Developing Optimal SDN Controller Placement in WAN Networks Based on Clustering Approach

Foziya Duri

A Thesis Submitted to Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Office of Graduate Studies
Adama Science and Technology University

June 23, 2023

Developing Optimal SDN Controller Placement in WAN Networks Based on Clustering Approach

Foziya Duri

PGE/22249/13

Advisor: **Dr.-Ing Frezewd Lemma**

A thesis Submitted to the department of Computer Science and Engineering

School of Electrical and Computing

Office of Graduate Studies
Adama Science and Technology University

June 23, 2023

DECLARATION

I declare that this thesis/dissertation entitled "**Developing Optimal SDN Controller Placement in WAN Networks Based on Clustering Approach"** is my own work and has not been submitted to any university for similar purpose. The references used in this proposal are duly recognized by proper citations.

_____	_____	_____
Name of student	Signature	Date

CERTIFICATE

I/we, the advisor(s) of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Developing Optimal SDN Controller Placement in WAN Networks Based on Clustering Approach** "prepared under my guidance by Foziya Duri Sani submitted in partial fulfillment of the requirements for the degree of Masters of Sciencein Computer Science and Engineering. Therefore, I recommend the submission of the revisedversion of the thesis to the department following the applicable procedures.

Major Advisor/Supervisor	Signature	Date
Co-advisor/Co-supervisor	Signature	Date

APPROVAL PAGE FOR THESIS

I/we hereby certify that the recommendation and suggestion given by the proposal review Committee are appropriately incorporated into the final thesis/dissertation entitled "**Developing Optimal SDN Controller Placement in WAN Networks Based on Clustering Approach**" by Foziya Duri Sani.

Major Advisor/Supervisor	Signature	Date
Co-advisor/ Co-supervisor	Signature	Date

We, the undersigned, members of the Board of Reviewers of the proposal open defense by Foziya Duri Sani have read and evaluated the Thesis/dissertation entitled "**Optimal SDN Controller Placement in WAN Networks Based on Clustering Approach**" and assessed the understanding of the candidate about the proposed research. This is, therefore, to certify that the Thesis/dissertation proposal is accepted and we recommend the implementation of the Proposal.

Chairperson	Signature	Date
Internal examiner	Signature	Date
External examiner	Signature	Date

Final approval and acceptance of the thesis/dissertation proposal is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

Department Head	Signature	Date
School Dean	Signature	Date
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

I would like to thank the almighty God (Allah) for providing me everything I need in my life. Next, my family because it owes all to them. Many Thanks!

I am grateful to my special friends, who have provided me through moral and emotional support in my life. A very special gratitude goes out to all down to my advisor Dr.-Ing Frezewd Lemma for the encouragement and support by giving me good comments and suggestions. I would not forget to remember his encouragement and more over for their timely support and guidance until the completion of the thesis. I am also grateful to my Lecture: Dr. Ketema Adere, for their motivating support.

Thanks for all your encouragement!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATION

LIST OF ABBREVIATIONS AND ACRONYMS

SDN	Software-Defined Network
ICMP	Internet Control Protocol
TC	Traffic Control
QOS	Quality of Service
WAN	Wide Area Network
SANRIN	South African National Research Institution Network
SDN-CP	Software-Defined Network Controller Placement
ML	Machine Learning
GPU	Graphic Processing Unit

ABSTRACT

This research looks into controller placement as a key factor in software-defined wide area network (WAN) deployment. We are trying to figure out the best number of controllers and where to place them in order to meet certain performance objectives, such as delay and cost of deployment. We are using ML algorithms and mathematical approaches to find the optimal placement of controllers to get the most out of the network. We set up two metrics to measure the network: average propagation latency between controllers and switches, and cost of deployment for each controller. Simulations were run on a real-world network called SANReN to test the minimum number of controllers and appropriate placement in order to improve the network's performance without needing expensive and time-consuming physical testing. We applied k means clustering, and compared our approach to previously used methods on the same dataset. Our results from SANReN showed that deploying the k means clustering method drastically decreases computational time used for controller deployment and reduces the chances of failure of controller based on delay of communication as it is one of controller failure reason. These findings will be helpful for network operators to deploy the right number of controllers and their locations for optimal performance.

Keywords: software defined networks, WAN, K-means

CHAPTER ONE
INTRODUCTION

1.1 Background of the study

WAN (Wide Area Network) is a telecommunications network that enables seamless connectivity across large geographical areas, often spanning countries, cities or continents. With the advent of Software-Defined Networking (SDN), a new level of flexibility, scalability, and centralized control has been introduced to WAN architecture, revolutionizing traditional networking methodologies. SDN-enabled WANs, or SD-WANs, provide an innovative approach to network management through the abstraction of control and data planes. This allows for better optimization and dynamic path selection, ensuring high-quality performance and a robust network infrastructure. In contrast to traditional WANs, SD-WANs leverage cloud-based services and virtualization to deliver cost-effective, agile, and easily manageable networking solutions.

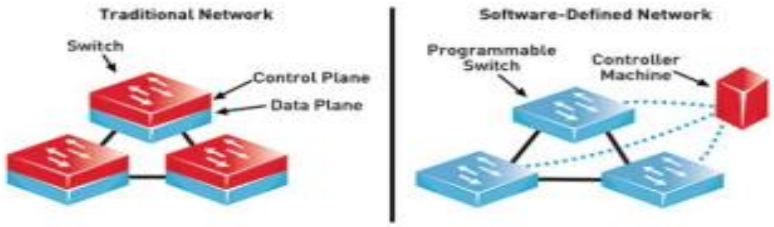

Figure 1. 1 Traditional versus Software Defined Networks. Image from [Dungay, 2016]

Also, Software-Defined Networking (SDN) has become a widely-accepted network paradigm that aims to increase flexibility, programmability, manageability, and agility in modern networks. By decoupling the control and data planes, and it is also called a recent networking Technology that directs network behavior to allow the transfer of information. Software-Defined Networks (SDN) have been introduced as an emerging paradigm whose primary advantage is giving developers greater control over the network traffic and administration (Tanha et al., 2016). SDN enables network administrators to have a centralized control system over the entire network, including WANs (Wide Area Networks).

One of the challenges in deploying SDN controllers in WAN networks is the placement of controllers in such a way that they provide efficient network management and optimal resource utilization. The importance of the optimal placement of controllers in SDN WAN networks comes from the crucial role controllers play in creating a reliable and efficient network based on the network architecture view. There are two different perspectives when analyzing network architecture are there, i.e. are The PoP level view and the router level view of a network are there, where router level view represents a very detailed and granular perspective. On the other hand, the PoP (Point of Presence) level view represents a more aggregated perspective. For SDN based Network Management the use of the PoP level view of the WAN network provided a more manageable and scalable approach to determine optimal SDN controller placement, facilitating the analysis and decision-making process. In large-scale distributed networks, SDN controllers must be placed strategically to meet the latency requirements, while minimizing network delays and resource utilization. Control overhead, reaction time, and system resilience are all factors that can be influenced by the optimal placement of controllers in SDN WAN networks. Therefore, determining the appropriate controller placement is an essential task in designing and implementing efficient SDN-based WANs.

In recent years, various mathematical models and machine learning techniques have been proposed for determining the optimal placement of SDN controllers. A common theme in these approaches is to create a solution space in which each point represents a possible controller location. These locations are then evaluated using one or multiple network performance metrics, such as latency and path diversity. Clustering algorithms have been particularly useful in this regard, as they enable researchers to identify promising regions in the solution space and reduce the total number of evaluation points. K-means clustering, a well-known machine learning technique, has been used extensively in these studies. K-means systematically partitions the data points into K clusters based on the Euclidean distance measure, with each partition corresponding to a candidate controller location. By using K-means clustering, researchers can identify areas that contain multiple high-quality candidates for controller placement. This approach allows for controllers to be placed more optimally while minimizing the trade-off between performance metrics.

Another approach for determining optimal controller placement is using integer programming and optimization techniques, like the Mixed Integer Linear Programming (MILP) model. This method employs mathematical formulations to minimize or maximize objective functions subject to specific constraints. By encoding network performance metrics and requirements in the objective function and constraints, researchers can create a mathematical model that captures the critical aspects of the SDN controller placement problem in a WAN setting.

Apart from these, other heuristic techniques like Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO) have been employed to solve the controller placement problem. These population-based metaheuristics explore the solution space by mimicking natural processes, like evolution and swarm behaviors. By iteratively refining their candidate solutions, these methods have proven effective in finding optimal or near-optimal controller placements in network domains.

The convergence of advances in mathematical models, machine learning algorithms, and heuristic optimization techniques indicates a promising future for determining the optimal SDN controller placement in WAN networks. Furthermore, the advent of network slicing and virtualization technologies has added another dimension to the problem, allowing researchers to explore more flexible, dynamic, and scalable techniques for accommodating diverse network requirements. By leveraging these advancements, network administrators and researchers can effectively incorporate SDN-based networks that deliver excellent performance and resource utilization in wide-area networking environments. In conclusion, the placement of SDN controllers in WAN networks is a crucial problem in modern networking. By utilizing advanced mathematical models, machine learning, and heuristic optimization techniques, optimal controller placements can be achieved, providing efficient network management and resource utilization. This research paper aims to investigate optimal SDN controller placement in WAN networks using a clustering approach, thereby promising an exciting and valuable contribution to the field of Software-Defined Networking

1.2 Motivation of the Study

The motive of this research work is to find efficiently placing the controller in network management scenario and attain the network that can achieve higher performance in terms of latency, throughput, and scalability. Additionally, it also attempts to lower the cost related with the adoption of the SDN, as well as reducing energy usage. Ultimately, the goal is to improve the overall performance of the network and make it easier to manage.

When using controllers in telecommunications, it's important to use the optimal number of controllers and to place them in the right locations. This will help ensure that all the components in the network are connected properly and that the system runs efficiently. To determine the optimal number of controllers, you'll need to consider the size of the network, the number of components that need to be connected, and the size of the data transfers that will occur. Once you have this information, you can determine the best placement for your controllers. Additionally, as they will enable network operators to deploy the ideal number of controllers and their locations for optimum network performance, the research's findings will be of great value to them.

Generally, it's best to place controllers near the core of the network and near the endpoints that need to be connected. You may also need to consider a distributed architecture to ensure that the controllers are placed in the right locations. This will help ensure that the network is balanced and that the controllers are connected to the right components.

1.3 Statement of Problem

WAN architecture now provides a higher level of flexibility, scalability, and centralized control thanks to the innovation of Software-Defined Networking (SDN). Recent studies have demonstrated that the placement of controllers in tiny networks using a combination of mathematical and emulation models is a slower and more computationally expensive technique. The SDN controller must be able to react fast to control requests because it serves as the network's brain. This results in controller placement having gaps because a way to balance load with a faster speed computing strategy is required. In order to maintain the accuracy of status information, control operations like data plane monitoring should be carried out as effectively as feasible. A controller plane optimization is necessary for this. Control plane to data plane performance has

been significantly impacted by propagation delay (switch-to-controller latency), making controller placement a crucial design consideration. Better network performance and efficiency are achieved because to controller placement, which controls how close the SDN controllers are to the data planes.

1.4 Research Question

This study is intended to answer the following four significant research questions

RQ1: Where should controller be placed in order to ensure a reasonable placement of controller?

RQ2: How many SDN controllers are needed in an effective SDN-enabled WAN

RQ3: How can k-means be combined with a mathematical approach for SDN-WAN network design that is used for placement of controller?

RQ4: What does Latency of communication of network nodes look like for k-means compared to Baseline clustering approach?

1.5 Objectives

1.5.1 General Objective

The general objective of the study is to find optimal placement of controllers using ML algorithms for better accuracy and efficiency.

1.5.2 Specific Objectives

In order to achieve the aforementioned general objective, this research will consist of the following specific objectives: -

- Design an optimal K-means ML approach to place the SDN controllers in the WAN network.
- Develop mathematical models that visualize distributed architecture of WAN
- Develop a cost-effective solution to determine the optimal number of SDN controllers needed in the WAN with use of SDN.
- To determine a way to balance the load between the SDN controllers.
- Develop an algorithm for SDN controller placement in WAN networks.
- Identify the advantages and disadvantages of the proposed approach for SDN controller placement.

- To create a mathematical model using a machine learning strategy to place the controller in the best possible location.
- Test Placement of controller with Emulation Platform

1.6 Scope and Limitation of the Study

1.6.1 Scope of the Study

The scope of the proposed work within the given time and resources includes-

- Develop a mathematical model that determines latency and distance between edge nodes that guarantee better network efficiency.
- Demonstrated Emulation platform with Mathematical results using long-running applications and Real WAN topology with in machine.

The developed algorithm is considered for determining the number of controllers and optimal placement

1.6.2 Limitations of the Study

This thesis does not cover the following due to time and resource limitations.

- ✓ The issue of high-performance devices capable of Wide area networks significantly impacted the outcome of this thesis. WAN requires a lot of computing, so it should be completed with high-performance technology such as graphics processing unit (GPU)
- ✓ Due to its historical nature, incomplete GML data from the Internet Topology Zoo may make accurate forecasts more difficult and may cause differences between the dataset and real network changes. It aids in consistently predicting the optimal outcomes of the suggested paradigm.
- ✓ Moreover, budget, deadline, and limitations of resources will be the constraints for greater practicality

1.7 Contribution of the Study

The contribution of this thesis is listed as follows

- Identify optimal controller placement and improve network performance without the need for expensive, time-consuming physical testing.
- Proposed a novel approach to software-defined networking (SDN) that combines mathematical modeling and fast machine learning (ML) algorithms to optimize

efficient network design.

- Evaluate placement of controllers with Mathematical Modeling and emulation software that is expected to provide a more accurate and efficient method for network design and optimization.
- proposed K-means machine learning with a mathematical model that was utilized to scale out with extensive real-world applications, which helped to enhance SDN technology.
- Improved the performance and accuracy of WAN network management by using SDN fastest ML clustering application network management concept by comparing with existing traditional network management.
- Design literature review based on finding optimal number of controllers and placement.
- Validated our mathematical and ML algorithms using Mininet software to emulate and test network design and optimization in a controlled, time-efficient manner.

1.8 Significance of the Study

The application of this thesis is to allow the operator to manage the network to improve overall network management tasks by implementing optimal placement with SDN-clustering approach. Inefficiency in the allocation of a controller placement results in over-expenditure, when either too many or too few controllers are allocated by the service provider, as it is either configured in static or dynamic without seeing load balancing. To balance the load inside the controller and assign resources in the event of a failure, controllers should use a quick allocation style.

The results of this research generally have a significant impact on cloud network management. In particular, as the thesis focuses only on the placement of controllers that provide better accuracy and performance, some of the examples that might apply to this.

- Find the optimal number of controllers and optimal placement of controllers with available controllers that are used to meet user requirements.
- Knowing the future to improve the effectiveness and reliability of networks, network management will switch to SDN network management technology.

- Determine the ML method that is utilized to place controllers efficiently and quickly.

1.9 Organization of the Thesis

The thesis is structured as follows:

Introduction (Chapter One) describes the background of the area and the motivation, the statement of the problem and the research question to be answered, the scope and limitations, and the application of the result.

Literature Review (Chapter Two) gives an overview of SDN, we covered the literature of research and related work previously done on SDN based on optimizing the parameters of increasing QoS, and comparison of different approaches taken by related work.

The methodology of the study (Chapter Three) discusses the various attributes used to create a solution and the way we can use those attributes. Dataset preparation and pre-processing processes, design tools, prototype creation systems, and platforms and evaluationmethods are also discussed.

Proposed Solution (Chapter Four) presents a used to Optimization approach to determine the optimal number of controllers and an unsupervised machine learning algorithm for finding optimal placement of controller

Implementation (Chapter Five) discusses the working environment setup and the implementation of the machine learning algorithm. It also discusses the Optimization approach with the integration of Unsupervised Machine Learning.

Results, Evaluation, and Discussion (Chapter Six) present the result of the Silhouette, Gap static and k-mean algorithm for the prediction of the client's needs.

Conclusion and Future Work (Chapter Seven) summarizes the work and concludes the result.It also presents future work.

CHAPTER TWO
LITERATURE REVIEW AND RELATED WORK

2.1 Introduction

SDN (Soft-ware Defined Network) is a network architecture that enables network administrators to programmatically configure and manage their networks. It works by separating the underlying hardware from the control layer, allowing administrators to make centralized changes to their networks without manually configuring each device individually. This makes SDN an essential tool for automation, scalability, and visibility in modern networks. The use of software defined networking (SDN) controllers for WAN networks is becoming increasingly popular due to its decentralized nature and scalability. The best location for SDN controllers has been determined using a variety of mathematical models and machine learning approaches in recent years. These methods all share the goal of building a solution space where each point represents a potential controller placement. One important factor in determining optimal SDN controller placement is finding the optimal number of controller nodes using optimization approach and machine learning (ML) approaches, which help to determine the total number of controllers needed by a network on a case-by-case basis and By leveraging machine learning approaches to mathematical Model is used to find an optimal number of controllers required for any given network architecture can ensure cost savings without compromising on throughput speeds or performance levels.

According to research on SDN controller placement, machine learning (ML) algorithms are capable of determining the best position for the controller based on a variety of factors, including device location, the existence of obstructions, connection stability, and throughput. By allowing the controller to be positioned closer to access points or end devices where traffic is concentrated, using ML techniques helps sustain performance by increasing the maximum attainable throughput. Additionally, because fewer hops are needed to process data packets leaving the master controller's network, quicker response times are also made possible. Clustering algorithms have shown to be very useful in this situation as they make it simple and quick for researchers to find interesting regions in the solution space while simultaneously reducing the number of evaluation points. The most widely used machine learning (ML) strategy for quick search and density determination is

based on clustering approach and the most well-known ML clustering techniques include K-means, K-medoids, hierarchical clustering, and DBSCAN.

This research has made substantial use of K-means clustering, a well-known machine learning approach. K-means systematically partitions the data points into K clusters based on the Euclidean distance measure, with each partition corresponding to a candidate controller location. By using K-means clustering, researchers can identify areas that contain multiple high-quality candidates for controller placement. This approach allows for controllers to be placed more optimally while minimizing the trade-off between performance metrics.

2.2 Modelling Approaches for the controller placement problem in SDN

To date, there has been numerous research studies directed towards addressing the controller placement problem in SDN. With in recent technology There are several modelling approaches for the controller placement problem in SDN. One approach is optimization technique and another approach is Mathematical formalization and heuristic approach to find minimum number of controller required in an SDN topology as well as their optimal placement location while keeping propagation latency from node to controller and inter controller under define limit and ensuring a balanced load distribution among them(Schutz et al., 2020). Another approach is machine learning Methodology to locate optimal placement of controller(Ramya et al., 2021)

2.2.1 Optimal Number of Controller

2.2.1.1 Optimization approach

Optimization approach is a set of mathematical methods used to identify the input variables that promote the achievement of an optimal solution. In the case of Software Defined Networking (SDN), optimization modeling can be used to find the optimal number of controllers in an SDN environment. By using optimization techniques, decision makers can maximize efficiency and cost-effectiveness while reducing network malfunctions or network delays. The number of controllers in an SDN environment affects decisions such as network control performance and traffic routing dynamics between each controller and its sub networks. Depending on the physical layout and given goals for the SDN, more or fewer controllers may be chosen to increase overall performance.

Using optimization models, engineers calculate controller thresholds for host overloading so that more decisive adaptation results can be generated. This includes mathematical algorithms that manipulate variables like bandwidth, communication latency, and node transmission variations among others to best determine what combination of workload percentages across multiple network nodes would be most beneficial. Utilizing these optimization models helps ensure that every possible mapping plate configuration is exhaustively tested before deployment so only the most effective configurations are ultimately utilized. This contributes substantially to improved resource utilization while also reducing network costs by ensuring resources used are optimized for value-recovery even at higher complexity levels, such as with larger networks with multiple controllers distributed across separate locations.

Genetic Algorithm-Based Modeling

Genetic Algorithm-Based Modeling (GABM) is an optimization technique that uses genetic algorithms to rapidly identify the optimal number of controllers in a Software-Defined Networking (SDN) system. GABM relies upon a set of fixed parameters and objective functions, as well as a graphical representation of the network to represent the topology. Once these are defined, the GA proceeds by attempting different combinations, or 'genes', until it finds an ideal configuration that produces the best results. The process usually takes several generations to complete and can be used to determine the ideal size for both central and decentralized SDN systems.

GABM has many advantages over traditional optimization techniques, such as numerical analysis or linear programming. It often converges on optimal solutions quickly and provides a high degree of accuracy when seeking controller numbers with limited resources. For example, running simulations using GABM can quickly provide insights into what number of controllers will optimize overall performance within specific constraints such as power and frequency. In addition, GABM is not only simpler to implement compared to other methods, but also more easily understood by engineers with less experience in SDN networks.

Genetic Algorithms that proposes by (Han et al., 2019) uses the application of Genetic Algorithms in SDN controller placement to enhance network management, optimize network traffic, and increase resilience against potential failures in Multi-controller SDN.

2.2.2.2 Mathematical approach

Mixed Integer Linear Programming (MILP)

Integer programming and optimization techniques like the Mixed Integer Linear Programming (MILP) model have also been used to determine optimal controller placement.

Gap statistics

Gap statistics is a modification of the Silhouette Analysis of clustering algorithms. It uses an algorithm to measure the intra cluster similarity and predicts the optimal number of clusters on a given dataset. Similar to Silhouette Analysis, Gap Statistics is a partition algorithm typically used in neural networks, to measure the quality of clustering measure based on average intra-cluster variation(Mohajer, n.d.). By combining this with Silhouette Analysis, the debate between the two conflicting results is resolved. The results suggested by Silhouette Analysis are enhanced by Gap Statistics, which adjusts any inaccuracies in clustering solutions identified by Silhouette Analysis. This method is used to determine the optimal number of SDN controllers to deploy in a network topology as it provides more accurate results than standard techniques used in evaluating clustering solutions.

Essentially, when using Gap statistics to determine an optimal number of SDN controllers for deployment in an SDN network topology, one can expect results which are highly accurate and reliable for practical implementation. Furthermore, since this method gives better understanding about performance measures of how well clusters respond under different set of scenarios, it becomes easier to design effective distributed applications and congestion avoidance techniques on such networks while still preserving scalability and reliability over time.

2.2.2.3 Heuristic approach

Heuristic algorithms provide an effective solution to optimize the location of an SDN controller. The algorithm considers criteria such as physical location, number of control points, resource availability, cost implications, and other relevant parameters before providing a recommendation. It also accounts for various physical constraints such as access restrictions, failure scenarios and geographic preferences when making a decision. By taking into account all these factors, the algorithm is able to determine the optimal location for placing an SDN controller.

Moreover, heuristic algorithms are also capable of accounting for dynamic parameters that may change over time such as network traffic patterns or changes in network topologies. As such, they can provide an ever-evolving optimization solution that adapts to changing conditions while keeping costs at a minimum. This means that an SDN controller placed with assistance from heuristic algorithms should perform optimally regardless of changes in the environment or network layout.

Heuristic Algorithms that proposes by (Yan-nan et al., 2012) addresses SDN controller placement using is "On the Placement of Controllers in Software-Defined Networks. They present a comprehensive comparison of various Heuristic Algorithms, such as the Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and K-means, for efficiently determining the ideal locations of SDN controllers. Through a thorough examination of these methods, the paper highlights the advantages and limitations of each, further ensuring that the best-suited algorithm is employed to enhance the performance and reliability of SDN networks in real-world scenarios.

Heuristic algorithm that proposes by (Li et al., 2020) uses a combination of greedy and random search methods to find the optimal placement of SDN controllers in a network. The algorithm takes into account factors such as network topology, traffic load, and controller capacity to determine the optimal placement of controllers. This approach improves the performance and efficiency of network management tasks.

The heuristic approach uses these algorithms to find good placements for controllers in SDN networks. The experimental results show that this approach can significantly reduce the number of controllers required to manage the network. The heuristic approach is one of the proposed solutions for this problem(Ramya et al., 2021)

Johnson's algorithm

Johnson's algorithm is a heuristic approach used to solve the single-source shortest path (SSSP) problem in dynamic or uncertain environments. The algorithm was proposed by Donald B. Johnson in 1977, and since then has been widely utilized for topology control in software defined networks (SDNs). At the core of Johnson's algorithm lies a modified version of Dijkstra's algorithm, which starts with a source node and expands outward via links that have been weighted

by estimated link distances to previous nodes. From this set of potential paths, an optimal solution can be found which maximizes total throughput while minimizing transmission latency among all nodes of the graph.

To find the best locations to place SDN controllers using Johnson's algorithm, first an SSSP route must be calculated between each node pair so that their distances can be approximated. These distances can then be used to weigh the links, with larger weights generally representing higher cost paths and smaller weights representing more cost-effective connections. With this network topology established, it is now necessary to identify which controller locations minimize the sum of cost over all paths from every node within the network. This is done through a combination of multi-objective optimization techniques such as linear programming and particle swarm optimization along with evolutionary computing approaches such as genetic algorithms.

Once all routes are optimized for cost and maximum throughput, it is possible to determine which nodes or cluster of nodes should host SDN controllers based on the results provided by Johnson's algorithm. Specifically, those points at which costs are minimized or transmission latency is lowest will typically provide ideal controller locations – supplementing available resources and improving overall network performance characteristics throughout the system architecture.

In the realm of Software-Defined Networking (SDN), finding the optimal controller location is a crucial factor in achieving better network performance and management. Johnson's algorithm stands out as an effective method to compute shortest paths between all pairs of nodes, which can then be utilized to determine the optimal SDN controller placement (Yan-nan et al., 2012)proposes Through leveraging Johnson's algorithm, the research effectively demonstrates the potential of the algorithm in identifying efficient controller placements in various network topologies, ultimately contributing to the improved performance and reduced latency within SDN environments.

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a metaheuristic optimization algorithm that has been used for SDN controller placement. PSO is a population-based optimization technique that replicates the cooperative behavior of fish schools and bird flocks. It has been used to optimize the placement of controllers in SDN networks. (Schütz et al., 2020)presents a mathematical formalization and a

heuristic approach to find the minimum number of controllers required in an SDN topology, as well as their optimal placement location, while keeping propagation latencies from nodes to controllers, and inter-controllers, under defined limits, and ensuring a balanced load distribution among them.

2.2.2 Optimal Controller Placement

2.2.2.1 Machine learning approach

Density cluster-based approach

Density cluster-based approach is a method used for solving the controller placement problem in SDN. It uses a density-based switch clustering algorithm to split the network into several sub-networks. As switches are tightly connected within the same sub-network and less connected from the switches in other sub-networks, one controller is deployed in each sub-network. This approach aims to minimize the average propagation latency between switches and controllers(Liao et al., 2017)(Qi et al., 2019).

Density cluster-based approach is a method used for solving the controller placement problem in SDN. It uses a density-based switch clustering algorithm to split the network into several sub-networks. As switches are tightly connected within the same sub-network and less connected from the switches in other sub-networks, one controller is deployed in each sub-network. This approach aims to minimize the average propagation latency between switches and controllers.

The entire network is divided into several sub-networks, each of which is managed by a controller. The method is based on modified density peaks clustering (MDPC) algorithm which aims to minimize the average propagation latency between switches and controllers.

Also, Density cluster-based approach is an effective method for solving the controller placement problem in SDN. It uses a density-based switch clustering algorithm to split the network into several sub-networks and deploys one controller in each sub-network. This approach aims to minimize the average propagation latency between switches and controllers. The experimental results show that this controller placement strategy can significantly reduce latency. as Density cluster-based approach is a method used for solving the controller placement problem in SDN.

Migration-Based Approach

Migration-Based Approach is a method used for solving the controller placement problem in SDN. It is based on three topologies and aims to reduce the number of controllers required to manage the network. This approach is based on the idea of migrating controllers from one sub-tree to another in order to balance the load and reduce the number of controllers required(Muller et al., 2014). In this approach, the network is divided into several sub-trees and each sub-tree is managed by a controller. The approach uses a migration-based algorithm to balance the load between controllers. The experimental results show that this approach can significantly reduce the number of controllers required to manage the network.

Silhouette analysis

Silhouette analysis is a type of machine learning algorithm that is used to find the optimal number of controllers in SDN (Software Defined Networking) technology. This type of analysis allows an engineer or system administrator to evaluate the data points gathered from various sources and identify the best selection for the network. 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 a varying number of partitions (Immermann et al., 2003) .The Silhouette method uses a combination of two factors: cohesion and separation, which are based on how much each point relates to a cluster of points. In other words, if there is low cohesion between two points, then they should be separated out into different clusters or controllers, whereas if there is high cohesion then they should be considered as one cluster/controller.

The Silhouette Analysis algorithm also helps in identifying outliers in the data set by looking at distances between points that should have similar attributes but have large differences. This algorithm can further refine the clustering process by detecting anomalous units in clusters that differ from their neighboring units far more than expected. Such variance can disrupt the smooth working of the network and hence needs to be addressed beforehand. The Silhouette Analysis algorithm works very well for addressing these issues with its robustness in uncovering outliers, thus providing an accurate prediction about which controllers need to be picked for optimal performance within SDN networks. The Silhouette Analysis algorithm also requires extensive

domain knowledge and experience when tuning parameters such as proximity metric weights and convergence thresholds depending on application-specific criteria like scalability, cost effectiveness and reliability of services provided via SDN networks. Moreover, since it is based on heuristics-based decisions, it might not provide accurate results unless proper parameters are tuned. Hence, engineers must understand its implementation thoroughly before taking any measures to build or improve upon existing SDN networks using Silhouette Analysis algorithms.

K-medoids clustering

K-medoid clustering is a clustering approach that has been used for SDN controller placement. It is a variant of the k-means algorithm that uses medoids instead of means. The medoid is defined as the data point that minimizes the sum of distances between itself and all other data points in the cluster. The k-medoids algorithm is more robust to noise and outliers than k-means because it uses actual data points as cluster centers instead of means. However, it is computationally expensive and does not scale well to large datasets.

K-means Clustering

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(Bouleimen et al., 2003) and Clustering Large Applications (CLARA)(Fahim et al., 2006)) or exhaustive algorithms (such as k-means(Foreman et al., 2017) and PAM. k-means is considered an unsupervised machine learning algorithm, as it is a clustering technique that does not require labeled data for training. It can be applied to dynamic controller placement in Software-Defined Networking (SDN) as a heuristic approach to optimize the placement. One big advantage of the k-means algorithm over PAM (Partitioning Around Medoids) for SDN (Software-Defined Networking) controller placement is its computational efficiency and scalability. K-means has lower time complexity compared to PAM, making it more suitable for large-scale networks and quicker in finding optimal controller placements. This is particularly important in SDN environments, where network configurations can be dynamic and a faster algorithm can help adapt

to changes more efficiently. Overall, k-means offers a trade-off between speed and accuracy, which can be a significant advantage in certain application scenarios.

K-means clustering, a widely utilized unsupervised machine learning technique in the realm of Software Defined Networking (SDN), has proven effective in determining the optimal placement of controllers within network architectures. By leveraging the inherent simplicity and speed provided by the K-means algorithm, researchers have been able to conduct insightful analyses of network topologies, highlighting critical factors such as latency, throughput, and controller-load balancing. In recent literature, various adaptations of the K-means clustering algorithm have been employed in combination with other optimization techniques, signifying its versatility in effectively tackling the multi-dimensional challenge of controller placement in SDNs. These enhancements aim to minimize response times, energy consumption, and operational costs in dynamic network environments, thus making K-means clustering an essential tool for network administrators seeking to optimize their SDN deployments.

In the context of SDN, K-means clustering can be used to find the optimal location for controllers in a network. (Kuang et al., 2018) discusses the use of K-means clustering to find the optimal placement of SDN controllers in a network. The paper proposes a novel approach that uses K-means clustering to group switches based on their traffic load and then places controllers in the center of each group. This approach improves the performance and efficiency of network management tasks. By implementing K-means clustering for SDN controller pl0acement, it significantly improves load balancing and reduces latency, enhancing overall network performance. (Ibrar et al., 2021) explore the utilization of K-means clustering for determining SDN controller locations and validate its efficiency in terms of latency and robustness. The integration of K-means clustering in SDN controller placement presents a promising method for optimizing network performance and ensuring smooth network management.

2.3 Simulation-based modeling

Simulation-based modeling is an effective and increasingly popular way to identify the optimal number of controllers for a Software Defined Network (SDN). By creating simulations that accurately reproduce the network environment, infrastructure, topology and workloads potential

system failures, it's possible to create complex models that can inform an engineer regarding how many controllers are needed to ensure maximum efficiency. Through these models, engineers can extrapolate data concerning reliability and latency related to various traffic patterns. This data can be used to assess the controller architecture of existing SDN deployments as well as provide guidance on what is needed to achieve desired levels of performance in new networks. Simulation-Based Modeling helps engineers identify redundancies and pinpoint performance bottlenecks when designing a controller network. It also helps them identify the optimal number of controllers that needs to be deployed based on their workload characteristics and any applicable scalability constraints. The resulting design may reduce controller cost by reducing the amount of hardware required while still meeting desired service level objectives.

2.4 Related work

In recent years, there has been a lot of research on the placement of controllers. Jalili et alquality.'s measurements for CPP expanded on the problem (2020). They assume that the controller location issue will be resolved by calculating the propagation latency between the switch and the controller using link utilization and other parameters. The article is anticipated to produce optimal controller placement, switch assignments to controllers, and network routes. The expense of deployment with an increasing number of controllers is ignored, though.

The controller placement issue has been framed using performance measurements, which are a crucial criterion in deciding where controllers should be placed (Sminesh et al., 2019). The Internet Topology Zoo has examined the Controller Unbalance Factor, Inter Controller Latency, Average Case Latency, and Worst-Case Latency for various Topologies. The aim of this effort was to position controllers in each network topology to achieve the highest possible throughput.

Deterministic annealing-based clustering algorithms that proposed by (Soleymanifar et al., 2020) uses two distinct that take a certain set of goal functions and provide the best viable position for the controller to address the issue of Edge Controller Placement (ECP) in wireless edge networks. The method proposed by (Firouz et al., 2021) provides a unique controller placement technique that takes advantage of network portioning and nature-inspired optimization techniques to account for network propagation latency between controllers and switches. Many goals have been

introduced and reviewed in these articles. The original controller placement algorithm and controller placement location have been evaluated in light of these objectives. The cost of controller deployment, one of the most important objectives of software-based networks, has not yet been solved.

The algorithm that proposed by (Dhar et al., 2021) put one controller in each cluster and recommended a mathematical strategy to design the clusters in order to lessen the worst-case latency. The "$-method" was introduced, which uses the distance matrix of the network nodes and inserts a "$" if it is discovered that a matrix component is farther than a predetermined maximum distance. That still holds true for small networks, though.

The solution proposed by (Amiri et al., 2021) present a model of the controller placement problem (CPP) as a Facility Location Problem to achieve a specific average propagation latency and load-balancing across controllers (FLP). The average controller processing delay, the average message arrival rate from each data plane, and the average propagation latency between the controllers and data planes are the three metrics we build to represent the network. The simulation results demonstrate that the solution to this optimization location problem correctly finds the smallest number of controllers and their appropriate placement in the network in order to satisfy network requirements.(Amiri et al., 2021) present a model of the controller placement problem (CPP) as a Facility Location Problem to achieve a specific average propagation latency and load-balancing across controllers (FLP). The average controller processing delay, the average message arrival rate from each data plane, and the average propagation latency between the controllers and data planes are the three metrics we build to represent the network. The simulation results demonstrate that the solution to this optimization location problem correctly finds the smallest number of controllers and their appropriate placement in the network in order to satisfy network requirements. In our thesis, a novel model for this objective function is presented in addition to addressing the significant assignment issue. Considering this goal, along with inter-controller latency, as useful goals in

Summary of Related Works

The following tables illustrate previous works on a SDN placement.

Table 2. 1 Summary of related works

Reference	Title	Placement Metrics	Expected Outcome	Gaps
(Jalili et al., 2020)	A new framework for reliable control placement in software-defined networks based on multi-criteria clustering approach	Flow setup time and inter-controller latency	Find the best location for controllers, Find best switch assignment to controller Find the best route in the network.	Do not consider cost of deployment for scaling the network with multi controller
(Sminesh et al., 2019)	Optimal multi-controller placement strategy in SD-WAN using modified density peak clustering	Utility value	minimum utility value and optimal placement of SDN	Number controller have to be predefined for computing placement of controller
(Soleymanifar et al., 2020)	A Clustering Approach to Edge Controller Placement in Software Defined Networks with Cost Balancing	Accuracy and speed.	Place controllers close to edge node clusters and not far away from other controllers to maintain a reasonable balance delay costs	Computationally complex and is used for wireless network

(Firouz et al., 2021)	A novel controller placement algorithm based on network portioning concept and a hybrid discrete optimization algorithm for multicontroller software-defined networks	Network propagation delay and convergence	Minimizing network propagation latency	it took more resource to get the proposed attribute.
(Dhar et al., 2021)	A new optimization technique to solve the latency aware controller placement problem in software defined networks	Distance matrix of the network nodes	Generate the minimum worst-case SC latency efficiently that one could ever achieve for a different number of controller placement.	Applicable for small scale network
(Mamushiane et al., n.d.-b)2021	Controller placement optimization for software defined wide area networks (SDN-WAN)	Average Switch to controller Latency and Worst-case Latency switch to Controller Balancing, Propagation+queuing+processing latency, signaling overhead	Placement of controller with the lowest average latency	Slower computational time for placement of controller that leads for inefficiency of placement method

(Amiri et al., 2021)	Optimized Controller Placement for Software Defined Wide Area Networks	Facility location problem	Average propagation latency and load-balancing among controllers	Finding the correct number of controllers is impossible because they are predetermined.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 Overview

This chapter goes to great lengths about the precise techniques, equipment, and steps required to assess the study problems. It offers a very succinct grounding in methodology that may be applied to the design, execution, and assessment of the suggested strategy. The processes for choosing samples, gathering data, and doing analyses were covered in more depth. The technique chosen to achieve the research's main purpose is most significantly included in this part.

This thesis seeks to locate the SDN Controller's ideal location utilizing an unsupervised machine learning strategy in cooperation with mathematical formalization, as directed by the research questions listed in sub-section 1.4 of this document. The goal of this study is achieved using the following strategies and techniques that we described in this chapter. This method is guided by the datasets utilized to assess the efficiency and efficiency of the suggested method are referred to as South African National Research Network (Sanren) gml format. To achieve the goal of this study, the strategies and techniques we used are mentioned in this chapter.

3.2 Data Collection

The datasets utilized to assess the effectiveness and efficiency of the suggested method are referred to as South African National Research Institution Network (Sanren) gml format. These datasets are used to train unsupervised machine learning algorithms for SDN controller placement. (Sanren)

3.2.1 Description of Each Data Features.

The following is a description of Sanren gml format dataset Attributes.

- **Edge:** represent connections between cities that mentioned in the gml data set format that needs for network topology development.
- **Source and Target attribute:** represent the cities connected by the edge.
- **Node:** Represent cities in South African Country, with their reprentive coordinates (Longitude and Latitude) and an Internal attribute set to 1.
- **Label:** the label of the network is SANReN
- **Country:** south Africa,

SANReN, serving as the National Research and Education Network (NReN) of South Africa, the South African National Research Institution Network offers a unique and advantageous real-world network topology for testing clustering approaches in SDN network management. Its large-scale, complexity, research focus, advanced infrastructure, data availability, diverse

network characteristics, and relevance to developing countries make SANReN an ideal choice for encouraging innovation in the field of SDN management.

3.2.2 Identify the Controller Placement Topology

In order to ensure realism in its application, our proposed solution was applied on a real-world Wide Area Network (WAN), the South African National Research Institution Network (SANReN). This network was selected to demonstrate the effectiveness of the proposed research in an emerging market context. The PoP level view rather than the router level view of this network is what was utilized for several reasons; it offers a more encompassing view of network links, informs users as to where they can access the SANReN and allows for optimal resiliency or redundancy configurations. To get data about SANReN topology and geographic locations (longitude and latitude), it was collected from Internet and we are arranging in Geography Markup Language (GML) format. This dataset contained information about 7 nodes and 7 fibre link connections configured in a ring topology.

3.3 Data Preprocessing

The data travels through a variety of pre-processing stages before being inserted into the recommended model. In order to develop a mathematical model that is used to determine the optimal number of controllers, one of unsupervised machine learning techniques called Silhouette is suggested. Algorithms that are unsupervised can learn from the input data without the need for labels. In order to decrease network latency, we use these algorithms to calculate the number of controllers to use in the controller placement algorithm. These are frequently used to assess how well data clusters fit together based on how far apart the data points are.

After determining how many controllers are required in the suggested architecture, we also employ Johnson's algorithm, a heuristic method for solving the placement problem of SDN controllers in the network, which determines the shortest path between any pair of switches. The best location for SDN controllers throughout the network may then be decided using this information. One of the elements for effective SDN Network management is the speed of the controller placement calculation. For theses, we use the South African National Research Institution Network (SANReN) as a case study and employ the quickest and best-known Clustering ML technique known as K-means clustering. 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 fibre where speed is approximately the speed of light in fibre (i.e., $2 \times 10^8 \frac{m}{s}$),

We compute propagation latency by taking the ratio of average distance (between nodes) to speed of light in fibre. Propagation latency is the time delay between the transmission of a signal and its reception at the destination. It is calculated by dividing the distance traveled by the signal (D) by the speed of light (C), which is the maximum speed at which information can be transmitted. The distances are calculated using the Haversine approach(Andrew et al., 1835). The results from our simulations and discussions are also presented in this section

$$Propagation\ latency\ =\ \frac{D}{C} \qquad \text{Equation 3. 1}$$

Where C stands for the speed of light in fiber $2 \times 10^8 \frac{m}{s}$ and D stands for Distance calculated by Harvesine formula. This equation is commonly used in telecommunications and networking to estimate the time it takes for a signal to travel from one point to another. The propagation latency can have a significant impact on the performance of communication systems, especially in high-speed networks where delays can cause data loss or errors.

3.4 Model Construction Based on ML and Mathematical Formalization

3.4.1 Optimal number of controllers

This section introduces the algorithms used to find the optimal number of controllers in a proposed wide area network topology (i.e SANReN).

3.4.1.1 Silhouette analysis

In the context of the controller placement problem, we adopt Silhouette Analysis ,it 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 a varying number of partitions.(Mamushiane et al., n.d.-a). In light of this, we suggest the silhouette technique, which may be applied to problems like determining how many controllers are required, given a wide area network design, to obtain the lowest possible intracluster propagation delay variation. Our goal function is shown in Equation 3.2

$$X\ =\ min\ \sum_{k=2}^{n} \boldsymbol{L(c)} \qquad \text{Equation 3. 2}$$

In this equation, X represents the minimum value of the sum of L(c) for k ranging from 2 to n. L(c) represents the intra-cluster propagation latency variation for a given cluster c. The study aims to minimize this value by determining the optimal number of controllers needed. The silhouette analysis algorithm is a clustering algorithm that measures how well each data point in a cluster is separated from other clusters and requires three input parameters, namely a clustering algorithm (cluster Algorithm) to cluster network data-plane nodes, distance function handle (disfunction), network topology graph (G (V, E, X)), where V denotes data-plane nodes (switches), E denotes edges (links between nodes), and X denotes the geographic locations (longitude, latitude) of nodes), and maximum number of controllers (maxNumControllers). It calculates a silhouette coefficient for each data point, which ranges from -1 to 1. A coefficient of 1 indicates that the data point is well separated from other clusters, while a coefficient of -1 indicates that the data point is more similar to data points in other clusters.

1.clustering algorithm – used in this ML algorithm that is called k-means clustering algorithm that used to groups a set of data points into a predetermined number (K) of clusters, based on the similarity's distances between each data point.

2. Distance function- as a reference to a function or Haversine Distance formula works by first converting the latitude and longitude of the two points to radians, then calculating the differences in latitude and longitude between the points. It then utilizes these differences, along with the Earth's radius, to calculate the distance using trigonometric functions. The resulting distance is expressed kilometers or miles. calculating distances between pairs of switches, using the Haversine distance approach ensures that the distances are accurate even if the switches are located at different longitudes and latitudes, considering the Earth's curvature. This can be important for applications like telecommunications or transportation, where accurate distance calculations between different locations on Earth are necessary. With in this work Haversine distance approach was used to compute the great circle distances between pairs of switches. 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).

$$D = 2(r)arcsin\{\sqrt{\boldsymbol{sin^2}\left(\frac{\boldsymbol{\alpha_2-\alpha_1}}{\boldsymbol{2}}\right) + \boldsymbol{cos(\alpha_1)\,cos(\alpha_2)\,sin^2}(\frac{\boldsymbol{\theta_2-\theta_1}}{\boldsymbol{2}})}\} \qquad \text{Equation 3. 3}$$

where α_1 and α_2 are the latitudes of points 1 and 2 respectively, θ_1 and θ_2 is the longitudes of point 1 and 2 respectively and r denotes the radius of the earth, a constant equal to 6371 km.

3. Network topology graph-Denoted by (G (V, E, X)) is a representation of the arrangement of interconnected devices or nodes within a computer network. This graphical representation helps to understand the structure and organization of a network, which is useful for designing, maintaining, and troubleshooting network issues.

were

- V denotes data-plane nodes (switches)- that help data packets find their way through the network by guiding them from one point to another.
- E denotes edges (links between nodes), and
- X denotes the geographic locations (longitude, latitude) of nodes), and maximum number of controllers (maxNumControllers).

With in Silhouette analysis the last step is to calculate the silhouette coefficient as specified by the maxNumControllers. for every given number of controllers, the number of iterations was set to 20 to maximize accuracy of the results. Then 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.

3.4.1.3 Cost-latency trade-off analysis

A tradeoff between cost and the QoS provided by the network is another aspect that affects the selection about the number of controllers. Our goal is to quantify this trade-off and provide network operators a useful recommendation for the right number of controllers to deploy while also taking cost and delay into account. A definition of this trade-off may be found in Eq. (3.7), where k is the number of controllers, CPXk is the normalized cost of deploying a single controller, and Lavg is the average latency when k controllers are deployed. One dollar per controller was maintained as the normalized cost of deployment (assuming that a company plans to install the same model of an SDN controller). Controllers to deploy is the price involved in adding new controllers to a network. This statistic is important since it affects overall CapEx and the amount of return on investment (ROI) network operators produce. On the other hand, there is a sizable.

$$Cost\ factor = \frac{k * CPx_k}{L_{avg}}\ \frac{\$}{ms} \qquad \text{Equation 3. 4}$$

3.4.2 Optimal controller location

This section describes the algorithms used to find the best locations to place SDN controllers.

3.4.2.1 Johnson's algorithm

To find the best locations for SDN controllers in a WAN, the shortest paths between each pair of switches must be determined. Johnson's algorithm is used along with the K-means algorithm to achieve this. The algorithm involves adding an arbitrary switch, checking for negative weight cycles, using the Bellman Ford algorithm to find shortest paths to the new switch, reweighting the graph, removing the new switch, and applying Dijkstra's algorithm to find shortest paths in the reweighted graph.

1. Bellman Ford algorithm-operates on a graph comprising vertices (nodes) and edges (links between nodes) with assigned weights (costs). Given a source vertex, the algorithm determines the shortest distance to every other vertex in the graph while also detecting negative weight cycles, which are cycles whose total weight is negative.

The algorithm iteratively relaxes all the edges in the graph, attempting to find a better path between vertices by updating their distance if the relaxation results in a shorter path. It does this by comparing the current path's sum of weights to the potential new path's sum of weights. If the new path offers a smaller total weight, the distance and predecessor of the vertex are updated.

The algorithm terminates after |V|-1 iterations, where |V| is the number of vertices in the graph. In each iteration, the algorithm traverses all edges, performing relaxation for each edge encountered. If relaxations continue after |V|-1 iterations, it indicates the presence of a negative weight cycle.

2. Applying Dijkstra's algorithm - to find shortest paths in the reweighted graph. Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge weights, producing the shortest path tree. Dijkstra's Algorithm original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

The basic idea behind Dijkstra's algorithm is to iteratively select the node with the minimum distance value, update the distance values, and then adjust the set of visited nodes until all nodes have been visited or the shortest path has been found. The algorithm maintains a set of unvisited nodes with tentative distance values and repeatedly selects the node with the minimum distance value and explores its neighbors.

3.4.2.2 Clustering Algorithm

Clustering algorithms have been shown to be particularly helpful since they allow researchers to quickly and easily locate interesting areas in the solution space while also minimizing the number of assessment points. Based on this on this thesis work, K-means clustering approach is used after the medoids to find optimal placement of SDN controller. Within these Two QoS parameters are considered in this work to find a solution for optimal controller placement. that is the average propagation latency (which is the overall propagation latency) and the worst-case propagation latency (which is the maximum network latency) which determines reaction times that considered as network performance indicators, which is calculated by taking the minimum distance between a node and a controller in the network. The equation for calculating the average propagation latency is given as:

$$L_{avg}(Z') = \frac{1}{(2 \times 10^8)N} \sum_{v \in V}^{z \in Z'} min\, d\,(v, z) \qquad \text{Equation 3. 5}$$

The worst-case propagation latency, on the other hand, is defined as the maximum network latency, which is calculated by taking the minimum distance between a node and the farthest controller in the network. The equation for calculating the worst-case propagation latency is given as:

$$L_{wc}(Z') = max_{v \epsilon V} min_{z \epsilon Z} d(v, z) \qquad \text{Equation 3. 6}$$

where $'Z'$ represents the set of candidate controller locations, L_{avg} (Z') is the average latency, N is the total number of nodes in the network, V is the set of all nodes in the network, and $d(v,z)$ is the distance between node v and controller z, 2 × 108 is the speed of light in fibre.

CHAPTER FOUR
PROPOSED OPTIMAL SDN CONTROLLER PLACEMENT

4.1. Overview

The WAN solution employing the Placement SDN Controller is presented in this chapter for the network operator in network management operation. This chapter also covers the detailed architecture of the suggested solution, including the block diagram, flow chart, and algorithm needed for a clear resolution of the issue. Additionally, it provides a succinct overview of network management processing, including the crucial steps of pre-managing, determining the optimal number of controllers, visualization, parameter settings, and describing the proposed ML model. It then moves on to describing the proposed architecture, along with mathematical expression and necessary diagrammatic representation.

One of the important considerations for placing controllers in its appropriate location is the ideal number of controllers. To find those effective number of controllers that are used for efficient placement of controllers it is important to also consider the cost-effective positioning of the controller. We might also state to locate the SDN controller in the suggested WAN network topology, two crucial steps must be taken. Preprocessing data is the first stage, which focuses on importing datasets, extracting node information, building relationships between dataset attributes based on various metrics, such as distance between nodes, and creating the topology based on the metrics used for evaluation. The second stage is model creation, which combines an ML technique with mathematical optimization of critical attributes utilized to locate controller location.

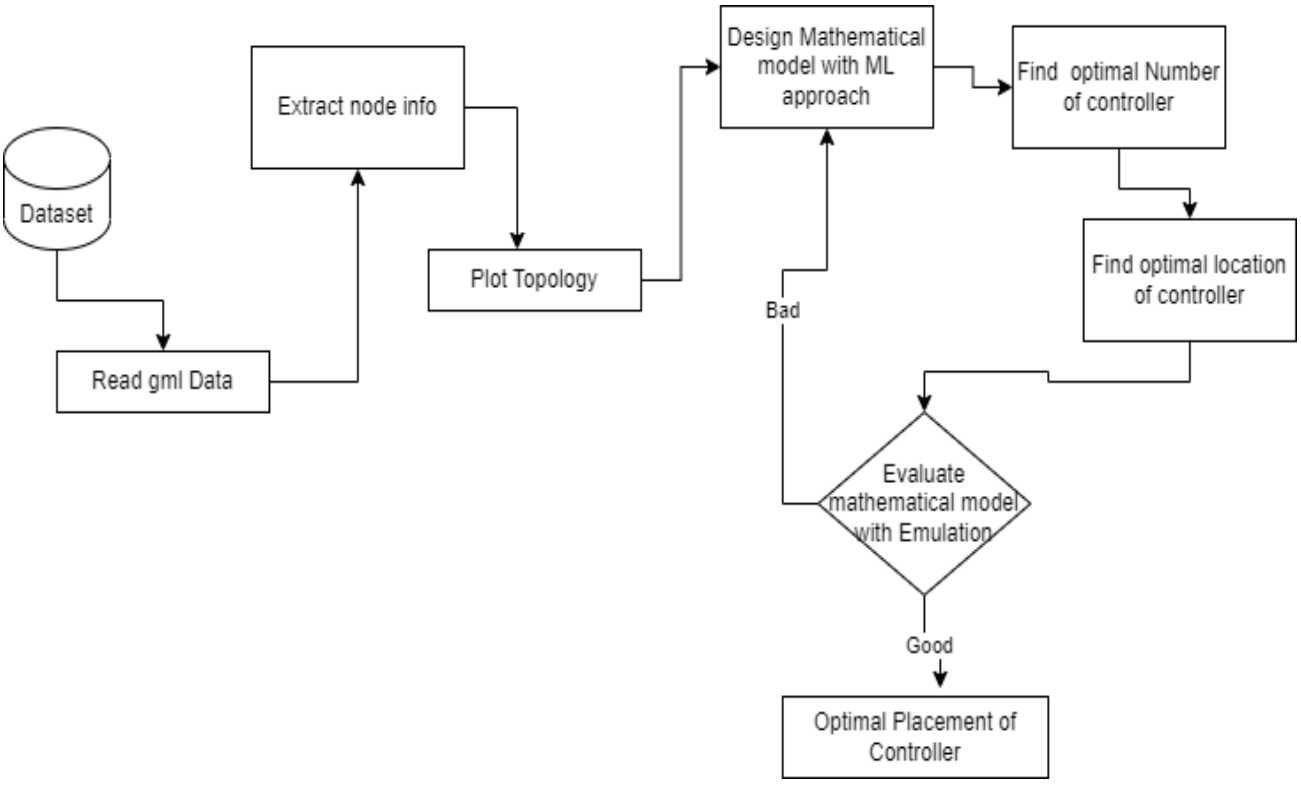

Figure 4. 1 The overall block diagram for the proposed controller placement model

The recommended system design based on the prior architecture is shown in Figure 4.2. The architecture imports the dataset containing the information imported from Internet Topology in step one, reads the gml data, extracts the node information, calculates the distance, and then starts optimizing the calculated attribute.

4.2 Dataset Preparation

4.2.1 Read the GML file containing network topology information.

SANReN topology datasets were used in this investigation from Section 3.2.1 describes the comprehensive data description. A sampling of the South African National Research Institution's data is shown in Table 4.1.

Network gml file trace with text line as follows.

```
Testbed 0
node [
  id 0
  label Johannesburg
  Country South-Africa
  Longitude 28.04363
  Latitude -26.20227
```

```
  ]
  node [
    id 1
    label Pretoria
    Country South-Africa
    Longitude 28.18783
    Latitude -25.74486
  ]
  node [
    id 2
    label Durban
    Country South-Africa
    Longitude 31.01667
    Latitude -29.85
  ]
  node [
    id 3
    label Bloemfontein
    Country South-Africa
    Longitude 26.2
    Latitude -29.13333
  ]
  node [
    id 4
    label East-London
    Country South-Africa
    Longitude 27.91162
    Latitude -33.01529
edge [
    source 0
    target 1
    id "e0"
  ]
  edge [
    source 0
    target 3
    id "e1"
  ]
```

4.3 Preprocessing and Data Formatting

4.3.1 Visualize the network topology as a graph

"Visualize the network topology as a graph" refers to displaying a network's structure including its components, such as computers, routers, switches, and other devices in a graphical format that demonstrates the connections and interconnections among them. The graph is a visual representation in this context made up of nodes (devices) and edges (the connections between

devices). This visual aid makes it easier to comprehend how the network is set up, see possible bottlenecks, and make plans for growth or improvement. and get the network nodes' coordinates (position or location) and related labels (which are often text annotations or IDs linked to the nodes) for further processing or analysis.

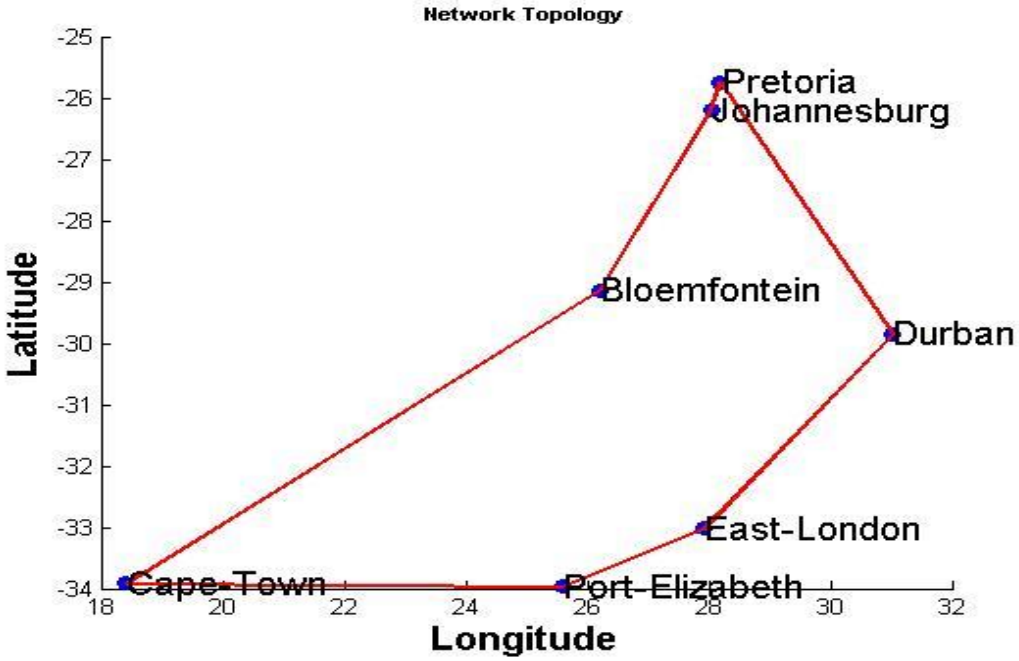

Figure 4. 2 Screenshot of graphical representation of network topology.

4.3.2 Calculate adjacency matrix

The process of producing a matrix that represents the links (edges) between the nodes (vertices) in a graph or network is known as "calculate adjacency matrix." It refers to computing a matrix using the network topology data that was retrieved from the GML file in the context of processing GML data.

The nodes of the graph are used to name the rows and columns of the adjacency matrix, which is a square matrix. N is the number of nodes in the graph, and N is the size of the matrix. The existence or absence of an edge between nodes i and j is represented by each item a (i, j) in the adjacency matrix:

- The entry a (i, j) is set to 1 if there is an edge between the nodes i and j.
- The item a (i, j) is set to 0 if there is no direct link between the nodes i and j. The rows and columns of the adjacency matrix are labeled by the nodes of the graph. N is the number of nodes

in the graph; hence the size of the matrix is NXN. The adjacency matrix's item a (i, j) indicates whether nodes i and j have an edge or not.

- The value of the entry a (i, j) is set to 1 if node i and node j are connected by an edge.
- The value of the entry a (i, j) is set to 0 if nodes i and j are not directly linked.

Finding the shortest pathways, analyzing connections, and calculating node centrality are just a few of the activities that the adjacency matrix may be used for in network analysis.

4.3.3 Calculate geographical distance matrix

The phrase "calculate geographical distance matrix" refers to the process of calculating a matrix based on the geographic coordinates of each pair of nodes in the network to represent the distances between them. This matrix can be helpful for comprehending the spatial interactions and separations between network nodes and can be used as an input for a number of network analysis activities that take into consideration the actual separations between entities.

For instance, we can determine the matrix of distances between all pairs of cities in our scenario as we have a network of cities with their latitude and longitude coordinates. This matrix may be used to examine how close or far away these cities are in space as well as to examine their connectedness depending on proximity to one another.

```
%Calculate geographical distance matrix
dMatrix1= []; dMatrix2= [];
for i=1: length(D)
    for j=1: length(D)
[dMatrix1(i, j) dMatrix2(i, j)] = lldistkm (D (i, :), D (j, :));
  end
end
dMatrix1;
dMatrix2;
```

4.3.4 Determine edge weights

In the context of network analysis or graph theory, "determine edge weights" refers to the process of giving the connections (edges) between the network's nodes (or points) a value or cost. Edge weights can reflect a number of connection-related characteristics, including time, cost, distance, and any other variable that characterizes the interaction between nodes.

These edge weights are often essential for a number of network analysis activities, including calculating the shortest way, lowering the total cost of traversing the network, identifying the most reliable or resilient paths, etc. In order to effectively depict the underlying problem, the procedure

of establishing edge weights often depends on the particular problem or application at hand. This may require the incorporation of extra data and characteristics.

%Weight Matrix

edgesDistKm = []; %distance of edges in km, // this represents weights
for i=1: length (graph. edge)
edgesDistKm(i,1) =graph. edge(i). source; %first two columns indicate edges
edgesDistKm(i,2) =graph. edge(i). target;
source = graph. edge (i). source+1;
target=graph. edge (i). target+1;

4.3.5 Shortest distance calculation

After processing the GML data and determining edge weights, we can apply Johnson's algorithm to calculate the shortest distances between all pairs of nodes in the network. We may use Johnson's approach to determine the shortest distances between all pairs of nodes in the network after processing the GML data and determining edge weights. The algorithm takes the graph, the number of nodes, and the source node as inputs and outputs the shortest path to all the other nodes in the graph. Then, the k-means clustering algorithm is used to find the optimal placement of controllers based on the results of the Johnson's Algorithm and the latency and controller location factors. The K-means algorithm was initialized with the number of clusters obtained from the Silhouette analysis

Table 4. 1 Johnson's Algorithm

Require: $G(V, E)$ {undirected weighted network graph}
1: Compute G' $where$ $V[G'] \leftarrow V[G] \cup q$ {G' is a 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 $v \in [G]$ **do**
10: set $h(v) \leftarrow \delta(q, v)$ compute shortest path using Bellman-Ford
11: end for
12: for $(u, v) \in [G']$ **do**
13: $wnew \leftarrow (u, v) + h(u) - h(v)$

14: **end for**
15: **for** $u \in [G]$ **do**
16: execute $Di(G, wnew, u)$ to compute $\delta new(u, v)$ for all $v \in V[G]$
17: **for** $v \in [G]$ **do**
18: $(u, v) \leftarrow (u, v) + h(u) - h(v)$
19: **end for**
20: **end for**
21: end if
22: return shortest path matrix

4.4 Optimal Number of Controllers

4.4.1 Clustering Quality

4.4.1.1 Silhouette analysis

With propagation delay as our primary performance indicator, we used our enhanced Silhouette method to determine the optimal number of controllers to deploy on the SANReN. In order to do this, we first acquired relevant information on the network's latency characteristics. Next, we used the Silhouette technique to find clusters with the greatest inter-cluster separation and the smallest intra-cluster distance. We then used this data to determine the appropriate number of controllers to install on the network's backbone. The results of our investigation are displayed in the table below.
Algorithms for silhouette Analysis

Table 4. 2 Algorithm for silhouette Analysis

Require: (V, E, X), maxNumControllers, disfun, Number of Cluster(k)

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

4.4.2 Cost-latency trade-off analysis

The other metrics that are considered in this work are to find out the efficient optimal number of SDN controllers with this research are the Cost-Latency trade-off analysis, which computes the cost-benefit trade-off between the price of putting in new controllers and their performance. This trade-off is used to locate the best location for SN controllers, with cost being one of the factors.

Table 4. 3 Algorithm for Cost-Latency

Require: Average Latency**(**L_average(i)), List of cost benefit with index(i), Number of Controller(k)

1. Initialize an empty list called cost benefit
2. Set controller cost to 1
3. Set k to 4
4. Loop from i = 1 to k:
   a. Compute cost benefit for the current i: (controller cost * i) / L_average(i)
   b. Store the computed cost benefit in the list at index i
5. Set k to a list [1, 2, 3, 4]

4.5 Optimal Controller Location

The algorithms used to locate SDN controllers in the ideal places are described in this section.

4.5.1 Calculate average and worst-case latency

In the context of network analysis, when we say "compute average and worst-case latency," we mean to assess the average time delay encountered in a network and the worst potential time delay under specific circumstances.

1.Average latency -is the average amount of time that data packets in a network delay as they move from a source to a destination. By taking into account all potential pathways (between all possible pairs of nodes), calculating the time delay for each path, and then averaging those values, the average latency is determined in this situation. This figure aids in our comprehension of the network's general performance.

2. Worst-case latency: This refers to the longest possible time delay a data packet might encounter while moving over the network. This number shows the farthest distance a data packet may travel between any two network nodes. The worst-case delay assists in recognizing possible bottlenecks and building networks to prevent such scenarios by giving insight into the least efficient path a packet would have to take.

4.5.2 Placement of SDN controller

4.5.2.1 K-means

We found out average time delay encountered in a network and the worst potential time delay under specific circumstances in the context of SDN controller placement, we use K-means clustering approach that used to group switches based on their average and worst-case latency to the controller and used to find out optimal placement of controller. The algorithm we propose placement of controller based on k-means clustering is as follows

Table 4. 4 Algorithm of K-means Clustering with Haversine Distance

Require: G (V, E), $NumClusters$
1: $k \leftarrow NumClusters$
2: $R\ i$ ($i \in [1 \ldots k]$) ← randomly select k objects (initial cluster centroids) from G (V, E)
3: Initialize old centroids to an empty list
4: while old centroids! = R do
5: old centroids ← R
6: for $So \in (V, E)$ do
7: Compute distance from So to each Ri ($i \in [1 \ldots k]$) using Haversine distance metric
8: Associate So to the nearest Ri (assign it to the cluster with the minimum distance)
9: end for
10: for $i \in [1 \ldots]$ do
11: Update $R\ i$ to be the mean of all So points associated with Ri (i.e., points in the current cluster)
12: end for
13: end while
14: return cl

This algorithm describes the application of K-means clustering to data points that have a geographic component, here using the Haversine distance metric to calculate distances between points on the surface of a sphere. The algorithm starts by randomly selecting the initial cluster centroids from the data points, and then iteratively associates each data point with the nearest centroid using the Haversine distance metric. After all data points are assigned to clusters, each cluster centroid is updated to be the mean of all data points belonging to that cluster. This process is repeated until the centroids no longer change, and the resulting clusters are returned. K-means with Haversine distance can be used to analyze and cluster data with geographic attributes effectively, such as recommended node locations based on distribution of controller with the fiber link connection across a geographic region of SANReN Topology Network

4.6 Controller Placement Problem Leveraging Emulation

The findings of controller placement that were previously displayed were entirely based on mathematical modeling. In this part, we'll go through a technique for locating SDN controllers' best and worst locations utilizing the Mininet orchestration platform for emulation, which is essential for simulating a true SDN deployment because it may incorporate many of the real implementation impacts. As a crucial performance measure, we employ controller to node latency (delay of propagation, queuing, and processing). Our major objective is to compare and validate the results of our mathematical formulation on the ideal places to deploy the controller in a wide area network (WAN). We chose a Ryu controller (version 1.3) for the controlplane because of its distributed core, which increases the controlplane's robustness by providing backup control in the case of network failure (Kobo et al., 2017). Additionally, Ryu's distributed core is self-coordinating and facilitates load sharing through dataplane fragmentation.

High intercontroller communication efficiency is ensured by the innovative east/westbound interface of this controller. The node-to-controller latency is also decreased when a geographically dispersed core is used, which enhances the controller responsiveness as experienced by the network nodes. A model of a local national backbone termed SANReN, the same network we utilized in the preceding procedure, is used to evaluate the suggested emulation strategy. However, it should be highlighted that our strategy is general and may be applied to improve any other network.

4.6.1 Experimental Evaluation

Experiments are assessed using the number of controllers that was discovered that was found out Mathematical Model above. The experiment will make use of emulation in an effort to address the controller placement difficulty. The experiment was tested within an effective number of controllers that we obtain using the technique previously mentioned above that was used to determine the ideal number of controllers:

4.6.1.1 Experimental setup

For each node, is utilized to calculate the average latency: We initially install the Ryu controller in the same region as the first Open Flow switch node in order to determine the best controller placements (using the Haversine great circle approach and the Linux TC utility). The following action is to cause a packetIn message to be sent to the controller. By creating traffic flows between all pairs, or between this node and every other node in the SANReN topology, this is accomplished.

To do this, we create an ICMP packet for each pair using the ping tool. The results of the ICMP pinging are then computed to get the overall average latency (roundtrip time) from the node to every other node in the network. Each node in the SANReN architecture goes through this process once more. We repeat the aforementioned process to make sure the findings are accurate and trustworthy. 5 times within a soft idle timeout of 5 seconds for the controller input (the soft idle timeout determines when a controller flow expires).

After we get optimal number of controllers in the stated topology, we are going to examine the topology with creating the network with real network simulator called Mininet and Ryu controllers and then we can check what placement of controller to get average and worst latency. Given that the connections between the switches are fiber, where speed is roughly equivalent to.

$$propagation\ latency\ (sec) = \ distance(m)/speed(m/s) \qquad \text{Equation 4. 1}$$

Using the nodes' precise GPS coordinates and the Haversine great circle method, the distances between nodes are determined. Running on a different workstation is the Mininet version 2.2.2 dataplane emulator with default settings for all trials (with 8 CPUs, 16 GB RAM and 1 TB HDD). There is a distinct data route ID for each switch in the dataplane (DPID). Through port 6633 of the controller and a sluggish WiFi router, the controlplane and dataplane are connected. The Linux Traffic Control (TC) software, which is installed on the system used for dataplane emulation, is used to set the control link parameters with the presumption that the best location for the controller is adjacent to one of the switches. Python 3.8 was the programming language utilized to create the software.

4.6.2 Controller Placement

Fig. 4.3 below is a flow chart that describes our methodology we are going on for controller placement. if as an example we take single controller situation as number of controllers.

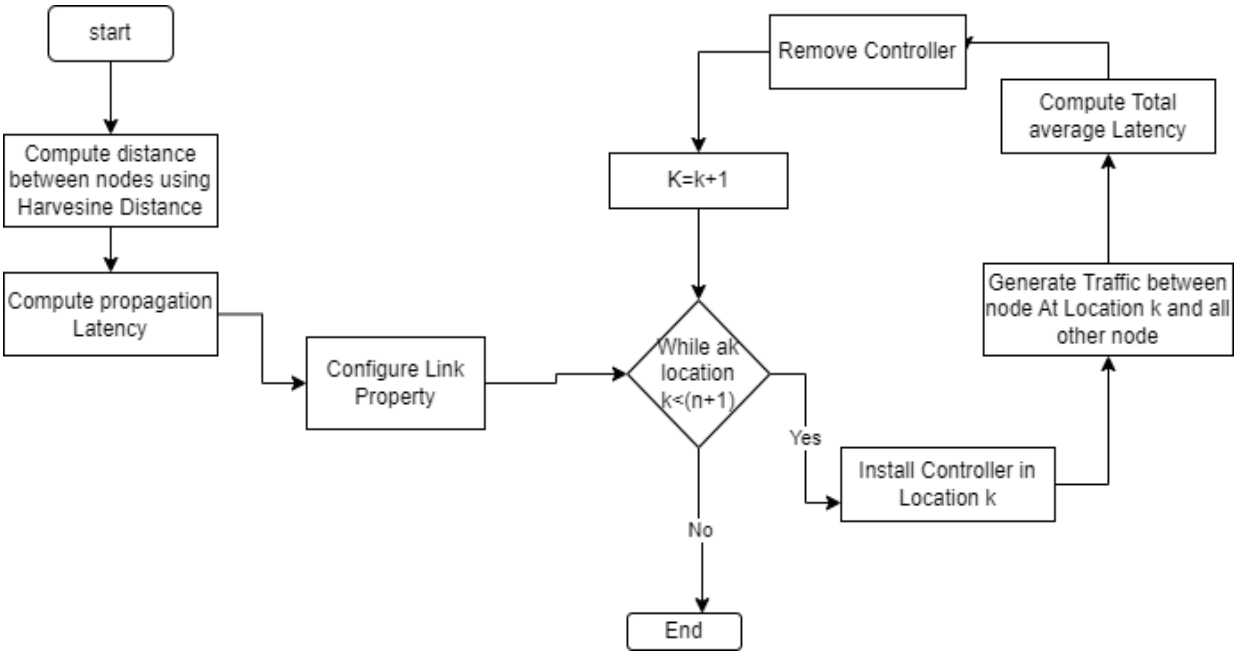

Figure 4. 3 Diagram of a flow chart for placing a controller with a certain number of controllers

we found out the flow chart for controller placement is as follows in the fig below. the same way of procedures we are going out for more than one controller case of placement (where n is the total number of possible controller placement sites, or the total number of nodes in a particular topology). There are a total of 7 possible places for controller installation in the SANReN network because n equals 7.

CHAPTER FIVE
IMPLEMENTATION DETAILS

5.1 Overview

This chapter describes how the suggested solution was put into practice. the use of the suggested adjacency matrix, unsupervised machine learning algorithms including silhouette analysis, gap statics, and code segmentation to determine the ideal number of controllers. Important code segments were provided in this area, but significant detail codes are shown in the appendix section.

5.2 Working Environments

5.2.1 Desktop Computers with Hardware Utilities

- Hard Disk: set to have two terabytes' worth of storage installed.
- SSD: 128GB configurable SSD.
- Inter(R) Core (TM) i7-7700HQ CPU at 2.80GHz (8 CPUs), about 2.8GHz
- Processing for graphics for use in training when computing time quickly. Super installed GeForce GTX 1050 from NVIDIA.
- Physical Memory (RAM): 16.00 GB installed memory.

5.2.2 Desktop Computers with Software Configuration

- (Ubuntu Desktop 20.04 LTS64 bit running on a computer with the following specs: Intel(R) Core (TM) i75600U CPU, with 4 cores (8 threads), 2.60 GHz clock speed, 8 GB of RAM, and 250 GB of storage space.
- (System Model: - Intel(R) Core (TM) i75600U CPU, with 4 cores and 8 threads, running at 2.60 GHz, 8 GB of RAM, and 250 GB of storage space.

5.3 Setup Environments

5.3.1 Design Tools

Design tools are media used for the production, expression, and interpretation of design concepts. Edraw Max is used for the construction of the proposed system. It is a lightweight and efficient graphic design platform for designing professional-looking flowcharts, network diagrams, UML diagrams, and more. This method is chosen because of.

- It has lots of high-quality shapes, example, and template,
- It easily visualizes complex information with a wide range of diagrams.
- Works with MS Office well and others.

5.3.2 Application Software

Development Environment and Tools - For successful completion of our thesis, the right Development Environment and Tools are absolutely essential. To ensure efficient development of our proposed problem and solution approach, we make use of Code Editors, Debugging Utilities, and Version Control Systems. The combination of these tools helps us to view different features and debug issues swiftly, while also allowing us to easily manage any changes made in our work, investing in the correct environment and tools is invaluable value for the success of this research!

Ryu Controller- is an open-source software-defined networking (SDN) controller designed to provide rich network control and flexibility for large, dynamic networks. Utilizing the Ryu controller in Mininet to examine SDN controller placement in WAN topologies proves more advantageous as its lightweight design, easier customization, and robust documentation. This ensures quicker deployment and more flexibility in meeting specific requirements while investigating SDN controller placement strategies in WAN environments.

Mininet emulator- is an open-source network emulator software Suite developed by Stanford University, which allows users to simulate a virtual network in order to test or develop protocols and applications. Mininet supports the emulation of traditional Layer 2/3 networks as well as Software Defined Networking (SDN). It provides a comprehensive platform for emulating, simulating and testing real-world networks.

Microsoft office packages: it is a technique used to write a thesis document, and Visio is used to design diagrams and flowcharts for the proposed system. Excel is used to mark images with their file name along with their corresponding label type number. **Mendeley Desktop**: it is an effective reference manager platform that acts as a scholarly, social network for the reference of related works.

MATLAB- software provides functions for plotting graphs, generating statistical models and performing numerical computations. Additionally, its powerful language allows users to write their own code for more personalized results. The main strength of MATLAB lies in its ability to solve complex problems quickly and accurately Additionally, MATLAB is relatively easier to use than other programming languages making it an ideal choice for us for coding to solve complex problems efficiently.

5.3.2 Integrated Development Environment and Editors

MATLAB Editor: is the default environment for writing, editing, debugging, and executing

MATLAB scripts, functions, and code.

Visual-Studio-Code is an IDE that may be used to edit Python, C++, and other programming language code. It is readily setup with various Python environments. In addition to the Jupiter notebook, version 1.51.1 has the community and 64-bit support installed for implementation.
Terminal window - that are available with in Linux OS to run script file.

5.3.3 Programming Language and Module Libraries

Python: Due to version incompatibility, version 3.8 of the programming language was utilized to implement the suggested remedy. On the most recent version of Python, several DL modules do not install correctly.

Python libraries- the following python libraries are used with this Research work

- **Igraph**: this is a library collection for creating and manipulating graphs and analyzing networks
- **Matplotlib**: A Python plotting library
- **Numpy**: a library that allows manipulation of large multi-dimensional arrays and matrix.

5.4 Training Procedure

The training procedure in this proposed research

- Collect massive dataset from Internet Topology Zoo
- Get the Graph structure by reading gml data
- find coordinates of node and then get Adjacent matrix
- Building Mathematical model
- Compile the model
- Evaluate the model

Let see all the above set of training procedures in a systematic and detailed manner.

5.5 Implementation Techniques for Dataset Processing

5.5.1 Dataset Description

In this thesis, the South African National Research Network is used as a case study. The datasets from this network were gathered in graphical markup language format from actual Internet2 network topology. Where The Geography Markup Language (GML) is an XML grammar defined by the Open Geospatial Consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic

transactions on the Internet. The code snippet for reading the South African National Research network trace dataset is shown in the accompanying Figure.

```
function graph = read_gml(Sanren.gml)
% READ_GML returns hierarchical datastructure of information in % .gml fil
    content = fileread(filename);
    content = strrep(content, sprintf('\n'), '');
    %for consistent input
    content = strrep(content, '[', ' [ ');
    content = strrep(content, ']', ' ] ');
    %remove multiple whitespaces in the content:
    loop = 1;
    i = 1;
    while loop
        i = i + 1;
        if size(strfind(content, repmat(' ', 1, i)), 2) == 0
                loop = 0;
            end
        end
    for ii = 1:(i-2)
         content = strrep(content, '  ', ' ');
        end
    content = regexp(content, ' ', 'split');
    [i, graph] = get_next_struct(content, 1);
     % make next line a comment if information doesnt contain an actual
    % graph.
    graph = graph.graph;
    end
```

Sample code 5.1 Reading the gml datasets

5.5.2 Get Graph Structure from Dataset

After we get our data set it must have to be prepared as the requirement to get the graph structure of the network topology. The following snipe of code is likely part of a larger program or analysis being conducted in the research paper. The purpose of this specific code is to read in a graph structure from a file for further analysis or manipulation.

```
clear all;
clc;
graph = read_gml('Sanren.gml'); % Get the Graph structure
```

Sample code 5.2 Feature read operations

5.5.3 Scaling and Transforming the Data

Before adding the data to the recommended model, it is pre-processed through a number of processes. First, information on node labels, coordinate matrices, and graph structure is discovered. A code snippet for Graph structure, getting the coordinates matrix, and getting node labels is shown in the accompanying Figure. These are used to test and train the suggested model.

```
% Get coordinates matrix
D=[];
for i=1:length(graph.node)
    D(i,1)=graph.node(i).Longitude;
    D(i,2)=graph.node(i).Latitude;
end
D;

%get the labels of nodes
L={};
%M=num2str(graph.node(1).label)
for i=1:length(graph.node)
    L{1,i}=num2str(graph.node(i).id); %node ids
    L{2,i}=num2str(graph.node(i).label); %node labels

end
L;

A = get_adjacency_matrix(graph); % Adjacency Matrix: row: Source, column: Target
```

Sample code 5.3 Scaling and transforming the data

5.5.4 Create Mathematical Model

To create the sequential model is used several features, distance, minimum intra-cluster propagation latency variation and metrics.

5.5.4.1 Distance Calculation

It mentions the calculation of a "graphical distance matrix" using a specific algorithm. The algorithm involves creating an empty matrix called "dratrix1" and a vector called "chatr" with dimensions 1x2. Then, a loop is initiated with the variable "i" starting at 1 and incrementing by 1 until it reaches the length of "maximum". Within the loop, two variables "A" and "V" are defined using the values of "i" and "j" (which starts at "i-1" and 0, respectively). The values of "A" and "V" are calculated using a function "D" with inputs "i", "j", and "n (j, :2)". The loop continues until all values in the matrix have been calculated. The purpose of this algorithm is to calculate a distance matrix for a set of data points, which can be used to analyze the relationships between the points.

```
%Calculate geographical distance matrix
dMatrix1=[]; dMatrix2=[];
for i=1:length(D)
    for j=1:length(D)
[dMatrix1(i,j) dMatrix2(i,j)]=lldistkm(D(i,:),D(j,:));
   end
end
dMatrix1;
dMatrix2;
%Verify results for Sanren backbone
latlon=[-29.8500  31.0167;-33.0153  27.9116];
[d1km d2km]=lldistkm(latlon(1,:),latlon(2,:));
```

Sample code 5.4 Creating the model

5.5.4.2 Latency Calculate

The average propagation latency (which is the overall propagation latency) and the worst-case propagation latency (which is the maximum network latency) code snipe is as follows

```
%Calculate average latency and worst-case latency.
L_average= (sum(sum(dist)))/(length(graph.node)); %average latency
L_worst=max(dist(:)); %worst-case latency
```

Sample code 5.5 Building the generator of the model

5.6 Compile Model

Once the network is built, the model will be trained with the required callback like clustering quality, average latency.

5.6.1 Silhouette Analysis

In software-defined wide area networks (WANs), controller placement and clustering play a critical role in optimizing network performance. The number of controllers and their optimal placement is essential to enable efficient distribution of traffic across the network. By utilizing the correct number of controllers, network operators can encourage the growth of high-quality clusters, which enable insightful analysis and informed decision-making. High-quality clusters allow operators to analyze data and interpret network behavior patterns, identify failures and optimize resource utilization, and anticipate potential issues before they occur. Accurately placing controllers in the network ensures that network components are efficiently and adequately monitored, minimizing delays and allowing fast and effective decision-making. Consequently, the clustering's quality plays a crucial role in software-defined WANs, and network operators must strategically choose the number of controllers to create high-quality clusters and ultimately achieve optimal network performance

```
%Evaluate best number of controllers to use
[idxk]=clust_med(D,k,n)
%Evaluate silhouette value for different no. of clusters
eva = evalclusters(D,idxk,'Silhouette','Distance',@dist_matrix)
%Plot silhouette results for varrying cluster count
figure;
plot(eva)
xlabel ('Number of clusters/controllers ','FontSize',30,'FontWeight','bold')
ylabel ('Silhouette Value','FontSize',30,'FontWeight','bold')
% title ('\fontsize{16}Evaluation of cluster quality for varrying number of controllers')
```

Sample code 5.6 find out optimal number of controllers using silhouette method

5.6.3 Cost-latency trade-off analysis

In software-defined networks, choosing the optimal placement of controllers is crucial to achieving high network performance. The number of controllers required and their optimal placement to minimize latency and distribute traffic across the network are carefully considered during the network planning phase. To optimize the deployment of controllers, network operators can use a specialized tool. The code snippet written in .m extension of MATLAB code generates information on the cost of deployment based on the number of controllers taken into consideration.

```
%Analysis tradeoff between cost of installing new controller and performace
cost_benefit=[];
controller_cost=1;
k=4;

for i=1:k
    cost_benefit(1,i)=(controller_cost*i)/(L_average(i));

end
%
k=[1 2 3 4];
figure;
AxesH = axes('Xlim', [1, 4], 'XTick', 1:1:4, 'NextPlot', 'add');
line(k,cost_benefit,'MarkerSize',20,'Marker','.','Color','b')
hold on
box on;
plot(k,cost_benefit, '-b')
set(gca,'fontsize',20)
% title('Optimal number of controllers based on cost benefit');
xlabel ('Number of controllers','FontSize',30, 'FontWeight','bold')
ylabel ('Cost benefit ($/ms)','FontSize',30,'FontWeight','bold')
```

Sample code 5.8 Number of controllers to use while accounting for deployment costs.

The cost of deployment often depends on the number of controllers utilized in a given network, and the proper placement of these controllers can have a significant influence on their effectiveness and efficiency. The code may use algorithms and mathematical models to identify the most cost-effective deployment strategies by iterating through different combinations of controllers and calculating their associated costs. This approach can inform decision-making processes by

providing valuable insights into the optimal number of controllers to deploy and their placement in the network. The result is a more streamlined and efficient network that improves overall performance and reduces costs associated with deployment. Therefore, leveraging the information generated by this code can have a significant impact on the success of a wide range of projects that rely on networks with multiple controllers.

5.7 Mathematical Model Evaluation with Emulation in WAN Network

5.7.1 Controller placement

The code snippet under evaluation is designed to examine the performance of a mathematical model in a wide area network (WAN) using emulation. The aim of this evaluation is to investigate the placement of a controller in the network, with a focus on optimizing its performance. To achieve this objective, the python script uses machine learning algorithms and mathematical approaches to determine the best placement of the controller that would meet specific performance objectives such as propagation latency and the cost of deployment.

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
import igraph as ig
import networkx as nx
import matplotlib.pyplot as plt
import haversine
import numpy as np
import haversine as hv


def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='192.168.0.180',
                         protocol='tcp',
                         port=6633)
```

Sample code 5.9

The results obtained will shed more light on the minimum number of controllers required and the most optimal location for the controller, without the need for expensive and time-consuming physical testing. This research is particularly relevant to network operators looking to optimize the performance of their network through the proper placement of controllers.

The code is a snipped are used to utilizes the Mininet platform to create a network topology and simulate network behavior. It begins with importing necessary modules such as Mininet, NetworkX, Matplotlib, and iGraph. Next, the script defines a function called myNetwork(), which creates a new Mininet object and sets up the network topology by adding switches and hosts. The script also defines two remote controllers and adds them to the Mininet object, the left part of the script also computes link delays by using the haversine library to compute the distance between the coordinates of the source and destination switches and continued in this way. Overall, this script defines a complex network topology and simulates its behavior, allowing for the testing of various controller placement.

CHAPTER SIX
RESULT, EVALUATION AND DISCUSSION

6.1 Overview

This chapter looks at how employing a mathematical model might produce the greatest results for network control. We especially use an ML technique called silhouette analysis to determine the network's optimal number of controllers in order to assess the clustering's quality. We look at the cost-latency trade-off to further clarify the impact of our results. The k-means algorithm is used to discover the best places to position each controller after determining the ideal number of controllers. This is achieved by considering the relationships between the number of controllers, latency, and controller location. To verify the accuracy of our mathematical model, we simulate the controller's position across a wide area network (WAN) and compare the results with our theoretical predictions.

6.2 Result of the Study

The outcomes of using the strategies outlined in Chapter 4 are shown and discussed in this section.

6.2.1 Finding the Optimal Number of Controllers from Dataset Preparation

Network topology datasets from the South African National Research Institution are used to evaluate the effectiveness of K-means clustering controller placement. The first stage is to create a mathematical model using machine learning to determine the clustering quality for determining the ideal number of controllers to be placed in the topology. The quality of cluster using silhouette utilized to determine the ideal number of controllers with the typical intracluster propagation latency is depicted in the figure 6.1,6.2,6.3 below. In this thesis, silhouette analysis a technique previously used to demonstrate the quality of clustering—is proposed(Mamushiane et al., n.d.-b).

6.2.1.1 Silhouette analysis

We may assess clustering quality at various SDN controller deployment densities by accounting for the average intra-cluster propagation time. The plots' blue horizontal bars each correspond to a switch and the corresponding silhouette score. The distance between each switch and every other switch, both inside and outside of its cluster, is represented by a silhouette score. Scores for silhouettes fall between [-1, 1]. A score that is closer to +1 is preferred since it denotes a collection of switches that are close to one another. In contrast, silhouette scores of -1 imply

inadequate grouping and significant internal dissimilarity. The switch is either engaged or extremely close to the decision boundary between two neighboring clusters when the value is zero (Lovmar et al., 2005). We were able to decide how many controllers to install in order to improve network accuracy and efficiency thanks to this clustering quality analysis. The clustering quality is represented in the graph below. The appropriate number of controllers to employ with the design depicted in the accompanying figure was determined using a procedure.

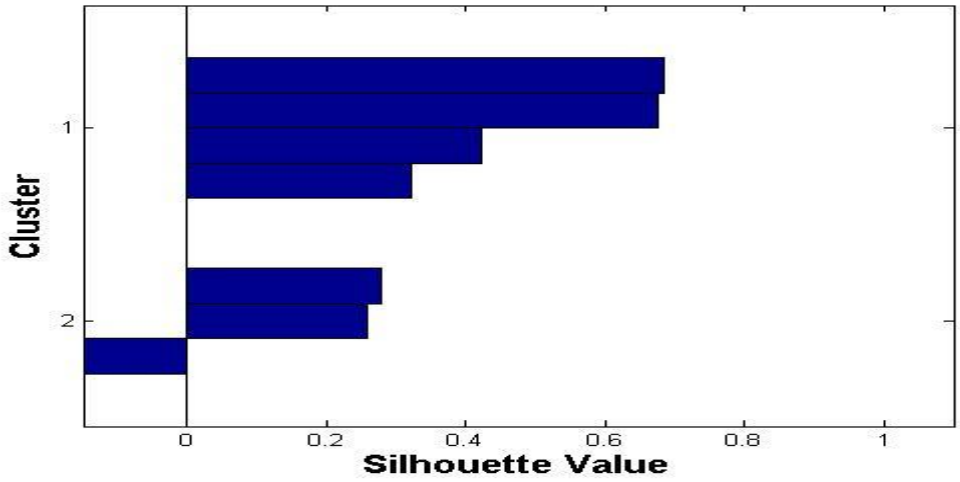

Figure 6. 1 clustering quality when two controllers are used

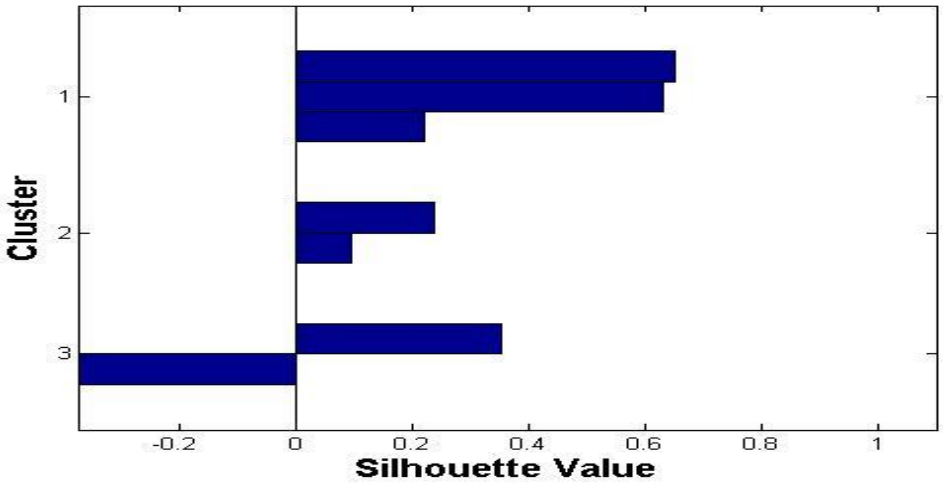

Figure 6. 2 clustering quality when Three controllers are implemented.

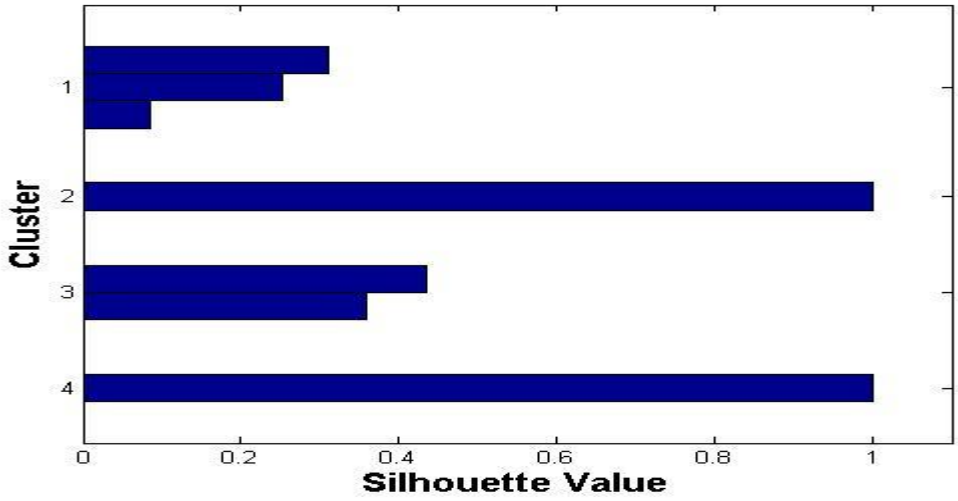

Figure 6. 3 clustering quality when Four controllers are implemented.

Parameters can be specified in many different ways. The first two architecture-related parameters for controller placement are the number of clusters, which is based on an earlier study(Mamushiane et al., n.d.-b), the silhouette score, and the number of nodes for the optimization of controller placement. i.e., the number of nodes is 7, as the suggested model can be scaled down, and when the model train has more nodes than 7, the computer's memory is insufficient. Table 6.1 displays the experiment results for the suggested model given the number of controllers and the silhotte value. The demonstrated experimental result shows the efficacy of the suggested strategy for clustering quality that yields high silhouette scores.

Table 6. 1 Silhotte value for clustering quality with different number of controllers

Number of controllers	Silhouette value for Clustering Quality
Number of controllers 2	0, 0.2, 0.4, 0.6
Number of controllers 3	-0.2, 0, 0.2, 0.4, 0.6, 0.8
Number of Controller 4	0, 0.2, 0.4, 0.6, 0.8, 1

Due to the occurrence of clusters with extremely low silhouette scores and large changes in the size of the silhouette plots, dividing into three clusters would produce poor clustering quality. When partitioning into 2 cluster as other metrics used to see clustering quality would be the

average intra-cluster propagation latency that shows proximity of switches within the same cluster with the found silhotte value. 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.

Our results indicate that deploying 3 or 4 controllers (shown in Fig. 6.1 and 6.2 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.

The factors listed in the previous sentence have a significant impact on the number of controllers (RQ2) and also Before deciding the placement of controller in the networks, the number of controllers with highest average silhotte value that have to be taken can help with finding optimal number of controllers we are used for placement of controller.

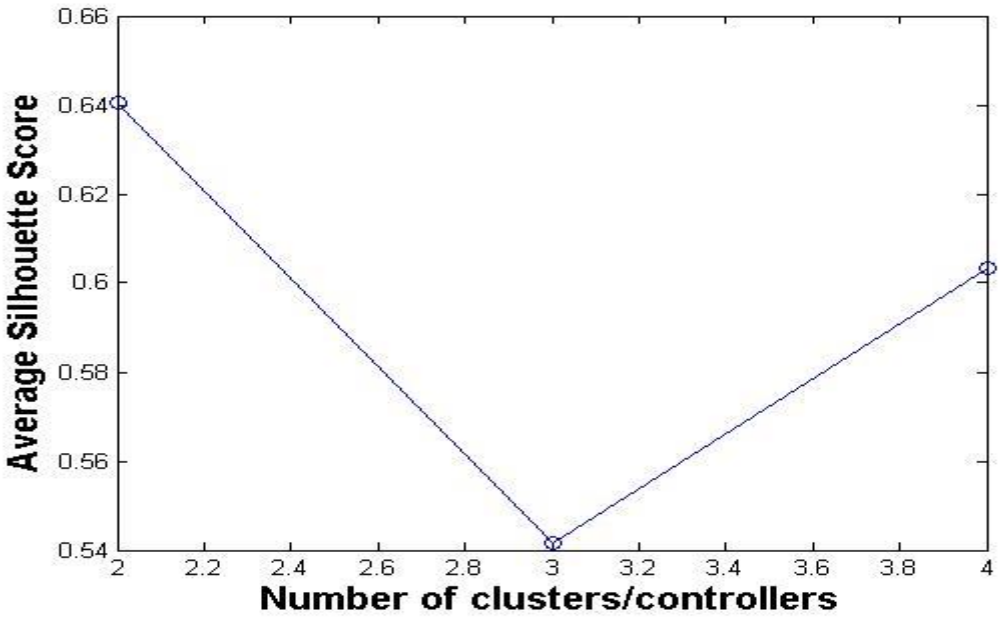

Figure 6. 4 Silhouette Analysis

The increased proximity of nodes within the same cluster is what causes the average silhouette score to grow from 3 to 4 controllers (as seen in Fig. 6.2.6.3). Deploying 4 controllers would result in the following network partitions, given that SANReN consists of 7 nodes:

- 2 clusters with 1 node per cluster;
- 1 cluster with 2 nodes; and
- 1 cluster with 3 nodes.

On the other hand, deploying 3 controllers would result in the following network partitions:

• 2 clusters with 3 nodes per cluster; and

• 1 cluster with 1 node.

Given the SANReN topology's sparse locations, it stands to reason that dividing the network into 4 clusters, which has fewer nodes per cluster than 3 clusters, will produce a higher average silhouette score. Deploying 2 controllers on the SANReN network is therefore the best choice as it will result in shorter propagation latency and a balanced switch-to-controller distribution. This is evident by the excellent silhouette score attained with a setting of two controllers. Although using 4 controllers will increase network stability and produce a fairly acceptable clustering quality, it is likely to cause high intercontroller latency (due to the frequent state information exchange between controllers). However, if latency and cost are topmost priority, then 2 controllers are recommended.

6.2.2 Cost-latency trade-off analysis

The average latency is the sum propagation delay computed for a variable number of controllers using the K-means algorithm mentioned in Section 4.3.2. Our examination of the trade-off between cost and controller count reveals a linear increase in deployment costs with more controllers put in the network, as illustrated in Fig. 6.5 below

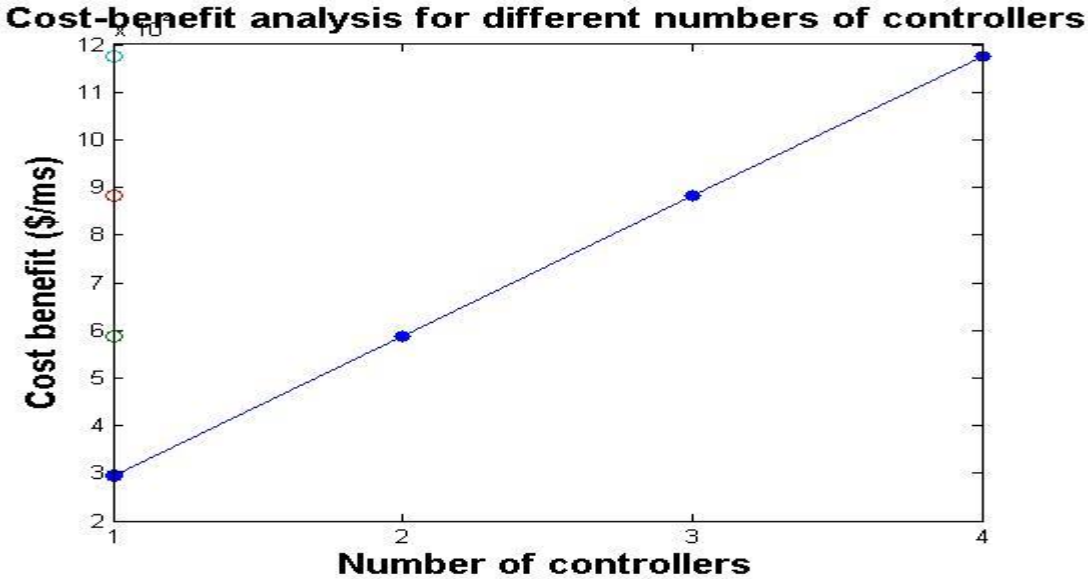

Figure 6. 5 Trade-off between cost and latency for varying number of controllers.

6.3 Overall Network Latency and Optimal Controller Placement

After using the Silhouette analysis to determine the appropriate number of controllers, the following step is to select the best locations to put the two SDN controllers that have been indicated. To find these locations, we use the suggested K-means technique from Section 4.3.2. The results (shown in Fig. 6.5) demonstrate that Johnsburg and East London are the ideal locations to position two controllers, with an average propagation latency of Lavg = 0.0016. The selection of these locations guarantees the best possible network performance for switch communication in the SANReN network. The network would function worse if the controllers were located in Port Elizabeth and Bloemfontein, with a propagation latency of Lwc = 0.0023 in the worst-case scenario.

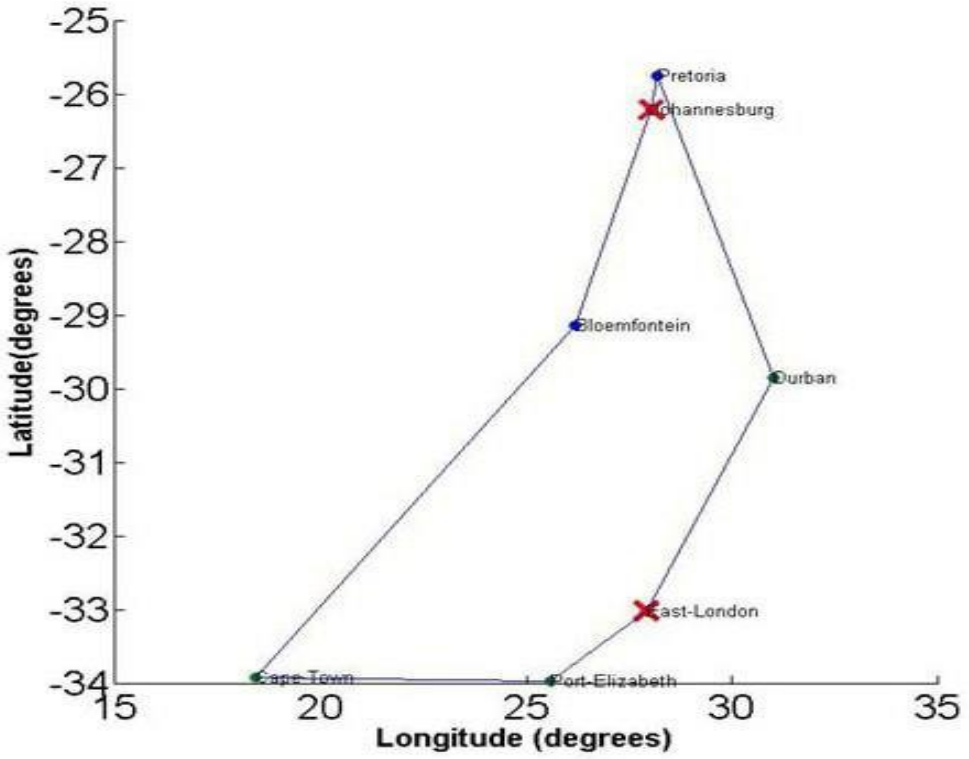

Figure 6. 6 Best and worst placements of two controllers on SANReN backbone

Table 3 presents the effect of increasing the number of controllers (k) on average and worst-case latency.

Controller Placement with K-means clustering

As k-means algorithm iteratively attempts to identify the optimal placement of the controllers by minimizing the total distance between the nodes and their closest controller. This allows the

network to be operated more efficiently, with a lower latency that used for for controller placement of controller as(RQ1,RQ3)and greater resiliency to faults. As such, k-means clustering can be a highly effective method for identifying optimal controller placement in various network architectures.

Table 6. 2 Average ($Lavg$) and worst-case (Lwc) latency for varying number of controllers with k-means clustering

	K=1	K=2	K=3	K=4
L_{avg}(ms)	0.003	0.0016	0.0008792	0.000484
L_{wrst}(ms)	0.007	0.0023	0.004	0.0002382
Names of Location for L_{avg}	Durban	Johannesburg East London	Cabe Town Durban East London Pretoria	Cabe Town Durban East London Johannesburg,
Names of Location for L_{wrst}	Bloemfontein	Bloemfontein Port-Elizabet	Durban Johannesburg Port-Elizabet	Bloemfontein Durban Johannesburg Port-Elizabet

The findings of the study reveal the significant impact of controller number on propagation delay in a network system. Our results indicate that there is a substantial reduction in latency when the number of controllers is increased from k=1 to k=2, with an average latency reduction of around 49% and worst-case latency reduction of 75%. The substantial reduction continues as the number of controllers is increased to k=3. However, the impact on latency becomes less significant when more than three controllers are implemented. This trend is illustrated in Figure 6.6, where the level of impact on latency reduction from deploying additional controllers diminishes after three. It is noteworthy that the optimal number of controllers for a given application may vary depending on various environmental factors, such as network architecture, transmission distance, and type of data being transmitted. Therefore, the results of this study can provide a valuable framework for optimizing controller count for efficiently reducing propagation delay to its lowest possible level while maintaining a reasonable cost-benefit ratio in network design.

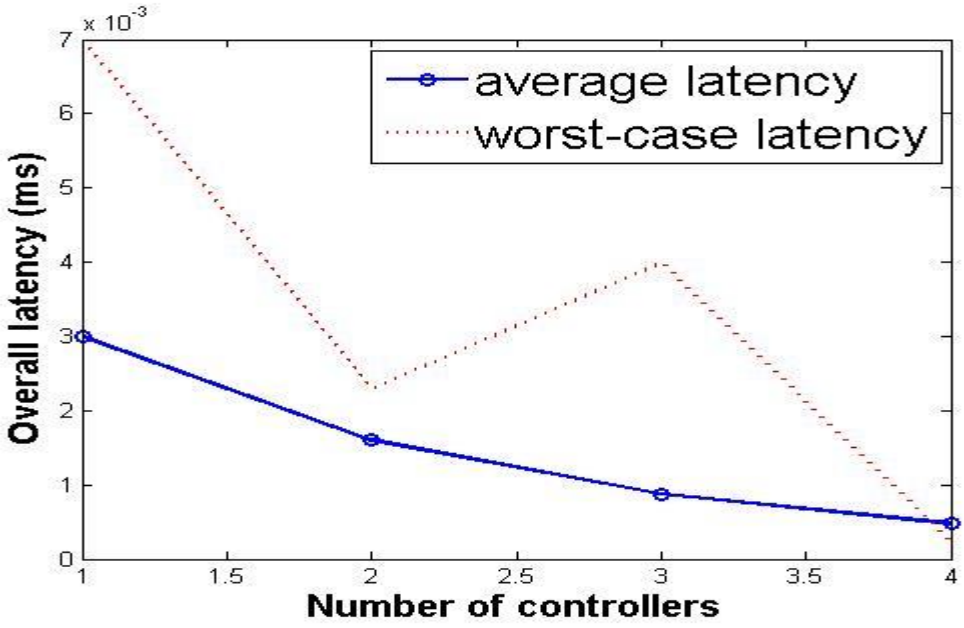

Figure 6. 7 Relation between number of controllers and latency

Controller Placement with K-Medoids Clustering

The k-medoids algorithm is a clustering technique that partitions a set of data points into K clusters. The process is iterative and seeks to assign the data points to the nearest cluster using the actual data point as the centroid, in contrast to the mean value of points as in the k-means algorithm. However, the k-medoids algorithm has a significant disadvantage over the k-means algorithm when it comes to computing average latency values. The k-medoids algorithm requires more computational resources to calculate the average latency from the data points within each cluster as shown in the table below and their value is high compared to the same data set used for k means clustering above (RQ4). This is because each data point must be compared to every other point within the cluster, and the distances between each data point and all other points must be evaluated. This additional processing inevitably increases the computational complexity, causing increased latency in the computation of the average latency values.

Table 6. 3 Average ($Lavg$) and worst-case (Lwc) latency with Kmedoids clustering

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

6.4 Controller Placement on Emulated WAN

The findings of controller placement that were previously displayed were entirely based on mathematical modeling. Under this section we are going to see the mathematical finding with emulation platform.

6.4.1 Controller placement

We employ a variety of controller counts to decide controller placement. The ideal position for a controller in the case of a single controller (where n is the total number of possible locations for a controller, or the total number of nodes in a particular topology). In the SANREN network, there are a total of 7 locations where controller installation is feasible. The following method is used to determine the typical latency for each node: To determine which controller efficiency are best,

- The first OpenFlow switch node and the Ryu controller are originally installed in the same region (using the Haversine great circle approach and the Linux TC utility).
- The following action is to cause a packetIn message to be sent to the controller. By creating traffic flows between all pairs, or between this node and every other node in the SANReN topology, this is accomplished. To do this, we create an ICMP packet for each pair using the ping tool.
- The next step will cause the controller to receive a packetIn message. This is done by establishing traffic flows between all pairs, or between this node and each other node in the SANReN topology. To accomplish this, we use the ping tool to generate an ICMP packet for each pair.

6.5 Evaluation Metrics

6.5.1 Average Latency

By examining the communication delay between network devices and SDN controllers, as shown in the aforementioned figure 6.7, average latency is utilized as an evaluation metric rather than worst case latency for Software-Defined Networking (SDN) controller placement in Wide Area Networks (WAN). As a lower latency value can be utilized as assessment metrics, this measure is crucial because greater latencies can cause congestion, longer reaction times, and overall lower network performance. The evaluation of latency in Figure 6.7 above compares the controller's average latency against the worst-case scenario, with the average latency being utilized to see the least latency number. This is used as a metric to assess the suggested model. The worst-case delay is much higher than average value. This value serves as an evaluation metric.

6.5.1 Cost of Installation

We determine the cost of installation for the topology we employ. Figure 6.5 demonstrates the model's effectiveness and ideal installation cost. Cost value will increase as the number of controllers increases. The model adjusts to any Controller deployment cost and learns any pattern or context. As a result, the smallest cost for controller deployment with the lowest possible average latency will be chosen as one of the assessment metrics in the suggested architecture.

6.6 Discussion

6.6.1 Discussion on Average Latency

The calculated Average Latency for the suggested model indicates that 0.0016 is the minimal deployment speed for calculation. is displayed in the following table. Therefore, we may choose the ideal number of controllers with the lowest average latency because the communication delays between switches are almost nil.

Table 6. 4 Average latency for Proposed Clustering approach

Clustering algorithm	Average latency in (ms) at K=1	Average latency in (ms) at K=2	Average latency in (ms) at K=3	Average latency in (ms) at K=4
K means	0.003	0.0016	0.0008792	0.000484

6.6.2 Discussion on Cost

Cost for deployment with good clustering quality, as shown in figure 6.5 above, is one of the factors considered to establish the appropriate location of the controller in this design. The findings indicate that using a single controller is the most effective strategy to reduce the trade-off between cost and network performance. However, we advocate using two controllers to ensure network scalability and failover. Our average latency at k=2 is the best suited value to deploy on SANReN as it has the least communication delay compared to other values. As the number of controllers increased, deployment costs increased linearly. This is largely due to the fact that two controllers are the second-best option with the lowest tradeoff. It is significant.

6.6.3 Controller Placement on Emulated WAN

The findings of controller placement that were previously displayed were entirely based on mathematical modeling. Under this section we are going to see the mathematical finding with emulation platform.

6.6.3.1 Evaluation Metrics

This study consists of evaluation metrics with the goal of solving the controller placement issue by utilizing emulation. Two scenarios are included in the assessment analyses: To see the placement with one controller, and to see Placements with the use of two controllers.

6.6.3.2 Controller placement

To determine the best position for a controller on a single controller case scenario (where n is the total number of possible locations for a controller, or the total number of nodes in a particular

topology). There are a total of 7 possible places for controller installation in the SANREN network because n equals 7. For each node, the method described below is used to calculate the average latency: To identify the best controller positions,

- The Ryu controller is initially set up in the same region as the first OpenFlow switch node (using the Haversine great circle approach and the Linux TC utility).
- The following action is to cause a packetIn message to be sent to the controller. By creating traffic flows between all pairs, or between this node and every other node in the SANReN topology, this is accomplished. To do this, we create an ICMP packet for each pair using the ping tool.
- The results of the ICMP pinging are then computed to get the overall average latency (roundtrip time) from the node to every other node in the network. Each node in the SANReN architecture goes through this process once more.

In the case of Two controller case the network is divided into two smaller administrative domains, clusters one and two, for the scenario of two controllers. Each of these domains is under the supervision of a separate Ryu instance. The values n1 and n2 represent, respectively, the total number of switches in clusters one and two. The partition results after running the mastership module are as follows: first Ryu instance (C1) is given three switch nodes in the regions of Pretoria, Bloemfontein, and Durban. While the other Ryu instance consists of switches in Johannesburg, Cape Town, East London, and Port Elizabeth,

6.7 Discussion

This section presents and discusses the results obtained from following the procedures described above.

6.7.1 Discussion on Controller placement

The SANReN network has been the subject of our investigation utilizing an emulation platform. The results of our study have led us to the discovery that controller placement without clustering based on the minimum average latency value suggests Pretoria to be the best location to implement a single controller with the lowest average latency (Lavg=0.03ms). However, this only holds true when only one controller is installed. On the contrary, Bloemfontein has been found to be the worst site to place the controller with the highest average latency.

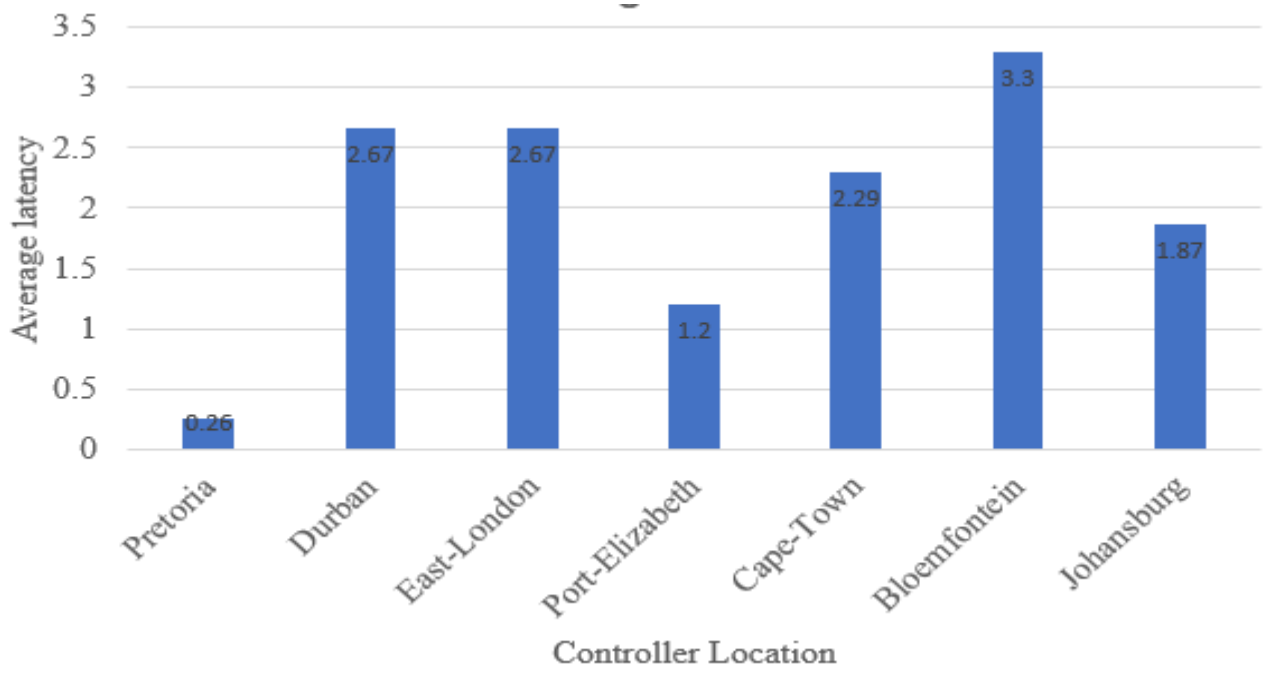

Figure 6. 8 Total average latency for the Ryu controller without clustering

In conclusion, our investigation into the SANReN network using an emulation platform suggests that controller placement without clustering based on the minimum average latency value indicates Pretoria to be the optimal location for a single controller.

In the case of clustering analysis, we find that the two clusters above in mathematical modeling have the best cluster quality. We test those clusters with emulation experiments and obtain the average latency value from there. With this, it divides our nodes into two clusters, and we will see the results of controller placement for each cluster.

The outcomes of the deployment of two controllers in our research, showed that Johannesburg is the optimal placement for cluster 1, and East London is the best location for cluster 2. The deployment of these two controllers helped to improve the efficiency of the system and ensure that the overall operations were running smoothly. So we can conclude that the placement of the controllers was crucial in ensuring that the system's performance was at the optimal level.

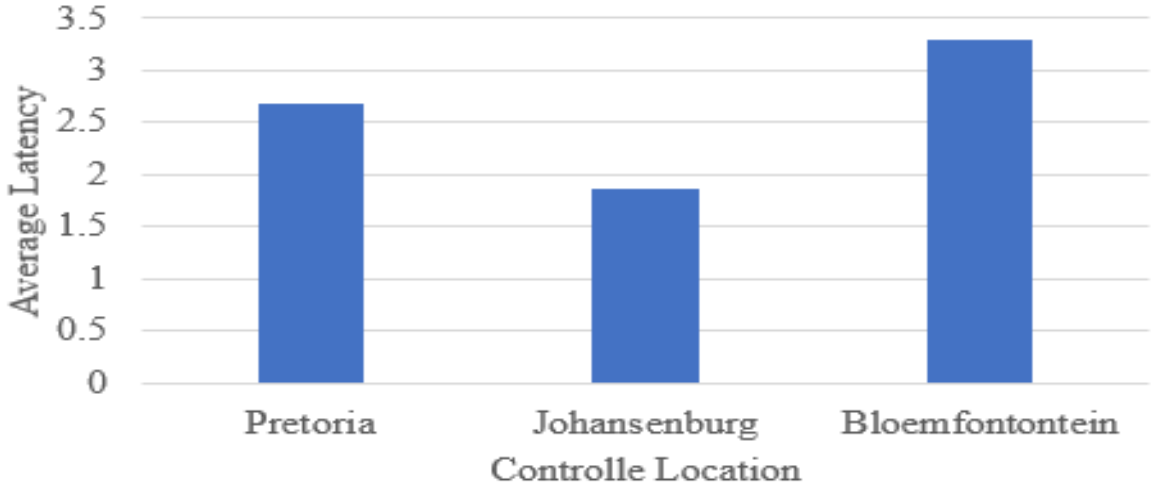

Figure 6. 9 Controller placement first cluster out of the two clusters

Figure 6. 10 Controller placement in second cluster

The findings of our research match those of our mathematical model in the above section. Our model took into account the various variables at play during the deployment of the two controllers and provided us with a framework for understanding the different outcomes we observed. Our research showed that Bloemfontein and Port Elizabeth were the worst sites for clusters one and two, respectively, confirming what our mathematical model predicted. Therefore, the results from the deployment of the two controllers matched the predictions we made in our mathematical model, providing us with confidence in our findings.

6.8 Comparison of the Experimental Result and Discussion

The results of the proposed model and baseline techniques are shown in Figure 6.11 below. The evaluation shows a decrease in latency as the number of K clusters increases. Medoids approach shows higher latency than the Kmeans approach as (RQ4). It is difficult as it requires higher computational complexity, reduced scalability, and difficulty in handling continuous data to make it less attractive for SDN controller placement. K-Means clustering is generally considered a more efficient and scalable solution for such problems.

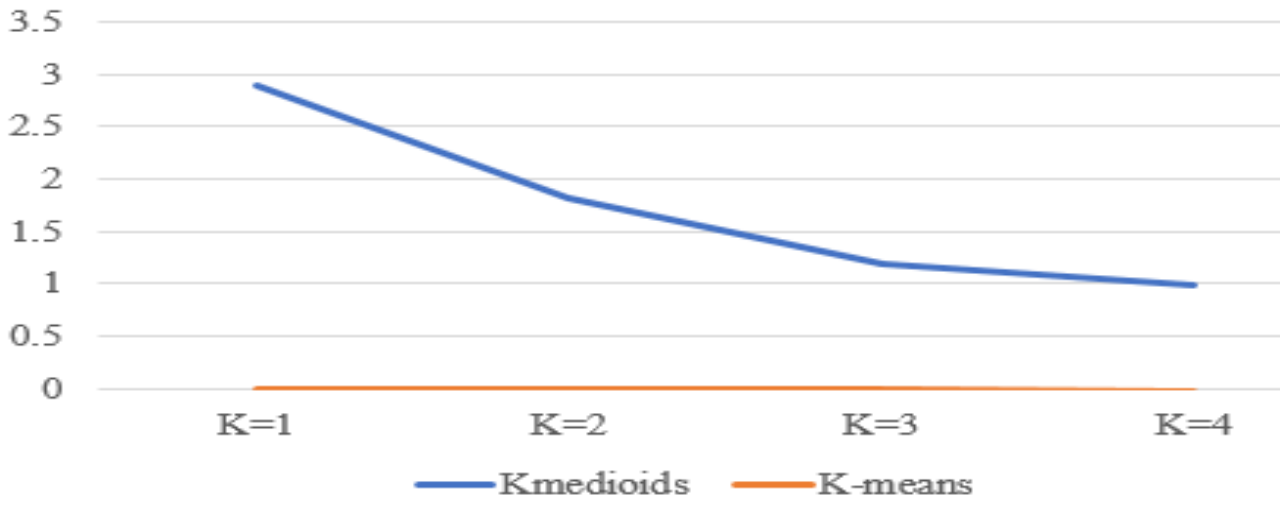

Figure 6. 11 computation latency between the proposed and baseline models

In comparison to earlier work, the Proposed Model still has great quick low computational competition efficiency, as seen in the figure. that was relevant to (RQ2). The value of Average Latency and Cost of Deployment achieved by the proposed model is lower than all the methods when the number of controllers increased. With the increase of number of controllers, the proposed model achieves more significant performance improvements compared to baseline

method. This is because these classic methods cannot effectively make long-term predictions for highly random workloads without obvious regularity. The proposed model can quickly identify the number of controllers and placements with minimum latency so that it can boost performance such as data congestion and slower response times. The suggested model scales out the procedure for large value computing and does not use computational cost. Proposed Model, however, can better handle this issue

CHAPTER SEVEN
CONCLUSSION AND FUTER WORK

7.1 CONCLUSSION

In order to solve the placement issue for SDN controllers, this research takes into account how many and where SDN controllers should be placed in a wide area network, as well as the performance and financial consequences that follow. The research is used using SANReN, a national network from a poor nation. The study consists of mathematical modeling and a technique for getting the findings by simulating a well-liked controller that may be used in actual deployments. The modeling was supported by the simulation, and it was also utilized to determine significant practical restrictions. K-means, Silhouette techniques were used. To determine the ideal number of controllers to deploy in a particular topology, "unsupervised" machine learning called Silhouette Analysis was deployed using graph modeling.

This study considers the trade-off between the price of installing a new SDN controller and performance since network operators are more worried about the cost related to network deployment. This is required to make it easier to choose the right number of controllers to install given the performance needs and financial restrictions. Using the K-means heuristic approach, the best places to deploy the controllers were found. Since there are little to no temporal limits and the applied algorithms are exhaustive, they are perfect for placing static controllers.

We employ exhaustive search on an emulator to solve the controller placement problem, simulating a genuine SDN deployment and verifying the results using our mathematical model. We utilize the Ryu SDN controller because of its built-in distributed self-coordination core. According to the findings of our simulation, using a single controller result in long reaction times since certain switches are placed too distant from the controller. Running a single controller is also insufficient to satisfy the requirements for resilience. Due to the network being separated into two administrative domains, the reaction time was significantly decreased when the number of controllers was raised to two. Additionally, the two controllers cooperated to save control overhead and guarantee network efficiency. The suggested method will be used to address the controller placement problem in any SDN-enabled dataplane. We believe that operators and service providers would benefit from our approach and analysis both during the original design of the SDN-enabled networks and throughout the subsequent incremental design.

7.2. Future Work

In the future, we intend to expand our work by evaluating the security aspect of SDN (Software Defined Networking) controllers. This expansion is driven by the fact that centralizing the network control intelligence creates a single point of attack or failure, thereby increasing the potential risk to the network's security. The assumption we take in this work that is fiber link between, is not valid for the actual SANReN (South African National Research Network) network. However, this simplification was made to facilitate the mathematical model formulation at the time. In the future, our goal is to consider a more complex scenario that takes into account different link bandwidths. As part of this work, we aim to develop a dynamic control traffic load balancing application based on the current resource utilization (such as RAM, CPU, and storage) of the switch. By doing so, we intend to enhance the overall network performance, reliability, and security.

REFERENCES

Amiri, E., & Alizadeh, E. (2021). *Optimized Controller Placement for Software Defined Wide Area Networks*.

Andrew, J., Cajori, F., & Inman, J. (1835). *Haversine formula*. 1–5.

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, *149*(2), 268–281. doi: 10.1016/S0377-2217(02)00761-0

Dhar, M., Bhattacharyya, B. K., Kanti Debbarma, M., & Debbarma, S. (2021). A new optimization technique to solve the latency aware controller placement problem in software defined networks. *Transactions on Emerging Telecommunications Technologies*, *32*(10), 1–17. doi: 10.1002/ett.4316

Fahim, A. M., Salem, A. M., Torkey, F. A., & Ramadan, M. A. (2006). Efficient enhanced k-means clustering algorithm. *Journal of Zhejiang University: Science*, *7*(10), 1626–1633. doi: 10.1631/jzus.2006.A1626

Firouz, N., Masdari, M., Sangar, A. B., & Majidzadeh, K. (2021). A novel controller placement algorithm based on network portioning concept and a hybrid discrete optimization algorithm for multi-controller software-defined networks. In Cluster Computing (Vol. 24, Issue 3). Springer US. doi: 10.1007/s10586-021-03264-w

Foreman, C., Russo, P., Davies, N., Hissaria, P., Proudman, S., Hughes, T., & Limaye, V. (2017). Use of intravenous immunoglobulin therapy for myositis: an audit in South Australian patients. *Internal Medicine Journal*, *47*(1), 112–115. doi: 10.1111/imj.13308

Han, B., Yang, X., & Wang, X. (2019). Dynamic controller-switch mapping assignment with genetic algorithm for multi-controller SDN. *Proceedings - IEEE 17th International Conference on Dependable, Autonomic and Secure Computing, IEEE 17th International Conference on Pervasive Intelligence and Computing, IEEE 5th International Conference on Cloud and Big Data Computing, 4th Cyber Scienc*, 980–986. doi: 10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00179

Ibrar, M., Wang, L., Muntean, G. M., Chen, J., Shah, N., & Akbar, A. (2021). IHSF: An Intelligent Solution for Improved Performance of Reliable and Time-Sensitive Flows in Hybrid SDN-Based FC IoT Systems. *IEEE Internet of Things Journal*, *8*(5), 3130–3142. doi: 10.1109/JIOT.2020.3024560

Immermann, F., & Huang, Y. (2003). An introduction to cluster analysis. In An Introduction to Toxicogenomics. doi: 10.1201/9781315373515-1

Jalili, A., Keshtgari, M., & Akbari, R. (2020). A new framework for reliable control placement in software-defined networks based on multi-criteria clustering approach. *Soft Computing*, *24*(4), 2897–2916. doi: 10.1007/s00500-019-04070-8

Kuang, H., Qiu, Y., Li, R., & Liu, X. (2018). A hierarchical K-means algorithm for controller placement in SDN-Based WAN architecture. *Proceedings - 10th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2018*, *2018-Janua*, 263–267. doi: 10.1109/ICMTMA.2018.00070

Li, Y., Guan, S., Zhang, C., & Sun, W. (2020). Parameter Optimization Model of Heuristic Algorithms for Controller Placement Problem in Large-Scale SDN. *IEEE Access*, *8*, 151668–151680. doi: 10.1109/ACCESS.2020.3017673

Liao, J., Sun, H., Wang, J., Qi, Q., Li, K., & Li, T. (2017). Density cluster based approach for controller placement problem in large-scale software defined networkings. *Computer Networks*, *112*, 24–35. doi: 10.1016/j.comnet.2016.10.014

Mamushiane, L., Mwangama, J., & Lysko, A. (n.d.-a). *CONTROLLER PLACEMENT OPTIMIZATION FOR SOFTWARE DEFINED WIDE AREA NETWORKS (SDWAN)*. Retrieved from https://www.itu.int/en/journal/j-fet/Pages/default.aspx.

Mamushiane, L., Mwangama, J., & Lysko, A. A. (n.d.-b). *Controller placement optimization for Software Defined Wide Area Networks (SDWAN)*. Retrieved from http://hdl.handle.net/10204/12005

Mohajer, M. (n.d.). *A comparison of Gap statistic definitions with and with- out logarithm function*.

Muller, L. F., Oliveira, R. R., Luizelli, M. C., Gaspary, L. P., & Barcellos, M. P. (2014). Survivor: An enhanced controller placement strategy for improving SDN survivability. *2014 IEEE Global Communications Conference, GLOBECOM 2014*, 1909–1915. doi: 10.1109/GLOCOM.2014.7037087

Qi, Y., Wang, D., Yao, W., Li, H., & Cao, Y. (2019). Towards Multi-Controller Placement for SDN Based on Density Peaks Clustering. *IEEE International Conference on Communications*, *2019-May*, 1–6. doi: 10.1109/ICC.2019.8761814

Ramya, G., & Manoharan, R. (2021). Enhanced optimal placements of multi-controllers in SDN.

Journal of Ambient Intelligence and Humanized Computing, *12*(7), 8187–8204. doi: 10.1007/s12652-020-02554-2

Schütz, G., & Martins, J. A. (2020). A comprehensive approach for optimizing controller placement in Software-Defined Networks. *Computer Communications*, *159*(February), 198–205. doi: 10.1016/j.comcom.2020.05.008

Sminesh, C. N., Mary Kanaga, E. G., & Roy, A. (2019). Optimal multi-controller placement strategy in SD-WAN using modified density peak clustering. *IET Communications*, *13*(20), 3509–3518. doi: 10.1049/iet-com.2019.0124

Soleymanifar, R., Srivastava, A., Beck, C., & Salapaka, S. (2020). A clustering approach to edge controller placement in software-defined networks with cost balancing. *IFAC-PapersOnLine*, *53*(2), 2642–2647. doi: 10.1016/j.ifacol.2020.12.379

Tanha, M., Sajjadi, D., & Pan, J. (2016). *Enduring Node Failures through Resilient Controller Placement for Software Defined Networks*.

Yan-nan, H. U., Wen-dong, W., Xiang-yang, G., Xi-rong, Q. U. E., & Shi-duan, C. (2012). On the placement of controllers in software-defined networks. *The Journal of China Universities of Posts and Telecommunications*, *19*(October), 92-97,171. doi: 10.1016/S1005-8885(11)60438-X

APPENDIX

Appendix A: Sample Code for Building, Creating and Compile Mathematical Model

```
clear all;
clc;
graph = read_gml('Internet2.gml'); % Get the Graph structure

% Show the Graph structure

show_graph = graph;

show_nodes = graph.node;

show_node2 = graph.node(2);

show_edges = graph.edge;

% Get coordinates matrix

D=[];

for i=1:length(graph.node)

   D(i,1)=graph.node(i).Longitude;

   D(i,2)=graph.node(i).Latitude;

end
D;
%get the labels of nodes
L={};

%M=num2str(graph.node(1).label)

for i=1:length(graph.node)

   L{1,i}=num2str(graph.node(i).id); %node ids

   L{2,i}=num2str(graph.node(i).label); %node labels

   end

L;

A = get_adjacency_matrix(graph); % Adjacency Matrix: row: Source, column: Target

%plot topology

figure;

for i=1:length(graph.node)
```

```
line(D(i,1),D(i,2),'MarkerSize',20,'Marker','.','Color','red')
end
hold on
gplot(A,D,'-r')
axis square
% title('SANREN Backbone');
xlabel 'Longitude'
ylabel 'Latitude'
for i=1:length(graph.node)
text(D(i,1),D(i,2),L{2,i},'FontSize',15,'Color','black');
end
 hold off
%Calculate geographical distance matrix
dMatrix1=[]; dMatrix2=[];
for i=1:length(D)
    for j=1:length(D)
[dMatrix1(i,j) dMatrix2(i,j)]=lldistkm(D(i,:),D(j,:));
   end
end
dMatrix1;
dMatrix2;
%Verify results for Sanren backbone
latlon=[-29.8500  31.0167;-33.0153  27.9116];
[d1km d2km]=lldistkm(latlon(1,:),latlon(2,:));
%Weight Matrix
edgesDistKm=[]; %distance of edges in km,this represents weights
```

```
for i=1:length(graph.edge)
edgesDistKm(i,1)=graph.edge(i).source; %first two colums  indicate edges
edgesDistKm(i,2)=graph.edge(i).target;
source=graph.edge(i).source+1;
target=graph.edge(i).target+1;%matlab doesn't work with zero indices. Therefore add "1" to each node id
edgesDistKm(i,3)=dMatrix1(source,target); %3rd column stores the distances between nodes in kms.
%This is used to represent the edge weights
end
edgesDistKm(:,1:3) = round( edgesDistKm(:,1:3));
%edgesDistKm(:,1:2)
source=(edgesDistKm(:,1)+1)';
target=(edgesDistKm(:,2)+1)';
Weight=[];
Weight(:,1)=edgesDistKm(:,3);
weights=Weight';

%plot topology with weights
% for i=1:length(graph.node)
% line(D(i,1),D(i,2),'MarkerSize',35,'Marker','.','Color','r')
% end
% hold on
% gplot(A,D,'-b')
% axis square
% hold on
% for i=1:length(graph.node)
% text(D(i,1),D(i,2),L{2,i},'FontSize',8,'Color','black');
```

```
% end
% hold on
% G=graph(source,target);
% h=plot(G)
% plot the weights on top of the figure
% labeledge(h,1:numedges(G),weights);
hold off
%Find the shortest distance matrix using Johnson's algorithm
maxnode = max([source, target]);
DG=sparse(source,target,Weight.',maxnode,maxnode);
UG = tril(DG + DG');
[dist]=graphallshortestpaths(UG,'directed',false);
%Calculate average latency and worst-case latency.
L_average= (sum(sum(dist)))/(length(graph.node)); %average latency
L_worst=max(dist(:)); %worst-case latency
%use k-medoiods for best placement of 4 controllers
k=4
[inds,cidx] = kmedioids(dist,k);
clf, hold on, axis square
c = lines(k);
pts=D';
Latency_matrix1=zeros(k,15);
%test=length(inds(1,inds==1))
for i=1:k

   %ptsi(:,length(inds(1,inds==1))) = pts(:,inds==i);
```

```
    ptsi= pts(:,inds==i);
    ctrpt = pts(:,cidx(i));
end
hold on
for i=1:length(graph.node)
text(D(i,1),D(i,2),L{2,i},'FontSize',10,'Color','black');
end
hold on
gplot(A,D,'-');
% title('\fontsize{16}Uncapacitated Optimal Controller Placement');
xlabel('Longitude (degrees)','FontSize',25,'FontWeight','bold');
ylabel ('Latitude(degrees)','FontSize',25,'FontWeight','bold');
% legend({'Cluster 1','Controller Placements','Cluster 2', 'Cluster 3','Cluster 4'}, 'FontSize',10)
set(gca,'fontsize',20)
hold off
plot(1:4,y, '--', 'Color','blue','LineWidth',2);
% title('\fontsize{25}Relation between number of controllers and latency');
xlabel('Number of controllers','FontSize',30,'FontWeight','bold');
ylabel ('Overall latency (ms)','FontSize',30,'FontWeight','bold');
hold on
plot(1:4,z, ':', 'Color','red','LineWidth',2);
legend({'average latency','worst-case latency'}, 'FontSize',20)
set(gca,'fontsize',20)
hold off
%Alternative approach for placement optimization using K-Medoids
n=length(graph.node);
```

```
% L_average=[];

D; %coordinates matrix

% for k=1:5 %number of clusters/controllers

% opts = statset('Display','iter');

% [idx,Clust,sumd,d,midx,info] = kmedoids(D,k,'Distance',@lldistkm,'Options',opts);

% L_average (1,k)=(sum(sumd))/length(graph.node) %evaluating the optimization results using k-medoids. Results are benchmarked against Johnson's results

% %######parameters description#########

% %idx: vector idx containing cluster indices of each observation

% %Clust:returns the k cluster medoid locations in the k-by-2 matrix C

% %sumd:  returns the within-cluster sums of point-to-controller distances in the k-by-1 vector sumd

% %d: returns distances from each point to every medoid in the n-by-k matrix D

% %midx: returns the indices midx such that Clust = X(midx,:). midx is a k-by-1 vector

% %info: returns a structure info with information about the options used by the algorithm when executed

% end

% % Plot clustering results

% figure;

% plot(D(idx==1,1),D(idx==1,2),'r.','MarkerSize',24)%everypoint in cluster idx(e.g. 1) plot and mark with red.

% hold on

% plot(D(idx==2,1),D(idx==2,2),'b.','MarkerSize',24)

% hold on

% plot(D(idx==3,1),D(idx==3,2),'g.','MarkerSize',24)

% hold on

% plot(D(idx==4,1),D(idx==4,2),'k.','MarkerSize',24)

%

% plot(Clust(:,1),Clust(:,2),'co',...
```

```
%      'MarkerSize',12,'LineWidth',2)
% legend('Cluster 1','Cluster 2','Cluster 3','Cluster 4','Controller Placements',...
%        'Location','NW');
% title('Uncapacitated Controller Placement');
% xlabel 'Longitude'
% ylabel 'Latitude'
% hold off
% title('Optimal number of controllers based on cost benefit');
xlabel ('Number of controllers','FontSize',30, 'FontWeight','bold')
ylabel ('Cost benefit ($/ms)','FontSize',30,'FontWeight','bold')
% edgeLatency for Mininet Emulation
edgesDistKm(:,3)=edgesDistKm(:,3)/2e2
edgeLatency=(edgesDistKm(:,1:3))
```

Appendix B: Sample Code for Compiling Emulation Code

```
from mininet.net import Mininet

from mininet.node import Controller, RemoteController, OVSController

from mininet.node import CPULimitedHost, Host, Node

from mininet.node import OVSKernelSwitch, UserSwitch

from mininet.node import IVSSwitch

from mininet.cli import CLI

from mininet.log import setLogLevel, info

from mininet.link import TCLink, Intf

from subprocess import call

import igraph as ig

import networkx as nx

import matplotlib.pyplot as plt

import haversine

import numpy as np

import haversine as hv

def myNetwork():

   net = Mininet( topo=None,

                build=False,

                ipBase='10.0.0.0/8')

   info( '*** Adding controller\n' )

   c0=net.addController(name='c0',

                  controller=RemoteController,

                  ip='192.168.0.180',

                  protocol='tcp',

                  port=6633)
```

```
    c1=net.addController(name='c1',
                         controller=RemoteController,
                         ip='192.168.0.181',
                         protocol='tcp',
                         port=6633)
    g=ig.Graph.Read_GML("Sanren.gml")
    #print(summary(g))
    #delays=g.node[0]["Longitude"]
    #print delays
#print nx.info(g)
#print g.nodes
#num_nodes=g.number_of_nodes()
    info( '*** Adding switches and hosts\n')
    nodes=g.vcount()
    print nodes
    for i in range(0,nodes):
        s="s"+str(i)
        h="h"+str(i)
        IP="10.0.0."+str(i+1)
        #print IP
        s = net.addSwitch(s, cls=OVSKernelSwitch)
        h = net.addHost(h, cls=Host, ip=IP, defaultRoute=None)
        net.addLink(h, s)
        #print i
        print s
```

```
info( '*** Connecting switches\n')
edges=g.get_edgelist() #read graph edges
print(edges)
s=[None]*0 #initialize/declare an array of fixed size (i.e. n=2)
#print s
for i in edges:
#print i  #i is an edge (subset of egdes)
   s=[None]*0
   for j in i:
   #print j  #j is either source or target (subset of i)
      r="s"+str(j)
      s.insert(0,r)  #insert results before position zero
   print s   #s represents target source pairs
   source=s[1]
   target=s[0]
   source_target="*"+"*"+source+target
   print source
   print target
   print source_target
   net.addLink(source, target, cls=TCLink)
##############configure link delays########
info( '*** Configuring link delays\n')
latlon=[]
lats=list(g.vs['Latitude']) #read the latitute attribute
lons=list(g.vs['Longitude']) #read the longitude attribute
```

```
lat=np.array([lats])
lon=np.array([lons])
#print np.concatenate((lat.T,lon.T),axis=1)
latlon= np.concatenate((lat.T,lon.T),axis=1)
lat_lon1=latlon.tolist()
print lat_lon1
m, n = 2, 2; #create a list of two lists each having 2 items all containing 0 entries
lonlat2 = [[0 for x in range(m)] for y in range(n)]
print lonlat2
edges=g.get_edgelist() #read graph edges
print(edges)
s=[None]*0
for i in edges:
   #lonlat2=[]
   print i  #i is an edge (subset of egdes)
   for j in range(0,2):
      #print j
      print i[j]
      k=i[j]
      #print lat_lon1[k]
      lonlat2[j][:]=lat_lon1[k] #read source destination pairs and store results in a 2X2 list of lists
   #print lonlat2
   link_delays=hv.distance(lonlat2)
   print link_delays
   for z in i:
```

```
        q="s"+str(z)
        s.insert(0,q)
    p=s[1]+s[0]
    link_delays2=str(link_delays)
    p = {'delay':link_delays2}
    print p
info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
    controller.start()
info( '*** Starting switches\n')
for i in range(0,nodes):
    s="s"+str(i)
    print s
    net.get(s).start([c0,c1])
info ('Optimizing controller placement through ping\n')
hosts=net.hosts
#server=hosts[0]
outfiles, errfiles = {}, {}
hosts_count=len(hosts) #get the size/length of the hosts array
for k in range(0,hosts_count):
    for h in hosts:
#create and/or erase output files
        server=hosts[k-1]
```

```
            outfiles[h]='/tmp/%s.out' % h.name
            errfiles[h]='/tmp/%s.err' % h.name
            h.cmd('echo >', outfiles[h])
            h.cmd('echo >', errfiles[h])
        #start pings
        for hpair in zip(hosts, hosts[1:]):
            net.ping(hpair, timeout='1')
        net.ping((hosts[-1], hosts[0]), timeout='1')
        #print hosts
        info( '*** Post configure switches and hosts\n')
        CLI(net)
        net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```