Scientific Research Publishing

Journal of Computer and Communications, 2024, 12, 84-100
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

# Comparative Performance Measurement of the Pareto Optimal Combination and Multi-Objective Combination Models for Controller Placement in Software-Defined Networks

**Mission Franklin, Constance Izuchukwu Amannah**

Department of Computer Science, Ignatius Ajuru University of Education, Port Harcourt, Nigeria
Email: aftermymsc@yahoo.com, aftermymsc@gmail.com

**How to cite this paper:** Franklin, M. and Amannah, C.I. (2024) Comparative Performance Measurement of the Pareto Optimal Combination and Multi-Objective Combination Models for Controller Placement in Software-Defined Networks. *Journal of Computer and Communications*, **12**, 84-100.
https://doi.org/10.4236/jcc.2024.123006

**Received:** June 25, 2023
**Accepted:** March 17, 2024
**Published:** March 20, 2024

## Abstract

The evolution of the current network has challenges of programmability, maintainability and manageability, due to network ossification. This challenge led to the concept of software-defined networking (SDN), to decouple the control system from the infrastructure plane caused by ossification. The innovation created a problem with controller placement. That is how to effectively place controllers within a network topology to manage the network of data plane devices from the control plane. The study was designed to empirically evaluate and compare the functionalities of two controller placement algorithms: the POCO and MOCO. The methodology adopted in the study is the explorative and comparative investigation techniques. The study evaluated the performances of the Pareto optimal combination (POCO) and multi-objective combination (MOCO) algorithms in relation to calibrated positions of the controller within a software-defined network. The network environment and measurement metrics were held constant for both the POCO and MOCO models during the evaluation. The strengths and weaknesses of the POCO and MOCO models were justified. The results showed that the latencies of the two algorithms in relation to the GoodNet network are 3100 ms and 2500 ms for POCO and MOCO respectively. In Switch to Controller Average Case latency, the performance gives 2598 ms and 2769 ms for POCO and MOCO respectively. In Worst Case Switch to Controller latency, the performance shows 2776 ms and 2987 ms for POCO and MOCO respectively. The latencies of the two algorithms evaluated in relation to the Savvis network, compared as follows: 2912 ms and 2784 ms for POCO and MOCO respectively in Switch to Controller Average Case latency, 3129 ms and 3017 ms for POCO and MOCO respectively in Worst Case Switch to Controller latency, 2789 ms and 2693 ms for POCO and MOCO respectively in Average Case Controller to Controller latency, and 2873 ms and 2756 ms for POCO and MOCO in Worst Case Switch to Controller latency respectively. The latencies of the two algorithms evaluated in relation to the AARNet, network compared as follows: 2473 ms and 2129 ms for POCO and MOCO respectively, in Switch to Controller Average Case latency, 2198 ms and 2268 ms for POCO and MOCO respectively, in Worst Case Switch to Controller latency, 2598 ms and 2471 ms for POCO and MOCO respectively, in Average Case Controller to Controller latency, 2689 ms and 2814 ms for POCO and MOCO respectively Worst Case Controller to Controller latency. The Average Case and Worst-Case latencies for Switch to Controller and Controller to Controller are minimal, and favourable to the POCO model as against the MOCO model when evaluated in the Goodnet, Savvis, and the Aanet networks. This simply indicates that the POCO model has a speed advantage as against the MOCO model, which appears to be more resilient than the POCO model.

**Keywords**

Latency, Measurement, Metrics, Performance POCO, MOKO, Architecture, Provision

## 1. Introduction

Software-defined networking (SDN) came into existence as a response to meeting these industry difficulties. The dynamic reaction to changes in usage patterns and availability of network resources is what SDN brings to the table. This ensures instant adjustment of Network architectures, making it respond to applications and user requests, and therefore, at a lower cost, services can far more rapidly and effortlessly be introduced [1]. This is because networks these days can expand beyond regional boundaries, with network devices located in different parts of the globe. These devices need to be configured to work efficiently with the other devices in the network, but the configuration is practically manual and programmability is not possible.

SDN exemplar is what offers the network a future life, to handle the challenges that the network faces. In SDN, the principal thing is the decoupling of the network devices into separate planes and allows the network to be programmable, with ease of maintainability and management. SDN is a fairly new concept that promises to address the problem of deficiency of programmability in current networking architectures and encourage quicker and stress-free network innovation. The SDN paradigm is to divorce the control plane from the data plane, and enable complex networking applications through software implementations over the control plane [2]. Thus, this would ensure that the data plane functions as forwarding devices, to transmit packets, while the control logic is moved out to a separate plane called the control plane with network intelligence. In the control plane, the function of control is then domiciled in a conventional computer system as a network-controlling software architecture component. The idea is that with the availability of cheaper hardware resources with lesser specifications, it would be possible to have software applications that can control the hardware through standardized interfaces. With the help of the available application programming interfaces (APIs), the programmability of these devices becomes flexible; therefore, we can dynamically programme additional operational features as network applications [2].

SDN architecture brings a lot more promises to the network allowing for programmability, maintainability and manageability. The idea is to move the complexity away from the hardware and implement the same functionality of flexibility and innovation in software. With this concept in place, the complexity of hardware design and functionality of forwarding traffic is now simplified, while the software is responsible for network management and network resources as well as the instructions to the forwarding devices, on how to route network traffic, which is now abstracted as a flow. SDN promises to advance the concept of programmability of these low-level devices. SDN provides split-up between data plane (Switches/routers) and the control plane (Controller). The interaction between both planes is achieved through a communication protocol that forwards instructions that effectively modify forwarding table flow entries in network Switches.

According to [2], the concept of SDN was adopted from operating systems of mobile phones such as Android (Google) and iOS (Apple), where the system allows dynamic addition of applications. While businesses grow and impact the performance of the network, the desire to reach low-level devices for quick reconfiguration becomes a challenge as the devices may be located in dispersed geography. Therefore, making changes to these devices at run time is a major problem that needs to be solved through the concept of software-defined networking. The new concept of divorcement of the data plane from the control plane creates a mechanism where the low-level devices can be controlled by a conventional computer system within a secure communication channel between the Switches and the network Controller [3].

With the nature of SDN deployment, the network administrators have means of innovatively managing the network. The network is centrally controlled by software to manage all the data plane devices that are responsible for forwarding. All network traffic is abstracted as a flow irrespective of whether it carries data over IP, Ethernet or other lower layers. Also, network management is flexible through dynamic updating of the forwarding rules in the forwarding devices. Since the Controller is software-based, traffic analysis is done based on the software that controls the behavior of the network through the analysis of the flows. SDN separates switching software from the actual network hardware and the controls of network devices from the data they transport.

The SDN architecture does not just separate the data plane from the control plane, it also provides a controlling entity that has an all-inclusive logical view of the entire network resources (Switches and routers), and their statuses, and the applications can communicate in real-time. The possibility of networks to interact with applications and efficiently reconfigure themselves on a need basis where necessary is enabled by a Controller, allowing multiple logical network topologies implementation on a single common network [1]. The abstraction of the central Controller, which ensures that the network topology and the network state are maintained consistently, also detects and identifies the pattern for the flows that transverse the network. Therefore, the network can be controlled from the impacts of security attacks due to malicious and mutated packets that transverse the network.

With SDN it is possible to optimize networks to respond swiftly to fluctuations in network usage without the need to reconfigure existing network infrastructure manually, or in some cases to procure a new hardware. These required modifications will only be sustained in the current Internet through enormous investment and changes in the architecture. It has become necessary to consider an evolution of the Internet architecture based on the concept of software-defined networks, which will lead to much more efficient use of available resources and provide a business environment that encourages investment and network programmability [4].

To make this possible, the Open Network Foundation has proposed and developed the OpenFlow design for communication protocol between the control plane and the data planes for effective and secure communication protocol [5]. OpenFlow has streamlined the communication protocol for interactions with the aid of an application programming interface (API) towards the southbound devices [6]. OpenFlow is popular though, but it is not the only implemented protocol for interaction with the forwarding devices. Much architecture exists to ensure improved capacitation and latency improvement trade-offs.

SDN deployment and research have increased due to its promises of network innovation, flexibility, maintainability, programmability, and manageability [7]. Several SDN deployments have improved services of networks like cloud services providers, data centre network management, automation of service provisions, etc. With software-defined networking (SDN) we can build and deploy changes with ease without time-consuming days of software upgrades, configuration nightmares and steep learning curves [8].

In the search for a reliable solution for the placement of controllers for the efficient management of an SDN network, we have conceived the idea of hybridization of the features of the Pareto optimal controller placement (POCP) and the multi-objective controller placement (MOCP) of the network parameters which serves as synergy of functionalities that may guarantee better placement, however, with limitations in scalability, resiliency and energy awareness. Our research then seeks to improve the hybridization of the POCP-MOCP by the enhancement of the hybrid controller through the introduction of additional features of energy-awareness, scalability and network resiliency to make our solution robust for efficient management of the network, making it fault-tolerant, with energy minimization and improved scalability.

The aim of the study was to measure the performance outputs of the Pareto optimal combination (POCO) and multi-objective combination (MOCO) in determining controller placement in a software-defined network. The study intended to:

1) Evaluate the performance of the Pareto optimal combination (POCO) in determining controller placement in a software-defined network;

2) Evaluate the performance of the multi-objective combination (MOCO) in determining controller placement in a software-defined network;

3) Compare the results of the POCO and the MOCO models, given the same environment and measurement metrics;

4) Justify the strengths and weaknesses of the POCO and MOCO models in determining controller placement in a software-defined network.

## 2. Related Literature

Transmission rates have improved. The capacity of transmission scheme and network devices' performance has also improved. However, traditional network has consistently proved to be inadequate [9]. This is due to the increasing complexity, expansive variability, and high volume of load that the system is currently absorbing, coupled with the quality of experience (QoE) and quality of service (QoS) imposed as network requirements by several applications on the network. The ability of the orthodox network to respond effectively to these demands requires an improved traffic load handling with sophistication and agility. Though the traditional Internetworking uses the TCP/IP protocol architecture approach, the approach is functionally dependent on these characteristics: 1) Two-level end system addressing, 2) Routing based on destination, and 3) Distributed, autonomous control [10].

The traditional network has heavy reliance on the identity of the network interface. The architecture at the networking level is a network of networks, with layers within layers. TCP/IP was used to support the networking of autonomous network, with distributed control. The architecture provides scale and reliability in terms of growth. Routes are discovered and used all over the Internet with the help of IP and distributed protocol. Also, using TCP which is a transport level protocol, decentralized and distributed algorithms can be implemented. These algorithms help to decongest the network [11] [12]. In traditional networks, routing was based on the packet's destination address (MAC/IP). The transmission uses a datagram technique of the packets traversing the network of routes with less delay to reach the destination. However, this technique has its draw back on the quality of service (QoS). The QoS requirements treat packets in terms of flows. A flow is a "sequence of packets between a source and destination that are recognized by the network as related and are treated in a uniform fashion [5]. Every packet that is related to a particular flow has definite QoS characteristics that define the routing for the entire flow. The orthodox networks were mainly static and end systems are in largely fixed location, due to the distributed and autonomous approach adopted in the development framework [2] [9].

There are limitations that the traditional network was ailed with, and that made it necessary to give way for a new model design. The open networking foundation (ONF) identifies four general limitations of out-of-date network architectures.

1) The static and complex nature of the architecture, making it difficult to manage different level of QoS, security requirement, changing traffic volume, which has created vendor dependent protocol to address the issues. However, issues are not fully addressed, rather more problems are generated like configuration of network devices (Switches, routers, firewall); updates on LAN setting, adjustment of QoS parameters based on changing user requirements and traffic patterns. In most cases manual configuration is adopted to achieve the objective.

2) Policy inconsistency is a problem when network wide security policies cannot be applied; in large network an unreasonable amount of time is spent to reconfigure devices across the network.

3) Scalability issue is also a major drawback, as demand on networks grows rapidly with varying features, creating challenges that are due to network expansion. This challenge is due to the static and complex nature of the network. Even with the strategy of oversubscription of the links, has not yield much solution, due to increased virtualization and the use of the increasing variety of multimedia applications, which has made the traffic pattern unpredictable.

4) Vendor lock-in/Lack of open interfaces, has left the enterprises with a relatively slow product cycles of vendor equipment, therefore the deployment of new capabilities and services that would respond rapidly to changing business needs of user demands is not available [9] [10].

The birth of software-defined networking (SDN) is necessitated by the complexity of the computing demands that impressed upon the services that the current network architecture supports. The current architecture is laden with a wide spectrum of networking services demand ranging from network applications, network control, bandwidth and other network resources, network agility, efficient network management and service innovation [13].

The idea of network programmability and management did not come from the desire to change network architecture. The notion stems from the challenges that the current network architecture is facing. Also, the growth and popularity of new network services models such as social media, cloud computing, big data analytics, and mobile applications; security of data and information has placed high demand for network efficiency, flexibility, reliability, effective access to storage and computational resources, and agility of the network environment are now very critical factors in the assessment of performance of the network [14].

In the midst of this challenge, and with the desire by both industry and academic professional to find solution to this problem, several networking technology solutions were proposed, developed and implemented with huge investments of financial and technical resources. This was done to enhance and expand the network infrastructure and capability to meet the increased demands of network services.

The conception of SDN was a novel idea to address the challenge that current networks face due to the overwhelming demand placed by networks users and applications because the network complexity that is observed with the current architecture is as a result of the network protocol solutions being defined without consideration of other existing problems and does not take advantage of fundamental coherence [1] [10].

It is this complexity that makes current network not able to meet the service requirements of future networks. Because the network today is limited by “static rigidity”, which is a source of concern for the operators of the network, as they sought to reduce the phenomenon of service disruption risk. Static rigidity negates network dynamism; therefore dynamic service provisioning is not supported by current networks, which is a requirement for future networks. In addition, the complexity in the current network adds to the difficulty experienced when enforcing consistent network policies across heterogeneous network devices [12].

The OpenFlow procedure for communication was an initiated protocol development by Stanford University researchers in 2008. The first implementation of this protocol was 2011 by Google, to support their network backbone. The success recorded at Google, necessitated its management by a consortium of networking companies under the umbrella body known as open networking foundation (ONF), which has worked from the initial version of OpenFlow 1.0 to its current version of OpenFlow 1.5 [15] [16].

OpenFlow is an application software based control mechanism that controls and manages flow processes in an SDN based network. The OpenFlow based Controller serves as the network operating system (NOS). It is regarded as the communication channel between the network applications and the network devices, interfaced by the network Controller. The network controller intermediates through its OpenFlow control protocol, by taking the information from the decisions of the applications, transform them into flow entries and pass it on to the network devices. The flow entries are to direct the network devices where to send the packet information, using the best path possible path, based on the instructions stored in the (forwarding table) flow table [17].

OpenFlow outlines the standard of communication between the intelligent Controller and the dumb data plane network devices. Basically OpenFlow is a standard southbound protocol for interactions between the Switch and the Controller, via the OpenFlow APIs. Due to the overall control and management of the Controller, applications information in the form of decisions are passed through the Controller to the Switches, via OpenFlow protocol implemented in APIs [18]. With the aid of OpenFlow the Switches and their ports can be monitored, for its statistics and management. The activity for which OpenFlow is involved does not affect any other part of the network, but strictly the interactions between the Controller(s) and the Switch(es). The inter-Switch interactions between Switches, do not reveal to each other their respective interactions with the Controller. The Switches obediently and confidentially perform their functions of forwarding the packets according to the instructions stored in their respective flow tables [19].

OpenFlow implementation is via software in the control plane, which resides in the central entity identified as the Controller that communicates with the data plane, where the network nodes are physically distributed. With the specifications of OpenFlow, a Switch can be programmed to operate like Switches in the conventional network with similar outcomes. This is achievable only with manual configuration and reconfiguration of the network devices at the data plane where the network changes.

OpenFlow protocol messages were intended to be brief, powerful and minimal in construct. There three type of OpenFlow messages: 1) Controller to Switch messages, 2) Switch to Controller, 3) symmetric messages.

1) Controller to Switch messages writes instructions (entries) to the flow tables that the Switch uses to forward packets, and also request for statistics from the Switch.

2) Switch to Controller messages are asynchronous message type, send packets or bytes counter for flows defined in the Switch, and send packets not matching a flow defined in the Switch.

3) Symmetric messages are messages for initiating interaction, originating from the Switch to the Controller. These messages includes hello (startup), experimental messages for extensions, and echoes (example, a measure of control path latency, heartbeats).

SDN architecture is a subdivision of the network system based on SDN concept into functional parts and the interfaces between the parts. The SDN architecture is consistent, open and useful, without the revelation of contractions when observed from multiple-perspectives. The SDN architecture is an open architecture because it is extensible. The SDN architecture describes the main components and the key functionalities of each component as well as the interaction interfaces between the components parts. The architecture of SDN is the principle component structure; it offers the procedures for SDN technical development. All network-related standard organizations like ONF, ITU-T, IRTF have developed specification for [9].

The main object of SDN was to offer open interfaces that defines data forwarding and processing operations, to be carried out by a set of traffic flows on the network resources through a software means. This is achievable by the separation of data forwarding function that is controlled by a devoted entity at the control plane called SDN Controller. The Controller offers a means of managing and controlling of network resources software, usually known as SDN applications [14] [17] [20].

The Controller provides controlling and management functionality to network resources through software which functions as SDN applications. Conceptually, the architectural characteristic components of the SDN are categorized and formalized into three distinct groups of structures: Data plane, Control plane, and Application plane [8]. While the interfaces which ensure that interaction of the layers are, the north bound interface, southbound interface, and the east-west interface [6] [11] [21] [22].

The SDN architecture generates challenges in terms of the controlling entity effectively controlling the data plane, this is due to the asynchronous nature of the communication channel between the Controller and the Switches; therefore, communications is initiated with different latencies. The communication between network applications and the Controller is also affected by different latencies; hence, it generates different control decisions. The configuration of the flow tables in the Switches is also impacted by different latencies depending on the load on the Controller or the Switches [23].

The method of information exchange among several Controllers in an SDN is what is referred to as inter-Controller communication. In this concept, a particular Controller is connected to a set of Switches, and these Switches communicate with their connected Controller through information published between each other to have a global network view. And Controllers send information to their neighbours about their local state to generate a global network state view. Centralized and distributed architectures implement well established routing protocols such as immediate system-immediate system (IS-IS), an internet gateway protocol (IGP), OSPF and BGP to have a consistent view, although generating a global network view has issues of consistency, due to time taken between updates of Controller, and the simultaneous reading of the updates by multiple Controller which impacts the network performance [12] [24].

Scholars suggested that only latency related issues are not sufficient to determine effective Controller placement; and that resilience of the Controller should be a major consideration for the Controller placement [25]. The authors posited that Controller failures, and reassignment to backup Controller would impact the latencies of the reassigned Switches on the network. Network disruptions can affect the stability of the network, as well as the topology of the network, and in some cases, affects a greater part of the network. Load imbalance may occur due to inconsistency in the assignment of network load on the Controller from connected Switches, due to queuing delay on the Controllers. Inter-Controller latency that results from Controller to Controller communication affects the synchronization of the global network state. The experimentation used POCO for the evaluation of the strategy adopted; evaluation, visualization of the network Controller placement scenario were looked at and analysis was performed with different use cases, the result show that POCO could be used to effectively analyze different options of placement to find an effective placement for the Controller based on certain parameters with respect to number of Controllers, nodes and Controller outages, and failure-free scenarios. The authors suggested the possibility of several parameters as inputs, however did not experiment with the identified parameters [20].

Experts suggested and presented a Pareto optimal controller (POCO) Placement framework, with Pareto optimal placement with regards to distinct performance metrics, where an exhaustive evaluation of possible placement were performed [19] [26]. For small and medium size network POCO is feasible but completely unrealistic for a large size network or networks with dynamic properties changing with time. The authors extended POCO with faster time complexity though less accurate. The experimentation was performed on a set of network topologies from the Internet Topology Zoo. The metric was the quantization of error introduced by the heuristics mechanism, with the trade-off of time and accuracy, and the method was extensible to virtual functions in virtualized network environment. The metrics of communication latencies includes Switch to Controller, and Controller to Controller latencies, resilience against link and node failures and well as load balancing. The algorithm did not take advantage of network partition to improve efficiency and reduce load on the controller throughput [5].

Zhang *et al.* formulated an algorithm for multi-objective optimization controller placement (MOCP) problem [27]. The object of the formulation was to determine a metric for maximizing Controller load balancing, network reliability and minimizing Controller latencies between Controller and Switches; also deciding an optimal location for Controller placement within nodes under the control of the SDN Controllers. A mathematical model was formulated with objective function for the optimization. Adaptive bacterial foraging optimization (ABFO) algorithm was developed as a computational model to compute complexity based on the current network state. Evaluation results after simulations on real network topologies show positivity with potentials to optimize the objective function efficiently. The ABFO, though allows for multiple inputs, but did not optimize the placement based on the parameters inputted; however, an enhancement may be needed to take advantage of the resourcefulness of the algorithm.

## 3. Method Adopted in the Study

The study adopted explorative and comparative investigation techniques in evaluating the POCO and MOCO models of controller placement in software-defined networks.

### 3.1. Analysis of the Existing Systems

The existing systems are two systems with varied functionality: Pareto optimal combination (POCO) and multi-objective combination (MOCO).

#### 3.1.1. Pareto Optimal Combination

The POCO controller functionality finds the optimal placement as a single objective function using the heuristics to search the database of network features in order to have a placement. However, that functionality is based only on a single parameter (distances of the nodes) in the network, but not any other parameters that would ensure the continuous and reliable existence of the network. The component architecture of POCO as an existing system is shown in **Figure 1**. POCO Controller takes an initial parameter with the injection of the function of Pareto optimality finder; the system searches the network database and finds an appropriate location for the placement of the Controller.

### 3.1.2. Multi-Objective Combination

The multi-objective input function does take some network input parameters to improve the placement objectives. MOCO takes latency and network load as parameters, however, it does not consider network resiliency, scalability, energy awareness. These functions highlighted above are stakeholder functionalities that have the capability to render the network useless. Energy usage, resiliency, and scalability issues due to latencies occur in a case of switch or controller failure.

The process of finding optimality in MOCO also requires improvement, when compared with POCO that uses Pareto optimality concept in arriving at an optimal solution for a single objective. The component architecture of MOCO as an existing system is shown in **Figure 2**. The MOCO Controller accepts multiple input parameters, with multi-objective search optimization function as a tradeoff, which the Controller uses to search the network database and finds an appropriate placement. However, the output of the controller did not create avenues for scalability, energy awareness and resiliency which are vital components for network survivability.

## 4. Results and Discussion

The study evaluated the POCO and MOCO controller algorithms against three network topologies:

1) Goodnet network;
2) AARNet network;
3) SAVVIS network.

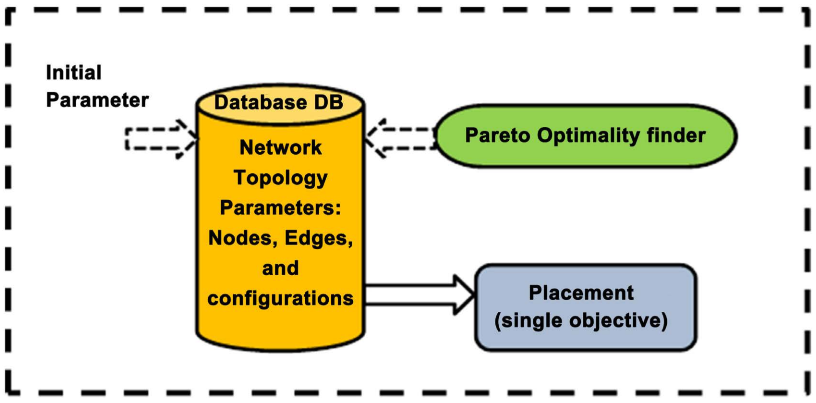
Initial Parameter
Database DB
Network Topology Parameters: Nodes, Edges, and configurations
Pareto Optimality finder
Placement (single objective)

Figure 1. Component architecture of the POCO controller.

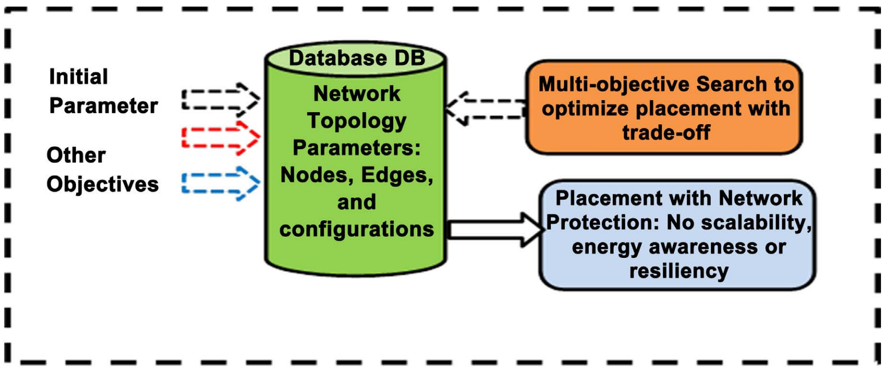
Initial Parameter
Other Objectives
Database DB
Network Topology Parameters: Nodes, Edges, and configurations
Multi-objective Search to optimize placement with trade-off
Placement with Network Protection: No scalability, energy awareness or resiliency

Figure 2. Component architecture of an existing system: MOCO controller.

The tabulated results in Table 1 show the performance of the models; POCO and MOCO. POCO-MOCO controller placement mechanisms for the Goodnet network are shown in Table 1, highlighting the visualization of the trends. The latencies of the algorithms were measured in microseconds against the positions and number of Controllers in the placements over the network.

The POCO-MOCO Controller executes individually in the four measurement scenarios, as the latencies improves with more controllers added within the network that reduced the latencies:

1) 3100 ms and 2500 ms for POCO and MOCO respectively in Switch to Controller Average Case latency;

2) 2598 ms and 2769 for POCO and MOCO respectively in Worst Case Switch to Controller latency;

3) 2776 ms and 2987 ms for POCO and MOCO respectively in Average Case Controller to Controller latency;

4) 2984 ms and 2759 for POCO and MOCO respectively in Worst Case Controller to Controller latency.

The result of the performance evaluation of the AARNet network is shown in Table 2, with the four metrics of measure of the algorithm on the network topology. The tabulated results in Table 2 show the performance of the algorithms; POCO and MOCO Controller placement mechanisms. Figure 3 shows the visualization of the trends. The latencies of the algorithms were measured in microseconds against the positions and number of Controllers in the placements over the network.

The POCO-MOCO Controller executes individually in the four measurement scenarios, as the latencies improves with more controllers added within the network that reduced the latencies:

1) 2473 ms and 2129 ms for POCO and MOCO respectively in Switch to Controller Average Case latency;

2) 2198 ms and 2268 ma for POCO and MOCO respectively in Worst Case Switch to Controller latency;

3) 2598 ms and 2471 ms for POCO and MOCO respectively in Average Case Controller to Controller latency;

Table 1. Test results of communication latencies for GoodNet.

| Network Name | GoodNet 2011 (USA) | |
|---|---|---|
| Network Nodes (N, E) | (12, 16) | |
| Parameters | POCO | MOCO |
| Average Case Latency: Switch to Controller | 3100 | 2500 |
| Worst Case Latency: Switch to Controller | 2598 | 2769 |
| Average Case Latency: Controller to Controller | 2776 | 2987 |
| Worst Case Latency: Controller to Controller | 2984 | 2759 |

Table 2. Test results of communication latencies for AARNet.

| Network Name | AARNet 2010 (Australia) | | |
|---|---|---|---|
| Network Nodes (N, E) | (14, 22) | | |
| Parameters | POCO | MOCO | HYBRID |
| Average Case Latency: Switch to Controller | 2473 | 2129 | 2078 |
| Worst Case Latency: Switch to Controller | 2198 | 2268 | 2124 |
| Average Case Latency: Controller to Controller | 2598 | 2471 | 2299 |
| Worst Case Latency: Controller to Controller | 2689 | 2814 | 2365 |

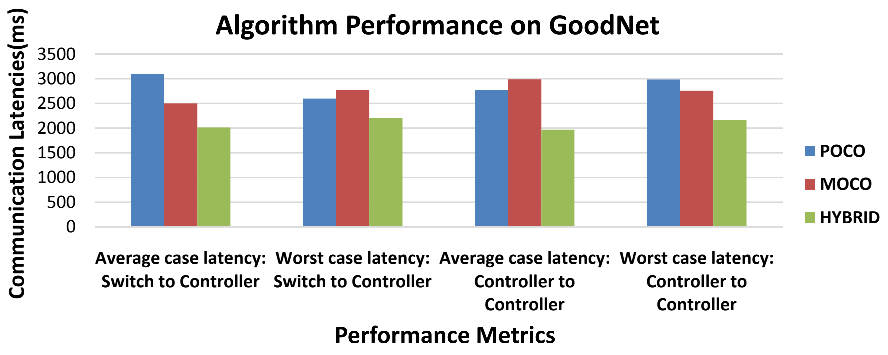

Figure 3. Trend analysis of communication latencies for GoodNet.

4) 2689 ms and 2814 for POCO and MOCO respectively in Worst Case Controller to Controller latency.

Table 2 shows the worst-case and average-case latencies of switch to controller, and worst-case and average-case latencies of controller to controller.

The result of the performance evaluation of the Savvis network is shown in Table 3, with the four metrics of measure of the algorithm on the network topology. The tabulated result in Table 3 shows the performance of the algorithms; POCO and MOCO Controller placement mechanisms. Figure 4 shows the visualization of the trends. The latencies of the algorithms were measured in microseconds against the positions and number of Controllers in the placements over the network.

The POCO and MOCO performed individually in the four measurement scenarios as follows:

Table 3. Test results of communication latencies for Savvis.

| Network Name | SAVVIS 2011 (USA) | | |
|---|---|---|---|
| Network Nodes (N, E) | (10, 14) | | |
| Parameters | POCO | MOCO | HYBRID |
| Average Case Latency: Switch to Controller | 2912 | 2784 | 2539 |
| Worst Case Latency: Switch to Controller | 3129 | 3017 | 2773 |
| Average Case Latency: Controller to Controller | 2789 | 2693 | 2598 |
| Worst Case Latency: Controller to Controller | 2873 | 2756 | 2614 |

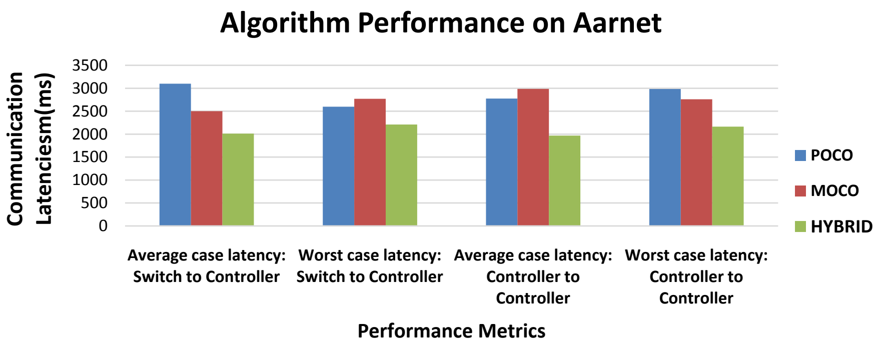

Figure 4. Trend analysis of communication latencies for AARNet.

1) 2912 ms and 2784 ms for POCO and MOCO respectively in Switch to Controller Average Case latency;

2) 3129 ms and 3017 for POCO and MOCO respectively in Worst Case Switch to Controller latency;

3) 2789 ms and 2693 ms for POCO and MOCO respectively in Average Case Controller to Controller latency;

4) 2873 ms and 2756 for POCO and MOCO respectively in Worst Case Controller to Controller latency.

## 5. Conclusions

The ultimate task of this study was to evaluate and compare the functionalities of the POCO and the MOCO models provided in controller placement in a software-defined networking. The localization of network Controller(s) within a defined topology that efficiently manages the network state and the global view of the network topology with reduced latency. The study is on software-defined networking (SDN) in general with a particular focus on the Controller placement. The study evaluated the POCO and MOCO algorithms that have the capability to address the issues of Controller placement. Although research findings show that algorithm design is greatly influenced by network features, architectural parameters, and topologies. This study found that other factors play significant roles in the proper placement of the Controllers, with the understanding that, continuous availability of the network requires: resiliency of the network, scalability of the network, and energy availability to keep the network alive, apart from the placement of the Controllers in the appropriate locations to control the data plane devices that focuses mostly on the packet forwarding.

The study identified specific influences and implementation challenges that affect the network sustainability in terms of the locational problems associated with Controller placement, while not overlooking the impact of architectural influences on the model of networks, which is expected to respond to dynamic changes of the network based on users and network environmental requirements.

When evaluated in relation to the Savvis network, POCO has a higher Average Case Switch to Controller latency (2912 ms) as against MOCO's (2784 ms). Similarly, POCO has a higher Worst-Case Switch to Controller latency (2789 ms) as against MOCO's (2693 ms). Also, POCO has a higher Average Case Controller to Controller latency (2789 ms) as against MOCO's (2693 ms). In the same vein, POCO has a higher Average Case Controller to Controller latency (2873 ms) as against MOCO's (2756 ms). The Average Case and Worst-Case latencies for Switch to Controller and Controller to Controller are minimal, and favourable to the POCO model as against the MOCO model when evaluated in the Goodnet, Savvis, and Aanet networks. This simply indicates that the POCO model has a speed advantage as against the MOCO model, which appears to be only more resilient in terms of storage than the POCO model.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

[1] Salvendy, G. and Wei, J. (2022) Design, Operation and Evaluation of Mobile Communications. Springer, Cham. https://doi.org/10.1007/978-3-031-05014-5

[2] Braun, W. and Menth, M. (2014) Software-Defined Networking Using Open Flow: Protocols, Applications and Architectural Design Choices. *Future Internet*, **6**, 302-336. https://doi.org/10.3390/fi6020302

[3] Long, Q., Chen, Y., Zhang, H. and Lei, X. (2019) Software Defined 5G and 6G Networks: A Survey. *Mobile Networks and Applications*, **27**, 1792-1812. https://doi.org/10.1007/s11036-019-01397-2

[4] Foukas, X., Marina, M.K. and Kontovasilis, K. (2014) Software Defined Networking Concepts. In: Liyanage, M., Gurtov, A. and Ylianttila, M., Eds., *Software Defined Mobile Networks* (*SDMN*): *Beyond LTE Network Architecture*, John Wiley & Sons, Ltd., Hoboken. https://doi.org/10.1002/9781118900253.ch3

[5] Open Network Foundation (2014) OpenFlow Switch Specification Version 1.5.0 (Protocol Version 0x06). https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.noipr.pdf

[6] Kreutz, D., Ramos, F.M.V., Verissimo, P., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S. (2014) Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, **103**, 14-76. https://doi.org/10.1109/JPROC.2014.2371999

[7] Stein, Y. and Haleplidis, E. (2015) SDN & NFV OpenFlow and ForCES: IETF-93. https://www.ietf.org/proceedings/93/slides/slides-93-edu-openflow-9.pdf

[8] Jammal, M., Singh, T., Shami, A., Asal, R. and Li, Y. (2014) Software Defined Networking: State of the Art and Research Challenges. *Computer Networks*, **72**, 74-98. https://doi.org/10.1016/j.comnet.2014.07.004

[9] Duan, Q., Ansari, N. and Toy, M. (2016) Software-Defined Network Virtualization—An Architectural Framework for Integrating SDN and NFV for Service Provisioning in Future Networks. *IEEE Network*, **30**, 10-16. https://doi.org/10.1109/MNET.2016.7579021

[10] Stallings, W., Agboma, F. and Jelassi, S. (2016) Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud. Pearson Education, Indianapolis.

[11] Dimogerontakis, E., Vilata, I. and Navarro, L. (2013) Software Defined Networking for Community Network Testbeds. 2013 *IEEE* 9*th International Conference on Wireless and Mobile Computing, Networking and Communications* (*WiMob*), Lyon, 7-9 October 2013, 111-118. https://doi.org/10.1109/WiMOB.2013.6673348

[12] Killi, B.P.R. and Rao, S.V. (2016) Capacitated Next Controller Placement in Software Defined Networks. *IEEE Transactions on Network and Service Management*, **14**, 514-527. https://doi.org/10.1109/TNSM.2017.2720699

[13] Lee, M. (2012) Introduction to Software-Defined Networking. UG3 Computer Communications & Networks (COMN). https://www.inf.ed.ac.uk/teaching/courses/comn/lecture-notes/lec18.pdf

[14] Duan, Q., Wang, Y., Bernstein, A. and Toy, M. (2017) Virtualization in Networking in Virtualized Software-Defined Networks and Services. Artech House, Boston.

[15] Amazonas, J.R.A., Santos-Boada, G. and Solé-Pareta, J. (2014) A Critical Review of OpenFlow/SDN-b. 16*th International Conference on Transparent Optical Networks* (*ICTON*)*,* Graz, 6-10 July 2014, 1-5.

[16] Rawal, A. (2021) The Supercloud Platform for API-First Applications. https://www.section.io/engineering-education/openflow-sdn/

[17] Allied Telesis (2021) OpenFlow™ Protocol Feature Overview and Configuration Guide. https://www.alliedtelesis.com/ng/en/documents/openflow-feature-overview-and-configuration-guide

[18] Hu, Y.N., Wang, W.D., Gong, X.Y., Que, X.R. and Cheng, S.D. (2012) On the Placement of Controllers in Software-Defined Networks. *The Journal of China Universities of Posts and Telecommunications*, **19**, 92-97, 171. https://www.sciencedirect.com/science/journal/10058885

[19] Ran, L., Taiyi, F., Yunfeng,G., Cong,Y.L., Yang,H., & Huilong, D. (2015) The Research of OpenFlow Management and Control Interface Protocols Based on SDN Technology. 2015 *IEEE International Conference on Computer and Communications* (*ICCC*), Chengdu, 10-11 October 2015, 45-49. https://doi.org/10.1109/CompComm.2015.7387538

[20] Li, T., Gu, Z., Lin, X., Li, S. and Tan, Q. (2018) Approximation Algorithms for Controller Placement Problems in Software Defined Networks. 2018 *IEEE Third International Conference on Data Science in Cyberspace*, Guangzhou, 18-21 June 2018, 250-257. https://doi.org/10.1109/DSC.2018.00043

[21] Borcoci, E. (2013) Software Defined Networking and Architectures. *NetWare* 2013 *Conference*, August 25 2013, Barcelona. https://www.iaria.org/conferences2013/filesAFIN13/NetWare%202013-SDN%20and%20Architectures%20v1.2-%20August%2025,%202013.pdf

[22] Mousa, M., Bahaa-Eldin, A.M. and Sobh, M. (2016) Software Defined Networking Concepts and Challenges. 2016 11*th International Conference on Computer Engineering & Systems* (*ICCES*), Cairo, 20-21 December 2016, 79-90. https://doi.org/10.1109/ICCES.2016.7821979

[23] Su, J., Wang, W. and Liu, C. (2019) A Survey of Control Consistency in Software-Defined Networking. *CCF Transactions on Networking*, **2**, 137-152.

[24] Das, S., Parulkar, G. and McKeown, N. (2012) Why OpenFlow/SDN Can Succeed Where GMPLS Failed. *European Conference and Exhibition on Optical Communication*, Amsterdam, 16-20 September 2012. https://doi.org/10.1364/ECEOC.2012.Tu.1.D.1

[25] Hock, D., Gebert, S., Hartmann, M., Zinner, T. and Tran-Gia, P. (2014) POCO-Framework for Pareto-Optimal Resilient Controller Placement in SDN-Based Core Networks. 2014 *IEEE Network Operations and Management Symposium* (*NOMS*), Krakow, 5-9 May 2014, 1-2. https://doi.org/10.1109/NOMS.2014.6838275

[26] Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., Hocky, D., Jarschelz, M. and Hoffmann, M. (2015) Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks. *IEEE Transactions on Network and Science Management*, **12**, 4-17. https://doi.org/10.1109/TNSM.2015.2402432

[27] Zhang, B., Wang, X., Ma, L. and Huang, M. (2016) Optimal Controller Placement Problem in Internet-Oriented Software Defined Network. 2016 *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Chengdu, 13-15 October 2016, 481-488. https://doi.org/10.1109/CyberC.2016.98