# CONTROLLER PLACEMENT OPTIMIZATION FOR SOFTWARE DEFINED WIDE AREA NETWORKS (SDWAN)

Lusani Mamushiane[1,2], Joyce Mwangama[2], Albert Lysko[1,2]
[1]Council for Scientific and Industrial Research (CSIR), [2]University of Cape Town, South Africa,

NOTE: Corresponding author: Lusani Mamushiane, Lravhuanzwo@csir.co.za

***Abstract*** – *Software Defined Networking (SDN) has emerged as a promising solution to revolutionize network deployment, operations and economic growth. This paradigm aims to address management and configuration complexities in legacy networks so as to reduce the total cost associated with deploying and running telecommunication infrastructures. At the heart of SDN is a controller which oversees orchestration of resources. An important problem that must be addressed during the initial design of an SDN-based network deployment is to find the optimal number of controllers and their locations, to achieve desired operational efficiency. This problem constitutes competing objectives such as latency, load balancing, and reliability. We apply Silhouette Analysis, Gap statistics and the Partition Around Medoids (PAM) algorithms and, unlike previous works, we add a new method for solving the controller placement problem using an emulation orchestration platform. Our approach aims to optimize controller-to-node latency, alleviate control plane signaling overhead and ensure control plane resiliency. Our results for South African national research network (SANReN) reveal that deploying two controllers yields lowest latency, reduces control plane signaling overhead and guarantees control plane resiliency. Our approach can be used by network operators as a guideline to start integrating SDN or plan a new SDN deployment, by helping them make quick automatic decisions regarding optimal controller placement.*

**Keywords** – Controller placement, Gap statistic, Partition Around Medoids, , Silhouette, Software Defined Networks, South African National Research and Education Network

## 1. INTRODUCTION

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

Software Defined Networking (SDN) has emerged as a promising candidate to revolutionize future telecommunication landscapes. Contrary to the traditional network architecture where the control and data plane of packet processing devices are tightly coupled, SDN presents a paradigm shift in networking by decoupling the control plane logic from the underlying physical infrastructure [5]. The control plane is then logically centralized in an external entity called a controller and interacts with the physical infrastructure via its southbound interface. By decoupling the control logic from the physical hardware, operators can programme new traffic engineering policies (such as bandwidth management, security, protection and restoration policies) without worrying about the constraints of closed proprietary hardware and firmware. Moreover, the abstraction of lower level functionality provided by SDN enables convergence of heterogeneous hardware thereby fostering a vendor-neutral ecosystem. In addition to enabling centralized network provisioning and holistic network management, SDN promises benefits such as security granularity (by providing a central point of control to holistically and consistently disseminate security information), savings in operational costs (by automating network administrative tasks), savings in capital expenditures (by capitalizing on commodity hardware) and cloud abstraction (which is critical to consolidate and facilitate management of massive data centers) [6]. According to [7] a huge portion of operational expenditure is from costs related to the

management and configuration of the telecommunication infrastructure. Therefore, leveraging SDN to automate management and configuration tasks is likely to improve return on investment (ROI).

This work presents a framework which can be used by network operators to optimize their SDN controller placement during deployment phase.

The paper is organized as follows: Section 2 presents the problem being addressed by this work, Section 3 describes related works and their drawbacks, and highlights our contributions, Section 4 describes the algorithms (based on mathematical modelling) used to solve the controller placement problem, Section 5 provides implementation details of the algorithms, Section 6 presents results from our mathematical modelling, Section 7 describes the emulation experiments conducted to verify the outcome of the mathematical modelling, Section 9 concludes the paper. Lastly, Section 10 describes future work.

## 2. PROBLEM STATEMENT

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

Another aspect to controller placement has to do with the number of controllers deployed in a given WAN. Deploying a certain number of controllers has an impact on several objectives such as propagation latency and reliability. Even though the number of controllers may be known in advance, the location of these controllers usually needs to be optimized to meet user requirements and contraints.

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

## 3. STATE OF THE ART

This section presents an analysis of state–of–the art controller placement solutions.

### 3.1 Related Work

To date there has been numerous research studies directed towards addressing the controller placement problem in SDN. These can be broadly classified into two categories: (i) studies that implemented exhaustive algorithms, as exampled by [8] –[9] and (ii) studies that implemented heuristic algorithms, as exampled by [16] –[30].

The controller placement problem was first introduced by Heller et al. [8] in 2012. The authors study the controller placement problem by investigating the impact of uncapacitated controller location on average and worst–case latency. The algorithm used in this study is k–center. To maintain realism, the authors tested their algorithm on the Internet2 OS3E topology [10]. Their results indicate that increasing the number of controllers decreases the overall network latency with a significant tradeoff between worst–case and average latency. The authors conclude that deploying one controller often suffices to meet existing latency requirements in campus networks. Expectantly, they also argue that one controller is not sufficient for large scale deployments with fault tolerant requirements.

Hu et al. [11] proposes the use of multiple controllers to ensure reliability in the control plane. To optimize controller placement, the authors carry out a comparative evaluation of optimization algorithms namely, random placement, l–w greedy and brute force. They focus their reliability metric on the "expected percentage of valid control paths", where a control path is defined as the interface between the switch and controller (southbound interface) as well as the connection between controllers (east/westbound interface). The algorithms were evaluated on Internet2 topology as well as various ISP topologies from the Rockefuel database [12]. From their simulations, random placement produced the least optimal results, while brute force produced optimal results after a significantly long runtime. As a result, the authors recommend the l–w greedy as the most optimal solution. This work is similar to Gong et al. [13] in that they both aim to optimize reliability in the event of node or link failure. However, latency (both switch–to–controller and inter–controller latency) and load balancing are not considered in these research works. Moreover, the number of controllers is assumed to be known in advance.

Tanha et al. [9] study the controller placement problem to optimize network resiliency in the event of controller failure while considering network deployment costs and satisfying switch–to–controller latency. In order to mimic

a production scenario, the authors take into account the capacity of the controller and assume a varying switch load. To maintain realism, they assessed their algorithms on real tier–1 service provider topologies. The outcome of their experiments demonstrated that controller resiliency is topology dependent. The drawback of this solution is that it is resource intensive and only ideal for small and medium network instances. The algorithm used in this study is the capacitated k–center algorithm.

The research work of Yao et al. [14] proposes a heuristic algorithm for capacitated controller placement in consideration of switch–to–controller latency and traffic load of switches. The main objective of this work is to optimize controller load balancing under heterogeneous data plane load while minimizing switch–to–controller latency. Resiliency is handled by deploying additional controllers in the network. The main shortcoming of this solution is that it is less accurate in larger deployments and therefore applicable only for small–scale networks.

Jimenez et al. [15], also proposes a capacitated controller placement solution to optimize load balancing. Contrary to Yao et al., this work is not limited to the size of the network and propose a divide and conquer philosophy to achieve scalability and robustness. Moreover, authors assume homogeneous traffic load on the data plane. The solutions proposed by Jimenez et al. and Yao et al. optimize controller placement based on fixed traffic observed initially, but do not adapt to the changing traffic load. This shortcoming is addressed by Bari et al. [16] and Jourjon et al. [17] who propose a heuristic algorithm for dynamic controller placement i.e. controller placement based on current data plane load. The metrics considered are switch–to–controller latency and controller processing load. The solutions proposed rely on trial and error and are not as reliable. Sanner et al. [18] propose a genetic algorithm leveraging the Non-dominated Sorting Genetic Algorithm (NSGA) II framework to optimize load balancing and inter–controller latency. Authors conclude that their solution consumes a lot of CPU resources and is only ideal for small and medium–sized networks.

Rath et al. [19] propose a Non–Zero–Sum game theory approach to optimize controllers' utilization. In this solution, controllers can be added or removed dynamically and can also go to sleep mode occasionally based on the traffic load on the controllers. This solution is intended to reduce network deployment costs (by minimizing the number of controllers deployed) and operation costs (by optimizing energy consumption through on–demand controller deployment). This solution does ignores controller placement in the network. Sallahi et al. [20] propose a mathematical formulation to find the optimum number of controllers to deploy. However, their approach suffers the same shortcoming as that proposed by Rath et al. in that it does not determine the optimal controller placement. Furthermore, both these research works are limited to small–scale networks.

Wendong et al. [21] study the tradeoff between reliability and latency using random placement, l–w greedy and simulated annealing. The results suggest that simulated annealing yields the most optimal solution in comparison with the other approaches. The outcome of the tradeoff analysis indicate a significant tradeoff between reliability and latency. Authors argue that the number of controllers must be chosen carefully. They demonstrate that using too few controllers has an adverse effect on reliability while using too many controllers can result in a broadcast storm on the east/westbound interface.

Hock et al. [22] and Lange et al. [23] advocate for careful consideration of latency (controller–to–controller) and reliability (defined as resilience in the event of a node or link failure and control plane load balancing) during controller placement. This work proposes a resilient Pareto–based Optimal Controller placement framework to achieve optimal controller placement. The authors use load imbalance as the key metric, which is the difference between the controller having more switch assignments and the controller having fewer number of switches under its supervision. The results from this work indicate that the optimal solution is achieved when 20% of all network nodes are controllers. The downside of this solution is that, instead of partitioning the network into small administrative domains, the authors treat the network as a whole with controllers working collaboratively. This means the controllers frequently share their network state information with their peers to maintain an accurate global view. This increases the probability of incurring a network broadcast storm which increases inter –controller latency. Therefore, this solution is restricted to small–and medium–scale network instances. Furthermore, this solution ignores the average switch–to–controller latency which is a critical parameter in SDN.

Ksentini et al. [24] consider three objectives in optimizing controller placement: (i) switch–to–controller latency, (ii) inter–controller latency and (iii) control plane load balancing simultaneously. The authors propose a bargaining game–based algorithm to optimize controller placement. Authors claim that their results outperform other mono–objective–based controller placement results. However, their algorithm is only suitable for small–scale networks and is less accurate for larger network instances.

Last but certainly not least, He et al. [26] formulate controller placement model to optimize flow setup time, where flow setup time is the total amount of time taken by the controller to install a flow instruction on the switch's flow table. The authors argue that dynamic controller placement is necessary to help reduce flow setup time.

The results from this work reveal that, for low flow densities, dynamic controller placement can reduce the flow setup time by up to 50% in comparison with static controller placement. However, for high flow densities, static controller placement produced better results.

As demonstrated by Heller et al. [8], Hock et al. [22] and Wendong et al. [21], there exists a significant tradeoff between load balancing, reliability (also known as resiliency) and latency. Therefore it is almost impossible to optimize one objective without sacrificing the other. This study attempts to address the controller placement problem in consideration of switch-to-controller latency metric. This metric has emerged as an important QoS determinant in SDN. This is primarily because the communication between the controller and data plane has to be seamless to ensure an accurate view of the network state and prompt data plane flow installations.

Table 1 provides a summary of the state of the art in research pertaining to SDN controller placement.

## 3.2 Contribution

From the state-of-the art review, it is apparent that most studies (with the exception of the work by Sallahi et al. [20]) assume the number of controllers to be known in advance. However, the model proposed by Sallahi et al. is ideal to plan a small–scale SDN and runs out of memory when solving larger instances. Moreover, most studies relied on heuristic algorithms to reduce algorithm runtime. However, this is achieved at the expense of solution accuracy. To the best of our knowledge, the only research studies that implement exhaustive algorithms are by Heller et al. [8] and Tanha et al. [9]. Both Heller et al. and Tanha et al. propose the use of *k*-center to solve the controller placement problem. However, *k*-center is sensitive to outliers and does not always consistently yield accurate results [27]. Perhaps more importantly, there is currently no analysis of the controller placement problem purely using an emulation platform to mimic a real SDN deployment. Most studies relied on mathematical modeling to address the controller placement problem, making it difficult to verify validity and reliability of the results.

Controller placement is a network planning problem, and is normally not time sensitive. Consequently, this study proposes exhaustive algorithms to optimize solution accuracy. In order to find the best locations to place SDN controllers, this study proposes the use of a classical machine learning algorithm called Partition Around Medoids (PAM) [28]. To determine the optimal number of controllers to deploy given a wide area network, this study proposes the use of Silhouette [29] and Gap statistics [30] algorithms. To mimic a real SDN

**Table 1** – Classification of existing controller placement solutions

| Solution | Topology(s) | Scale of Network | Environment | Algorithm(s) | Placement Metric(s) | Network Partitioning |
|---|---|---|---|---|---|---|
| Heller et al. [8] | Internet2 OS3E | Large–scale | Static | k–center | average switch–to–controller latency<br>worst–case latency | No |
| Hu et al.[11] | Internet2 OS3E | Small and medium-sized | Static | l–w greedy | Reliability | No |
| Tanha et al. [9] | Sprint<br>ATT NA<br>PSINET<br>UUNET | Large-scale | Static | Capacitated k-center | switch–to–controller latency<br>Reliability | No |
| Yao et al. [14] | Internet Zoo | Large–scale | Dynamic | Linear relaxation | switch–to–controller latency<br>Load balancing | No |
| Jimenez et al. [15] | Sparse<br>Medium<br>Dense | Large–scale | Dynamic | k–critical | Load balancing | Yes |
| Bari et al. [16] | RF-I | Large–scale | Dynamic | DCP-GK | switch–to–controller latency<br>Load balancing | Yes |
| Jourjon et al. [17] | Not discussed | Large–scale | Dynamic | LiDy+ | switch–to–controller latency<br>Load balancing | Yes |
| Sanner et al. [18] | Internet2 OS3E | Large–scale | Dynamic | NSGA | inter–controller latency<br>load balancing | Yes |
| Rath et al. [19] | Random network with 28 switches | small-scale | Dynamic | Non–zero–<br>Sum Game | Load balancing | No |
| Sallahi et al. [20] | Random network with 10, 20, 30, 40, 50, 75, 100, 150 switches | small-scale | Dynamic | CPLEX | Load balancing | No |
| Wendong et al. [21] | Internet2 OS3E | Large–scale | Static | l–w greedy | switch–to–controller latency<br>Reliability | No |
| Hock et al. [22] | Internet2 OS3E | Small and medium–sized | Static | POCO | switch–to–controller latency<br>Reliability<br>Load balancing | No |
| Lange et al. [23] | Internet2 OS3E<br>Internet Zoo | Large–scale | Dynamic | Simulated Annealing | switch–to–controller latency<br>Reliability<br>Load balancing | No |
| Ksentini et al.[24] | Ring<br>Binary Tree | Large–scale | Static | No specific name | switch–to–controller latency<br>Inter–controller latency<br>Load balancing | Yes |
| Mamushiane et al.[25] | SANReN | Small-scale | Static | Partition Around Medoids (PAM)<br>Gap Statistics<br>Silhouette Analysis<br>Johnson's Algorithm<br>Emulation | average switch–to–controller latency<br>worst–case latency<br>switch-to-controller balancing<br>propagation +queuing + processing latency<br>signalling overhead | Yes |

deployment, the controller placement problem is studied using an emulation orchestration platform. This is something that to the best of our knowledge has not been done, and we consider it necessary to verify the outcome of the mathematical modeling. Finally, a mechanism to manage control plane overhead is proposed.

The key performance indicators used to gauge network performance are: (i) network latency (propagation + queuing + processing latency), (ii) reliability (in the event of link and/or node failure) and (iii) control plane signaling overhead.

## 4. MATHEMATICAL MODELLING

In order to compute the optimal number of controllers, we propose two "unsupervised" machine learning approaches namely, Silhouette and Gap Statistic. Unsupervised algorithms learn from input data that has no labeled responses [31]. These algorithms are classically used to analyze cluster quality through the metric of minimum distances between data points. In the context of controller placement, we leverage these algorithms to find the number of controllers that minimizes overall network propagation latency (i.e. switch-to-switch latency). To find the best locations for these controllers, we extend and apply a facility location algorithm called Partition Around Medoids algorithm (PAM), with propagation latency (i.e. controller-to-switch latency) as our main objective function. For realism, we use the South African National Research Network (SANReN) as a case study. The choice of this topology was mainly motivated by the fact that it represents the emerging market case study which is the key use case of this study.

Since the links between SANReN's switches are known to be fiber where speed is approximately the speed of light in fiber (i.e. $2 \times 10^8\ m/s$), we compute propagation latency by taking the ratio of average distance (between nodes) to speed of light in fiber. The distances are calculated using the Harvesine approach [32]. The results from our simulations and discussions are also presented in this section.

### 4.1 Assumptions

The following assumptions apply to the proposed algorithms:

- Switch-to-controller communication is assumed to happen out-of-band (control and regular traffic do not share the same links) ;
- The bandwidth for all connection links is constant;
- Control path security and reliability has been perfectly solved;
- Controllers are co-located with some of the switches;
- Switches incur a fixed load.

### 4.2 Optimal number of Controllers

This section introduces the algorithms used to find the optimal number of controllers to deploy given a wide area network. Table 2 defines some of the notation used in this section.

**Table 2** – Mathematical symbols

| Symbol | Definition |
|---|---|
| $C_k$ | $k_{th}$ cluster |
| $L(C_k)$ | Intra-cluster propagation latency variation |
| $G(V,E,X)$ | Network topology graph |
| $V$ | Data plane nodes |
| $E$ | Links between nodes |
| $X$ | Geographic locations of nodes |
| $\varphi$ | Latitude of a node |
| $\lambda$ | Longitude of a node |
| $r$ | Radius of the earth |
| $k$ | Number of clusters |
| $B$ | Randomy generated reference dataset of network topology |
| $s$ | Standard deviation |

#### *4.2.1 Silhouette Analysis*

Silhouette Analysis is a method of interpretation within existing clusters, used to measure the quality of a cluster (how close each point in a cluster is to its adjacent clusters) for varying number of partitions [33]. In the context of the controller placement problem, we adopt and extend this algorithm to answer this question: given a wide area network topology, how many controllers are needed to achieve minimum intra-cluster propagation latency variation? Eq. *(1)* shows our objective function.

$$X = min \sum_{k=2}^{n} L(C_k) \qquad (1)$$

Algorithm 1 outlines the Silhouette approach. The algorithm requires three input parameters namely, a clustering algorithm (clustAlg) to cluster network data plane nodes, distance function handle (disfun), network topology graph ($G(V,E,X)$), where $V$ denotes data plane nodes (switches), $E$ denotes edges (links between nodes), and $X$ denotes the geographic locations (longitude, latitude) of nodes), and maximum number of controllers (maxNumControllers). The clustering algorithm used is called Partition Around Medoids (PAM) described in Section 4.3.2 [33]. The Harvesine distance approach was used to compute the great circle distances between pairs of switches [34]. 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)[35] [36] . An alternative method to compute geographic distances is the Law of Cosines, which is optimal for shorter distances and is not as accurate for longer distances [37]. To compute the

**Algorithm 1** Silhouette Analysis

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

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

great-circle distance, Eq. *(2)* which defines the Harvesine approach is used, where $\varphi_1$ and $\varphi_2$ are the latitudes of points 1 and 2 respectively, $\lambda_1$ and $\lambda_2$ is the longitudes of point 1 and 2 respectively and $r$ denotes the radius of the earth, a constant equal to 6371 km.

$$Distance = 2(r)arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + cos(\varphi_1)cos(\varphi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \quad (2)$$

The procedure to compute the optimal number of controllers using Silhouette (with steps/instructions enumerated from 1 to 12 in Algorithm 1) is as follows: First, a cluster model is created from input network data using PAM and Harvesine approach (Instruction 4). Next, the average propagation latency from each switch to its cluster centroid is calculated (Instruction 6), to find the intra-cluster propagation latency variation (intraClustVar). To this end, a model from the centroids is created (Instruction 7). Next, the average propagation latency between each centroid to the global center (Instruction 8-9) is calculated. In this way we obtain the inter-cluster propagation latency variation (interClustVar). The last step is to calculate the silhouette coefficient (Instruction 11). This procedure is repeated as specified by the maxNumControllers input parameter in order to calculate the silhouette coefficient for each number of controllers. Moreover, for each number of controllers (Instruction 3), the number of iterations was set to 20 to maximize accuracy of the results.

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

### *4.2.2 Gap Statistics*

Similar to Silhouette Analysis, Gap Statistic is a partition algorithm typically used in neural networks, to measure the quality of clustering measure based on average intra-cluster variation [38] [30]. We adopt and enhance this algorithm to verify the results from our Silhouette Analysis. Therefore our goal is to determine the optimal number of SDN controllers to deploy given a network topology, and compare the outcome of the simulation with the results from the Silhouette Analysis.

The Gap Statistic algorithm constitutes the following steps (enumerated by instructions from 1 to 12 in Algorithm 2): First the network topology is partitioned (using the PAM algorithm), by varying the number of controllers $k$ (which corresponds to the number of clusters) from 2 to the maximum user-defined value (Instruction 3).

**Algorithm 2** Gap Statistics

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

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

This is followed by the computation of the average intra-cluster propagation latency variation (intraClusVar denoted by $L(C_k)$ in Eq. *(3)*) between the switches (Instruction 4). Next a reference dataset ($rRef$ denoted by $B$ in Eq. *(4)*) of the network topology is randomly generated (Instruction 6). The average intra-cluster latency variation of the reference dataset (intraClusVarRef denoted by $L^*(C_{kb})$ in Eq. *(4)*) is computed (Instruction 7). The gap statistics is calculated

using Eq. *(3)* and *(4)*. Finally, the standard deviation of B Monte Carlo replicates is calculated [30]. The optimal number of controllers is one that meets the condition in Eq. *(5)*, where $s_{k+1}$ is denotes the standard deviation of B Monte Carlo replicates.

$$gap_n(k) = E_n^* log(L^*(C_k)) - log(L(C_k)) \quad (3)$$

where

$$E_n^* log(L^*(C_k)) = \frac{1}{B}\sum_b log(L^*(C_{kb})) \quad (4)$$

$$gap(k) \geq gap(k+1) - s_{k+1} \quad (5)$$

## 4.3 Optimal controller location

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

### *4.3.1 Johnson's Algorithm*

In order to determine the best locations to place SDN controllers in a WAN, the shortest paths between each pair of switches must be known. Johnson's algorithm [39] provides a means to find the shortest paths between node pairs and has become a popular method for addressing SDN optimization problems [40]. Therefore we used the results from this algorithm alongside the PAM algorithm to determine the best places to deploy controllers. A pseudocode of this algorithm is as shown in Algorithm 3 and consists of the following steps: First a new arbitrary switch (denoted by $q$) is added to the network graph, connected by zero-weight links to all other switches (denoted by $v$) in the network graph (Instructions 1-5). If this step detects a negative weight-cycle (i.e. a cycle whose weight sums to a negative number), the algorithm is terminated (Instruction 6-7). Second, a single source shortest path algorithm called Bellman-Ford algorithm is evoked, to find the shortest path $h(v)$ from each switch $v$ in the network to the new switch (Instructions 9-11). Next, the graph is reweighted to find new link weights $w_{new}$ (Instruction 12-14). Finally, the new switch is removed, and Dijkstra's algorithm is used to compute the shortest paths $p(u, v)$ from each each node to every other node in the reweighted graph (Instructions 15-22).

**Algorithm 3** Johnson's Algorithm

**Require:** $G(V, E)$ {undirected weighted network graph}
1: Compute $G'$ $where$ $V[G'] \leftarrow V[G] \cup q$ {$G'$ is 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** $v \in V[G]$ **do**
10: set $h(v) \leftarrow \delta(q, v)$ compute shortest path using Bellman-Ford
11: **end for**
12: **for** $(u, v) \in E[G']$ **do**
13: $w_{new} \leftarrow w(u, v) + h(u) - h(v)$
14: **end for**
15: **for** $u \in V[G]$ **do**
16: execute $Dijkstra(G, w_{new}, u)$ to compute $\delta_{new}(u, v)$ for all $v \in V[G]$
17: **for** $v \in V[G]$ **do**
18: $p(u, v) \leftarrow \delta_{new}(u, v) + h(u) - h(v)$
19: **end for**
20: **end for**
21: **end if**
22: **return** shortest path matrix

### *4.3.2 Partition Around Medoids (PAM)*

After determining the optimal number of controllers to use given a WAN topology, the next step is to find the best locations to place the controllers such that the QoS is maximized. This can be achieved by leveraging "unsupervised" machine learning heuristic algorithms (such as Simulated Annealing [41] and Clustering LARge Applications (CLARA)[42]) or exhaustive algorithms (such as k-means [43] [44] and PAM [45] [46][47]. However, heuristic algorithms are suboptimal in the sense that they are primarily focused on optimizing runtime over solution accuracy. Therefore, heuristic algorithms are more ideal for scenarios requiring dynamic controller placement. However, this study explores static controller placement, where the controller placement problem is addressed during network planning. Therefore, the accuracy of the optimization algorithm is significantly more important than the speed of computation. From the available exhaustive algorithms, we opted for the PAM algorithm. This is mainly because the k-means algorithm is very sensitive to outliers which can lead to solution inaccuracy [48]. Unlike k-means, PAM is more stable and more accurate [49].

Algorithm 4 describes the steps we followed to compute the optimal locations of SDN controllers. Our approach assumes co-location of controllers and switches. First, $k$

arbitrary switches (where $k$ is the number of controllers to place)) are selected as the potential controller locations (Instruction 3). This is followed by the association of each switch to the closest controller (Instructions 4-6). While the cost of configuration (the overall propagation latency) decreases, the controller location $R_i$ and switch $S_o$ are swapped (Instructions 7-9), and each switch is reassigned to their closest controller location (Instructions 4-6).

**Algorithm 4** Partition Around Medoids (PAM)

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

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

If an increase in configuration cost is detected, the swap is undone and the optimal controller locations that optimize QoS are found (Instructions 12-18). Two QoS parameters are considered in our solution, that is the average propagation latency (which is the overall propagation latency) and the worst-case propagation latency (which is the maximum network latency). Eq. *(6)* and *(7)* define how these parameters are defined, where $L_{avg}(Z')$ is the average latency, $L_{wc}(Z')$ is the worst-case latency, $d(v,z)$ is the shortest distance from the switch (node $v \in V$) to the controller (node $z \in Z$ ), $N = |V|$ denotes the number of nodes and $2 \times 10^8$ is the speed of light in fiber.

$$L_{avg}(Z') = \frac{1}{(2X10^8)N} \sum_{v\in V} \min_{z\in Z'} d(v,z) \qquad (6)$$

$$L_{wc}(Z') = \max_{v\in V} \min_{z\in Z'} d(v,z) \qquad (7)$$

## 5. IMPLEMENTATION OF MATHEMATICAL MODELLING

This section explains our implementation for solving the controller placement problem using the algorithms described in Sections 4.2 and 4.3. These algorithms are implemented in MATLAB 2018b. The primary objective is to establish the number of controllers for the achievement of minimum propagation latency and to determine the best locations to place these controllers in a WAN topology. The results from our simulation experiments are also presented in this section.

### 5.1 Topologies

To maintain realism, our proposed solution is applied on a real-world WAN called South African Research Network (SANReN), operated by CSIR'S Next Generation Enterprises and Institutions (NGEI) cluster [50]. The reason for choosing the SANReN network was so that we could demonstrate our proposed solution on an emerging market use case. However, it may be noted that our solution is topology-agnostic and can easily be used to test other networks of different configurations and sizes.

The SANReN network constitutes a core national backbone, with each Point of presence (PoP) connecting a metropolitan network. This work only focuses on the PoP-level instead of the router-level view of the SANReN network. This is because the router-level view is proprietary and not publicly available. Moreover, the PoP-level view has been deemed more useful [51] for several points: it provides a larger scale view of network links, which are most interesting for network optimization; it shows end users where they can connect to the network and it's the level where resiliency and redundancy are critical. The PoP-level geographical map of the SANReN topology comprises 7 nodes and 7 fiber links configured in a ring topology. The data set of this topology was downloaded from a repository called The Internet Topology Zoo [52]. The format of the data set is in Geography Markup Language (GML) and includes geographic locations (longitude, latitude) of nodes and topological configuration of the SANReN network.

### 5.2 Hardware and software used for modelling

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

### 5.3 Flowchart of proposed solution

The flowchart depicted in Figure 1 summarizes the steps in our proposed controller placement solution. First, network graph modelling is used to model the

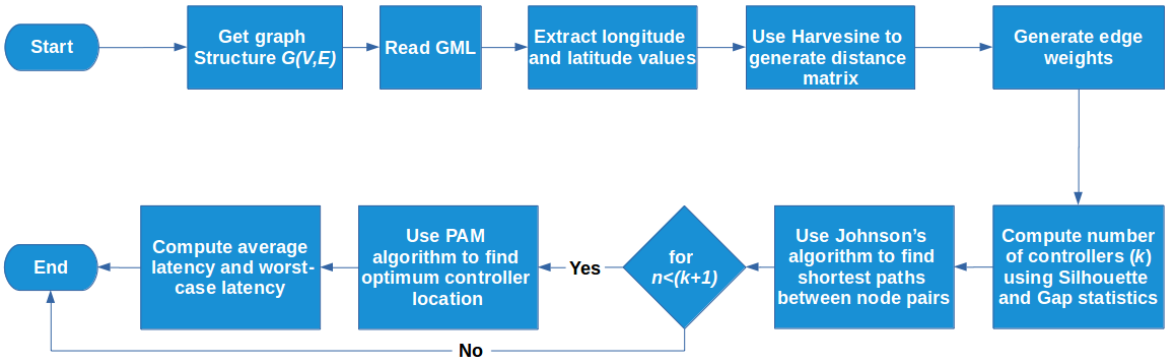


**Fig. 1** – Flow chart of proposed method.

network topology as an undirected graph $G(V, E)$, where $V$ denotes network switches and $E$ represents fiber links (edges) connecting the switches. This is followed by the extraction of the geographic location data using the input dataset. Next, the Harvesine approach is applied on the location data to generate the distance matrix. To determine edge weights, an adjacency matrix is implemented between all connected switches. Then, computation of the number of controllers that minimize intra-cluster latency is carried out using Silhouette algorithm as described in Section 4.2.1, Algorithm 1. To verify the results from Silhouette, Gap Statistic is implemented as described in Section 4.2.2, Algorithm 2. This is followed by computation of the shortest path matrix by applying Johnson's algorithm outlined in Algorithm 3. The results from Silhouette, Gap Statistic and Johnson's algorithm, are used as inputs to the PAM algorithm discussed in Section 4.3.2, Algorithm 4, which is used to find the best locations that minimizes propagation latencies, namely the average latency and worst-case latency defined in Section 4.3.2 (Eq *(6)* and *(7)*). The key factor in our mathematical formulation is the distance (under the assumption of constant bandwidth across all fiber links). Therefore under constant bandwidth, propagation latency is directly proportional to distance.

## 6. RESULTS FOR MATHEMATICAL MODELLING

This section presents and discusses the results obtained after applying the approaches described in Section 4.

### 6.1 Optimal number of controllers

#### *6.1.1 Silhouette Analysis*

In order to determine the optimal number of controllers to deploy on the SANReN backbone, we applied our enhanced Silhouette algorithm with propagation latency as our key performance indicator. The results from our Silhouette analysis are as depicted in Figure 2.

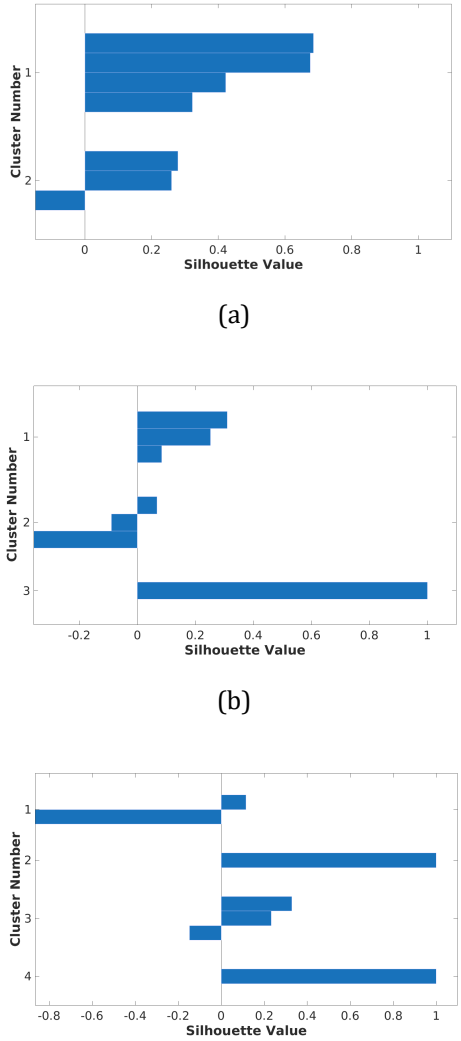


**Fig. 2** – Silhouette analysis to determine optimal number of controllers for (a) $k = 2$(b) $k = 3$(c) $k = 4$.

These plots show the clustering quality when different number of SDN controllers are deployed. For instance, Figure 2 $(a)$ illustrates the clustering quality when 2 controllers are deployed. The metric used to measure

clustering quality is the average intra-cluster propagation latency. Each blue horizontal bar in the plots represent a switch and its corresponding silhouette score. A silhouette score reveals the proximity of a switch to all other switches outside and within its cluster. Silhouette scores lie in the range of [-1,1]. The desired score is one that is closer to +1 as it indicates high proximity of switches within the same cluster. On the other hand, silhouette scores near -1 indicate high dissimilarity within a cluster and is a sign of poor clustering quality. A value of 0 shows that the switch is on or very close to the decision boundary between two adjacent clusters [53].

Our results indicate that deploying 3 or 4 controllers (shown in Figure 2 ($b$) and ($c$), respectively) would result in poor clustering quality due to the presence of clusters with very low silhouette scores and the high fluctuations in the size of the silhouette plots. The increase in average silhouette score from 3 to 4 controllers (as shown in Figure 3(a) ) is caused by the high proximity of nodes in the same cluster. Given that SANReN constitutes 7 nodes, deploying 4 controllers would result in the following network partitions:

- 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 sparse locations of the SANRen topology, it only makes sense that partitioning the network into 4 clusters would yield a higher average silhouette score than 3 clusters since there are fewer nodes per cluster and fewer outliers. Therefore, 2 controllers are the ideal number of controllers to deploy on the SANReN network as this will ensure lower propagation latency and a fair switch-to-controller distribution. This is seen from the high silhouette score obtained when the number of controllers is set to 2. Although deploying 4 controllers would yield a fairly good clustering quality and improve network reliability, it is likely to result in high inter-controller latency (due to the frequent state information exchange between controllers) and require high CapEx. However, if latency and cost are topmost priority, then 2 controllers are recommended. Moreover, 2 controllers would still suffice to meet reliability requirements unless the network has stringent requirements. However, different results are observed for different topologies.

### 6.1.2 Gap Statistic

To verify the results from our Silhouette algorithm, we applied the Gap Statistic algorithm on the SANReN topology. With Gap Statistic the optimal number of controllers corresponds to the highest gap value with the statistical deviation, as it reflects a low intra-cluster propagation latency. Figure 3(b) indicates that the optimal number to deploy on SANReN is 2 controllers. These results match the outcome of our Silhouette analysis.

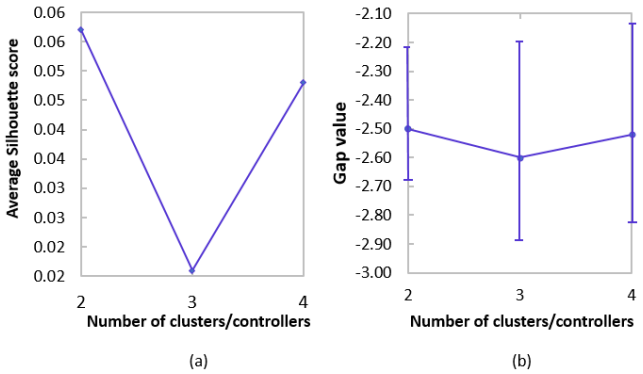


**Fig. 3** – (a) Silhouette and (b) Gap Statistic evaluation summary.

### 6.1.3 Cost-Latency Tradeoff Analysis

Another factor that influences the decision regarding the number of controllers to deploy, is the cost associated with installing new controllers in a given network. This metric is critical as it contributes to the overall CapEx and determines how much return on investment (ROI) network operators generate. However, there exists a considerable tradeoff between cost and the QoS delivered by the network. Our intention here is to quantify this tradeoff so as to provide a practical guideline to network operators, regarding the ideal number of controllers to use taking into account cost and latency. This tradeoff is termed "cost factor" and is defined in Eq. *(8)*, where $k$ is the number of controllers, $CPX_k$ is the normalised cost of deploying a single controller and $L_{avg}$ is the average latency when $k$ controllers are deployed. The normalised cost of deploying a single controller was kept at a constant value of 1$ per controller (assuming that a company plans to install the same model of an SDN controller).

$$cost\ factor = \frac{k * CPX_k}{L_{avg}} \quad \left[\frac{\$}{ms}\right] \qquad (8)$$

The average latency is the overall propagation latency computed using the PAM algorithm described in Section 4.3.2 for varying number of controllers. Figure 4 shows our results from analyzing the tradeoff between cost and network performance. As expected, the results indicate that deploying 1 controller is an ideal choice to ensure minimal tradeoff between cost and network performance. However, to ensure network scalability and failover, we recommend using 2 controllers. This is primarily because 2 controllers are the second best option that provides the least tradeoff, and our Silhouette and Gap Statistic analysis recommend 2 controllers as the optimal number to deploy on SANReN. It is important to note that our

proposed approach does not provide a comprehensive cost analysis, but only provides a basis for one.

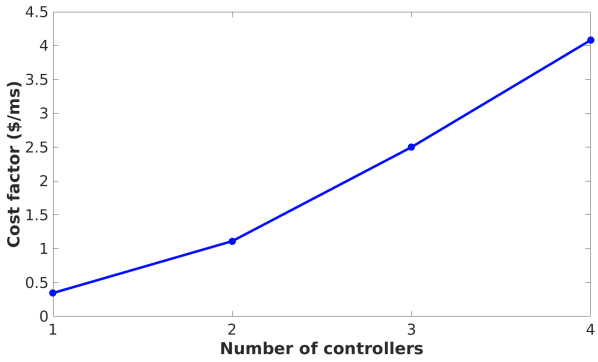


**Fig. 4** – Tradeoff between cost and latency for varying number of controllers.

## 6.2 Optimal controller locations

After determining the optimal number of controllers using the Silhouette analysis and Gap Statistic, the next step is to determine the best locations to place the recommended two SDN controllers. To find these locations, we use our proposed PAM algorithm described in section 4.3.2. The results (depicted in Figure 5) indicate that the optimal locations to place two controllers are Pretoria and East London with the average propagation latency of $L_{avg}$ = 1.81. The selection of these locations guarantees the best network performance with respect to the southbound communication in the SANReN network. In contrast, deploying the controllers in Port Elizabeth and Bloemfontein would result in poor network performance, with the worst-case propagation latency being $L_{wc}$ = 3.92.

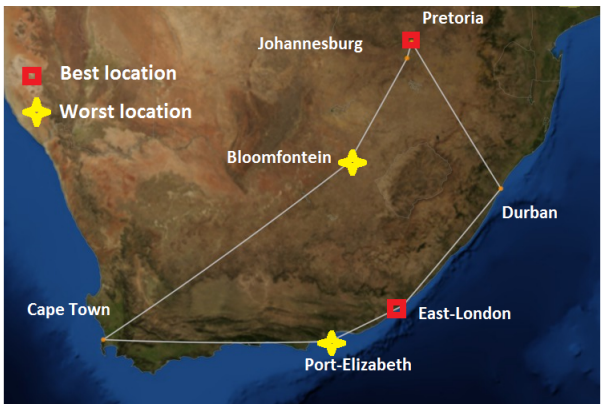


**Fig. 5** – Best and worst placements of two controllers on SANReN backbone.

Table 3 presents the effect of increasing the number of controllers ($k$) on average and worst-case latency. These results were obtained by applying the PAM algorithm. The results indicate that, varying the number of controllers from $k$=1 to $k$=2 significantly reduces propagation latency (approximately 38% reduction of average latency and 42% reduction of worst-case latency). A further reduction is observed when the number of controllers is set to $k$=3. However, increasing the number of controllers beyond 3 controllers has a much less significant impact on latency (as depicted in Figure 6).

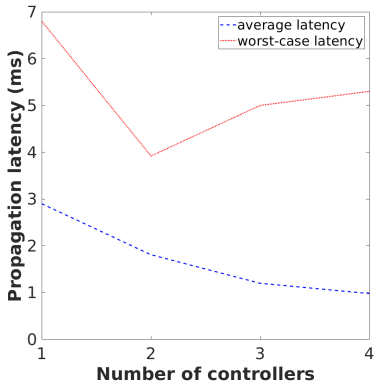


**Fig. 6** – Relation between number of controllers and latency.

# 7. CONTROLLER PLACEMENT ON EMULATED WAN

The controller placement results presented in section 6 relied strictly on mathematical modeling. In this section, we describe a method for finding optimal and worst locations of SDN controllers using an emulation orchestration platform called Mininet, which is able to include many of the practical implementation effects and so critical to mimic a real SDN deployment. We use controller-to-node latency (propagation + queuing +processing latency) as a key performance indicator. Our main goal is to match and verify the outcome from our mathematical formulation regarding the best locations to place the controller in a wide area network (WAN). To further optimize network performance, we also consider control plane resiliency, as well as propose a means to alleviate signaling overhead on the control channel.

For the control plane, we implement ONOS controller (version 1.14) because of its distributed core which improves the robustness of the control plane, by providing backup control in the event of network failure [54]. Moreover, ONOS' distributed core is self-coordinating and enables load sharing through fragmentation of the data plane. This controller has an advanced east/westbound interface to ensure high inter-controller communication efficiency. Finally, employing a geographically distributed core reduces the node-to-controller latency, thus improving the controller reactivity as perceived by the network nodes. Last but not least our decision to choose ONOS is influenced by the results from our ealier controller benchmarking experiments in [55] which confirm ONOS scalability features making it ideal for carrier grade deployments.

The evaluation of the proposed emulation approach is carried out on a model of a local national backbone called SANReN, the same network we used in section 5. It may

**Table 3** – Average ($L_{avg}$) and worst-case ($L_{wc}$) latency for varying number of controllers

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

however be noted that our approach is generic and can be used to optimize any other network.

## 7.1 Experimental Setup

The experiment setup is as illustrated by Figures 7 and 8 (captured from Miniedit). Node c0 and c1 are ONOS SDN controller instances running on a dedicated remote machine (with 8 CPUs, 16 GB RAM and 1 TB HDD and no swap partition), and h0-h6 are hosts attached to SDN Open Virtual Switches (OVS 2.9.90) running OpenFlow version 1.3. A built-in application for reactive flow instantiation is activated to set the ONOS controller to reactive operational mode. The red dash-dotted lines show connection (over WiFi) between switches and controllers and the blue solid lines are links between the switches. The switch-to-controller communication is assumed to happen out-of-band. Since the links between the switches are known to be fiber, where speed is approximately the speed of light in fiber i.e. about $2 \times 10^8 m/s$, we use the latency formula Eq. *(9)* to configure the link properties.

$$propagation\ latency\ (sec) = \frac{distance\ (m)}{speed\ \left(\frac{m}{sec}\right)} \qquad (9)$$

The distances between nodes are calculated using the Harvesine great circle approach and the actual GPS coordinates of the nodes.

The data plane emulated on Mininet version 2.2.2 (with default settings, for all experiments) is running on a separate machine (with 8 CPUs, 16 GB RAM and 1 TB HDD). Each switch in the data plane has a unique datapath ID (DPID). The connection between the control plane and data plane is via port 6633 of the controller over a slow WiFi router. The control link parameters are configured using the Linux Traffic Control (TC) utility (installed on the machine used for data plane emulation) under the assumption that the optimal controller placement is co-located with one of the switches. The programming language used to develop the software is Python 2.7.14.

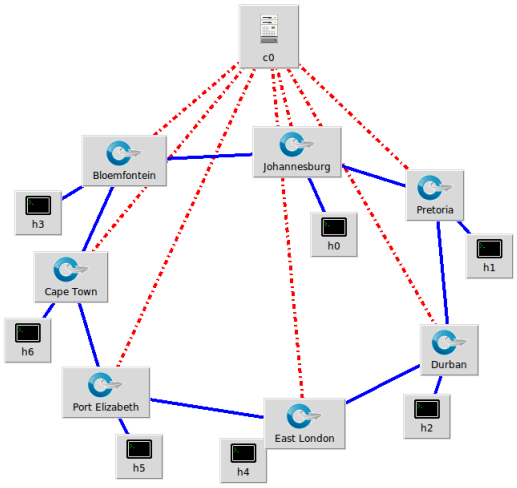


**Fig. 7** – Experiment setup with one ONOS controller

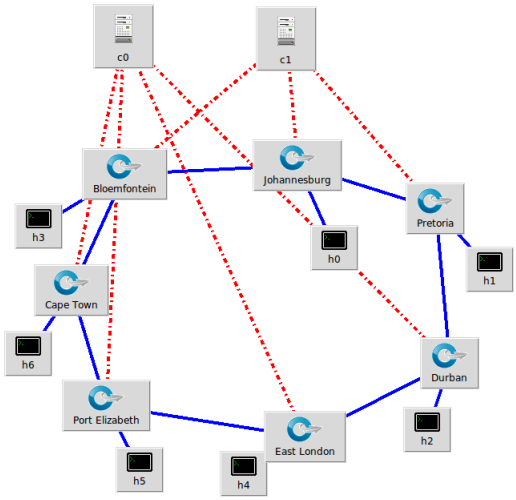


**Fig. 8** – Experiment setup with two self-coordinating ONOS controllers

## 7.2 Methodology

This work constitutes two independent experiments. The first experiment is carried out with the intention to address the controller placement problem leveraging emulation. The first experiment analyses two scenarios: (1) when only one controller is used and (2) when two controllers are used. The second experiment presents different approaches through which signaling overhead on the control channel can be reduced, in consideration of control plane resiliency.

### *7.2.1 Controller placement*

On example of one controller case, Figure 9 summarizes our approach in a flow chart (where $n$ is the total potential controller placement locations, i.e. the total number of nodes in a given topology). For the SANReN network, $n$ is 7, meaning there are a total of 7 potential controller placement locations in the network.

The following procedure (outlined in Figure 9) is used for each node to determine average latency: To find optimal controller locations, first we install the ONOS controller in the same geographic location as the first OpenFlow switch node (using the Harvesine great circle approach and the Linux TC utility). The next step is to trigger a packet-In message to the controller. This is done by generating traffic flows between all pairs, i.e. between this node and all other nodes in the SANReN topology. To do this we generate a ICMP packet using the ping utility for each pair. This is followed by computation of the ICMP pinging results to obtain the total average latency (round-trip time) from the node to all other nodes in the network. This step is repeated for all nodes in the SANReN topology. To ensure valid and reliable results, we repeat the above procedure 5 times under a soft idle timeout for the controller entry of 5 seconds (the soft idle timeout defines the expiry time of a controller flow rule when there is no flow activity) and compute the average results. The soft idle timeout is set to ensure generation of control traffic upon pinging reiterations.

For the case of two controllers (see Figure 10), the network is partitioned into two smaller administrative domains, namely cluster one and cluster two, each supervised by a dedicated ONOS instance. The parameters $n1$ and $n2$ denote the total number of switches in cluster one and two, respectively. After executing the mastership module, the partition results are as follows: The first ONOS instance ($c1$) is assigned three switch nodes in region Pretoria, Bloemfontein and Durban, while the other ONOS instance constitutes switches located in Johannesburg, Cape Town, East London and Port Elizabeth. In order to optimize the placement of these two controllers, an exhaustive search is carried out by iterating through all possible combinations (within the limits defined by each controller domain). In other words, $c1$ is placed at different regions within its domain. For each placement

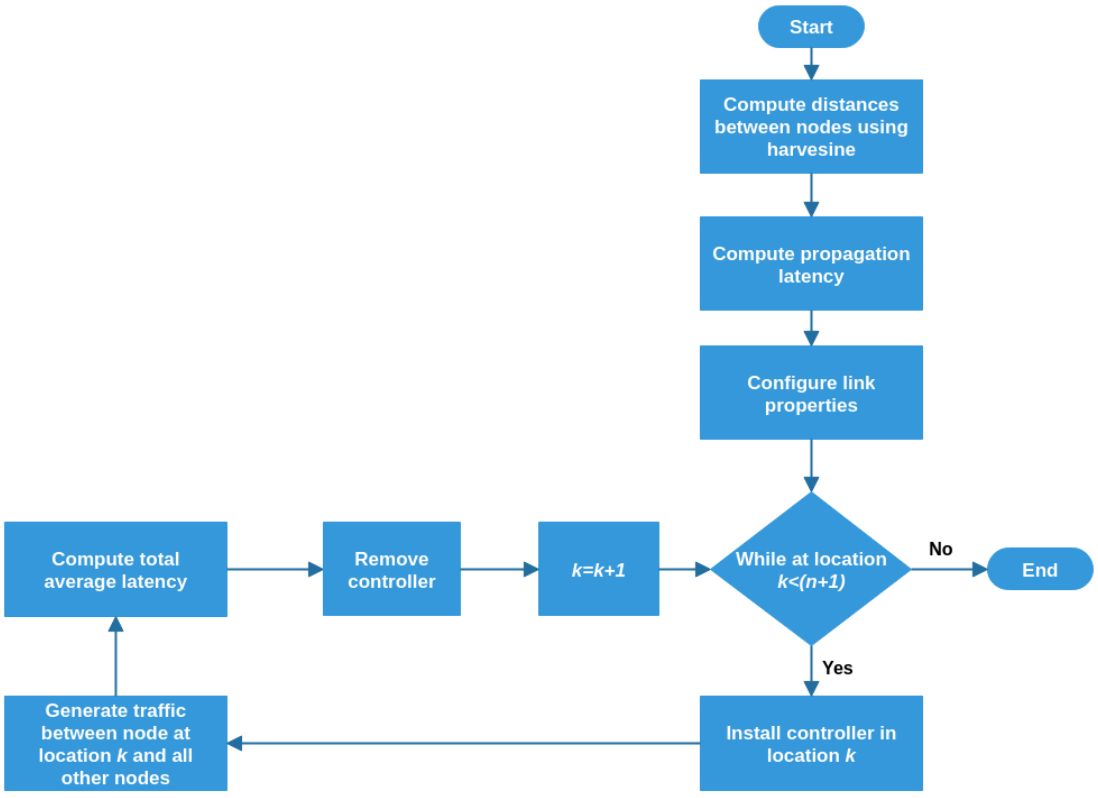


**Fig. 9** – Flow chart of proposed method for one controller[56].

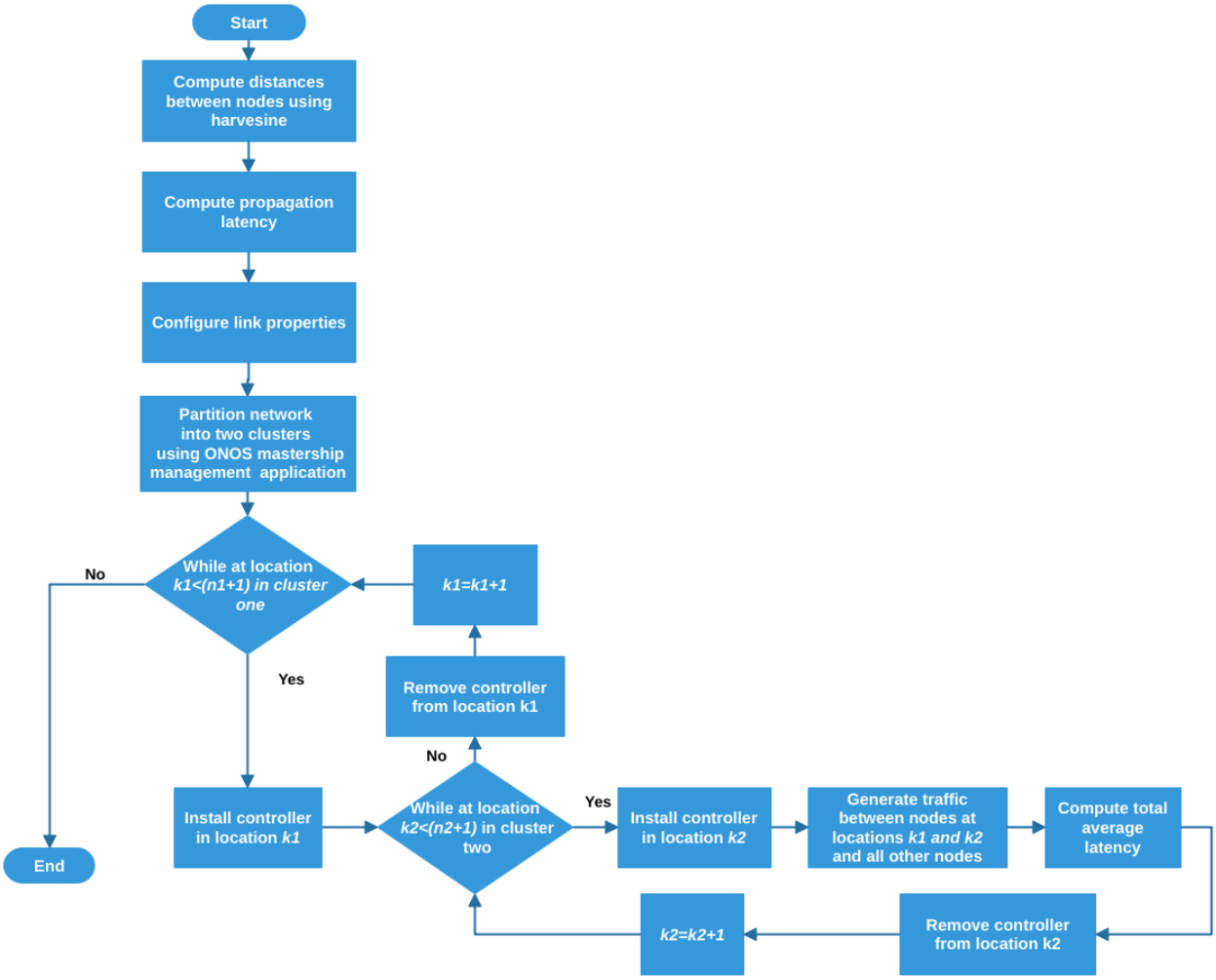


**Fig. 10** – Flow chart of proposed method for two controllers.

of $c1$, $c2$ is then placed at different regions within its domain. For each set of placement, the average latency is computed following the same procedure as described above.

### *7.2.2 Control plane signaling overhead and failover*

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

To quantify the impact of switch-to-controller placement, we generate variable traffic between two virtual hosts (the client connected to Johannesburg and the server connected to Cape Town), a distance of 1399 km from each other. This is carried out using the Distributed Internet Traffic Generator (D-ITG) tool. The transport protocol is set to UDP and the number of packets per second is varied from 50 000 to 200 000 in increments of 50 000. The packet size is set to 512 bytes. The link bandwidth was kept at 10 Mb/s. The duration for the generation process is set to 5 minutes. The key performance indicators are delay,

jitter and packet loss all monitored at the server end. This procedure is carried out for two scenarios: 1) when the switch-to-controller placement is imbalanced (switch-to-controller assignment is two and five switches for controller one and two respectively) and 2) for the scenario where switch-to-controller placement is balanced (switch-to-controller assignment is three and four for controller one and two respectively).

In addition to switch-to-controller balancing, the control plane has several tuneable parameters in the control plane, such as polling frequency and soft idle timeout [57]. Polling frequency is a parameter that specifies how frequently statistics requests are sent to the data plane. Soft idle timeout specifies the total time an inactive flow entry is stored in the flow tables before deletion. Tuning these parameters impacts control plane overhead. In other words, increasing polling frequency is likely to decrease the control plane overhead (of course at the expense of data plane protection and restoration) while increasing the soft idle timeout results in more flow rules in the flow tables and reduces control plane overhead (with the switch resource (e.g. memory and storage) exhaustion as a tradeoff). In an operational environment, OpenFlow switches with TCAM (Ternary Content Addressable Memory) support are typically preferred for fast processing [58]. However, TCAM is very expensive with very limited memory space [59]. Therefore, the soft idle timeout can only be increased up to a certain threshold to maintain the switch memory utilization around acceptable levels.

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

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

## 7.3 Results and Discussion

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

### *7.3.1 Controller placement*

Figures 11 and 12 present the results obtained from our analysis of the SANReN network. As per Figure 11, our results show that the optimum controller location when one controller is deployed is Cape Town since this node has the lowest average latency ($L_{avg}$=88.78 ms). Similarly, the worst location to place the controller when one controller is deployed is Bloemfontein since this location yields the highest average latency ($L_{avg}$=164.4 ms).

Figure 12 presents the results obtained when two controllers are deployed. These results are intepreted as follows: the blue bars indicate a scenario where one controller is placed in Pretoria (a region belonging to cluster one as described in section 7.2.1), while the other controller's location is iterated between Johannesburg, East London, Port Elizabet and Cape Town (regions belonging to cluster two). Similarly, the red and green bars indicate controller placement in Durban and Bloemfontein (regions belonging to cluster one) while the

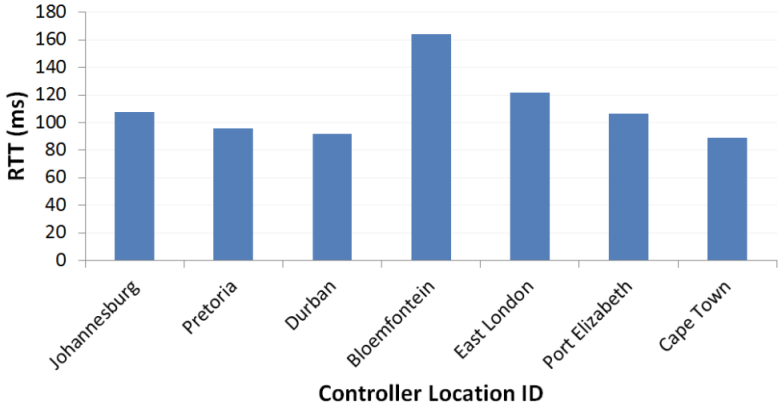


**Fig. 11** – Total average latency for the ONOS controller without clustering.

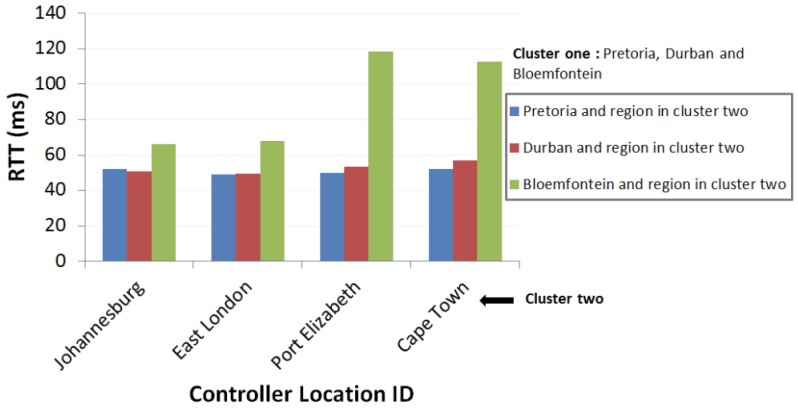


**Fig. 12** – Total average latency for ONOS controller when network is partitioned into two clusters.

other controller is placed in all regions within cluster two. Our results shows that when two controllers are deployed and the mastership management module is activated, the optimum controller locations are Pretoria for cluster one and East London for cluster two, with $L_{avg}$=48.9 ms. The worst locations are Bloemfontein and Port Elizabeth for cluster one and cluster two respectively, with $L_{avg}$=118.4 ms. These results coincide with the results from our mathematical formulation in section 6.2.

### *7.3.2 Control plane failover and signaling overhead*

The outcome of our failover tests was positive in that all nodes could reach each other regardless of the failed control node. This means that the switch nodes under the supervision of the failed controller were automatically reassigned to the active controller in the other cluster. The reassignment took approximately 0.5 seconds. The reassignment time was measured by carrying out the ONOS performance benchmark test case provided in [60]. The failure recovery time increases significantly with the number of disconnected switches [61]. The recovery time can potentially be improved by using more powerful servers with more RAM and CPU resources [62].

Figure 13 and 14 depict the results we obtained both before and after switch-to-controller placement balancing. As expected (see Figure 13), the average delay is in overall lower after switch-to-controller placement balancing compared to the case of imbalance. We believe this is primarily because, after balancing the switch-to-controller placement, data-plane monitoring traffic is fairly divided between the controller nodes thus improving overall network performance.

The decline in average delay (both before and after switch-to-controller balancing) is a result of an increased matching probability of preserved flow rules with newly arriving packets, which reduces the number of packet-In messages to the controller, resulting in a reduction in network delay. When the number of packets is increased

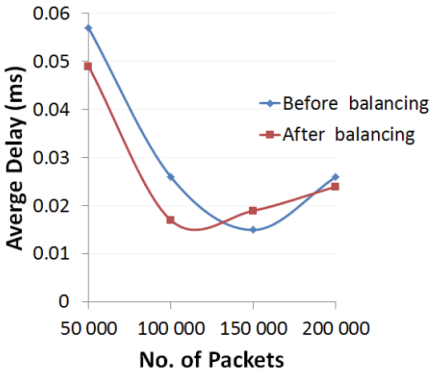


**Fig. 13** – Average latency.

to 150 000 packets, an increase in average delay is observed. This is likely because during the transmission of the first 150 000 packets, the switch has matching entries in its flow tables on how to route traffic, which eliminates the need to forward incoming packets to the control plane for routing decisions. After a certain time (from 150 000 packets upwards), the switch reaches a hard timeout and clears its flow tables leading to an additional processing delay. The additional processing delay is likely the cause of the increase in average delay. Similar results are observed with regards to network jitter (as shown in Figure 14). Last but certainly not least, when the number of packets is increased

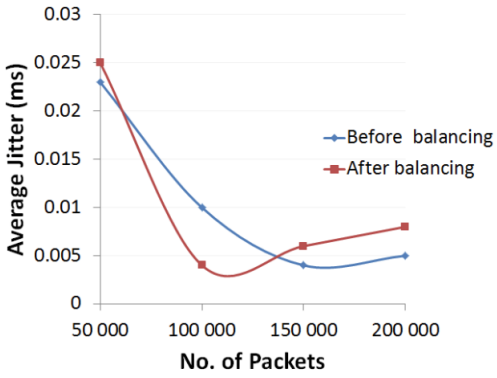


**Fig. 14** – Average jitter.

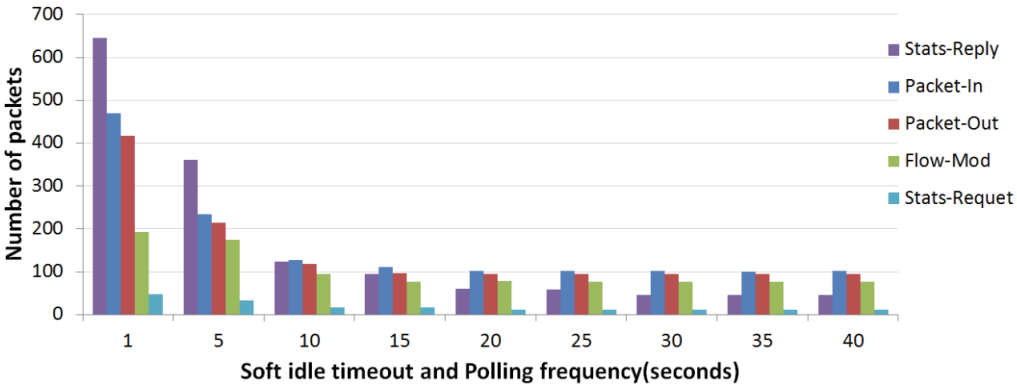


**Fig. 15** – Impact of soft idle timeout on control plane overhead.

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

Figure 15 depicts the impact of tuning the soft idle timeout and polling frequency on control plane overhead. The results indicate that, increasing the polling frequency and soft idle timeout decreases the number of control packets (synonymous with control plane overhead) generated during reactive flow instantiation. However, from 20 seconds forward, the number of control packets remains constant. Therefore, we can conclude that configuring the polling interval and idle timeout to 20 seconds would be the sufficient choice to achieve an acceptable control overhead and a potentially lower switch memory utilization. Increasing the timeout beyond 20 seconds would not change the load on the control channel but will potentially lead to a higher memory utilization.

## 8. SOURCE CODES

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

## 9. CONCLUSION

This study considers determining the number and location of SDN controllers in a wide area network, and associated performance and cost implications and is intended to be used to address the SDN controller placement problem. The work is applied to a national network from a developing country, SANReN. The work includes mathematical modelling and a method for obtaining the results through emulation on a popular controller suitable for real world deployments. The emulation confirmed the modelling and is also used to derive important practical limits. The modeling included Silhouette, Gap and PAM approaches. Using graph modeling, two "unsupervised" machine learning algorithms namely, Silhouette and Gap Statistic algorithms were applied to optimize the number of controllers to deploy in a given topology. Given the fact that network operators are more concerned about the cost associated with network deployment, this study also takes into consideration the trade-off between cost of installing a new SDN controller and performance. This is necessary to facilitate decision making regarding the number of controllers to deploy, based on performance requirements and cost constraints. To determine the optimal locations to install the controllers, a classical algorithm called PAM was used. The applied algorithms are exhaustive making them ideal for static controller placement with minimal to no time constraints. In order to mimic a real SDN deployment and also to verify the outcome from our mathematical model, we use exhaustive search on an emulator to address the controller placement problem. This approach also takes into account resiliency and control plane overhead metrics. We use the ONOS SDN controller due to its inherent self-coordinating distributed core. Our emulation results show that running a single controller yields high reaction times as some switches are located too far away from the controller. Moreover, running a single controller is not enough to meet resiliency requirements. When the number of controllers was increased to two, the reaction time was reduced considerably since the network was subdivided into two administrative domains. Moreover, the two controllers worked collaboratively to alleviate control overhead and ensure resiliency in the network. Leveraging our controller placement results as well as balancing the switch-to-controller placement, we also investigated the impact of soft idle timeout and polling frequency on control plane overhead. Our finding suggested that a large soft idle timeout and polling frequency reduces the overall control plane overhead. In reading the above conclusions, it should be noted that the solution to the controller placement problem is topology dependent and thus the results presented by this work only apply

to the SANReN topology studied. However, the proposed approach is "protocol" agnostic and can be adopted to solve controller placement in any SDN-enabled data plane (such as data planes that support protocols such as P4 Runtime, a successor to OpenFlow). We believe that our method and analysis would be beneficial for operators and service providers, not only during the initial design, but also during the incremental design of the SDN-enabled networks.

## 10. FUTURE WORK

In future we intend to extend our work to address dynamic controller placement which is necessary to meet 5G requirements such as ultra reliable low latency communications achievable through dynamic placement of the mobile edge computing node. We also plan to evaluate the security aspect of SDN controllers. This is motivated by the fact that centralizing the network control intelligence presents a single point of attack/failure.

The assumption that the bandwidth for all connection links is constant (see section 4.1) is not valid for the actual SANReN network. This assumption was made to simplify the mathematical model formulation. In future, a more complicated scenario with different link bandwidths will be considered.

In this work, traffic load was set to 512 bytes which does not reflect the actual traffic exchanges between SANReN nodes. As future work, a characterisation of the actual traffic profiles combined with a rerun of the evaluation in section 7 will be considered. We also intend to use the proposed approach to optimize data planes running P4 Runtime protocol. Last but not least, we intend to develop a dynamic control traffic load balancing application based on current switch resource (RAM, CPU, and storage) utilization.

## REFERENCES


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

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

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

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

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

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

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

[8] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[25] L. Mamushiane, A. A. Lysko, and J. Mwangama, "Controller placement optimization for Software Defined Networks." ITU Journal, 2021.

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

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

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

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

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

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

[32] G. S. Vincent, "The haversine formula," Last accessed September 2018). [Online]. Available: http://www.longitudestore.com/haversine-formula.html

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

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

[35] A. Prakhar, "Haversine formula to find distance between two points on a sphere," Last accessed February 2019. [Online]. Available: https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

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

[37] J. D. Donnay, *Spherical trigonometry*. Read Books Ltd, 2013.

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

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

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

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

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

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

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

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

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

[47] P. O. Olukanmi, F. Nelwamondo, and T. Marwala, "k-means-lite: Real time clustering for large datasets," in *2018 5th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, 2018, pp. 54–59.

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

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

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

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

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

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

[54] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE access*, vol. 5, pp. 1872–1899, 2017.

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

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

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

[58] S. Xu, X. Wang, G. Yang, J. Ren, and S. Wang, "Routing optimization for cloud services in sdn-based internet of things with tcam capacity constraint," *Journal of Communications and Networks*, vol. 22, no. 2, pp. 145–158, 2020.

[59] J.-Y. Huang and P.-C. Wang, "Tcam-based ip address lookup using longest suffix split," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 976–989, 2018.

[60] "Test case on the role intimation time for controller cluster at setup," Last accessed January 2019. [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=11864465

[61] "The role intimation time for controller cluster after master failover," Last accessed September 2018). [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=11864470

[62] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

## AUTHORS

**Lusani Mamushiane** received BEng degree in Electrical and Electronic Engineering from the University of Johannesburg and MSc in Electrical Engineering from the University of Cape Town (UCT), South Africa, in 2014 and 2019 respectively. In 2015, she joined Microsoft where she worked as an Electronic and Telecoms Engineer for Intelligent Traffic Systems. In 2016 she then joined Advanced and Network Architecture Systems research group in Next Generation Enterprises and Institutions (NGEI) cluster, CSIR as a Researcher. Ms Mamushiane's research interests include but not limited to Optical Networks, Software Defined Networks (SDN), Network Function Virtualization (NFV), Intelligent Traffic Systems and 5G Mobile Architectures and Services.

**Joyce B. Mwangama** received BSc degree in Electrical and Computer Engineering and MSc in Electrical Engineering from the University of Cape Town (UCT), South Africa, in 2008 and 2011 respectively. In 2015, she joined the Department of Electrical Engineering as a Lecturer and received the PhD degree in Electrical Engineering from the University of Cape in 2017. Her current research interests include Future Internet Technologies, 5G Mobile Network Architectures and Services, Software Defined Networks, Network Function Virtualization, Machine-to-Machine Communications and the Internet of Things. Dr Mwangama is a member of the IEEE and the South African Institute of Electrical Engineers (SAIEE).

**Albert A. Lysko** received MSc in Radiophysics from St-Petersburg State Technical University, Russia in 1998 and PhD in ICT from Norwegian University of Science and Technology (NTNU), Norway in 2010. Dr Lysko started working before his first degree and has over 25 years of combined industrial and academic experience gained in Russia, Norway and South Africa. From 2007, he is with the Council for Scientific and Industrial Research (CSIR) in South Africa, where he is a Principal Researcher. He is also an Adjunct Professor with the University of Cape Town and Extraordinary Researcher with the North-West University, South Africa. His research interests include networking, especially software-defined networking; numerical modelling; wireless communications and especially television white spaces (TVWS). He authored and co-authored a book, two book chapters, three patents, several technology demonstrators, numerous technical reports, and over 100 research, popular science and news publications. He has made contributions to national regulations in South Africa. His work affected TVWS regulations in the USA. The TVWS work was recognised with a national-level award. Dr Lysko has organised over 100 events and 3 international conferences. He has shared three best paper awards, given numerous invited talks including a plenary and keynote talks, and served on multiple national and international funding and research/standardisation panels/committees. He is an award-winning volunteer and a Fellow of South African Institute of Electrical Engineers (SAIEE) and a Senior member of IEEE. He also represented South Africa in two EU Cooperation in Science and Technology (COST) Actions IRACON and VISTA.