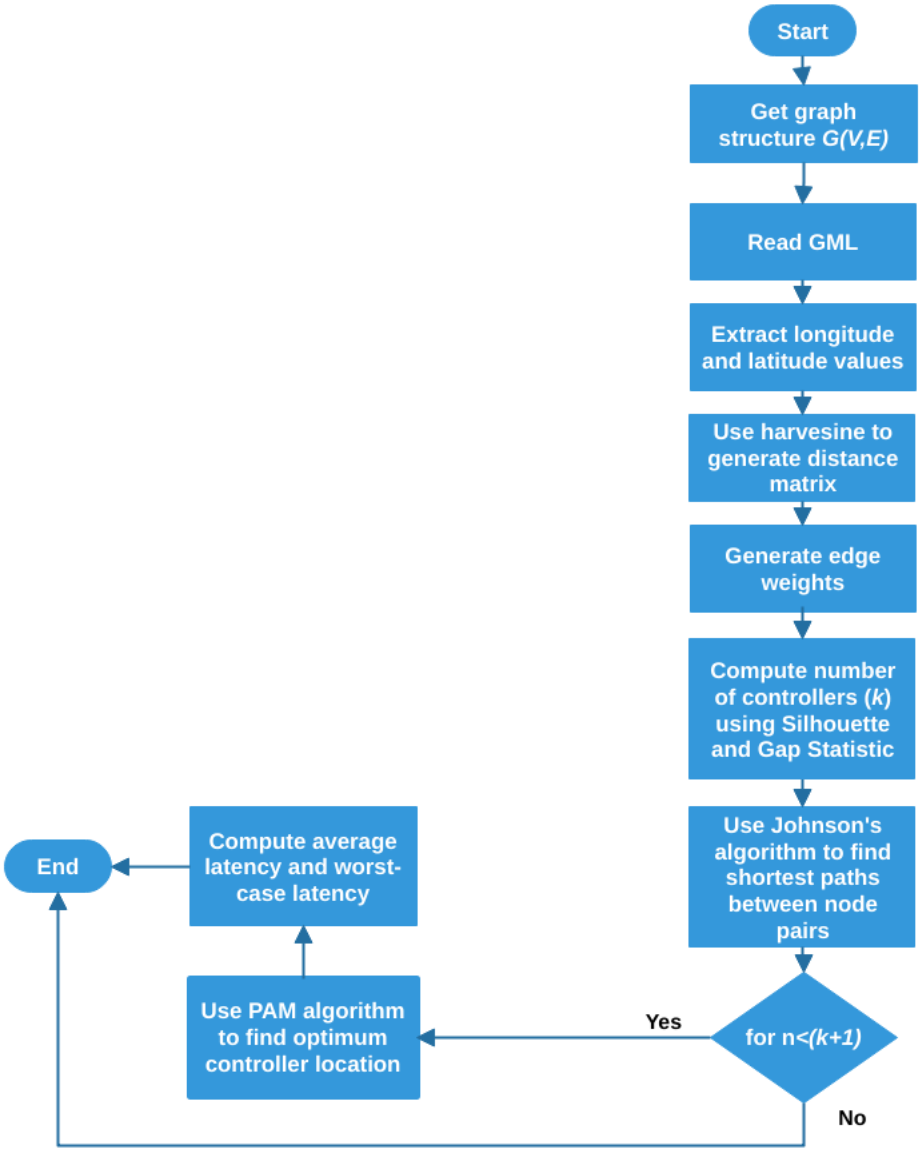


Figure 3.3: Flow chart of proposed method [109].

3.4 Results and Discussion

3.4.1 Optimal number of controller

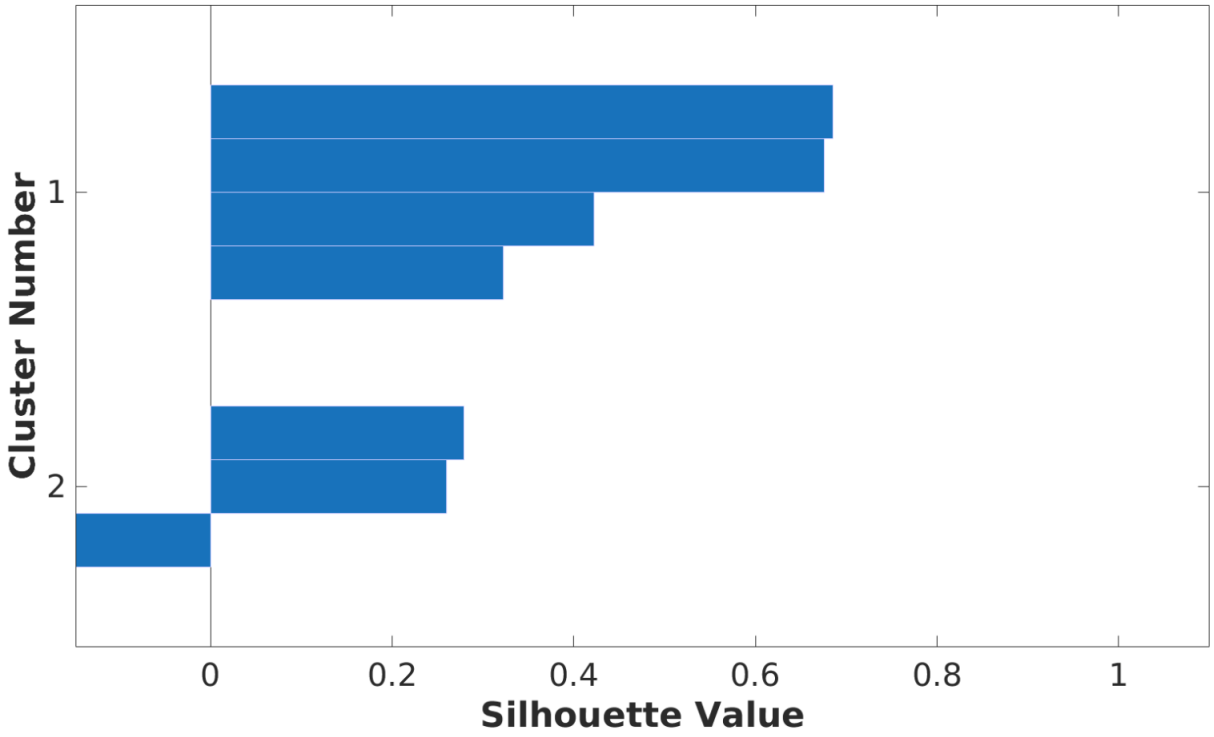
Silhouette Analysis

In order to determine the optimal number of controllers to deploy on the SANReN backbone, we applied our enhanced Silhouette algorithm with propagation latency as our key performance indicator. The results from our Silhouette analysis are as depicted in Figure 3.4. These plots show the clustering quality when different number of SDN

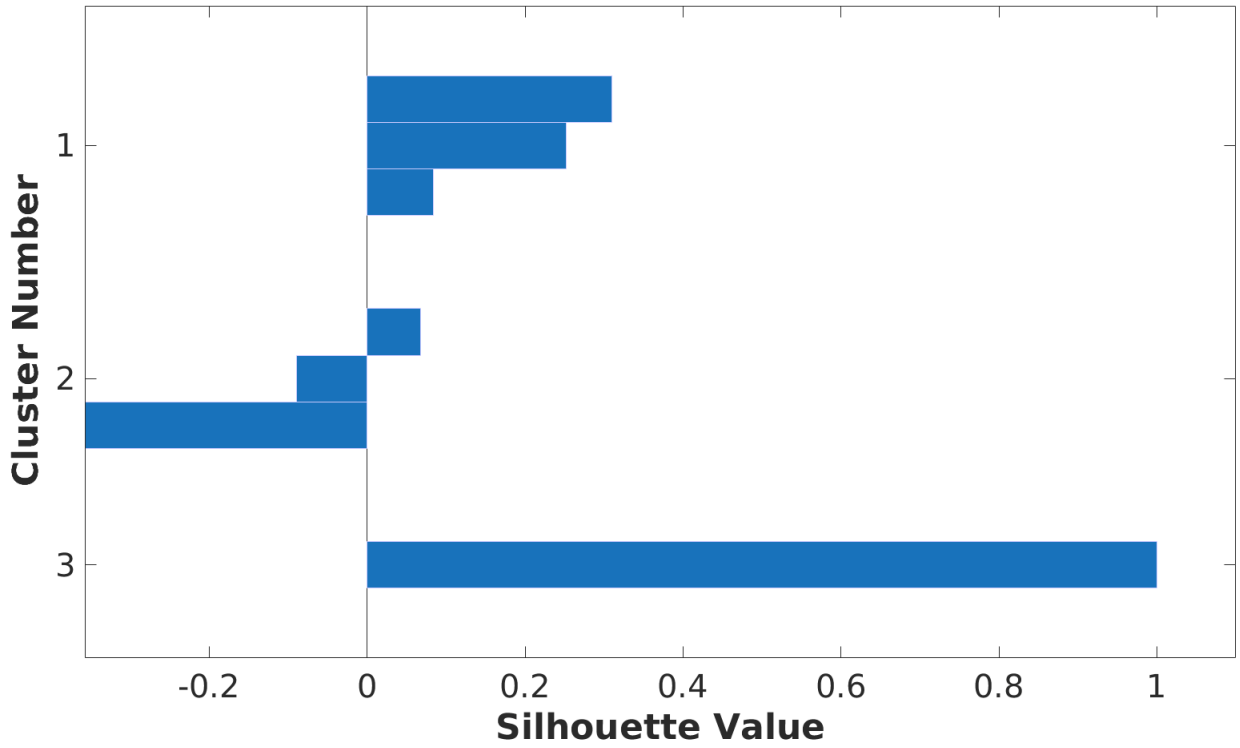
controllers are deployed. For instance, Figure 3.4 (*a*) illustrates the clustering quality when 2 controllers are deployed. The metric used to measure clustering quality is the average intra-cluster propagation latency. Each blue horizontal bar in the plots represent a switch and its corresponding silhouette score. A silhouette score reveals the proximity of a switch to all other switches outside and within its cluster. Silhouette scores lie in the range of [-1,1]. The desired score is one that is closer to +1 as it indicates high proximity of switches within the same cluster. On the other hand, silhouette scores near -1 indicate high dissimilarity within a cluster and is a sign of poor clustering quality. A value of 0 shows that the switch is on or very close to the decision boundary between two adjacent clusters [110].

Our results indicate that deploying 3 or 4 controllers (shown in Figure 3.4 (*b*) and (*c*), respectively) would result in poor clustering quality due to the presence of clusters with very low silhouette scores and the high fluctuations in the size of the silhouette plots. Furthermore, the cluster size (the number of switches per cluster) when the number of controllers is set to 3 and 4 is imbalanced. Therefore, 2 controllers are the ideal number of controllers to deploy on the SANReN network as this will ensure lower propagation latency and a fair switch-to-controller distribution.

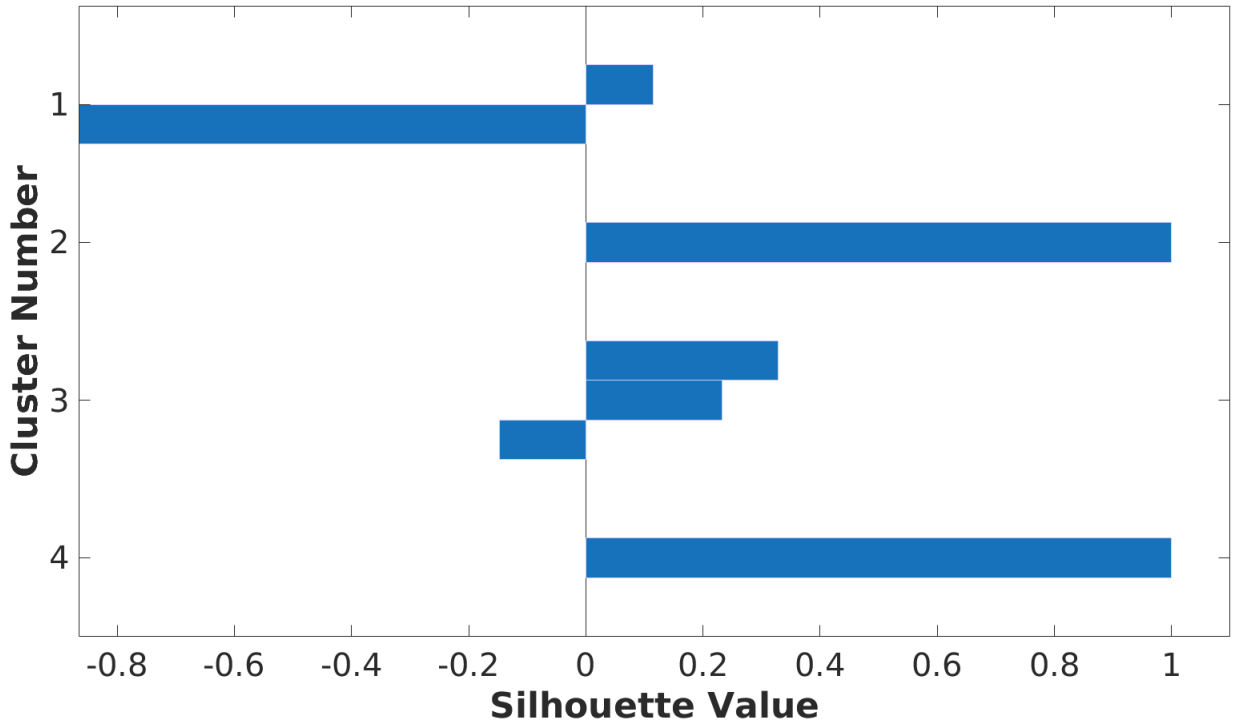
Figure 3.5 depicts the overall evaluation results from our Silhouette analysis based on overall propagation latency. These results show that deploying 2 controllers is an ideal choice. This is seen from the high silhouette score obtained when the number of controllers is set to 2. Although deploying 4 controllers would yield a fairly good clustering quality, it is likely to result in high inter-controller latency (due to the frequent state information exchange between controllers) and high CapEx. On the other hand, 4 controllers would offer more resiliency improving network reliability. However, if latency and cost are topmost priority, then 2 controllers are recommended. Deploying 2 controllers in the SANReN network would suffice to meet reliability requirements unless the network has stringent requirements.


Cluster Number
1
2
0
0.2
0.4
0.6
0.8
1
Silhouette Value

(a)


Cluster Number
1
2
3
-0.2
0
0.2
0.4
0.6
0.8
1
Silhouette Value

(b)



(c)

Figure 3.4: Silhouette analysis to determine optimal number of controllers for (a) $k = 2$ (b) $k = 3$ (c)$k = 4$

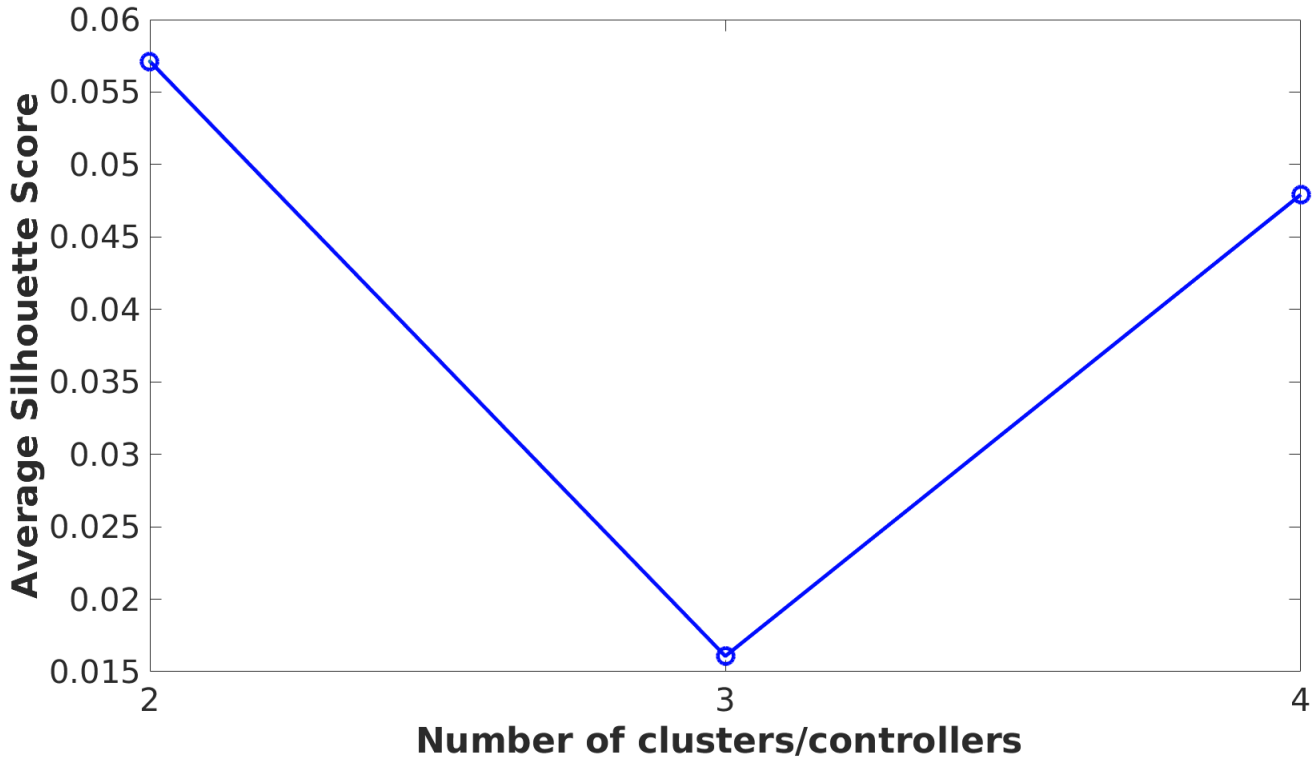


Figure 3.5: Silhouette evaluation summary

Gap Statistic

To verify the results from our Silhouette algorithm, we applied the Gap Statistic algorithm on the SANReN topology. With Gap Statistic the optimal number of controllers corresponds to the highest gap value with the statistical deviation, as it reflects a low intra-cluster propagation latency. Figure 3.6 indicates that the optimal number to deploy on SANReN is 2 controllers. These results match the outcome of our Silhouette analysis.

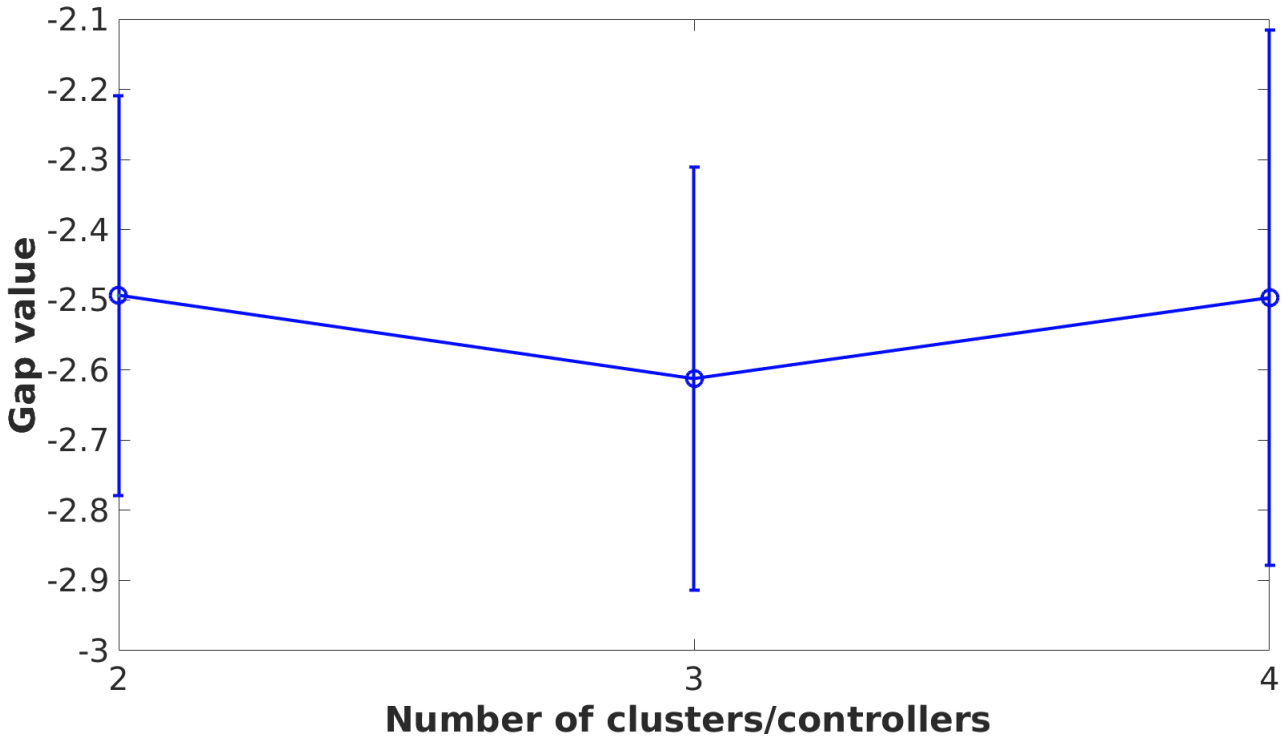


Figure 3.6: Gap Statistic evaluation summary

Cost-Latency Tradeoff Analysis

Another factor that influences the decision regarding the number of controllers to deploy, is the cost associated with installing new controllers in a given network. This metric is critical as it contributes to the overall CapEx and determines how much return on investment (ROI) network operators generate. However there exists a considerable tradeoff between cost and the QoS delivered by the network. Our intention here is to quantify this tradeoff so as to provide a practical guideline to network operators, regarding the ideal number of controllers to use taking into account cost and latency. This tradeoff is termed "cost factor" and is defined in equation 3.8, where k is the number of controllers,

C_k is the cost of deploying a controller and L_{avg} is the average latency when k controllers are deployed.

$$cost\ factor = \frac{k \times C_k}{L_{avg}} \qquad \left[\frac{\$}{ms}\right] \tag{3.8}$$

The average latency is the overall propagation latency computed using the PAM algorithm described in section 3.2.2 for varying number of controllers. Figure 3.7 shows our results from analyzing the tradeoff between cost and network performance. As expected, the results indicate that deploying 1 controller is an ideal choice to ensure minimal tradeoff between cost and network performance. However, to ensure network scalability and failover, we recommend using 2 controllers. This is primarily because 2 controllers are the second best option that provides the least tradeoff, and our Silhouette and Gap Statistic analysis recommend 2 controllers as the optimal number to deploy on SANReN. However, this does not provide a comprehensive cost analysis but only provides a basis for one.

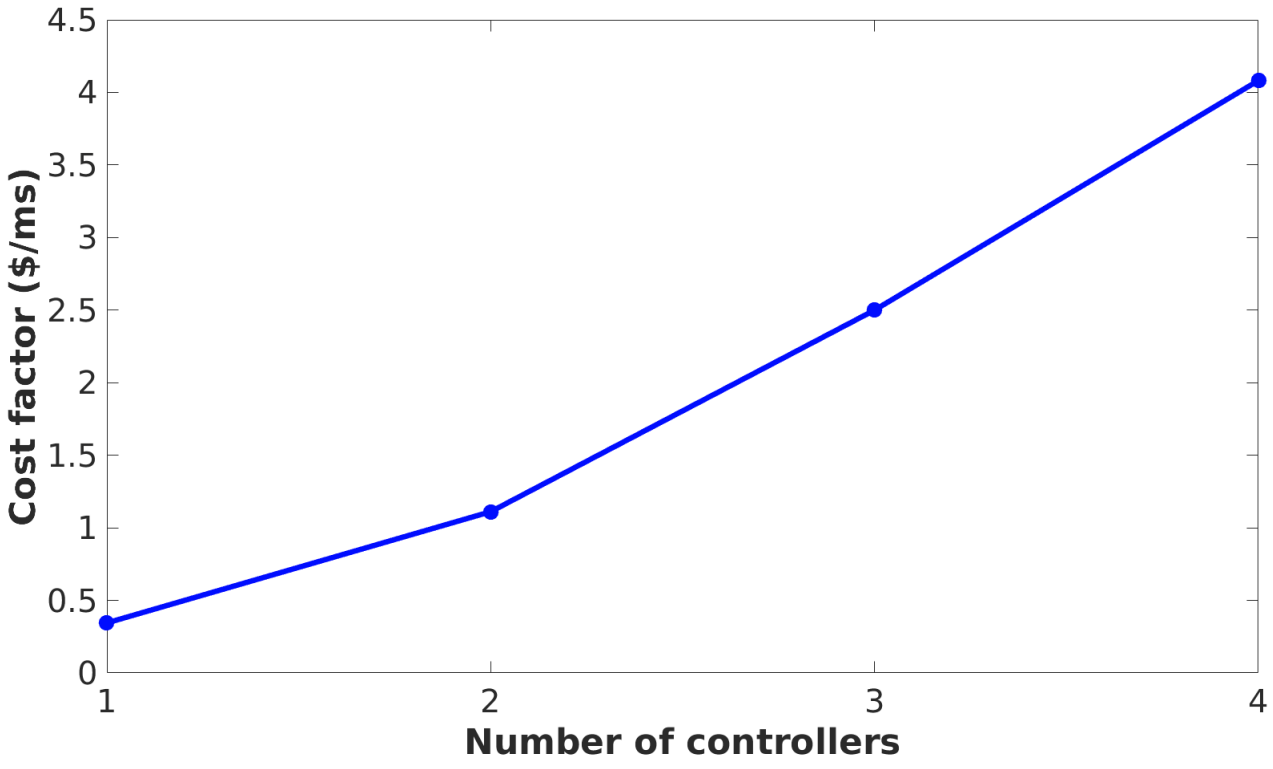


Figure 3.7: Tradeoff between cost and latency for varying number of controllers

3.4.2 Optimal controller locations

After determining the optimal number of controllers using the Silhouette analysis and Gap Statistic, the next step is to determine the best locations to place the recommended two SDN controllers. To find these locations, we use our proposed PAM algorithm described in section 3.2.2. The results (depicted in Figure 3.8) indicate that the optimal locations to place two controllers are Pretoria and East London with the average propagation latency of $L_{avg} = 1.81$. The selection of these locations guarantees the best network performance with respect to the southbound communication in the SANReN network. In contrast, deploying the controllers in Port Elizabeth and Bloemfontein would result in poor network performance, with the worst-case propagation latency being $L_{wc} = 3.92$.

Table 3.1 presents the effect of increasing the number of controllers (k) on average and worst-case latency. These results were obtained by applying the PAM algorithm. The results indicate that, varying the number of controllers from k=1 to k=2 significantly reduces propagation latency (approximately 38% reduction of average latency and 42% reduction of worst-case latency). A further reduction is observed when the number of controllers is set to k=3. However, increasing the number of controllers beyond 3 controllers has a much less significant impact on latency (as depicted in Figure 3.9).

Table 3.1: Average and worst-case latency for varying number of controllers

	$k=1$	$k=2$	$k=3$	$k=4$
L_{avg} (ms)	2.9	1.81	1.2	0.98
L_{wc} (ms)	6.8	3.92	5	5.3
Names of locations for L_{avg}	Durban	Pretoria East London	Pretoria Johannesburg Port Elizabeth	Johannesburg Durban East London Port Elizabeth
Names of locations for L_{wc}	Bloemfontein	Port Elizabeth Bloemfontein	Cape Town East London Durban	Pretoria Cape Town Bloemfontein Port Elizabeth

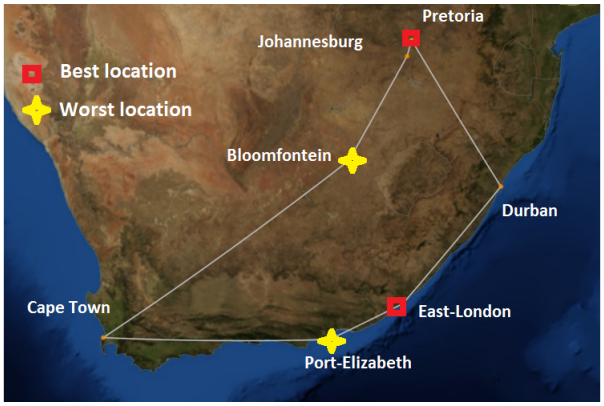


Figure 3.8: Best and worst placements of two controllers on SANReN backbone

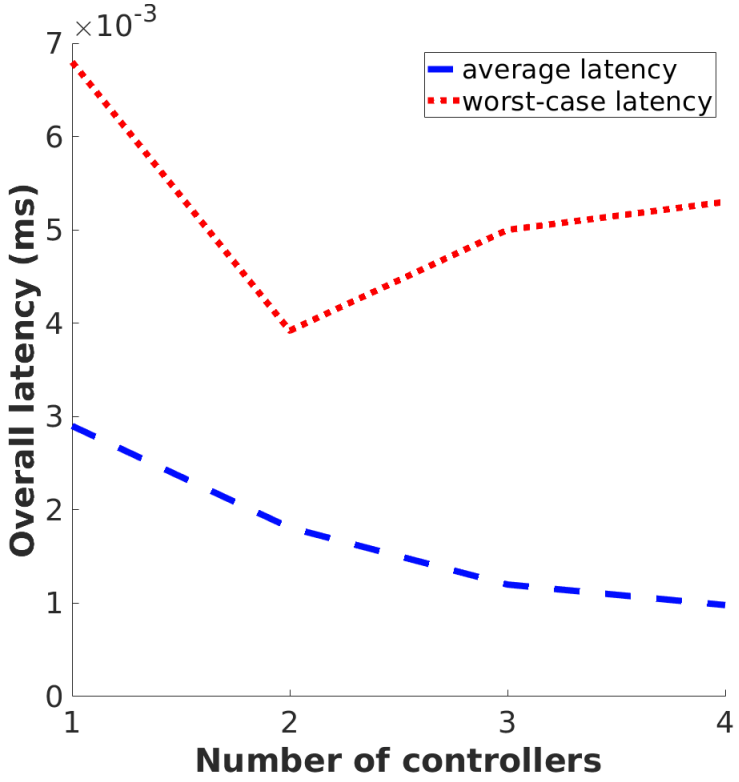


Figure 3.9: Relation between number of controllers and latency

3.5 Chapter Summary

SDN has appeared as a promising solution to bridge the digital divide in emerging markets, by enabling network programmability through separation of the control plane and the data plane. This separation poses several challenges with respect to network scalability, reliability and performance. During SDN planning, a decision must be made regarding the number of controllers to deploy and the locations in which to place them. This chapter proposed three mathematically-based algorithms namely, Silhouette, Gap Statistic and PAM to facilitate controller placement decision making. These are

exhaustive algorithms which are ideal from static controller placement with minimal to no time constraints. Unlike heuristic algorithms, the proposed algorithms are more accurate. These algorithms are generic enough to test topologies of various sizes and configurations. Our evaluation of the number of controllers to deploy was based on latency and cost of deploying new controllers. For our SANReN case study, we discovered that two controllers are ideal to achieve a low controller-to-switch propagation latency at a reasonable cost. As controller placement is topology dependent, the results presented in this chapter only apply to the SANReN topology. Our proposed solution can be leveraged by potential service providers who would like to migrate to SDN, to mitigate their concerns regarding SDN performance.

Chapter 4

Controller Benchmarking

4.1 Introduction

The controller placement problem constitutes three aspects that must be optimized during the network planning phase namely, the optimal number of controllers to deploy, the best location(s) to place the controller(s) and the type of controller to deploy. Chapter 3 strictly focused on optimizing the number of controllers and their locations with propagation latency as the main objective function. The solution presented in chapter 3 is based on mathematical modelling and does not take into account other real network dynamics such as controller processing latency, throughput and scalability. These factors are integral to the performance of a real world network.

To mimic a real SDN deployment, a decision regarding which controller to deploy has to be made. This decision is mainly influenced by two factors: (i) the features promised by the controller and (ii) the controller performance. Some of the important features typically considered are southbound APIs supported (necessary for interoperability), security, architecture (distributed/centralized core), support documentation and partnership (this reflects controller maturity), modularity (to enable application customization) and orchestration support. In terms of network performance, two metrics are typically considered namely, switch-to-controller latency (this is a measure how fast a controller responds to packet-In messages) and throughput (this is indicative of how many packet-In messages a controller can process per second).

At this juncture, there has been a plethora of open source SDN controllers proposed by different communities. The most prominent ones (based on adoption

coverage) appear to be Ryu [63], Floodlight [111], OpenDayLight [112] and ONOS [31]. A feature-based comparison of these controllers is presented in chapter 2 section 2.1.3. The results from this comparison indicate that Floodlight and Ryu are ideal for campus networks since they have a centralized core and lack support for east/westbound protocols such as BGP-LS (this protocols are necessary to deploy a distributed control architecture). The feature-based comparison also revealed that OpenDayLight supports the most southbound protocols (including legacy protocols) making it an ideal choice in multi-vendor systems. Last but certainly not least, it was discovered that ONOS has high modularity, and a distributed core making it ideal to use in wide area SDN deployments.

A feature-based comparison only presents a qualitative analysis of SDN controllers and is not the only dimension in choosing controllers, as the controllers may be slow for given tasks. There is a need to carry out a performance evaluation of SDN controllers subject to the same traffic conditions. Controller benchmarking has been studied in the past as can be seen in [113], [114], [115], [116], [62], [117], [118]. However, given the rapid introduction of new versions of SDN controllers, the past evaluations are seemingly obsolete necessitating the need to re-evaluate controller performances. As a result, this chapter presents a comparative performance evaluation of the most prominent controllers namely, Floodlight, Ryu, OpenDayLight and ONOS, using a benchmarking tool called Cbench [33]. The key performance indicators considered in our simulations are latency and throughput.

4.2 Simulation Tools

As mentioned above, the aim of this chapter is to evaluate the performance of four SDN controllers namely, Floodlight, Ryu, OpenDayLight and ONOS. To achieve this, a benchmarking tool called Cbench is used. Cbench is a program for evaluating the performance of an OpenFlow-compatible controller by stressing it with packet-In messages. Cbench emulates OpenFlow switches which connect to a controller, push packet-In messages and wait for packet-Out messages, whilst measuring various metrics. Cbench mainly operates in two modes namely, throughput and latency. Figure 4.1 illustrates the list of simulation tools used for the experiments.

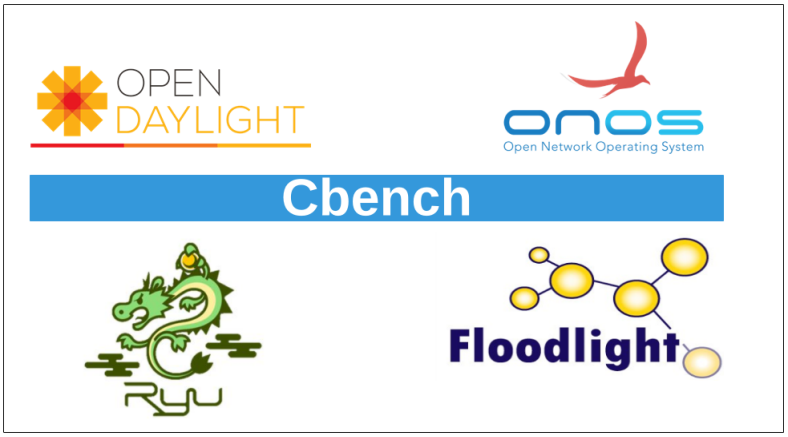


Figure 4.1: Simulation tools

4.3 Test Environment

To overcome the Ethernet interface speed limitations, both Cbench and controller under test (Floodlight, Ryu, ONOS and OpenDayLight) are implemented under a shared Ubuntu server 16.04 LTS-64 bit on a virtual machine with 4 CPUs (8 threads) and 8 GB of RAM.

4.4 Methodology

Figure 4.2 illustrates the setup of our experiment. Cbench is used to emulate the data plane. Our experiment is divided into two parts. First, we investigate the impact of control load on controller latency (time it takes to receive a packet-Out message after evoking a packet-In message). Second, we perform the same evaluation to measure the impact on throughput.

In the first part of the experiment, the number of switches is varried from 1 to 4 and then from 4 to 32 in increments of 4. Each time, the switches are connected to the controller under test using OpenFlow and packet-In messages are sent to the controller. Using cbench's statistic and performance probe modules, the packet-Out messages are counted and latency is measured. Here, the number of MACs (a shorter version of media access control address) identifiable as hosts is fixed at 1000 MACs while varying the number of switches. This experiment is repeated under varying number of MACs per switch (1K, 10K, 100K, 1000K, 10000K) while the number of switches is fixed at 16. The decision to keep the number of switches at 16 while varrying the number of MACs was

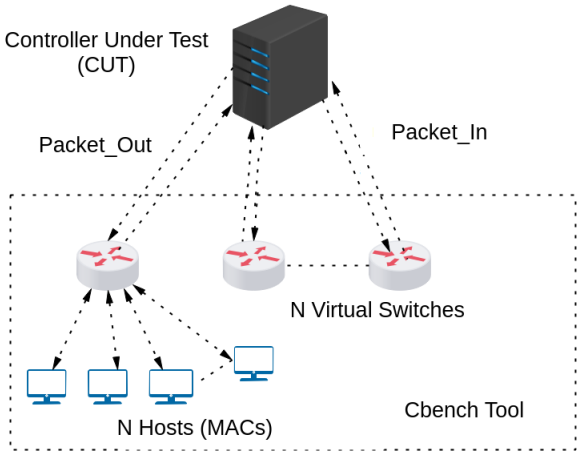


Figure 4.2: Setup of the benchmarking experiment [65]

recommended in [119]. This is mainly because the test suite reaches a timeout when the number of switches and MACs are increased beyond 16 and 100 000 respectively. Nonetheless, the limitation to 16 switches is not significant to the experimental results.

The second part of the experiment is performed to measure throughput by varying the number of switches and MACs in the same manner as the first experiment. The number of worker threads was set to 8 to maintain a fairly low runtime. To reduce influence of time sharing in virtual environment, 14 iterations are performed per test in both experiments. The duration of each test is set to 10 seconds. The first two loops are considered the controller warm-up and their results are ignored. The following example command is used to carry out the test:

./cbench $-c$ localhost $-p$ 6633 $-l$ 14 $-m$ 10000 $-M$ 1000 $-s$ 8 $-t$

where

- c : controller IP or hostname (localhost);
- p : controller port number (6633);
- l : number of iterations (14);
- m : test duration in ms (10000);
- M : number of MACs per switch (1000);
- s : number of switches (8);
- t : throughput mode.

4.5 Results and Discussion

This section presents the results obtained by running the tests described above. The raw data is as shown in Appendix A Tables A4 and A5.

4.5.1 Controller throughput

As shown in Figure 4.3, there is significant difference in the throughput exhibited by various controllers under varying number of switches. It is clear that Ryu's throughput performance is the poorest. This can be attributed to the fact that Ryu is resource-intensive and uses CPU and RAM to optimum resulting in performance degradation in the presence of increasing number of switches [120]. OpenDayLight and Floodlight also exhibit a low throughput when the number of switches are increased beyond 5 and 10 switches, for OpenDayLight and Floodlight respectively. Similar to Ryu, these controllers are more resource intensive and require activation of hyper-threading to improve performance. ONOS exhibit the best throughput performance probably because its multi-threaded and adding more switches leads to better CPU utilization.

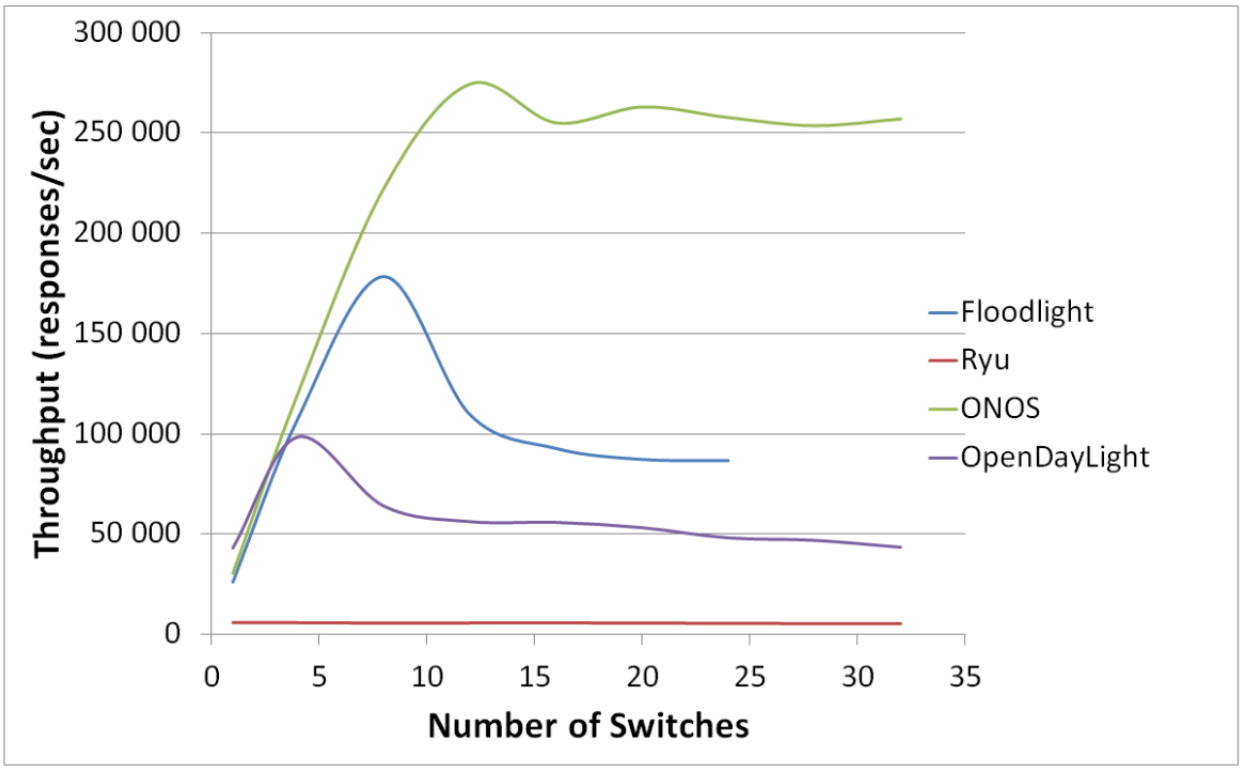


Figure 4.3: Average number of responses per second under varying number of switches (MACs =1000, threads=4)

4.5.2 Controller scalability

Figure 4.4 depicts the results obtained when the number of MACs per switch is varried under a fixed number of switches. This experiment was conducted to measure SDN controller's scalability under high traffic volumes. The results indicate that OpenDayLight and Ryu throughput remains virtually unchanged under varying number of MACs. Thus, these controllers are not ideal to use for large scale SDN deployments. Floodlight exhibits high throughput for smaller number of MACs (up to 10 000 MACs) and show diminishing performance when the number of MACs is set beyond 10 000. Similar to the results obtained in 4.3, ONOS has the best throughput performance compared to its rivals. This is likely because ONOS has a data plane contention management module which divides the MAC address table among a collection of hash tables, thereby improving ONOS's responsiveness [121]. The results presented in Figure 4.4 coincide with those in Figure 4.3 controller throughput performance under various sizes of traffic volumes and data planes.

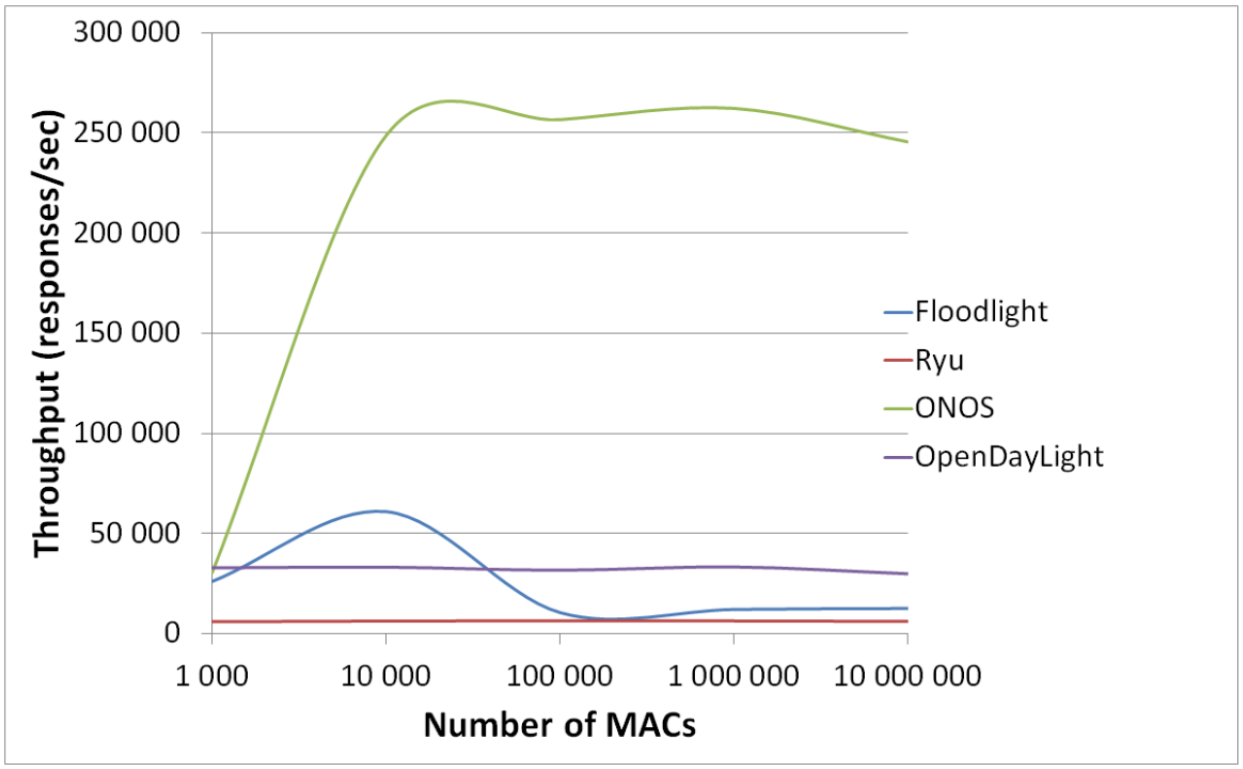


Figure 4.4: Average number of responses per second under varying number of MACs (switches=16, threads=4)

4.5.3 Controller latency

The results obtained in throughput mode are as shown in Figure 4.5 and Figure 4.6. The results show that ONOS exhibits the worst latency performance both under varying number of switches (as shown in Figure 4.5 and varying number of MACs (as shown in Figure 4.6). Ryu stands out as it exhibits the best latency performance. The latency remains low and constant under varying number of switches. This is probably because Ryu has limited support for multi-threading. OpenDayLight performance is better than Floodlight under varying number of switches. However, when the number of MACs is increased beyond 100 000, OpenDayLight's performance becomes significantly poorer than Floodlight showing its scalability limitations. Therefore, networks using Floodlight require more time to find the route and send instructions for new flows.

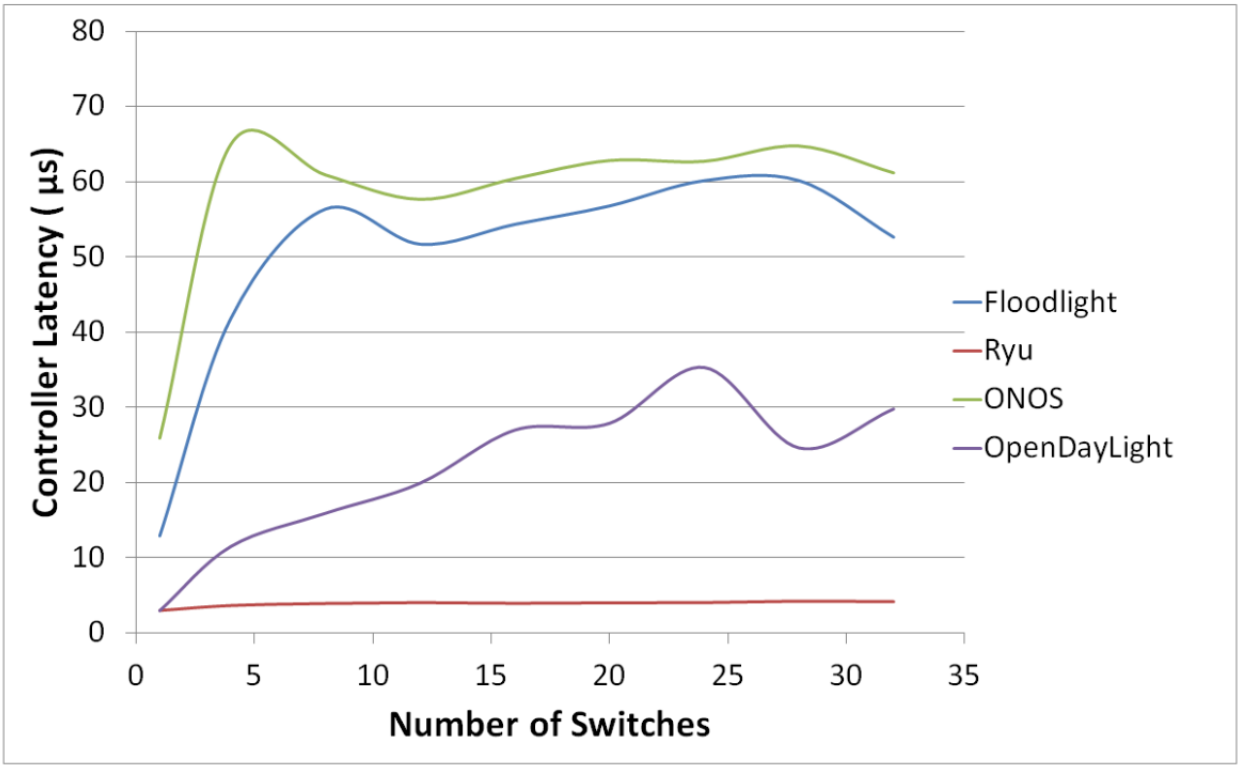


Figure 4.5: Average latency under varying number of switches (MACs=1000, thread=4)

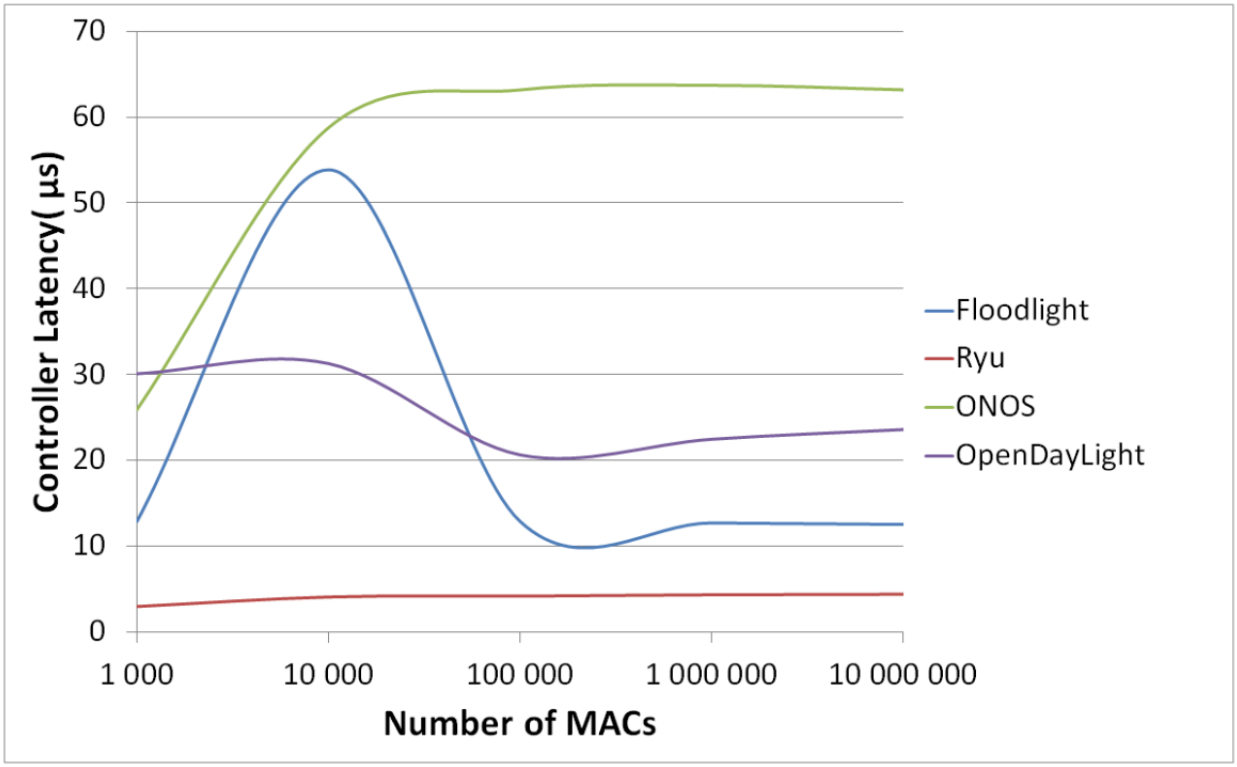


Figure 4.6: Average latency under varying number of MACs (switches=16, threads=4)

4.5.4 Compound performance

We define an additional figure of merit for compound performance by taking a ratio of throughput to latency (see equation 4.1). The results from this analysis are presented in Figure 4.7 and 4.8 for varying number of switches and MACs. The results show that ONOS has in overall the best performance, especially in large-scale SDN deployments. This is indicative of ONOS scalability benefits. Floodlight has the least compound performance in comparison to Ryu and OpenDayLight whose performances are virtually the same for larger data plane instances.

$$compound\ performance = \frac{throughput}{latency} \quad \left[\frac{responses}{sec^2}\right] \tag{4.1}$$

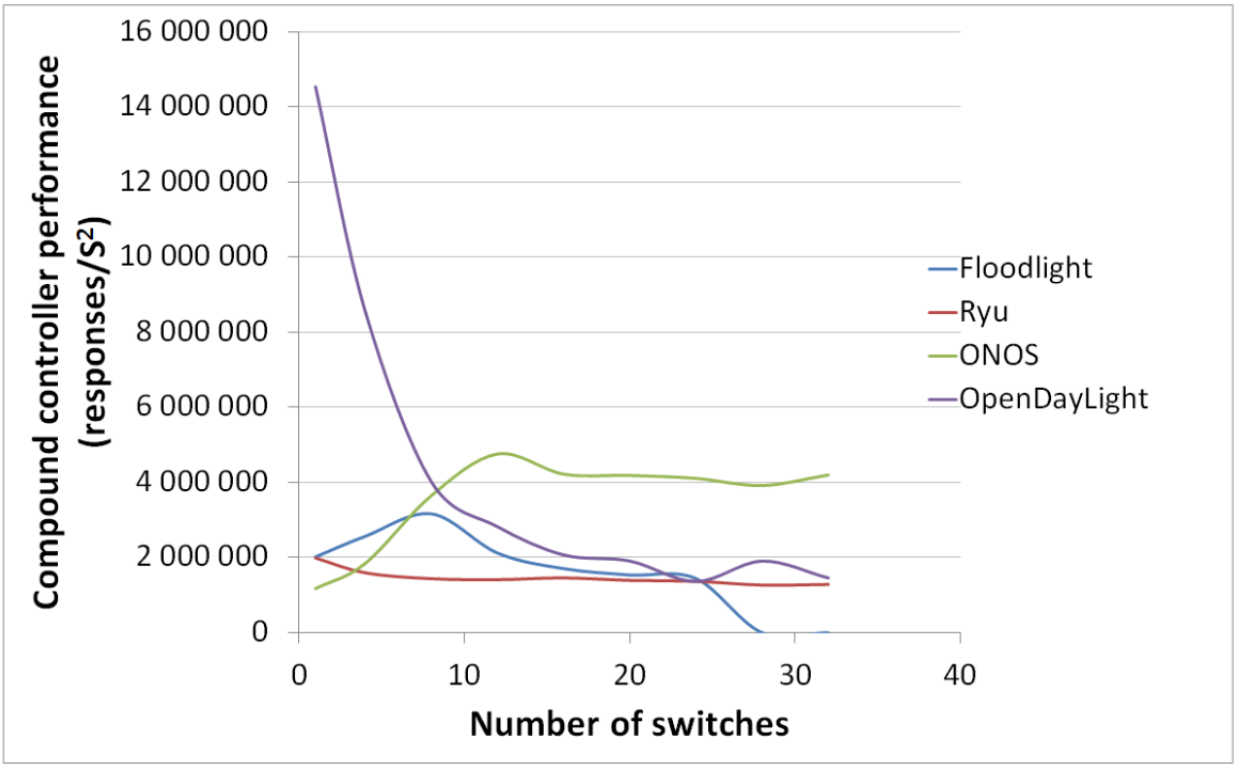


Figure 4.7: Compound controller performance (MACs=1000, threads=4)

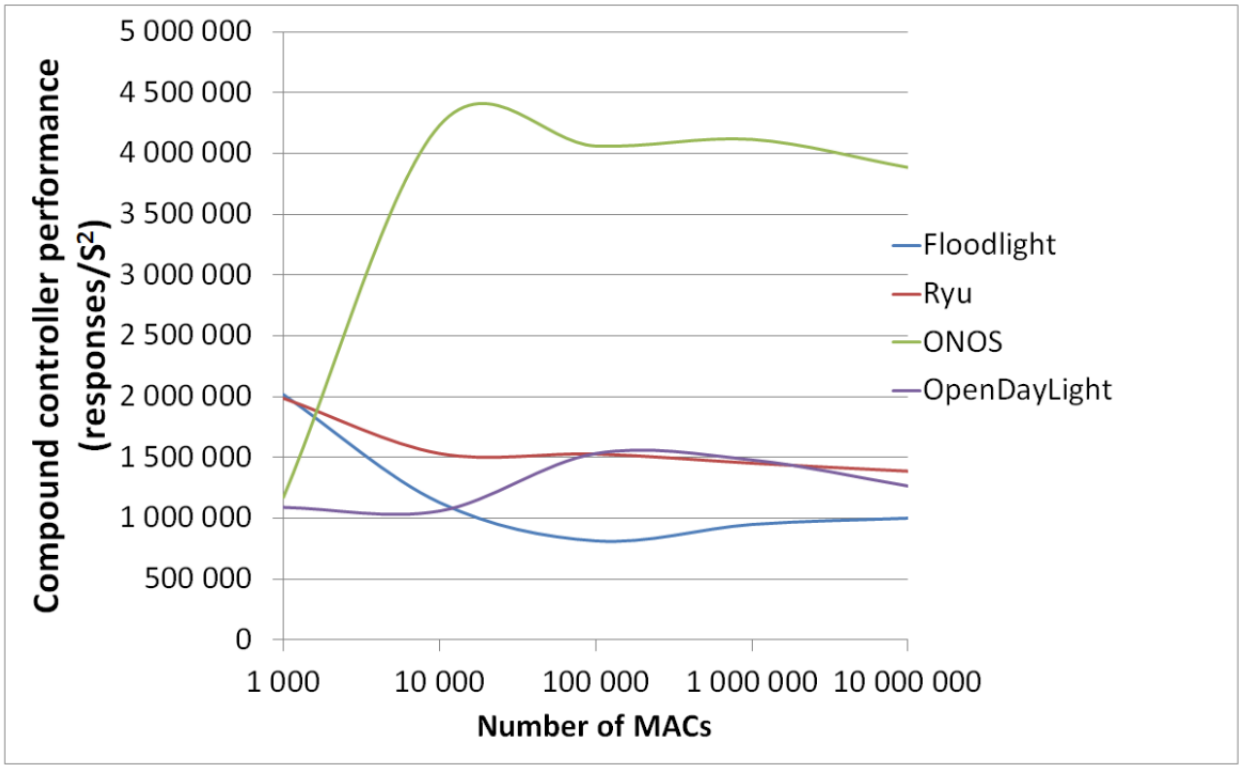


Figure 4.8: Compound controller performance (switches=16, threads=4)

4.6 Chapter Summary

One of the most important design question in controller placement is choosing which controller to deploy. This decision is influenced by a combination of the features offered by a controller as well as its performance. Some of the features typically considered include the protocols supported both on southbound and northbound interfaces, documentation, architecture, security protocols and modularity. In terms of performance, two key metrics are typically prioritized namely, throughput and latency. This chapter presented a performance evaluation of the most popular open source SDN controllers namely, Ryu, Floodlight, ONOS and OpenDayLight, in consideration of latency and throughput.

Our results indicate that ONOS has high throughput and scalability showing that it can process more packet-In messages per second than the other controllers. However Ryu displayed very low controller latency making it ideal for latency-sensitive applications. However given Ryu's poor resource utilization, it is not ideal for large-scale deployments but rather suitable for smaller-scale proof-of-concept demonstrations. OpenDayLight is feature-rich (as discussed in chapter 2 section 2.1.3 but its performance is not outstanding.

From the above observations, our conclusion is that the decision regarding which controller to deploy is entirely dependent on user specification requirements. However, in the context of controller placement for large networks, we recommend ONOS due to its high modularity, support for multi-threading, distributed core and a very high throughput.

Chapter 5

Controller Placement using Emulation Platform

5.1 Introduction

The controller placement results presented in Chapter 3 relied strictly on mathematical modeling. In this chapter, we propose a method for finding optimal and worst locations of SDN controllers using an emulation orchestration platform called Mininet, critical to mimic a real SDN deployment. We use controller-to-node latency (propagation + queuing +processing latency) as a key performance indicator. Our main goal is to match and verify the outcome from our mathematical formulation regarding the best locations to place the controller in a wide area network (WAN). To the best of our knowledge (based on the state-of-the art review), there is currently no work that studies the controller placement problem using emulation. To further optimize network performance, we also consider control plane resiliency, as well as propose a means to alleviate signaling overhead on the control channel.

For the control plane, we implement ONOS (version 1.14) because of its distributed core which improves the robustness of the control plane, by providing backup control in the event of network failure. Moreover ONOS distributed core is self-coordinating and enables load sharing through fragmentation of the data plane. This controller has an advanced east/westbound interface to ensure high inter-controller communication efficiency. Finally, employing a geographically distributed core reduces the node-to-controller latency, thus improving the controller reactivity as perceived by the network nodes. Last but not least our decision to choose ONOS is influenced by the

results from our controller benchmarking experiments in Chapter 4 which confirm ONOS scalability features making it ideal for carrier grade deployments.

The evaluation of the proposed emulation approach is carried out on a model of a local backbone called SANReN (the South African national research and education network) as we did in Chapter 3. It may however be noted that our solution is generic and can be used to optimize any other network.

5.2 Experiment Setup

The experiment setup is as illustrated by Figures 5.1 and 5.2 (captured from Miniedit). Node c0 and c1 are ONOS SDN controller instances running on a dedicated remote machine (with 8 CPUs, 16 GB RAM and 1 TB HDD), and h0-h6 are hosts attached to SDN Open Virtual Switches (OVS 2.9.90) running OpenFlow 1.3. A built-in application for reactive flow instantiation is activated to set the ONOS controller to reactive operational mode. The red dash-dotted lines show connection (over WiFi) between switches and controllers and the blue solid lines are links between the switches. The switch-to-controller communication is assumed to happen out-of-band. Since the links between the switches are known to be fiber, where speed is approximately the speed of light in fiber i.e. about $2 \times 10^8 m/s$, we use the latency formula (equation 5.1) to configure the link properties.

$$propagation\ latency\ (sec) = \frac{distance\ (m)}{speed\ \left(\frac{m}{sec}\right)} \tag{5.1}$$

The distances between nodes are calculated using the harvesine great circle approach and the actual GPS coordinates of the nodes. The dataset of the SANReN topology was downloaded from the Internet Topology Zoo [22] (a database of WANs published by network operators).

The data plane emulated on Mininet is running on a separate machine (with 8 CPUs, 16 GB RAM and 1 TB HDD). Each switch in the data plane has a unique datapath ID (DPID). The connection between the control plane and data plane is via port 6633 of the controller over a WiFi router (see Figure 5.3). Since the control plane and data plane are hosted on separate machines, the WiFi router is used to connect the host machines. The control link parameters are configured using the Linux Traffic Control (TC) utility

under the assumption that the optimal controller placement is co-located with one of the switches. The programming language used to develop the software is Python 2.7.14.

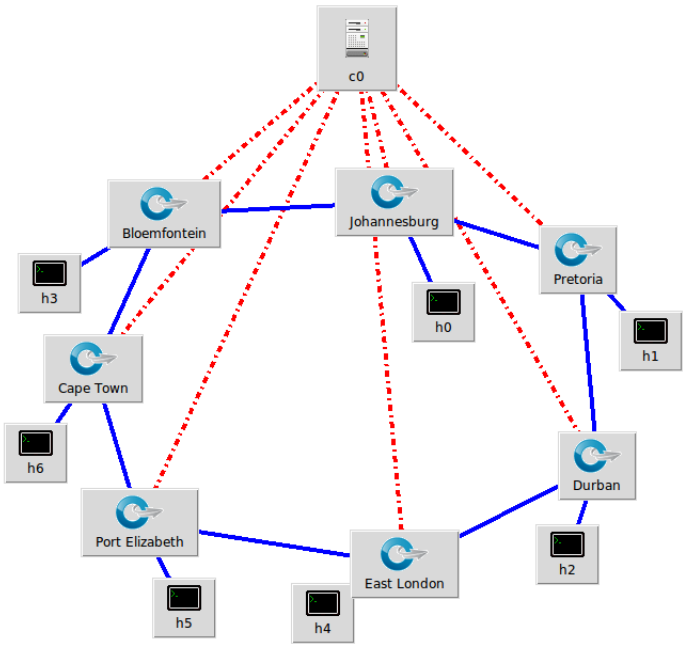


Figure 5.1: Experiment setup with one ONOS controller

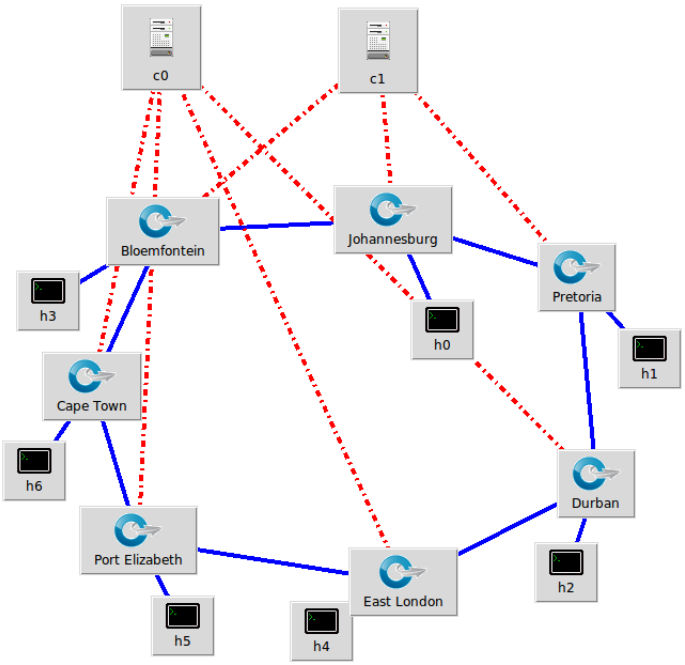


Figure 5.2: Experiment setup with two self-coordinating ONOS controllers

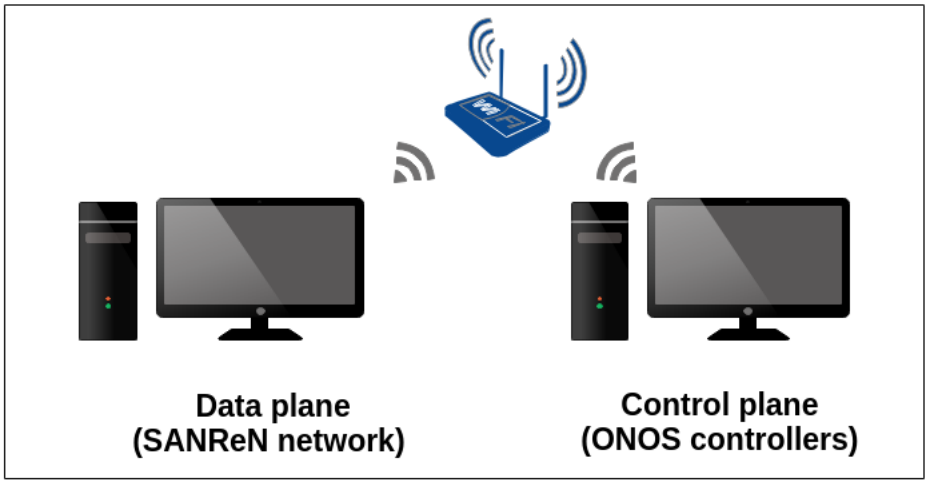


Figure 5.3: Experiment network setup

5.3 Methodology

This work constitutes two independent experiments. The first experiment is carried out with the intention to address the controller placement problem leveraging emulation. The second experiment presents different approaches through which signaling overhead on the control channel can be reduced, in consideration of control plane resiliency.

5.3.1 Controller placement

When confronted with the question of where to optimally place SDN controllers given a network topology, the first step to take is to determine how many controllers to place for a given topology. The answer to this question is presented in Chapter 3, where two controllers are recommended as the most optimal number to deploy in SANReN. Therefore, the number of controllers deployed in our experiment is two as per Figure 5.2. However we also analyze a scenario where only one controller is used as illustrated in Figure 5.1. This is because, according to [68], one controller often suffices to meet existing flow setup time requirements (though certainly not resiliency requirements).

On example of one controller case, Figure 5.4 summarizes our approach in a flow chart (where n is the total potential controller placement locations, i.e. the total number of nodes in a given topology). For the SANReN network, n is 7, meaning there are a total of 7 potential controller placement locations in the network.

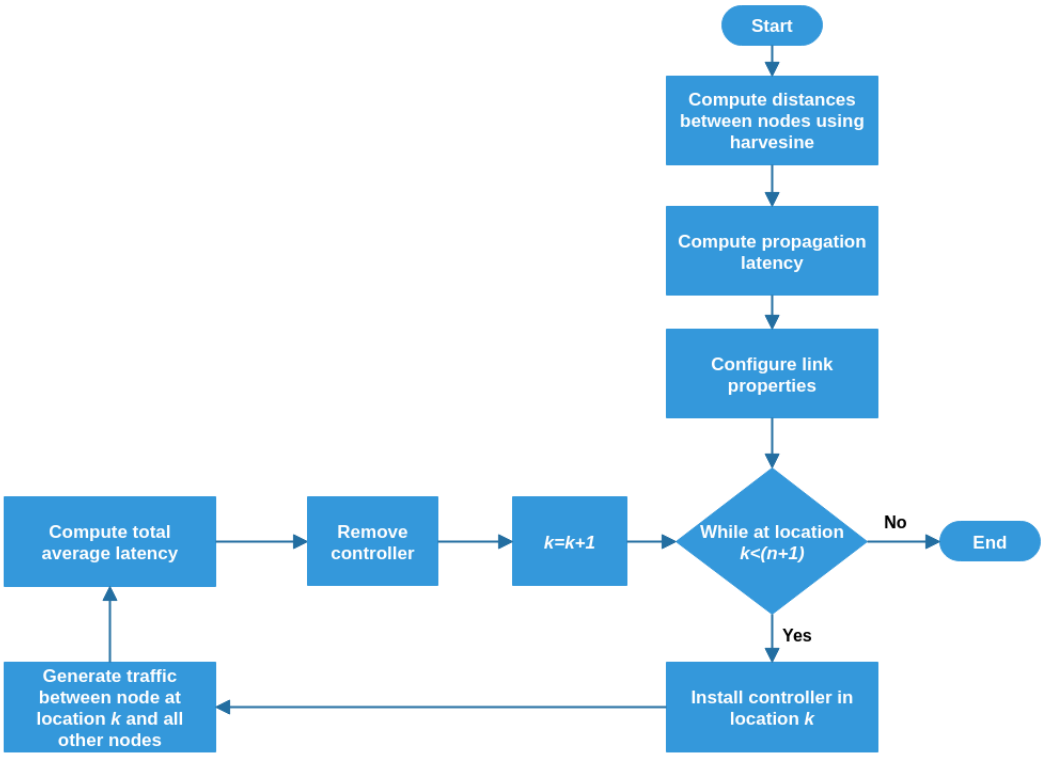


Figure 5.4: Flow chart of proposed method for one controller[109].

The following procedure (outlined in Figure 5.4) is used for each node to determine average latency: To find optimal controller locations, first we install the ONOS controller in the same geographic location as the first OpenFlow switch node (using the harvesine great circle approach and the Linux TC utility). The next step is to trigger a packet-In message to the controller. This is done by generating traffic flows between all pairs, i.e. between this node and all other nodes in the SANReN topology. To do this we generate a ICMP packet using the ping utility for each pair. This is followed by computation of the ICMP pinging results to obtain the total average latency (round-trip time) from the node to all other nodes in the network. This step is repeated for all nodes in the SANReN topology. To ensure valid and reliable results, we repeat the above procedure several times under a soft idle timeout for the controller entry of 5 seconds (the soft idle timeout defines the expiry time of a controller flow rule when there is no flow activity) and compute the average results. The soft idle timeout is set to ensure generation of control traffic upon pinging reiterations.

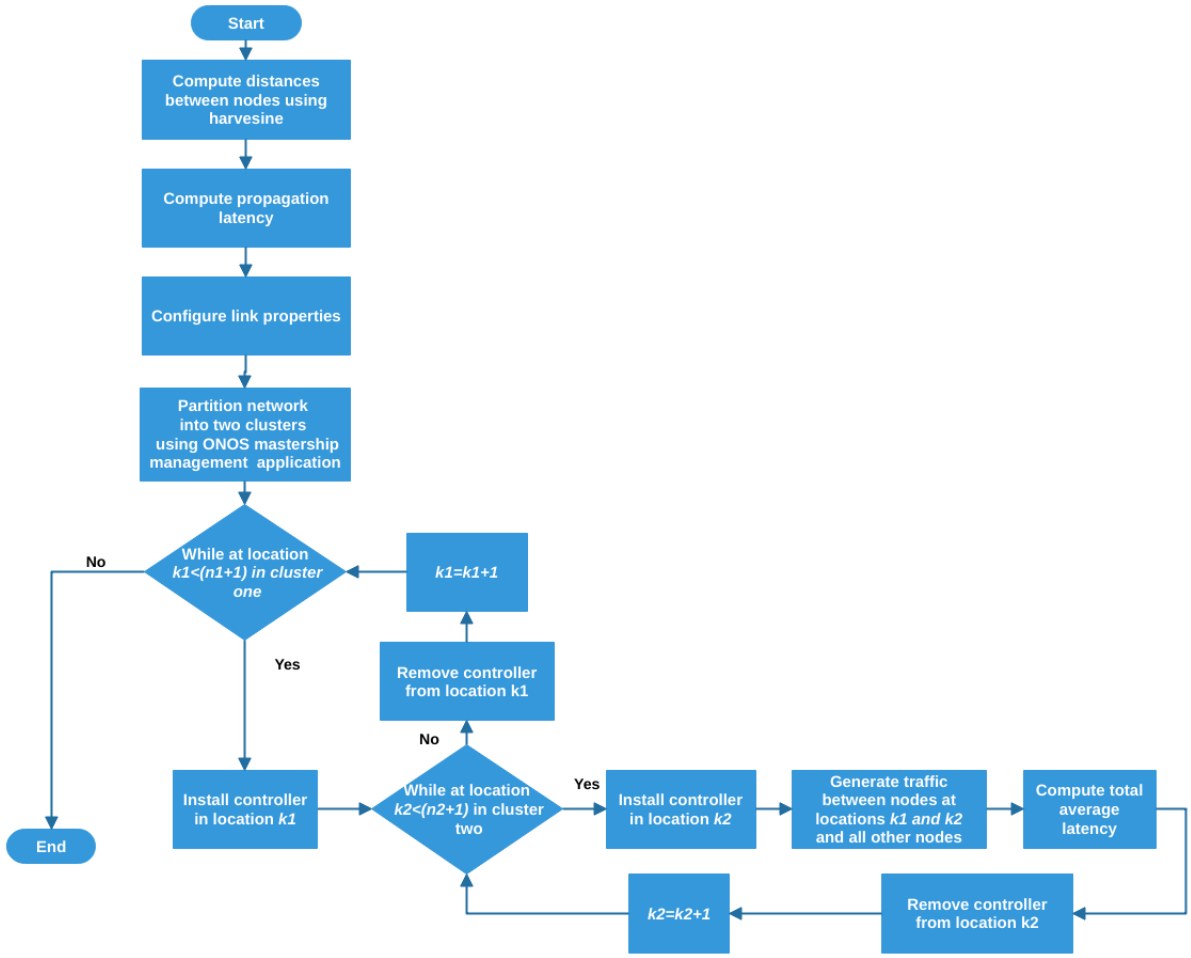


Figure 5.5: Flow chart of proposed method for two controllers.

For the case of two controllers (see Figure 5.5), the network is partitioned into two smaller administrative domains, namely cluster one and cluster two, each supervised by a dedicated ONOS instance. The parameters $n1$ and $n2$ denote the total number of switches in cluster one and two, respectively.

After executing the mastership module, the partition results are as follows: The first ONOS instance ($c1$) is assigned three switch nodes in region Pretoria, Bloemfontein and Durban, while the other ONOS instance constitutes switches located in Johannesburg, Cape Town, East London and Port Elizabeth. In order to optimize the placement of these two controllers, an exhaustive search is carried out by iterating through all possible combinations (within the limits defined by each controller domain). In other words, $c1$ is placed at different regions within its domain. For each placement of $c1$, $c2$ is then placed at different regions within its domain. For each set of placement, the average latency is computed following the same procedure described above.

5.3.2 Control plane overhead and failover

The centralized control scheme SDN adopts puts the control channel at risk of incurring very high signaling overhead generated during data-plane monitoring (e.g. Stats-Request and Stats-Reply) and reactive flow instantiation (such as packet-In, packet-Out and Flow-Removed). In order to manage this rapid influx of traffic on the control channel, the following procedure is used: First we configure a cluster of two ONOS instances each managing a segment of the network. The cluster is configured using the REST API of each separate ONOS. Upon data plane instantiation, the switch-to-controller placement (in terms of the number of switches per cluster) is imbalanced. This is because switch-to-controller placement is based solely on best effort (meaning the controller that completes the handshake with the switch first, gets mastership of the switch). By partitioning the data plane into two clusters, the traffic induced by data plane monitoring (code-named polling) is reduced. Specifically, after clustering, the controller sends and receives monitoring data from just a fraction of data plane nodes. To balance the switch-to-controller placement, we activate the ONOS mastership management module. This results in a more balanced monitoring load which we expect to further decrease control plane overhead.

To quantify the impact of switch-to-controller placement, we generate variable traffic between two virtual hosts (the client connected to Johannesburg and the server connected to Cape Town), a distance of 1399 km from each other. This is carried out using the Distributed Internet Traffic Generator (D-ITG) tool. The transport protocol is set to UDP and the number of packets per second is varied from 50 000 to 200 000 in increments of 50 000. The packet size is set to 512 bytes. The link bandwidth was kept at 10 Mb/s. The duration for the generation process is set to 5 minutes. The key performance indicators are delay, jitter and packet loss all monitored at the server end. This procedure is carried out for two scenarios: 1) when the switch-to-controller placement is imbalanced (switch-to-controller assignment is two and five switches for controller one and two respectively) and 2) for the scenario where switch-to-controller placement is balanced (switch-to-controller assignment is three and four for controller one and two respectively).

In addition to switch-to-controller balancing, the control plane has several tuneable parameters in the control plane, such as polling frequency and soft idle timeout [122]. Polling frequency is a parameter that specifies how frequently statistic requests are sent to the data plane. Soft idle timeout specifies the total time an inactive flow entry is stored in the flow tables before deletion. Tuning these parameters impacts control plane

overhead. In other words, increasing polling frequency is likely to decrease the control plane overhead (of course at the expense of data plane protection and restoration) while increasing the soft idle timeout results in more flow rules in the flow tables and reduces control plane overhead (at the expense of switch resource (e.g. memory and storage) utilization).

To determine how the soft idle timeout affects control plane overhead, we gradually increase the soft idle timeout and polling frequency (from 5 s to 40 s in increments of 5 s) and measure the number of packets (i.e. Packet-In, Packet-Out, Flow-Mod, Stats-Request and Stats-Reply). In order to evoke control traffic we generate 200 000 packets between two hosts (one connected to the node in Johannesburg and the other connected to a node in Cape Town). The duration, packet size and bandwidth are the same as for the switch-to-controller placement experiment. This experiment leveraged the results from the controller placement experiment (for the case when two control instances are deployed). In other words, two control instances were deployed at optimal locations to minimize propagation latency. Additionally, the ONOS mastership management module was activated to balance the switch-to-controller placement.

Failover is evaluated by shutting down one controller in the cluster and calling the "pingall" function. If no packet loss is observed, then it means all hosts can reach each other and switch reassignment to the active controller was successful. We also take note of the time it takes for controller to take mastership of the "controller-less" switches.

5.4 Results and Discussion

This section presents the results obtained from following the procedures described above. The raw data is as shown in Appendix A Tables A1, A2 and A3.

5.4.1 Controller placement

Figures 5.6 and 5.7 present the results obtained from our analysis of the SANReN network. As per Figure 5.6, our results show that the optimum controller location when one controller is deployed is Cape Town since this node has the lowest average latency (L_{avg}=88.78 ms). Similarly, the worst location to place the controller when one controller is deployed is Bloemfontein since this location yields the highest average latency

(L_{avg}=164.4 ms).

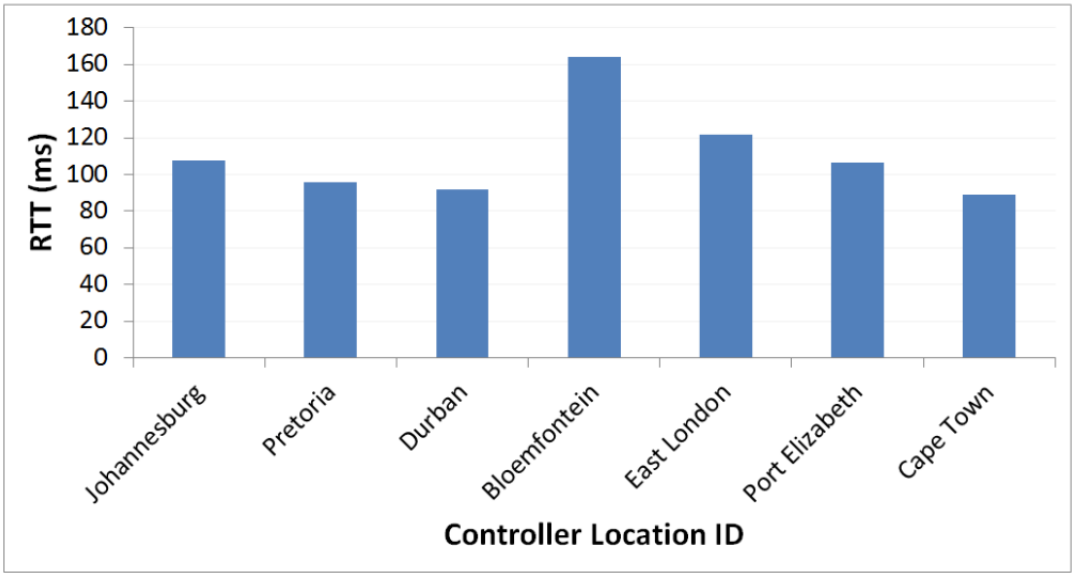


Figure 5.6: Total average latency for the ONOS controller without clustering

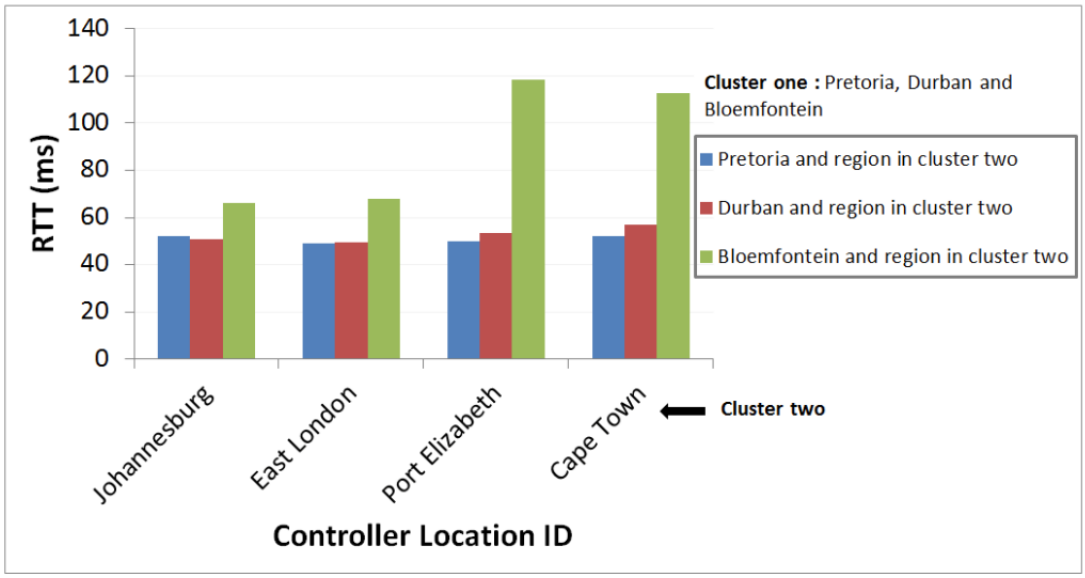


Figure 5.7: Total average latency for ONOS controller when network is partitioned into two clusters

Figure 5.7 presents the results obtained when two controllers are deployed. These results are intepreted as follows: the blue bars indicate a scenario where one controller is placed in Pretoria (a region belonging to cluster one as described in section 5.3.1), while the other controller's location is iterated between Johannesburg, East London, Port Elizabet and Cape Town (regions belonging to cluster two). Similarly, the red and green bars indicate controller placement in Durban and Bloemfontein (regions belonging to cluster one) while the other controller is placed in all regions within cluster two. Our results shows that when two controllers are deployed and the mastership

management module is activated, the optimum controller locations are Pretoria for cluster one and East London for cluster two, with L_{avg}=48.9 ms. The worst locations are Bloemfontein and Port Elizabeth for cluster one and cluster two respectively, with L_{avg}=118.4 ms. Our results coincide with the results from our mathematical formulation in Chapter 3, section 3.4.2.

5.4.2 Control plane overhead and failover

The outcome of our failover tests was positive in that all nodes could reach each other regardless of the failed control node. This means that the switch nodes under the supervision of the failed controller were automatically reassigned to the active controller in the other cluster. The reassignment took approximately 0.5 seconds.

Figure 5.8 and 5.9 depict the results we obtained both before and after switch-to-controller placement balancing. As expected (see Figure 5.8), the average delay is in overall lower after switch-to-controller placement balancing compared to the case of imbalance. We believe this is primarily because, after balancing the switch-to-controller placement, data-plane monitoring traffic is fairly divided between the controller nodes thus improving overall network performance.

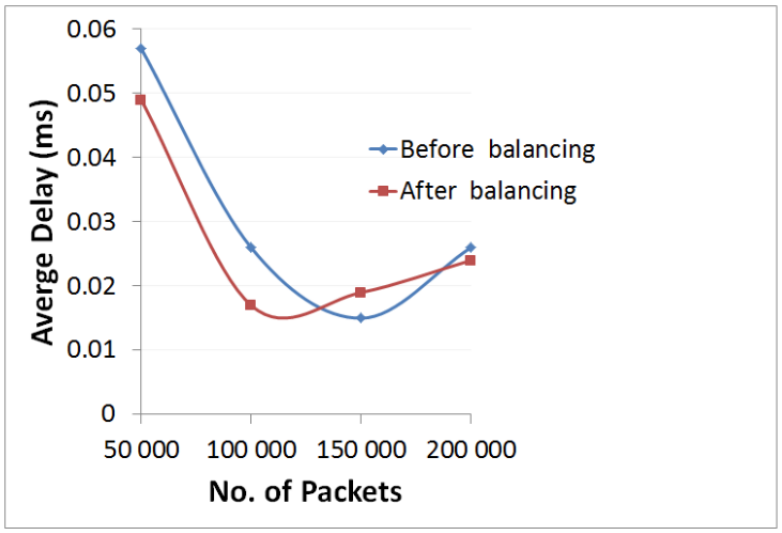


Figure 5.8: Average latency

The decline in average delay (both before and after switch-to-controller balancing) is a result of an increased matching probability of preserved flow rules with newly arriving packets, which reduces the number of packet-In messages to the controller, resulting in a reduction in network delay. Similar results are observed with regards to network jitter (as shown in Figure 5.9). Last but certainly not least, when the number

of packets is increased to 150 000 and 200 000, we observe a percentage packet drop of 0.19% and 0.53% (respectively) before switch-to-controller placement balancing, and 0.14% and 0.07% (respectively) after switch-to-controller placement balancing.

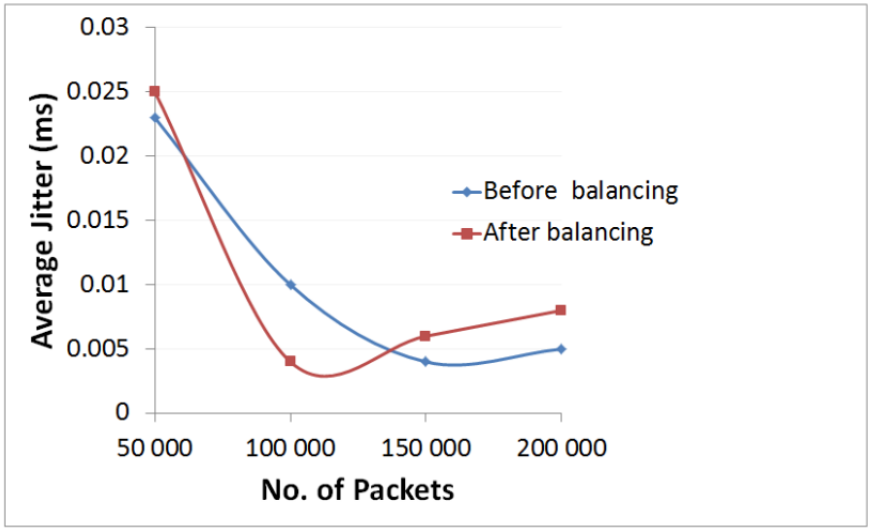


Figure 5.9: Average jitter

Figure 5.10 depicts the impact of tuning the soft idle timeout and polling frequency on control plane overhead. The results indicate that, increasing the polling frequency and soft idle timeout decreases the number of control packets generated during reactive flow instantiation. However from 20 seconds forward, the number of control packets remains constant. Therefore, we can conclude that configuring the polling interval and idle timeout to 20 seconds would be the ideal choice to reduce overall control channel load. However, it is important to note that increasing the soft idle timeout has a huge impact on data plane resource utilization. In other words, with a larger soft idle timeout, the data plane resource utilization (specifically RAM) also increases. Unlike hardware OpenFlow switches that use TCAM (a shorter version of ternary content-addressable memory) [123], Open Virtual Switches present a restriction in terms of memory utilization. Thus there is a need to develop a mechanism to restrict data plane memory utilization around a certain threshold while maintaining a low control plane load. There is currently an ONOS application called control plane management (CPMan) [124] used for collecting statistics regarding system metrics such as CPU, RAM, disk I/O and network I/O and control messages such as Packet-In, Packet-Out, Stats-Request etc. Setting the soft idle timeout to 10 seconds yields an acceptable control plane overhead and potentially a lower switch memory utilization, thus we recommend 10 seconds as the ideal timeout for the SANReN network.

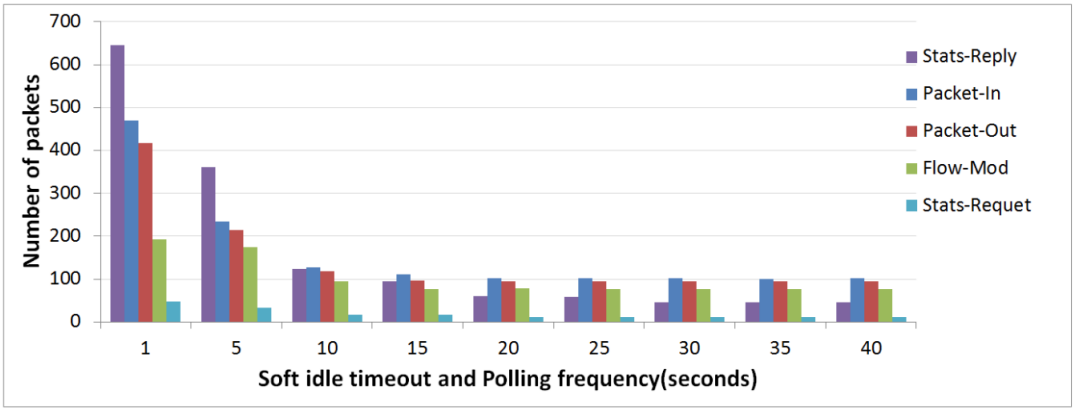


Figure 5.10: Impact of soft idle timeout on control plane overhead

5.5 Chapter Summary

This chapter proposed a new method for determining the best locations to place SDN controllers, which takes into account average latency (propagation, queuing + processing latencies), resiliency and control plane overhead metrics. Instead of only relying on mathematical modeling, this approach uses emulation to mimic a real SDN network deployment. To ensure load sharing and resiliency we use the ONOS SDN controller due to its inherent self-coordinating distributed core. We use the SANReN backbone as a case study for the experiments. Our emulation results show that running a single controller yields high reaction times as some switches are located too far away from the controller. Moreover, running a single controller is not enough to meet resiliency requirements. When the number of controllers was increased to two, the reaction time was reduced considerably since the network was subdivided into two administrative domains. Moreover, the two controllers worked collaboratively to alleviate control overhead and ensure resiliency in the network. Leveraging our controller placement results as well as balancing the switch-to-controller placement, we also investigated the impact of soft idle timeout and polling frequency on control plane overhead. Our finding suggested that a sufficiently large soft idle timeout and polling frequency reduces the overall control plane overhead.

We believe that our method and analysis would be beneficial for operators and service providers, not only during the initial design, but also during the incremental design of the SDN-enabled networks. The algorithm presented in this chapter can be applied to an emerging market use case where SDN adoption and deployment is still in embryonic stage. However, our algorithm is generic enough to solve larger network instances i.e. topologies of different sizes with various configurations.

5.6 Source Codes

The source codes for the proposed solution have been made publicly available on Github, a world's leading code repository. The source codes can be downloaded from this link: https://github.com/Lusani/SDN-Controller-Placement

Chapter 6

Summary and Conclusions

SDN paradigm has emerged as a potential candidate to address the ICT inequality challenge in emerging markets. SDN decouples control and data plane thereby simplifying network management and configuration, reducing network complexity and accelerating innovation. This is indispensable to achieve a stronger telecom infrastructure which is pivotal to economic growth. At the heart of SDN is the controller which oversees orchestration of resources based on its global view of the network's current utilizations. It is critical that this controller is placed in a manner that optimizes network performance. This design choice is commonly known as the controller placement problem and should be addressed during SDN rollout planning. The controller placement problem aims to answer the question regarding the number of controllers to deploy and their best locations in an SDN-enabled network.

This study presented a framework that can be used to address the controller placement problem using the emerging market as a case study. Using graph modeling, two "unsupervised" machine learning algorithms namely, Silhouette and Gap Statistic algorithms were applied to optimize the number of controllers to deploy in a given topology. Given the fact that network operators are more concerned about the cost associated with network deployment, this study also takes into consideration the tradeoff between cost of installing a new SDN controller and performance. This is necessary to facilitate decision making regarding the number of controllers to deploy based on performance requirements and cost constraints. To determine the optimal locations to install the controllers, a classical algorithm called PAM was used. The applied algorithms are exhaustive making them ideal for static controller placement with minimal to no time constraints. The algorithms utilized in this work are more accurate than heuristic algorithms. These algorithms (i.e. Silhouette, Gap Statistic and PAM) were applied on

the SANReN topology, but are generic enough to test other topologies of various sizes and configurations. From our SANReN case study, we discovered that 2 controllers are ideal to achieve low latency at a reasonable cost of controller deployment. In terms of the best locations to place these controllers, our results recommend Pretoria and East London as the optimal locations that minimize propagation latency. This solution was based on mathematical modeling.

In order to mimic a real SDN deployment and verify the outcome from our mathematical model, we used an emulation to study the controller placement problem. To achieve this, we first had to choose the controller to use in our analysis. This decision was influenced by the results from our controller benchmarking experiments and feature-based comparison featuring four popular controllers namely, Ryu, Floodlight, ONOS and OpenDayLight. The benchmarking experiments were carried out using a tool called Cbench, used to stress the controller by pushing high control traffic volume on its southbound interface. Based on the outcome of these analysis, a decision was made to implement a carrier grade SDN controller called ONOS, due to its scalabilty, distributed core and maturity. To optimize ONOS controller placement, an emulation platform called Mininet was used. The results from this experiment recommended Pretoria and East London as the ideal locations to place the controllers, matching the results from our mathematical modeling.

To further optimize controller placement, we also considered resiliency and switch-to-controller placement. Resiliency was achieved by leveraging ONOS' "form-cluster" module with restoration time measured to be within 0.5 s. To balance the switch to controller assignment, the ONOS load balancing application was activated. Leveraging the emulation controller placement results as well as balancing the switch-to-controller placement, we also investigated the impact of soft idle timeout and polling frequency on control plane overhead. Our finding suggested that a sufficiently large soft idle timeout and polling frequency of 10 seconds or more reduces the overall control plane overhead.

The controller placement is topology dependent and the results presented in this study are specific to the SANReN topology. However, our solution is topology-agnostic making it possible to test other topologies through readily.

This work can be extended to solve other placement problems such as (i) edge node (mobile edge node, fog node and cloudlet) placement which appears in the context of edge computing, (ii) hypervisor placement which appears in the context of network slicing and (iii) base-band processing unit (BBU) placement to enable the cloud RAN use case. All these technologies are integral to the envisaged 5G network and require

optimization which can partly be achieved through proper placement.

6.1 Summary of Contributions

The main contributions of this work are as follows:

- This study provides a thorough state of the art review of SDN controller placement research and various solution implementations. This includes reviewing the merits and faults of various algorithms proposed to address the controller placement. This is necessary to identify improvement opportunities within this research space.
- Although SDN is anticipated to revolutionize network operations and economic growth, the idea of centralizing network control has been received with a lot of skepticism, especially in emerging markets where SDN adoption is still in its infancy. Most operators are not convinced that a centralized controller will, at the very least, match up the performance of legacy networks. This study aims to mitigate these concerns by qualitatively demonstrating the performance capabilities of centralizing network control. We demonstrate how software defined network planning can be improved through proper SDN controller placement. Operators can consume some of the techniques proposed in this study to facilitate transition to SDN.
- This study attempts to create a practical guideline on controller placement optimization by approaching the SDN controller placement problem from all its possible facets. Controller placement features various competing objectives that (depending on user-defined requirements and constraints) must be considered for realistic deployments. This study takes into account objectives namely, latency (controller-to-node latency), reliability and control plane overhead to address the controller placement problem.
- During SDN planning, a decision must be made regarding the controller to deploy. To date, a plethora of SDN controllers have been proposed within the research community and industry, making it difficult to choose a suitable controller. This study provides a decision making guideline regarding a controller to deploy, given a set of user requirements and constraints.
- This study is tailored for emerging markets (particularly African economies) faced with the challenge of ICT inequality. Therefore this study features a South African research and education network (SANReN) to demonstrate the viability of migrating to SDN.

6.2 Future Research Work

In future we intend to extend our work to address dynamic controller placement which is necessary to meet 5G requirements such as ultra reliable low latency communications achievable through dynamic placement of the mobile edge computing node.We also plan to evaluate the security aspect of SDN controllers. This is motivated by the fact that centralizing the network control intelligence presents a single point of attack/failure. Last but not least, we intend to develop a dynamic control traffic load balancing application based on current switch resource (RAM, CPU, and storage) utilization.

The assumption that the bandwidth for all connection links is constant (see Chapter 3 section 3.1.1) is not valid for the actual SANReN network. This assumption was made to simplify the mathematical model formulation. In future, a more complicated scenario with different link bandwidths will be considered.

In Chapter 5, the traffic load was set to 512 bytes which does not reflect the actual traffic exchanges between SANReN nodes. As future work, a characterisation of the actual traffic profiles combined with a rerun of the evaluation in Chapter 5 will be considered.

Bibliography

[1] M. D. Chinn and R. W. Fairlie, "ICT use in the developing world: an analysis of differences in computer and internet penetration," *Review of International Economics*, vol. 18, no. 1, pp. 153–167, 2010.

[2] Cisco, "Cisco visual networking index: Forecast and methodology, 2016–2021," Tech. Rep., 2017.

[3] M. Bourreau and T. Valletti, "Enabling digital financial inclusion through improvements in competition and interoperability: What works and what doesn't," *CGD Policy Paper*, vol. 65, pp. 1–30, 2015.

[4] "Why SDN or NFV now?" accessed June 2018). [Online]. Available: https://www.sdxcentral.com/sdn/definitions/why-sdn-software-defined-networking-or-nfv-network-functions-virtualization-now/

[5] W. E. Forum, "Internet for all: A framework for accelerating internet access and adoption," Tech. Rep., 2016.

[6] G. Cruz, "Enabling rural coverage regulatory and policy recommendations to foster mobile broadband coverage in developing countries," Tech. Rep., 2018.

[7] "Enabling rural coverage, regulatory and policy recommendations to foster mobile broadband coverage in developing countries," accessed January 2018. [Online]. Available: https://www.gsma.com

[8] S. A. Government, "South africa national development plan," accessed December 2017. [Online]. Available: https://www.gov.za/issues/national-development-plan-2030

[9] D. Kreutz, F. M. 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.

[10] S. Tomovic, M. Pejanovic-Djurisic, and I. Radusinovic, "SDN based mobile networks: Concepts and benefits," *Wireless Personal Communications*, vol. 78, no. 3, pp. 1629–1644, 2014.

[11] E. Hernandez-Valencia, S. Izzo, and B. Polonsky, "How will NFV/SDN transform service provider opex?" *IEEE Network*, vol. 29, no. 3, pp. 60–67, 2015.

[12] "An introduction, benefits, enablers, challenges & call for action, ETSI, oct. 2012."

[13] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[14] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," Tech. Rep., 2015.

[15] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latré, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, 2016.

[16] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5G and future mobile networks: mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, 2017.

[17] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5g network slice broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.

[18] L. Mamushiane, A. A. Lysko, and S. Dlamini, "SDN-enabled infrastructure sharing in emerging markets: Capex/opex savings overview and quantification," in *2018 IST-Africa Week Conference (IST-AFRICA)*. IEEE, 2018, pp. Page–1.

[19] L. Mamushiane and S. Dlamini, "Leveraging SDN/NFV as key stepping stones to the 5G era in emerging markets," in *Wireless Summit (GWS), 2017 Global*. IEEE, 2017, pp. 23–27.

[20] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815*, 2015.

[21] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: a survey, use cases, and future directions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[22] B. Liang, *Mobile edge computing*. Cambridge University Press, 2017.

[23] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[24] E. Ahmed and M. H. Rehmani, "Mobile edge computing: opportunities, solutions, and challenges," 2017.

[25] J. Mwangama and N. Ventura, "Accelerated virtual switching support of 5G NFV-based mobile networks," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on*. IEEE, 2017, pp. 1–7.

[26] "openepc," 2009(accessed October 2018). [Online]. Available: https://www.openepc.com/

[27] F. Fokus, "Open5gcore," accessed October 2018. [Online]. Available: https://www.open5gcore.org/

[28] "Onap: Open network automation platform," accessed October 2018. [Online]. Available: https://www.onap.org/

[29] F. Fokus, "Open baton: An extensible and customizable nfv mano-compliant framework," accessed October 2018. [Online]. Available: https://openbaton.github.io

[30] "Opendaylight," accessed October 2018. [Online]. Available: https://www.opendaylight.org/

[31] "Onos project," accessed October 2018. [Online]. Available: https://onosproject.org/

[32] H. Li, G. Shou, Y. Hu, and Z. Guo, "Mobile edge computing: progress and challenges," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2016 4th IEEE International Conference on*. IEEE, 2016, pp. 83–84.

[33] "cbench." [Online]. Available: https://github.com/mininet/oflops/tree/master/cbench

[34] T. U. of Adelaide, "The internet topology zoo," accessed October 2018. [Online]. Available: http://www.topology-zoo.org/dataset.html

[35] 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.

[36] J. Guillermo, "Multilayer data connectivity orchestration: Exploring the cengn proof of concept," 2016, accessed December 2017. [Online]. Available: https://www.cengn.ca

[37] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[38] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN," *International Journal of Network Management*, vol. 28, no. 3, p. e2018, 2018.

[39] ONF, "Onf, software-defined networking: The new norm for networks," 2012, accessed December 2017. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[40] "SDN architecture overview," 2014, accessed December 2017. [Online]. Available: https://www.opennetworking.org

[41] T. Čejka and R. Krejčí, "Configuration of open vswitch using of-config," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 883–888.

[42] S. Wallin and C. Wikström, "Automating network and service configuration using netconf and yang," in *Usenix Large Installation System Administration Conference: 04/12/2011-09/12/2011*. USENIX-The Advanced Computing Systems Association, 2011, pp. 267–279.

[43] M. Brandt, R. Khondoker, R. Marx, and K. Bayarou, "Security analysis of software defined networking protocols—openflow, of-config and ovsdb," in *The 2014 IEEE Fifth International Conference on Communications and Electronics (ICCE 2014), DA NANG, Vietnam*, 2014.

[44] 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.

[45] A. Rodriguez-Natal, M. Portoles-Comeras, V. Ermagan, D. Lewis, D. Farinacci, F. Maino, and A. Cabellos-Aparicio, "Lisp: a southbound SDN protocol?" *IEEE Communications Magazine*, vol. 53, no. 7, pp. 201–207, 2015.

[46] A. Sgambelluri, F. Paolucci, F. Cugini, L. Valcarenghi, and P. Castoldi, "Generalized SDN control for access/metro/core integration in the framework of

the interface to the routing system (i2rs)," in *Globecom Workshops (GC Wkshps), 2013 IEEE*. IEEE, 2013, pp. 1216–1220.

[47] M. Rouse, "Path computation element protocol (pcep)," 2017, accessed December 2017. [Online]. Available: https://whatis.techtarget.com/definition/Path-Computation-Element

[48] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven SDN controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[49] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "SDN and openflow evolution: A standards perspective," *Computer*, vol. 47, no. 11, pp. 22–29, 2014.

[50] K. Bhamre, "Is pcep/bgp-ls-based SDN approach ideal choice for service providers?" 2014, accessed December 2017. [Online]. Available: https://www.ixiacom.com/company/blog/pcepbgp-ls-based-sdn-approach-ideal-choice-service-providers

[51] F. Khan, "Need a quick recipe for SDN in WAN? mix BGP-LS with PCE," 2016, accessed October 2018. [Online]. Available: https://www.telcocloudbridge.com/

[52] D. Samociuk, "Secure communication between openflow switches and controllers," *AFIN 2015*, vol. 39, 2015.

[53] ONF, "Openflow switch specification," 2012, accessed October 2018. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf

[54] ——, *OpenFlow Switch Specification*, 2015, Accessed October 2018. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

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

[56] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, 2016.

[57] "Openstack," accessed October 2018. [Online]. Available: https://www.openstack.org

[58] "Opnfv," accessed October 2018. [Online]. Available: https://www.opnfv.org

[59] "Opendaylight," accessed October 2018. [Online]. Available: https://www.opendaylight.org/view/unimgr:Documentation

[60] accessed October 2018. [Online]. Available: https://www.opendaylight.org/what-we-do/current-release/carbon

[61] MEF, "Service operations specification- lifecycle service orchestration (lso): Reference architecture and framework, page 9,15," Tech. Rep., 2016.

[62] S. Rowshanrad, V. Abdi, and M. Keshtgari, "Performance evaluation of SDN controllers: Floodlight and opendaylight," *IIUM Engineering Journal*, vol. 17, no. 2, pp. 47–57, 2016.

[63] s Rao, "SDN series part four: Ryu,a rich-featured open source SDN controller supported by NTT lab," accessed July 2018. [Online]. Available: https://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/

[64] M. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *2013 Advanced Information Networking and Applications (AINA)*. IEEE, 2013.

[65] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," in *Wireless Days (WD), 2018*. IEEE, 2018, pp. 54–59.

[66] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," *IEEE Communications Letters*, vol. 20, no. 6, pp. 1108–1111, 2016.

[67] "An introduction to computer networks," accessed December 2017. [Online]. Available: intronetworks.cs.luc.edu/1/html/packets.html

[68] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[69] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–7.

[70] "Internet2 network infrastructure topology," accessed January 2018. [Online]. Available: https://www.internet2.edu/media/medialibrary/2018/07/16/Internet2-Network-Infrastructure-Topology-All-legendtitle.pdf

[71] Y.-N. Hu, W.-D. Wang, X.-Y. Gong, X.-R. Que, and S.-D. Cheng, "On the placement of controllers in software-defined networks," *The Journal of China Universities of Posts and Telecommunications*, vol. 19, pp. 92–171, 2012.

[72] "Rocketfuel," accessed March 2018. [Online]. Available: https://github.com/CS236340/RocketFuel

[73] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 672–675.

[74] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[75] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in *Networking Conference, 2014 IFIP*. IEEE, 2014, pp. 1–9.

[76] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 18–25.

[77] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 63–76, 2017.

[78] J.-M. Sanner, Y. Hadjadj-Aoul, M. Ouzzif, and G. Rubino, "An evolutionary controllers' placement algorithm for reliable SDN networks," in *IFIP International Workshop on Management of SDN and NFV Systems (ManSDNNFV'2017)*, 2017.

[79] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha, "Optimal controller placement in software defined networks (SDN) using a non-zero-sum game," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. IEEE, 2014, pp. 1–6.

[80] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE communications letters*, vol. 19, no. 1, pp. 30–33, 2015.

[81] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Communications*, vol. 11, no. 2, pp. 38–54, 2014.

[82] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 115–116.

[83] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[84] A. Ksentini, M. Bagaa, and T. Taleb, "On using SDN in 5G: the controller placement problem," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[85] M. He, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in sdn: Evaluation for dynamic flows," in *IEEE International Conference on Communications (ICC)*, 2017.

[86] P. O. Olukanmi and B. Twala, "Sensitivity analysis of an outlier-aware k-means clustering algorithm," in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech), 2017*. IEEE, 2017, pp. 68–73.

[87] T. M. Kodinariya and P. R. Makwana, "Review on determining number of cluster in k-means clustering," *International Journal*, vol. 1, no. 6, pp. 90–95, 2013.

[88] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411–423, 2001.

[89] A. Bhat, "K-medoids clustering using partitioning around medoids for performing face recognition," *International Journal of Soft Computing, Mathematics and Control*, vol. 3, no. 3, pp. 1–12, 2014.

[90] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The SDN controller placement problem for WAN," in *Communications in China (ICCC), 2014 IEEE/CIC International Conference*. IEEE, 2014, pp. 220–224.

[91] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[92] C. C. Aggarwal and C. K. Reddy, *Data clustering: algorithms and applications*. CRC press, 2013.

[93] J. J. Mwemezi and Y. Huang, "Optimal facility location on spherical surfaces: algorithm and application," *New York Science Journal*, vol. 4, no. 7, pp. 21–28, 2011.

[94] A. Prakhar, "Haversine formula to find distance between two points on a sphere," accessed February 2018. [Online]. Available: https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

[95] L. Mamushiane, A. Lysko, and J. Mwangama, "Optimum placement of SDN controllers in african backbones: SANREN and ZAMREN as a case study," in *Southern Africa Telecommunication Networks and Applications Conference (SATNAC), 2018*, 2-5 September 2018.

[96] M. Mohajer, K.-H. Englmeier, and V. J. Schmid, "A comparison of Gap statistic definitions with and without logarithm function," *arXiv preprint arXiv:1103.4767*, 2011.

[97] N. Damak, "Distance problems in networks—theory and practice," *Classical shortest-path algorithms*, pp. 1–9, 2010.

[98] M. Bindhu and G. Ramesh, "Load balancing and congestion control in software defined networking using the extended johnson algorithm for data centre," *International Journal of Applied Engineering Research*, vol. 10, no. 17, p. 2015, 2015.

[99] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268–281, 2003.

[100] A. Fahim, A. Salem, F. A. Torkey, and M. Ramadan, "An efficient enhanced k-means clustering algorithm," *Journal of Zhejiang University-Science A*, vol. 7, no. 10, pp. 1626–1633, 2006.

[101] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999.

[102] M. Steinbach, G. Karypis, V. Kumar *et al.*, "A comparison of document clustering techniques," in *KDD workshop on text mining*, vol. 400, no. 1. Boston, 2000, pp. 525–526.

[103] Q. Zhang and I. Couloigner, "A new and efficient k-medoid algorithm for spatial clustering," in *International conference on computational science and its applications*. Springer, 2005, pp. 181–189.

[104] L. S. Zhang, M. J. Yang, and D. J. Lei, "An improved PAM clustering algorithm based on initial clustering centers," in *Applied Mechanics and Materials*, vol. 135. Trans Tech Publ, 2012, pp. 244–249.

[105] S. S. Singh and N. Chauhan, "K-means v/s k-medoids: A Comparative Study," in *National Conference on Recent Trends in Engineering & Technology*, vol. 13, 2011.

[106] E. Schubert and P. J. Rousseeuw, "Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms," *arXiv preprint arXiv:1810.05691*, 2018.

[107] "South African National Research Network," accessed October 2018. [Online]. Available: https://www.sanren.ac.za/backbone/

[108] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[109] L. Mamushiane, A. A. Lysko, and J. Mwangama, "Resilient SDN controller placement optimization applied to and emulated on south african national research network (sanren)," in *submitted for Wireless Communications and Networking Conference*. IEEE, 2019.

[110] L. Lovmar, A. Ahlford, M. Jonsson, and A.-C. Syvänen, "Silhouette scores for assessment of SNP genotype clusters," *BMC genomics*, vol. 6, no. 1, p. 35, 2005.

[111] "Project floodlight," accessed October 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[112] "Opendaylight platform overview," 2016, accessed October 2018. [Online]. Available: https//www.opendaylight.org/what-we-do/odl-platform-overview

[113] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks." *HOT-ICE*, vol. 12, pp. 1–6, 2012.

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

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

[116] M. P. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013, pp. 1009–1016.

[117] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[118] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th central & eastern european software engineering conference in russia*. ACM, 2013, p. 1.

[119] "Cbench," accessed April 2018). [Online]. Available: https://github.com/oflops/cbench

[120] S. Asadollahi, B. Goswami, and M. Sameer, "Ryu controller's scalability experiment on software defined networks," in *Current Trends in Advanced Computing (ICCTAC), 2018 IEEE International Conference on*. IEEE, 2018, pp. 1–5.

[121] J. Lam, S.-G. Lee, H.-J. Lee, and Y. Eko Oktian, "Design, implementation, and performance evaluation of identity-based cryptography in ONOS," *International Journal of Network Management*, vol. 28, no. 1, p. e1990, 2018.

[122] J. Li, J.-H. Yoo, and J. W.-K. Hong, "Dynamic control plane management for software-defined networks," *International Journal of Network Management*, vol. 26, no. 2, pp. 111–130, 2016.

[123] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," in *International conference on distributed computing and networking*. Springer, 2013, pp. 439–444.

[124] J. Li, J.-H. Yoo, and J. W.-K. Hong, "CPMan," accessed MAY 2018. [Online]. Available: https://wiki.onosproject.org

Appendices

A Raw Data from Simulations

Table A1 presents the latency (i.e. round-trip time (RTT)) results obtained from our emulation experiments when one ONOS controller is deployed.

Table A1: Total average latency for one controller

Switch ID	Region Name	RTT (ms)
s1	Johannesburg	107.58
s2	Pretoria	95.62
s3	Durban	91.87
s4	Bloemfontein	164.4
s5	East London	121.63
s6	Port Elizabeth	106.68
s7	Cape Town	88.78

Table A2 presents the latency (i.e. round-trip time (RTT)) results obtained from our emulation experiments when two ONOS controllers are deployed.

Table A2: Total average latency for two controller

Switch ID	Region Name	RTT (ms)
s2 & s1	Pretoria & Johannesburg	52.316
s2 & s5	Pretoria & East London	48.933
s2 & s6	Pretoria & Port Elizabeth	49.95
s2 & s7	Pretoria & Cape Town	52.23
s3 & s1	Durban & Johannesburg	51
s3 & s5	Durban & East London	49.62
s3 & s6	Durban & Port Elizabeth	53.62
s3 & s7	Durban & Cape Town	56.87
s4 & s1	Bloemfontein & Johannesburg	66.02
s4 & s5	Bloemfontein & East London	68.13
s4 & s6	Bloemfontein & Port Elizabeth	118.35
s4 & s7	Bloemfontein & Cape Town	112.67

Table A3 presents the results obtained from our experiments regarding the impact of switch-to-controller placement balancing on SDN controller performance. The key performance indicators considered in this experiments are controller average delay, average packet loss and jitter.

Table A3: Effect of switch placement on controller performance

No. of Packets	**Before switch-to-controller placement balancing**			**After switch-to-controller placement balancing**		
	% Packet loss	Average Delay	Average Jitter	% Packet loss	Average Delay	Average Jitter
50 000	0	0.057	0.023	0	0.049	0.025
100 000	0	0.026	0.01	0	0.017	0.004
150 000	0.19	0.015	0.004	0.14	0.019	0.006
200 000	0.53	0.026	0.005	0.07	0.024	0.008

Tables A4 and A5 present the results obtained from our controller benchmarking experiments. Table A4 results were obtained by varying the number of virtual switches under a fixed number of MACs (i.e. Number of MACs = 1000). Table A5 shows results obtained for varying number of MACs whilst the number of switches is fixed at 16 switches.

Table A4: Benchmarking results under varrying number of switches

No. of Switches	Throughput			Latency				Compound Performance		
	Floodlight	Ryu	ONOS	OpenDayLight	Floodlight	Ryu	ONOS	OpenDayLight	Floodlight	Ryu	ONOS	OpenDayLight
1	26051.21	5942.35	30407.51	43082.72	12.90	2.98	25.91	2.96	2018866.40	1988392.25	1173620.06	14534006.69
4	107236.75	5810.87	119699.82	98418.04	41.82	3.64	64.98	11.49	2564167.71	1595608.20	1842116.95	8565874.21
8	178396.00	5634.29	222293.52	63988.61	56.37	3.92	60.92	15.92	3164561.80	1437744.33	3649203.33	4019819.35
12	109641.47	5701.41	274439.70	56219.76	51.71	4.04	57.67	19.92	2120473.75	1412019.51	4758907.09	2822311.11
16	92692.79	5748.85	255122.81	55840.85	54.34	3.94	60.43	26.98	1705658.94	1458762.06	4221577.60	2070045.74
20	87245.83	5590.35	263027.37	53217.50	56.79	4.02	62.83	27.88	1536296.45	1392057.55	4186264.78	1908481.13
24	86658.10	5523.29	257741.71	48140.64	60.16	4.04	62.72	35.31	1440405.85	1367465.78	4109239.13	1363486.36
28	-	5357.17	253676.33	46904.44	60.16	4.22	64.76	24.61	-	1267726.19	3917061.27	1905740.20
32	-	5334.92	257037.10	43420.51	52.64	4.16	61.19	29.78	-	1283719.50	4200312.25	1457954.03

Table A5: Benchmarking results under varrying number of MACs

No. of MACs	**Throughput**				**Latency**				**Compound Performance**			
	Floodlight	Ryu	ONOS	OpenDayLight	Floodlight	Ryu	ONOS	OpenDayLight	Floodlight	Ryu	ONOS	OpenDayLight
1000	26051.21	5942.35	30407.51	32867.56	12.90	2.99	25.91	30.11	2018866.42	1988392.25	1173620.06	1091679.67
10 000	60919.16	6286.60	248668.61	33192.59	53.86	4.10	58.77	31.29	1130969.65	1534099.09	4231100.87	1060803.02
100 000	10521.75	6421.19	256598.40	31677.42	12.90	4.20	63.17	20.65	815799.53	1530483.63	4062212.72	1534037.78
1 000 000	12092.92	6331.94	262218.59	33298.36	12.72	4.35	63.70	22.48	950817.12	1454107.46	4116280.45	1481317.57
100 000 000	12568.29	6129.53	245473.89	29894.65	12.55	4.42	63.17	23.60	1001805.41	1388351.45	3885715.37	1266729.44