



## Article

# Minimizing the Density of Switch–Controller Latencies over Total Latency for Software-Defined Networks

Andres Viveros <sup>1</sup>, Pablo Adasme <sup>1,\*</sup>, Ali Dehghan Firoozabadi <sup>2,\*</sup> and Enrique San Juan <sup>1</sup>

<sup>1</sup> Department of Electrical Engineering, Universidad de Santiago de Chile, Avenida Víctor Jara N° 3519, Santiago 9170124, Chile; andres.viverosll@usach.cl (A.V.); enrique.sanjuan@usach.cl (E.S.J.)

<sup>2</sup> Department of Electricity, Universidad Tecnológica Metropolitana, Av. Jose Pedro Alessandri 1242, Santiago 7800003, Chile

\* Correspondence: pablo.adasme@usach.cl (P.A.); adehghanfiroozabadi@utem.cl (A.D.F.)

**Abstract:** This study examines the problem of minimizing the amount and distribution of time delays or latencies experienced by data as they travel from one point to another within a software-defined network (SDN). For this purpose, a model is proposed that seeks to represent the minimization of the distances between network switches in proportion to the total nodes in a network. The highlights of this study are the proposal of two mixed-integer quadratic models from a fractional initial version. The first is obtained by transforming (from the original fractional model) the objective function into equivalent constraints. The second one is obtained by splitting each term of the fraction with an additional variable. The two developed models have a relationship between switches and controllers with quadratic terms. For this reason, an algorithm is proposed that can solve these problems in a shorter CPU time than the proposed models. In the development of this research work, we used real benchmarks and randomly generated networks, which were to be solved by all the proposed models. In addition, a few additional random networks that are larger in size were considered to better evaluate the performance of the proposed algorithm. All these instances are evaluated for different density scenarios. More precisely, we impose a constraint on the number of controllers for each network. All tests were performed using our models and the computational power of the Gurobi solver to find the optimal solutions for most of the instances. To the best of our knowledge, this work represents a novel mathematical representation of the latency density management problem in an SDN to measure the efficiency of the network. A detailed analysis of the test results shows that the effectiveness of the proposed models is closely related to the size of the studied networks. Furthermore, it can be noticed that the performance of the second model compared to the first one presents better behavior in terms of CPU times, the optimal solutions obtained, and the reduced Mipgaps obtained using the solver. These findings provide a deep understanding of how the models operate and how the optimization dynamics contribute to improving the efficiency and performance of SDNs.

**Keywords:** latency density; mixed-integer programming; software-defined networking; models and algorithms approaches



**Citation:** Viveros, A.; Adasme, P.; Dehghan Firoozabadi, A.; San Juan, E. Minimizing the Density of Switch–Controller Latencies over Total Latency for Software-Defined Networks. *Algorithms* **2024**, *17*, 393. <https://doi.org/10.3390/a17090393>

Academic Editor: Roberto Montemanni

Received: 25 July 2024

Revised: 19 August 2024

Accepted: 2 September 2024

Published: 5 September 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

In the current context, wireless networks face complex challenges due to the high expectations of users and the diversity of their needs [1]. This represents a significant obstacle for network administrators who must ensure an optimal service that meets these demands. In this scenario, SDNs stand out as a smart solution. These architectures separate the control and data transmission functions within the network, similar to a team with specific roles for decision-making and task execution. This structure enhances system flexibility, allowing for agile adjustments and immediate adaptations to changes in the environment. Essentially, SDNs provide an adaptable framework that evolves effectively

to address diverse circumstances. Within this environment, latency density is an important factor in ensuring the minimum latency levels required by specific users, allowing service operators to guarantee the viability of technologies such as telemedicine and autonomous driving. In the case of telemedicine, where communication between doctors and patients depends on the instantaneous transmission of critical medical data, a low latency density ensures that consultations are carried out smoothly and effectively, without delays that could compromise medical care [2]. On the other hand, in the field of autonomous driving, low latency density is critical for instantaneous communication between vehicles and centralized control systems. This enables autonomous vehicles to make fast and accurate decisions based on real-time data, improving road safety and transportation efficiency. In short, optimized latency density provides service operators with the confidence that advanced technologies such as telemedicine and autonomous driving can be deployed effectively and safely, meeting the demanding response time requirements for these critical applications [3]. The following is a summary of related work supporting these approaches.

One known problem in software-defined networks is referred to as the controller placement problem. This problem involves figuring out the best locations for controllers to ensure the network runs smoothly, scales well, and stays reliable. The goal is to reduce delays between controllers and switches, balance the workload, and ensure that the network remains available even if some controllers fail. It also needs to consider costs and network policies. Solving this problem involves using mathematical models, simulations, and optimization techniques to find the best controller locations for the network's needs. This study introduces two new models for solving a variant of the aforementioned SDN problem, starting from an initial complex model. The first derived model is obtained by changing the original problem objective into constraints. The second one is stated by breaking down each part of the complex problem using extra variables, leading to a bi-linear nonconvex problem. Both models involve quadratic terms related to switches and controllers, and they are NP-Hard problems that are difficult to solve optimally. In order to solve them faster, this study proposes new models and a novel meta-heuristic algorithm. The research uses both real and randomly generated networks to test the models. It also uses larger random networks to better assess the performance of the algorithm. The models are tested under different conditions, particularly by setting limits on the number of controllers in each network. All tests are conducted using the Gurobi solver [4] to find the optimal solutions. This work constitutes a new approach to the management of latency in SDNs, particularly when measuring network efficiency. The results show that the model's effectiveness depends on the size of the networks being studied. The second model performs better than the first one, with faster solution times and smaller gaps between the best solutions obtained and the lower bounds given by the Gurobi solver. These findings help us understand how the models contribute to improving SDN performance.

This study is divided into several sections to help readers understand the research thoroughly. Section 2 examines the previous studies and approaches relevant to the topic. Section 3 provides a feasible solution and the mathematical formulations used to address the problem. Section 4 explores the algorithmic approach used to solve these models effectively, discussing the methodologies and strategies applied. In Section 5, detailed and substantial computational experiments are presented to validate the models and algorithms through the specific scenarios tested, and the numerical results are reported. By focusing on analyzing and discussing the outcomes of these experiments, we highlight the key findings and insights gained from the study. Finally, Section 6 summarizes the conclusions drawn from the research findings, outlining their implications for the field. This section also suggests potential areas for future research and development based on the study outcomes and identified limitations.

## 2. Related Work

Some published works related to SDN networks, latency management, and other quality of service (QoS) parameters, which are closely related to this study, can be described

as follows. In [5], by using an artificial intelligence approach applied to software-defined networking architecture, challenges in underwater things (IoUT), such as load balancing and QoS, are addressed. The study aims to enhance network performance, flexibility, and scalability, using a balancing strategy multi-controller (CASM). Additionally, an adaptive routing protocol (SQAR) using reinforcement learning is proposed to select paths for different services while optimizing network QoS. The results show that the balancing strategy improves migration and achieves efficient load-balancing, benefiting SQAR directly. SQAR has highlights in terms of energy efficiency, convergence, and QoS when compared to other established protocols. Overall, the proposed framework achieves significant and effective rates in both QoS and load balancing. In [6], a methodology is introduced to enable the dynamic creation and adjustment of network segments to meet the varying needs of different user applications. This approach focuses on efficiently allocating resources, such as bandwidth and energy, ensuring optimal performance and adequate QoS in future 5G networks. In [7], two quadratic programming models and their linear equivalent counterparts are proposed. These models address the problem of user assignment in a software-defined network while aiming to minimize the connectivity costs and connection latencies between controllers and switches. The results show that the linear models have shorter processing times than the quadratic models for obtaining optimal solutions. The study compares user connection strategies, and the findings suggest that users should connect to switches. In [8], by using reinforcement learning with constraints, a model is proposed to optimize resource user segmentation while also meeting the requirements of existing users and applications in the network. In [9], methods such as Controller adaption and migration decision (CAMD) and dynamic and adaptive load-balancing (DALB) are used to address load imbalance in an SDN network. The results show that CAMD is the most efficient method. This algorithm utilizes the inherent variance in controller load status to identify under-utilized controllers, optimizing the migration process to balance network load. The performance metrics considered include response time, migration space, controller throughput, and packet loss. CAMD outperforms controller adaption and migration decisions, with average improvements of 7.4% in controller throughput, 6.4% in packet loss, 5.6% in migration space, and 5.7% in response time. In [10], the authors analyze an overview of the evolution of SDN controllers. Here, the authors develop advancements in architecture, protocol, and key features. In [11,12], the problem of latency minimization in an SDN is addressed. The first article focuses on the controller placement problem in a software-defined network to minimize the maximum latency for switches and controllers as well as communication between controllers. Additionally, it considers reducing the number of controllers needed for network operation, taking into account a fixed installation cost for each located controller. The second study proposes mathematical models to minimize latency in SDN networks. Both articles [11,12] aim to improve performance and quality of service by reducing network latency and managing messages between each pair of nodes for the existing arcs, considering the controllers. In [13], an approach for the cost-effective and efficient placement of switches and controllers in a hybrid SDN is proposed. Elements of QoS and implementation costs are considered to determine the optimal placement, maximizing QoS, and minimizing infrastructure deployment costs. In [14], the growing energy consumption of communication technologies, such as 5G and beyond, is addressed. The existing proposals for energy savings and how network management technologies could lead to undesired energy consequences, such as load balancing or link saturation problems, are evaluated. The conflicting nature of energy-saving and load-balancing objectives makes simultaneous optimization a challenge. In order to tackle this task, an energy-efficient and load-balanced online flow routing framework based on software-defined networking is proposed, followed by an optimization problem to balance energy savings and load. A route-updating algorithm that is flexible to changes in flow scheduling is also proposed. The experimental results demonstrate the superiority of the proposed algorithm over benchmarks in energy savings, load balancing, and reducing link congestion risks.

In [15], the study examines how Boolean algebra concepts can be used to solve problems in operations research. This technique helps transform complex binary quadratic problems into simpler linear ones. In [16], the migration of a traditional corporate network to the SDN architecture is studied to demonstrate the advantages of an SDN network compared to a conventional network. For this purpose, a classic corporate network was simulated, and then the simulation was migrated to an SDN architecture. The network simulation was carried out using a virtual environment (MININET). By comparing the implementation process of these two networks, the simplicity of the SDN network implementation compared to the classic network was shown. In order to study the complexity of maintenance, two scenarios were used: the migration from one virtual local area network (VLAN) to two VLANs and the evolution of routing rules in both networks. In [17], in response to the slow but steady adoption of SDN architectures and the current solutions to manage them, this work proposes a new efficient SDN architecture based on a flat multi-controller multi-domain network that provides efficient load balancing in the network. Its performance was demonstrated and verified through simulations in a Mininet environment with Ryu controllers.

In [18], the authors introduce a solution designed to improve load balancing in SDNs with multiple controllers, thereby enhancing network performance. It uses a greedy approach to migrate switches from overloaded controllers to underloaded ones based on deviations in controller loads from the average. This ensures each controller has a balanced load with minimal switch migrations. The algorithm's effectiveness is demonstrated through the analysis of time and space complexities, along with simulations and experimental studies showing superior performance in terms of switch migrations, response time, throughput, and delay compared to traditional methods. In [19], as data flows, industrial applications grow exponentially, achieving low-latency and reliable data transmission simultaneously poses a challenge. Time-sensitive software-defined networking (TSSDN) has emerged as a technology for combining the real-time network configuration capabilities from SDN with deterministic flow delivery from time-sensitive networking (TSN), which is ideal for industrial internet of things (IIoT) applications. In order to address this challenge, a frame replication and elimination for reliability (FRER) mechanism is proposed for TSSDN-based IIoT systems. However, FRER introduces stress on already limited network resources by creating redundant paths. In order to tackle this concern, an end-to-end delay-bound model and reliability model are proposed for analysis. An optimization problem is formulated to maximize overall system utility while meeting commercial flow transmission requirements and hardware resource constraints. Numerical simulations validate the effectiveness and performance superiority of the proposed approach over existing methods. Reference [20] explores the management of users across various segments within an SDN, aiming to minimize the latency among switches, controllers, and user access nodes. It introduces two mathematical models and a heuristic algorithm. This study represents a pioneering effort in integrating network slicing into SDN, enhancing network management efficiency and security to accommodate services based on specific segment needs. The models and algorithm proposed contribute to significant advancements in handling user connections to controller or switch-type nodes based on their segment affiliation, ensuring robust and adaptive network deployment.

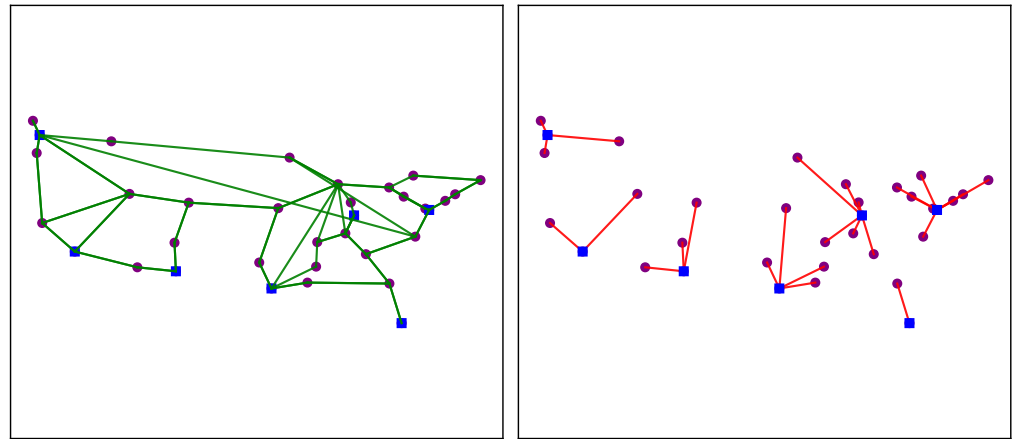
### 3. A Feasible Solution and the Mathematical Formulations

In this section, we start by introducing the optimization problem for network planning that we are investigating. We include an example of an input graph network along with its corresponding optimal solution. Then, we proceed to detail and explain each of the proposed models.

#### 3.1. A Feasible Solution to the Problem

Figure 1 displays an output of a feasible solution to the problem. Indeed, it is the optimal solution. It corresponds to the instance known as Internet2 OS3E. In the left figure,

the green lines indicate the arcs that can be used to connect network nodes. Switches are represented by purple circles, and network controllers are shown as blue squares. On the right side of Figure 1, the connections between switches and controllers are depicted with red lines. It is important to note that the entire network is connected, but for the sake of clarity, the communications between controllers in the backbone are not shown in the figure.



**Figure 1.** Representation of the Internet2 OS3E instance, which is divided into the input graph and separation between the switch and controller nodes on the left side of the figure. The graph of network Internet2 OS3E shows purple dots for the network switches, blue squares for the network controllers, and green lines for the installed network links. The right figure represents the connections between switches and controllers in the network solution. In particular, purple dots denote the network switches, the blue squares denote the network controllers, and the red lines, the connections between the switches and controllers, which are indirect and follow the shortest path according to the installed network.

### 3.2. Mathematical Formulations

We consider a network composed of switches and controllers (nodes), where these nodes are represented using a graph denoted by  $G = (V, A)$ , with  $V$  representing the set of nodes of the network and  $A$  the set of arcs in the network. The connection rules for the networks under study are as follows: switches can only connect to a single controller, and communications between controllers form a single network component that spans the entire graph. The networks are evaluated under different scenarios considering that the number of controllers varies according to the following percentages {20%, 30%, and 40%} of the total network nodes. In each scenario, the goal is to minimize the latency of connections between switches and controllers relative to the total active connections in the network.

In order to mathematically represent the aforementioned elements, the following parameters and variables are introduced.

#### 3.2.1. Parameters

- $D_{ij}$ : The shortest path distance between nodes  $i$  and  $j \in V$  in the network. Notice that matrix  $D = (D_{ij})$  is symmetric, and its diagonal has only zero values. Moreover, notice that these distances are computed using Dijkstra's algorithm [21].

#### 3.2.2. Variables

- $x_{ij}$ : This is a binary variable that is equal to one if node  $i \in V$  is assigned to controller  $j \in V$ ; otherwise,  $x_{ij}$  equals zero.
- $y_i$ : This is a binary variable that equals one if node  $i \in V$  acts as a controller node in the backbone SDN, and it equals zero otherwise.

### 3.3. Mathematical Formulations

The fractional mixed-integer nonlinear model can, thus, be written as follows

$$F_1 : \min_{\{x,y\}} \frac{\sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij}}{\sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij} + \sum_{\substack{i,j \in V \\ (i < j)}} D_{ij}y_i y_j} \tag{1}$$

$$\text{st : } \sum_{j \in V} x_{ij} + y_i = 1, \quad \forall i \in V, (i \neq j) \tag{2}$$

$$\sum_{j \in V} y_j = K \tag{3}$$

$$x_{ij} \leq y_j, \quad \forall i, j \in V, (i \neq j) \tag{4}$$

$$x_{ij} \leq 1 - y_i y_j, \quad \forall i, j \in V, (i \neq j) \tag{5}$$

$$x \in \{0, 1\}^{|V|^2}, y \in \{0, 1\}^{|V|} \tag{6}$$

It should be noted that we are referring to a fractional programming model (1) that is very hard to deal with. As such, we cannot solve it directly, but we can transform it to derive two quadratic models. The aim is to minimize the latency between the connections between switches and controllers concerning the latency of all connections in the network. In constraint (2), it is allowed that the connection of a node,  $i \in V$ , can only be assigned to a single node,  $j \in V$ , provided that  $i \neq j$ , with the only exception being when that node is a controller without connections. Constraint (3) ensures that the number of controllers equals  $K$ . Constraints (4) impose that the connections can only occur if the destination node of that link is a controller. In constraint (5), it is ensured that two controllers cannot communicate directly. Finally, constraint (6) denotes the domains of the decision variables.

Now, the transformation of the fractional model (1)–(6) into the intermediate model (7)–(9) can be derived by imposing an upper bound variable,  $t$ , in the constraint (8). Furthermore, the domain definition of the variable,  $t$ , is added in constraint (9).

$$\min_{\{x,y,t\}} t \tag{7}$$

$$\text{st : } \frac{\sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij}}{\sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij} + \sum_{\substack{i,j \in V \\ (i < j)}} D_{ij}y_i y_j} \leq t \tag{8}$$

$$\sum_{j \in V} x_{ij} + y_i = 1, \quad \forall i \in V, (i \neq j)$$

$$\sum_{j \in V} y_j = K$$

$$x_{ij} \leq y_j, \quad \forall i, j \in V, (i \neq j)$$

$$x \in \{0, 1\}^{|V|^2}, y \in \{0, 1\}^{|V|}, t \in [0, \infty) \tag{9}$$

The first equivalent model that we consider and evaluate by using the benchmark and random instances is presented below. Compared to the previous intermediate model, constraint (8) is replaced by constraints in (11) and (12). In addition to removing the fractional part of (8), the quadratic term  $y_i y_j$  is also replaced using variable  $z_{ij}$  for all  $i, j \in V, (i < j)$ . This is carried out to avoid having a cubic term in the third sum of constraint (11). Finally, the domain of the decision variable  $z$  is included in constraint (13).

$$M_1 : \min_{\{x,y,t,z\}} t \tag{10}$$

$$\text{st : } \sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij} - \sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}tx_{ij} - \sum_{\substack{i,j \in V \\ (i < j)}} D_{ij}tz_{ij} \leq 0 \tag{11}$$

$$\sum_{j \in V} x_{ij} + y_i = 1, \quad \forall i \in V, (i \neq j)$$

$$\sum_{j \in V} y_j = K$$

$$x_{ij} \leq y_j, \quad \forall i, j \in V, (i \neq j)$$

$$z_{ij} = y_i y_j, \quad \forall i, j \in V, (i < j) \tag{12}$$

$$x_{ij} \leq 1 - y_i y_j, \quad \forall i, j \in V, (i \neq j)$$

$$x \in \{0,1\}^{|V|^2}, y \in \{0,1\}^{|V|}, t \in [0, \infty), z \in \{0,1\}^{\frac{|V|(|V|-1)}{2}} \tag{13}$$

Another variant of the fractional model is also presented below, where constraint (11) is replaced by the constraints (15)–(17), and the domain definitions of variables  $z_1, z_2$  are added in constraint (22).

$$M_2 : \min_{\{x,y,t,z_1,z_2\}} t \tag{14}$$

$$\text{st : } z_1 = \sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij} \tag{15}$$

$$z_2 = \sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij}x_{ij} + \sum_{\substack{i,j \in V \\ (i < j)}} D_{ij}y_i y_j \tag{16}$$

$$tz_2 = z_1 \tag{17}$$

$$\sum_{j \in V} x_{ij} + y_i = 1, \quad \forall i \in V, (i \neq j) \tag{18}$$

$$\sum_{j \in V} y_j = K \tag{19}$$

$$x_{ij} \leq y_j, \quad \forall i, j \in V, (i \neq j) \tag{20}$$

$$x_{ij} \leq 1 - y_i y_j, \quad \forall i, j \in V, (i \neq j) \tag{21}$$

$$x \in \{0,1\}^{|V|^2}, y \in \{0,1\}^{|V|}, t, z_1, z_2 \in [0, \infty) \tag{22}$$

Finally, to analyze the relevance of model  $M_2$ , we weigh the latency of switch–controller connections versus the latencies of controller–controller links. For this purpose, the following variant is introduced, which modifies constraints (15) and (16) using parameter  $\alpha \in [0;1]$  to weigh the switch–controller communication latencies and  $(1 - \alpha)$  scales the latencies between controllers.

**Observation 1.** Note that when  $\alpha = 1$ , constraint (24) eliminates the consideration of connections between controllers. Similarly, when  $\alpha = 0$ , we eliminate the switch–controller connections in constraint (24).

$$M_3 : \min_{\{x,y,t,z_1,z_2\}} t$$

$$\text{st : } z_1 = \alpha \sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij} x_{ij} \tag{23}$$

$$z_2 = \alpha \sum_{\substack{i,j \in V \\ (i \neq j)}} D_{ij} x_{ij} + (1 - \alpha) \sum_{\substack{i,j \in V \\ (i < j)}} D_{ij} y_i y_j \tag{24}$$

$$tz_2 = z_1$$

$$\sum_{j \in V} x_{ij} + y_i = 1, \quad \forall i \in V, (i \neq j)$$

$$\sum_{j \in V} y_j = K$$

$$x_{ij} \leq y_j, \quad \forall i, j \in V, (i \neq j)$$

$$x_{ij} \leq 1 - y_i y_j, \quad \forall i, j \in V, (i \neq j)$$

$$x \in \{0, 1\}^{|V|^2}, y \in \{0, 1\}^{|V|}, t, z_1, z_2 \in [0, \infty)$$

Notice that all the proposed models are NP-hard and, thus, are difficult to solve in polynomial time. In the next section, we present a meta-heuristic algorithm that uses model ( $M_2$ ) to obtain approximate solutions in a short CPU time.

#### 4. Algorithmic Approaches

In this section, an algorithm is introduced that decomposes the models into two parts. An auxiliary linear programming model is initially solved to obtain a first solution,  $y_i$ , for all  $i \in V$ . For this purpose, we randomly generate an initial input vector,  $c_i$ , for all  $i \in V$  that acts as the coefficients for the objective function of the following auxiliary optimization problem.

$$M_{A1} : \min_{\{y\}} \sum_{i \in V} c_i y_i \tag{25}$$

$$\text{st : } \sum_{j \in V} y_j = K \tag{26}$$

$$y \in \{0, 1\}^{|V|} \tag{27}$$

Notice that solving  $M_{A1}$  is trivial since it is a linear programming problem. Additionally, notice that we impose a predefined number of controllers,  $K$ , in the output solution of the problem. Once we have obtained the output vector,  $y$ , we introduce it as an input parameter in the following optimization problem. This parameter is denoted by  $\bar{y}$ :

$$M_{A2} : \min_{\{x\}} \sum_{\substack{i \in V \\ (i \neq j)}} D_{ij} x_{ij} \tag{28}$$

$$\text{st : } \sum_{j \in V} x_{ij} + \bar{y}_i = 1, \quad \forall i \in V, (i \neq j) \tag{29}$$

$$x_{ij} \leq \bar{y}_j, \quad \forall i, j \in V, (i \neq j) \tag{30}$$

$$x_{ij} \leq 1 - \bar{y}_i \bar{y}_j, \quad \forall i, j \in V, (i \neq j) \tag{31}$$

$$y \in \{0, 1\}^{|V|} \tag{32}$$

Observe that model  $M_{A2}$ , which is linear now, is a decomposed version of model  $M_2$ . This model aims to minimize the latency of connections between switch–controller pairs based on the given set of controllers obtained using  $M_{A1}$ . By performing iterative swaps with the values of the input vector,  $c$ , to solve model  $M_{A1}$ , Algorithm 1 can be written:



**Algorithm 1:** Proposed Local Search Meta-heuristic algorithm.**Data:** An instance SDN input graph network. $Iter = 0, IterMax = MaxValue, MinGlobal = \infty$ **Result:** A feasible solution and its objective function value are obtained for the SDN network.Randomly generate the vector  $c$  that weights each controller.**while** ( $Iter \leq IterMax$ ) **do**    Randomly generate two different positions in vector  $c$  and denote them by  $pos1$  and  $pos2$ , respectively.     $aux = c[pos1]$      $c[pos1] = c[pos2]$      $c[pos2] = aux$     Solve the reduced model (25)–(27) to generate a feasible solution  $y$ .    Solve the reduced model (28)–(32) to obtain a feasible solution  $x$ .    Compute the objective function of  $F_1$  using  $y$  and  $x$ . Denote it by  $Obj_m$ .    **if** ( $MinGlobal > Obj_m$ ) **then**         $y_G = y$          $MinGlobal = Obj_m$          $x_G = x$          $C_G = c$     **else**         $c = C_G$      $Iter = Iter + 1$ 

Return the best feasible solution obtained and its objective function value.

The algorithm starts by initializing iteration counters and setting a maximum number of iterations. It randomly generates a vector “ $c$ ” that weights controllers in the network. In each iteration, two positions in “ $c$ ” are randomly selected and swapped. The algorithm then solves two models to produce feasible solutions: “ $y$ ” and “ $x$ ”. The objective function value, denoted as “ $Obj_m$ ”, is calculated. If “ $Obj_m$ ” is lower than the current global minimum, “ $MinGlobal$ ”, the best solutions and associated parameters are updated. If not, the vector “ $c$ ” is reverted to its previous best configuration. The process continues until the maximum number of iterations is reached, at which point the best solution is found, and its objective value is returned. Finally, it is mentioned that Algorithm 1 has polynomial solving complexity in the order of  $O(IterMax * r^3)$  to  $O(IterMax * r^3 \log(r))$  since we solve two linear programming subproblems [22].

## 5. Results and Discussion

In this section, we conduct a series of computational experiments to compare the performance of all the proposed models and algorithms. Initially, we evaluated 13 real instances of the benchmark networks obtained from (<https://www.topology-zoo.org/> (accessed on 4 June 2024)). They range from networks with  $n = 4$  to  $n = 87$  nodes. In total, we considered 31 connected networks, including randomly generated ones. Notice that the benchmark network from <https://www.topology-zoo.org/> (accessed on 4 June 2024)). has been used in references [23,24]. A random network is created with random co-ordinates within  $1 \text{ km}^2$ . A randomly generated network is a type of network or graph that is created using a process that includes some element of randomness. Instead of following a fixed pattern, the connections (or edges) between the points (or nodes) in the network are established according to certain probabilities. For example, in a simple, randomly generated network, each possible connection between two nodes might be added with a certain probability, like flipping a coin for each pair of nodes to decide whether they are connected or not. This randomness makes the structure of the network unpredictable and can result in a wide variety of different shapes and connection patterns. We consider only connected random networks. After generating the co-ordinates for each node, we connect the network either sparsely or fully. A sparse network means it

is connected, but not every pair of nodes is linked directly, whereas a fully connected network means every pair of nodes is connected directly. Random networks are useful in various scenarios, including disaster situations where the exact node positions are initially unknown. The parameters used in Algorithm 1 are as follows. *IterMax* and *MinGlobal* are initially set to the values of 50 and  $\infty$ . Next, the entries of vector “c” are drawn randomly from the interval (0;1). We implement a Python program using the Gurobi Solver version 11.0.1. Gurobi solver is a powerful optimization tool widely used in mathematical optimization. It is capable of solving a broad range of problems, including linear programming, mixed-integer programming, quadratic programming, quadratic constraints programming, and mixed-integer quadratic programming. Recognized for its efficiency and speed, Gurobi employs advanced algorithms and optimization techniques to solve complex problems quickly and effectively [4]. Because it is designed to enhance modern computational architectures, it is well-suited for large-scale optimization tasks.

Substantial numerical experiments were conducted to compare the performance of the proposed models  $M_1$ ,  $M_2$ , and  $M_3$  and the proposed algorithm. Python code was implemented using the Gurobi solver for this purpose. The maximum CPU time allowed is limited to 1 h. Hence, if an objective value reported takes 3600 s or more, it means the solver is reporting the best solution obtained within 1 h. Otherwise, it corresponds to an optimal solution to the problem.

The experiments were conducted on a Mac with the following technical specifications: 8 GB of RAM, dual-core Intel Core i5 processor running at 1.8 GHz, and operating system Mac OS Monterey 12.7.5. The experiments involved 13 real benchmark networks with known latitude and longitude locations, where the distance matrix,  $D$ , corresponds to the Dijkstra distance between these locations. Additionally, 18 randomly generated networks were created as fully connected graphs. Each node and user co-ordinate was generated within a square area of 1 km<sup>2</sup> using a uniform distribution function. Consequently, each entry in the input distance matrix,  $D$ , was computed using these co-ordinates. The experiments were evaluated under three scenarios, with each obtained while varying the density of controllers in the network. The densities used were computed as 20%, 30%, and finally, 40% of the total nodes of the network using the ceiling function.

In Table 1, we describe each of the benchmark networks under study. More precisely, we displayed the name of each network in each column. The first, second, and third rows show the number of nodes, the number of edges, and the density of each network, respectively. The density is calculated by dividing the total number of edges in the network by the total amount of edges in the graph as if it were a fully connected graph. Finally, the fourth row indicates the number of users to be connected to the SDN network.

**Table 1.** Real benchmark networks (<https://www.topology-zoo.org/> accessed on 4 June 2024).

Network Name	$N_1$	$N_2$	$N_3$	$N_4$	$N_5$	$N_6$	$N_7$	$N_8$	$N_9$	$N_{10}$	$N_{11}$	$N_{12}$	$N_{13}$
Nodes	4	11	13	18	20	21	27	33	33	34	34	35	87
Edges	4	14	24	50	30	22	28	48	38	41	52	39	89
Density (%)	66.7	25.5	30.8	32.7	15.8	10.5	8	9.1	7.2	7.3	9.3	6.6	2.4
Users	24	66	78	108	120	126	162	198	198	204	204	210	522

The terms  $N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9, N_{10}, N_{11}, N_{12}$ , and  $N_{13}$  represent the following networks: Arpanet196912, Abilinet, Aarnet, Ans, HurricaneElectric, Atmnet, Bbnplanet, Bics, CrlNetworkServices, Internet2OS3E, Geant2009, NetworkUsa, and Vtl-Wavenet2008, respectively.

In Table 2, numerical results are reported for models  $M_1$  and  $M_2$  across the three proposed scenarios using real benchmark networks. The first column of the table lists the names of the studied networks. From the second to the seventh column, the following information is presented: the objective value of model  $M_1$ , the best bound, the MipGap, the processing time in seconds, the number of controllers (NC), and finally, the number of nodes from the branch and bound ( $B\&B$ ) method, respectively. Similarly, from the eighth

to the thirteenth column, the same information is listed as in columns two to seven, but for the  $M_2$  model.

**Table 2.** Numerical results for the real benchmark networks obtained with models  $M_1$  and  $M_2$  across the three density scenarios.

Network Name	$M_1$ with 20% Controllers Density						$M_2$ with 20% Controllers Density					
	Solution	Best Bound	MipGap	CPU	NC	B&B Nodes	Solution	Best Bound	MipGap	CPU	NC	B&B Nodes
Arpanet196912	1	1	0	0.03	1	1	1	1	0	0.01	1	1
Abiline	0.4	0.4	0	0.33	3	1	0.4	0.4	0	0.16	3	1
Aarnet	0.48	0.48	0	0.72	3	304	0.48	0.48	0	0.29	3	1
Ans	0.25	0.25	0	1.16	4	1	0.25	0.25	0	0.56	4	1
HurricaneElectric	0.18	0.18	0	1	4	1	0.18	0.18	0	0.21	4	1
Atmnet	0.25	0.25	0	3.93	5	544	0.25	0.25	0	1.68	5	131
Bbnplanet	0.16	0.16	0	5.24	6	61	0.16	0.16	0	8.88	6	1385
Bics	0.19	0.19	0	58.55	7	1904	0.19	0.19	0	16.36	7	1645
CrINetworkServices	0.11	0.11	0	12.99	7	1	0.11	0.11	0	13.98	7	1339
Internet2 OS3E	0.17	0.17	0	28.66	7	683	0.17	0.17	0	26.95	7	1025
Geant2009	0.21	0.21	0	25.13	7	585	0.21	0.21	0	14.52	7	1618
NetworkUsa	0.18	0.18	0	41.19	7	1075	0.18	0.18	0	56.45	7	1033
VtIWavenet2008	0.02	0	0.93	3600.08	18	1148	0.02	0	0.91	3600.01	18	2202
Network Name	$M_1$ with 30% controllers density						$M_2$ with 30% controllers density					
	Solution	Best bound	MipGap	CPU	NC	B&B nodes	Solution	Best bound	MipGap	CPU	NC	B&B nodes
Arpanet196912	0.27	0.27	0	0.01	2	1	0.27	0.27	0	0	2	1
Abiline	0.22	0.22	0	0.36	4	1	0.22	0.22	0	0.17	4	1
Aarnet	0.26	0.26	0	0.58	4	212	0.26	0.26	0	0.26	4	1
Ans	0.09	0.09	0	0.97	6	1	0.09	0.09	0	0.23	6	136
HurricaneElectric	0.04	0.04	0	0.99	6	1	0.04	0.04	0	0.16	6	1
Atmnet	0.1	0.1	0	6.2	7	782	0.1	0.1	0	2.71	7	2165
Bbnplanet	0.04	0.04	0	3.57	9	1	0.04	0.04	0	2.03	9	709
Bics	0.07	0.07	0	34.37	10	693	0.07	0.07	0	13.74	10	1601
CrINetworkServices	0.04	0.04	0	50.95	10	1385	0.04	0.04	0	31.23	10	8093
Internet2 OS3E	0.05	0.05	0	141.51	11	3363	0.05	0.05	0	27.48	11	1113
Geant2009	0.05	0.05	0	10.22	11	1	0.05	0.05	0	8.39	11	404
NetworkUsa	0.05	0.05	0	176.59	11	4353	0.05	0.05	0	19.99	11	1074
VtIWavenet2008	0.01	0	0.88	3600.03	27	804	0.01	0	0.79	3600.03	27	1302
Network Name	$M_1$ with 40% controllers density						$M_2$ with 40% controllers density					
	Solution	Best bound	MipGap	CPU	NC	B&B nodes	Solution	Best bound	MipGap	CPU	NC	B&B nodes
Arpanet196912	0.269	0.269	0	0.01	2	1	0.269	0.269	0	0.01	2	1
Abiline	0.108	0.108	0	0.43	5	157	0.108	0.108	0	0.09	5	70
Aarnet	0.061	0.061	0	0.32	6	1	0.061	0.061	0	0.07	6	13
Ans	0.036	0.036	0	1.1	8	1	0.036	0.036	0	0.19	8	57
HurricaneElectric	0.019	0.019	0	1.41	8	1	0.019	0.019	0	0.23	8	1
Atmnet	0.045	0.045	0	9.82	9	1522	0.045	0.045	0	1.98	9	222
Bbnplanet	0.018	0.018	0	4.36	11	1	0.018	0.018	0	2.82	11	361
Bics	0.024	0.024	0	12.88	14	1	0.024	0.024	0	5.3	14	330
CrINetworkServices	0.012	0.012	0	13.72	14	1	0.012	0.012	0	10.1	14	1806
Internet2 OS3E	0.026	0.026	0	410.15	14	20,685	0.026	0.026	0	35.43	14	1562
Geant2009	0.026	0.026	0	11.7	14	1	0.026	0.026	0	4.27	14	174
NetworkUsa	0.023	0.023	0	203.24	14	4016	0.023	0.023	0	18.57	14	1767
VtIWavenet2008	0.003	0.001	0.787	3600.02	35	715	0.003	0.001	0.661	3600.02	35	1092

First, we observe that the solutions obtained using models  $M_2$  and  $M_3$  decrease when the network node increases. It should also be noted that the data in the solution, best bound, MipGap, and CPU time fields are generally approximated to two decimal places, except for the scenario with a controller density of 40%, where, for the solution, best bound, and MipGap columns, it is necessary to have approximations to three decimal places. It can be noted that in almost all the studied networks, the optimal solution was found. Only in the case of the VtIWavenet2008 network was a close-to-optimal solution found. For this network, the solution time used exceeds the given maximum time, which is much higher than that used for solving the other networks under study. It should also be mentioned that the number of B&B nodes used to solve model  $M_1$  does not follow the trend of the network under study, and it is related to the CPU used. For example, in the scenario with a controller density of 20%, the networks CrINetworkServices and Bics have the same number of nodes. However, Bics has a larger number of edges than CrINetworkServices, and due to this difference, the resolution time and B&B nodes are significantly larger than those of the Bics network. From the information presented in Table 2, we can also mention that Model  $M_2$  is more efficient in most of the studied networks, except for the

Bbnplanet, CrlNetworkServices, and NetworkUsa networks in terms of CPU time for the 20% controller density scenario. Additionally, it can be observed that Model  $M_2$  is more efficient for all studied networks in scenarios with 20% and 30% controller densities. Finally, regarding the time used and changes in MipGap across all networks, we can conclude that the scenario reporting the highest complexity in its resolution is the 20% controller density scenario, followed by the 30% scenario, and finally, the 40% scenario. Similarly, from the review of models  $M_1$  and  $M_2$  on randomly generated networks, we can derive from the information shown in Table 3.

**Table 3.** Numerical results obtained for the random networks with models  $M_1$  and  $M_2$  across the three density scenarios.

Network Name	$M_1$ with 20% Controllers Density						$M_2$ with 20% Controllers Density					
	Solution	Best Bound	MipGap	CPU	NC	B&B Nodes	Solution	Best Bound	MipGap	CPU	NC	B&B Nodes
Random (20)	0.44	0.44	0	2.55	4	71	0.44	0.44	0	0.6	4	1
Random (25)	0.38	0.38	0	6.7	5	366	0.38	0.38	0	3.18	5	1
Random (30)	0.32	0.32	0	38.4	6	1879	0.32	0.32	0	3.74	6	1
Random (35)	0.29	0.29	0	112.89	7	1282	0.29	0.29	0	36.67	7	956
Random (40)	0.26	0.26	0	220.94	8	1030	0.26	0.26	0	191.11	8	1039
Random (45)	0.23	0.23	0	394.44	9	1114	0.23	0.23	0	61.52	9	888
Random (50)	0.12	0.12	0	609.12	10	1159	0.12	0.12	0	180.75	10	998
Random (55)	0.19	0.19	0	1205.55	11	2006	0.19	0.19	0	307.58	11	1597
Random (60)	0.17	0.17	0	1829.16	12	1288	0.17	0.17	0	301.42	12	1331
Random (65)	0.09	0.07	0.2	3600.03	13	3537	0.09	0.09	0	1384.99	13	2922
Random (70)	0.15	0.11	0.23	3600.02	14	2171	0.15	0.15	0	1233.56	14	3199
Random (80)	0.06	0.01	0.89	3600.05	16	2074	0.06	0.01	0.91	3600.01	16	3339
Random (100)	0.05	0	0.91	3600.02	20	890	0.05	0	0.91	3600.01	20	1313
Network Name	$M_1$ with 30% controllers density						$M_2$ with 30% controllers density					
	Solution	Best bound	MipGap	CPU	NC	B&B nodes	Solution	Best bound	MipGap	CPU	NC	B&B nodes
Random (20)	0.19	0.19	0	2.12	6	1	0.19	0.19	0	0.82	6	1
Random (25)	0.12	0.12	0	19.56	8	1436	0.12	0.12	0	8.06	8	4438
Random (30)	0.13	0.13	0	27.05	9	1006	0.13	0.13	0	19.95	9	1124
Random (35)	0.1	0.1	0	162.48	11	1979	0.1	0.1	0	14.88	11	991
Random (40)	0.1	0.1	0	222.32	12	1033	0.1	0.1	0	53.54	12	900
Random (45)	0.08	0.08	0	763.97	14	14,726	0.08	0.08	0	32.39	14	867
Random (50)	0.04	0.04	0	788.28	15	1224	0.04	0.04	0	68.32	15	926
Random (55)	0.06	0.06	0	686.62	17	1286	0.06	0.06	0	89.44	17	895
Random (60)	0.06	0.06	0	1581.11	18	1965	0.06	0.06	0	271.4	18	860
Random (65)	0.03	0.03	0	1686.76	20	943	0.03	0.03	0	234.77	20	827
Random (70)	0.05	0.05	0	1033.16	21	899	0.05	0.05	0	218.22	21	873
Random (80)	0.02	0	0.82	3600.04	24	984	0.02	0	0.8	3600.02	24	1347
Random (100)	0.02	0	0.89	3600.07	30	572	0.02	0	0.81	3600.04	30	983
Network Name	$M_1$ with 40% controllers density						$M_2$ with 40% controllers density					
	Solution	Best bound	MipGap	CPU	NC	B&B nodes	Solution	Best bound	MipGap	CPU	NC	B&B nodes
Random (20)	0.09	0.09	0	6.87	8	938	0.09	0.09	0	1.85	8	973
Random (25)	0.06	0.06	0	35.55	10	3800	0.06	0.06	0	6.35	10	3466
Random (30)	0.06	0.06	0	197.58	12	25,821	0.06	0.06	0	9.43	12	1209
Random (35)	0.05	0.05	0	108.88	14	1255	0.05	0.05	0	11.29	14	1012
Random (40)	0.04	0.04	0	212.14	16	1077	0.04	0.04	0	22.31	16	951
Random (45)	0.04	0.04	0	554.05	18	1432	0.04	0.04	0	31.16	18	896
Random (50)	0.02	0.02	0	825.09	20	1001	0.02	0.02	0	42.41	20	930
Random (55)	0.03	0.03	0	1482.45	22	966	0.03	0.03	0	68.38	22	906
Random (60)	0.03	0.03	0	1180.69	24	937	0.03	0.03	0	97.33	24	839
Random (65)	0.01	0.01	0	927.38	26	876	0.01	0.01	0	102.15	26	865
Random (70)	0.02	0.02	0	1308.33	28	891	0.02	0.02	0	162.19	28	848
Random (80)	0.01	0.01	0.16	3600.02	32	902	0.01	0.01	0	969.72	32	872
Random (100)	0.01	0	0.86	3600.12	40	3908	0.01	0.01	0	2218.93	40	827

For the scenario with a 20% controller density, it can be observed that for all networks, Model  $M_2$  is more efficient in terms of CPU processing time, except for the Randomic networks with 80 nodes and 100 nodes, where an optimal solution cannot be obtained, and the best near-optimal solution is found after 1 h of searching. In the case of the 80-node Randomic network, the MipGap obtained in  $M_1$  is better than in  $M_2$ , while in the 100-node Randomic network, the MipGap is approximately the same. It is noteworthy that for  $M_1$  in this scenario, optimal solutions are achieved from the 20-node network up to the 60-node network; this result is worse than that obtained using  $M_2$ , which also achieves optimal solutions for networks with 65 and 70 nodes. For the 30% controller density scenario, it can be observed that both models achieve optimal results, with the random networks

ranging from 20 nodes to 70 nodes, achieving near-optimal solutions for the 80-node and 100-node networks, respectively. It is also noticeable that model  $M_2$  is more efficient for all networks analyzed in this scenario, either in terms of CPU computing times or in the MipGaps obtained for networks where an optimal solution could not be found. In the 40% controller density scenario, it can be seen that model  $M_1$  obtains optimal solutions for networks from 20 nodes up to the network with 70 nodes, finding near-optimal solutions for the 80-node and 100-node networks, respectively. In contrast, model  $M_2$  achieved optimal solutions for all the studied networks. In this case, model  $M_2$  is more efficient in terms of CPU computing times for all studied networks. As it can be observed in Table 4, the model reporting better results is the one for the scenario using 40% controller density, the following study was conducted using  $M_3$  in that scenario.

**Table 4.** Numerical results obtained with  $M_3$  for real benchmark and random instances varying parameter  $\alpha$  for different controller densities.

Network Name	$M_3$ with $\alpha = 0.25$			$M_3$ with $\alpha = 0.50$			$M_3$ with $\alpha = 0.75$			$M_3$ with $\alpha = 1$		
	Solution	MipGap	CPU	Solution	MipGap	CPU	Solution	MipGap	CPU	Solution	MipGap	CPU
Arpanet196912	0.109	0	0.01	0.269	0	0.01	0.525	0	0	1	0	0.07
Abiline	0.039	0	0.12	0.108	0	0.12	0.266	0	0.12	1	0	0.19
Aarnet	0.021	0	0.07	0.061	0	0.07	0.164	0	0.09	1	0	0.25
Ans	0.012	0	0.21	0.036	0	0.32	0.102	0	0.27	1	0	0.53
HurricaneElectric	0.006	0	0.23	0.019	0	0.24	0.054	0	0.23	1	0	0.73
Atmnet	0.015	0	2.82	0.045	0	2.68	0.123	0	2.73	1	0	0.62
Bbnplanet	0.006	0	3.05	0.018	0	2.88	0.052	0	2.8	1	0	1.51
Bics	0.008	0	8.4	0.024	0	15.13	0.068	0	14.54	1	0	1.85
Cr1NetworkServices	0.004	0	25.81	0.012	0	20.91	0.036	0	14.77	1	0	2.73
Internet2 OS3E	0.009	0	32.84	0.026	0	44.54	0.074	0	36.61	1	0	2.03
Geant2009	0.009	0	3.22	0.026	0	5.22	0.074	0	4.49	1	0	1
NetworkUsa	0.008	0	33.85	0.023	0	17.41	0.065	0	15.24	1	0	1.87
VtlWavenet2008	0.001	0.65	3600.03	0.003	0.67	3600.02	0.009	0	3005.59	1	0	3.63
Network Name	$M_3$ with $\alpha = 0.25$			$M_3$ with $\alpha = 0.50$			$M_3$ with $\alpha = 0.75$			$M_3$ with $\alpha = 1$		
	Solution	MipGap	CPU	Solution	MipGap	CPU	Solution	MipGap	CPU	Solution	MipGap	CPU
Random (20)	0.032	0	1.45	0.09	0	2.29	0.229	0	2.61	1	0	0.89
Random (25)	0.023	0	9.7	0.065	0	8.18	0.172	0	2.27	1	0	0.85
Random (30)	0.021	0	12.31	0.06	0	4	0.162	0	24.32	1	0	1.94
Random (35)	0.018	0	19.17	0.051	0	20.84	0.138	0	9.52	1	0	1.41
Random (40)	0.015	0	32.93	0.044	0	26.51	0.121	0	27.84	1	0	1.98
Random (45)	0.013	0	33.06	0.037	0	39.65	0.103	0	33.44	1	0	2.14
Random (50)	0.006	0	79.48	0.016	0	43.74	0.048	0	42.42	1	0	1.81
Random (55)	0.009	0	80.57	0.027	0	67.87	0.078	0	61.76	1	0	2.25
Random (60)	0.009	0	111.52	0.026	0	98.15	0.074	0	111.69	1	0	2.39
Random (65)	0.004	0	177.58	0.012	0	111.89	0.034	0	111.68	1	0	5.72
Random (70)	0.007	0	180.3	0.021	0	159.06	0.059	0	143.39	1	0	2.08
Random (80)	0.003	0	1037.72	0.008	0	1195.13	0.025	0	666.69	1	0	5.48
Random (100)	0.002	0.63	3600.01	0.007	0.67	3600.02	0.021	0	2190.03	1	0	1.74

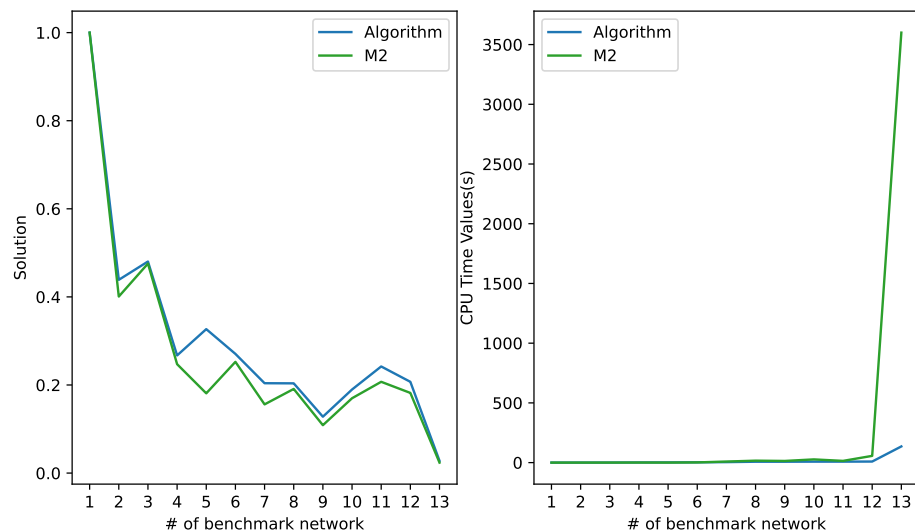
This evaluates the behavior of this model, which is similar to  $M_2$  but with weights on its components for switch–controller communication and controller–controller communication (note that Model  $M_2$  obtained the best results in the previous review).

Firstly, it is noted that for  $\alpha = 1$ , the complexity of the evaluated model is drastically lower because, in this case, the latency of communications between controllers is not considered in the problem under study. Therefore, no further observations or comparisons can be made against this scenario. The first analysis to be conducted is focused on the benchmark networks. In general, it can be observed that the objective solutions are higher for  $\alpha = 0.75$  compared to  $\alpha = 0.5$  and  $\alpha = 0.25$ , which is consistent because giving more weight to switch–controller latencies, which are more numerous than the connections between controllers, naturally leads to higher solution values.

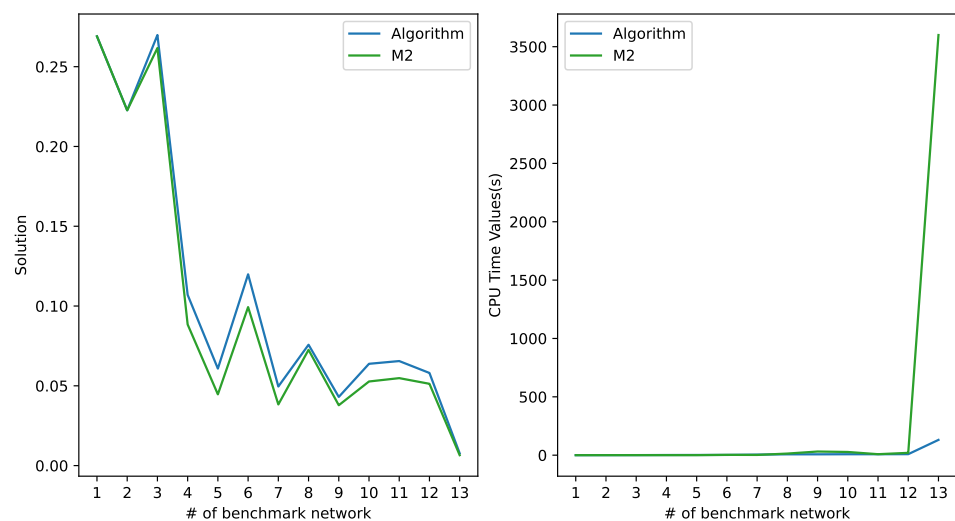
It is also noticeable that, in general, the scenario with  $\alpha = 0.5$  is worse than the scenario with  $\alpha = 0.25$  and  $\alpha = 0.75$ , either in terms of CPU processing times and MipGaps obtained. Furthermore, it can be appreciated that  $\alpha = 0.75$  achieves more efficient results in terms of processing times for larger networks compared to  $\alpha = 0.25$ . For  $\alpha = 0.75$ , optimal solutions are obtained for all benchmark networks studied, unlike  $\alpha = 0.25$  and  $\alpha = 0.5$ , where an optimal solution is not obtained for the VtlWavenet2008 network. Regarding the study

of random networks, the results are similar. It is noted that  $\alpha = 0.75$  achieves optimal solutions for all the studied networks, whereas  $\alpha = 0.25$  and  $\alpha = 0.5$  do not achieve optimal solutions; instead, the solutions are close to optimal for the 100-node network. Additionally, for  $\alpha = 0.25$ , less efficient results were obtained when compared to  $\alpha = 0.5$  for all the networks except for those with 20, 35, 45, 80, and 100 nodes. Overall,  $\alpha = 0.75$  yields better results in terms of CPU processing times.

Regarding this analysis, it can be determined that as the weighting of switch–controller communication relative to controller–controller communication increases, the objective values approach one. After the previous observations and to improve the performance of the methods used so far, the best model,  $M_2$ , was compared to the proposed algorithm. For the scenario with 20% controller density, it can be observed in Figure 2, on the left side, that the solution value found by the algorithm is not better than that of model  $M_2$ . However, it is still very close, with small differences in the majority of the studied networks. In contrast, the CPU times used for networks with fewer nodes are similar, whereas for the larger networks, they are drastically lower, as observed in network #13 (following the same order as the benchmark networks in Table 1). Similarly, the comparison results can be observed for the 30% controller density scenario. This can be observed in Figure 3.

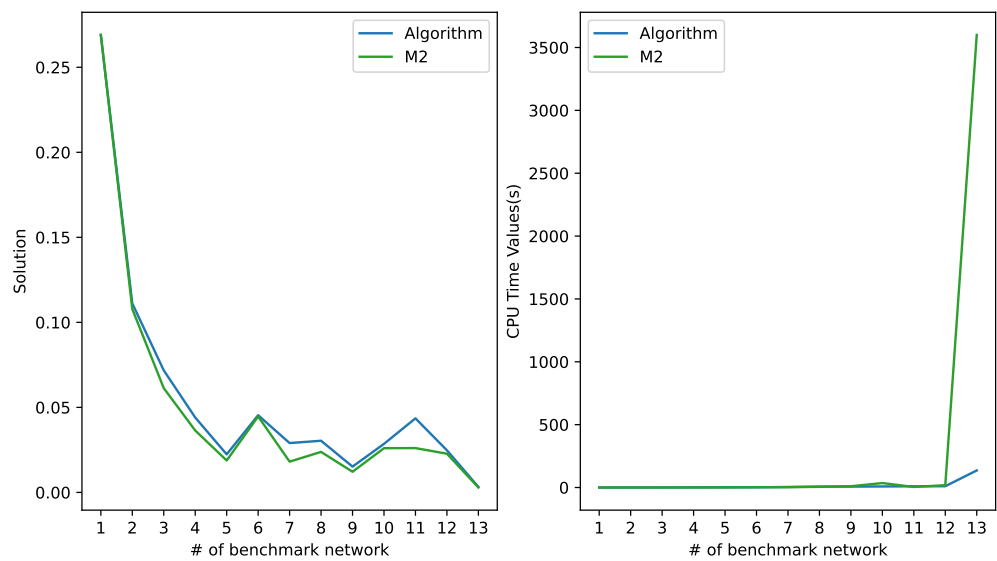


**Figure 2.** This figure represents both the objective function values on the left and the CPU times on the right side for the benchmark networks studied at a 20% controller density.

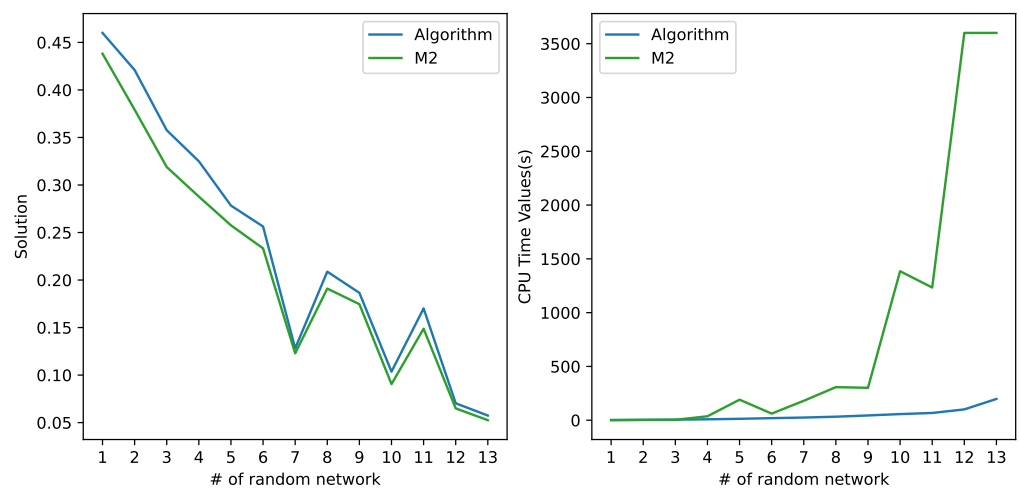


**Figure 3.** This figure represents both the objective function values on the left and the CPU times on the right side for the benchmark networks studied at a 30% controller density.

For this scenario, the solutions found by the algorithm align more closely with those obtained using model  $M_2$ , with considerably better times in the majority of the studied networks and noteworthy performance in larger networks, such as network #13. The trend of aligning with model  $M_2$  is highlighted more effectively in the 40% controller density scenario. In these cases, the proposed algorithm achieves solutions that are very close to those of model  $M_2$ , with significantly more efficient times. It performs best in the largest networks, where the solution found is very close to optimal and has drastically lower CPU times. Observe network #13 in Figure 4. The same previous analysis is conducted with the studied random networks. In this case, there are certain differences compared to the benchmark networks. While the algorithm found solutions that are very close to those found by model  $M_2$ , the CPU processing times are much lower than those required by  $M_2$  and are comparatively more efficient than in the case of the benchmark networks for the 20% density scenario, as shown in Figure 5.



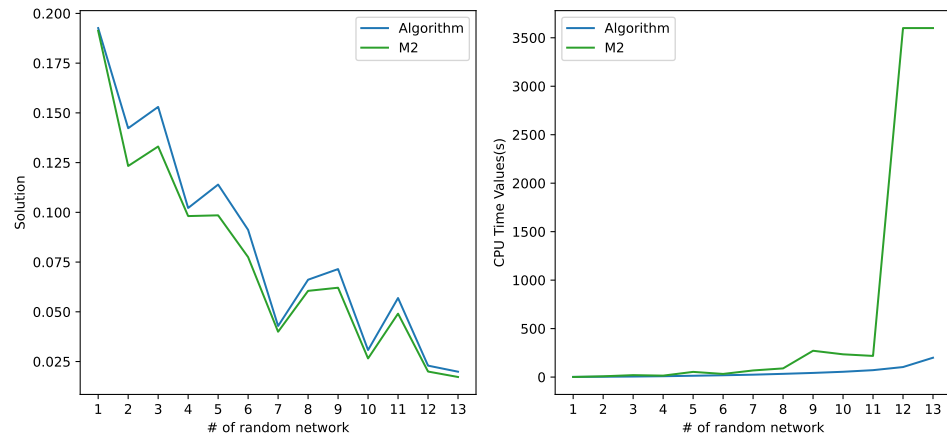
**Figure 4.** This figure represents both the objective function values on the left and the CPU times on the right side for the benchmark networks studied at a 40% controller density.



**Figure 5.** This figure represents both the objective function values on the left and the CPU times on the right for the random networks at a 20% network density.

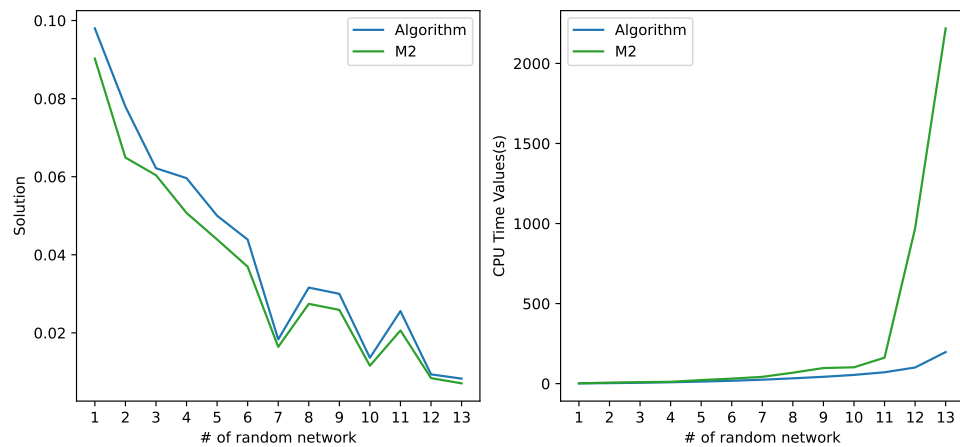
For the 30% controller density scenario, the comparison of solutions found is even closer. Note that in Figure 6, the scale of the Y-axis on the left is smaller, and although

the CPU times are better than  $M_2$ , compared to the previous scenario, the performance is slightly lower.



**Figure 6.** This figure represents both the objective function values on the left and the CPU times on the right for the random networks at a 30% network density.

Finally, in the 40% controller density scenario, the trend of closeness in the solutions found by the algorithm compared to  $M_2$  continues, and this closeness becomes stronger for larger networks. Similarly, the processing time used by the algorithm is much lower than that required by  $M_2$  for the majority of the studied networks, with the advantage being clearer for larger networks. From these analyses, including the trends presented in Figure 7, it is suggested that the proposed algorithm can be used for larger networks.



**Figure 7.** This figure represents both the objective solutions on the left and the CPU times on the right for the random networks at a 40% network density.

Based on this final suggestion, a last experiment was conducted with larger random networks, as can be seen in Table 5 below.

**Table 5.** Analysis of Algorithm 1 versus model  $M_2$  for larger random networks.

Network Name	20% Controllers Density				30% Controllers Density				40% Controllers Density			
	$M_2$		Algorithm		$M_2$		Algorithm		$M_2$		Algorithm	
	Solution	CPU	Solution	CPU	Solution	CPU	Solution	CPU	Solution	CPU	Solution	CPU
Random (150)	0.029	3600	0.044	24.1	0.009	3600.1	0.014	37.7	0.004	3600.1	0.006	23.1
Random (200)	0.019	3600.1	0.029	41.9	0.006	3600.1	0.009	50.8	0.002	3600.1	0.004	41
Random (300)	0.01	3600.1	0.016	95.3	0.003	3600.1	0.005	106.4	0.001	3600.3	0.002	93.4
Random (400)	0.007	3600.1	0.011	168.6	0.002	3600.2	0.003	171.3	0.001	3600.1	0.001	164.7
Random (500)	0.006	3600.2	0.008	263.8	0.002	3600.2	0.002	270.5	0.001	3600.3	0.001	261.2



From Table 5, it can be observed that the behaviors for all scenarios are similar to the above tables. In the scenario with a 40% controller density, the results show reduced CPU processing times. Additionally, upon analyzing each scenario and comparing the results obtained with  $M_2$  and the proposed algorithm, it can be noted that as the size of the network increases, the difference between the solution of  $M_2$  and the proposed algorithm decreases. Furthermore, there is a significant reduction in CPU processing time compared to model  $M_2$ , with an average between 1% and 7% of the CPU time required by model  $M_2$ . The results obtained from analyzing each scenario lead to the conclusion that the use of Algorithm 1 is recommended for larger networks again. This is because the solution found will increasingly approach the optimal solution found by model  $M_2$  and with shorter CPU times. This can be observed, for example, in the results of random networks with 400 and 500 nodes for the scenario using 40% controller density.

## 6. Conclusions

In this study, we focused on reducing latency density in software-defined networks, which is crucial for enhancing user experience and operational efficiency for network administrators. Lowering latency density in SDNs provides significant benefits for users, such as faster response times, improved connection reliability, and better resource utilization. For network decision-makers, this translates into more efficient traffic management and an enhanced ability to meet service-level agreements. In order to tackle this challenge, we introduced two novel mathematical models,  $M_1$  and  $M_2$ . These models are specifically designed to optimize resource distribution and management in SDNs, aiming to minimize latency and enhance overall network performance. Our evaluations consistently highlight that the  $M_2$  model outperforms  $M_1$  in terms of CPU times and MipGaps across all studied instances, demonstrating its effectiveness and robustness in various network configurations. In addition to theoretical modeling, we conducted comprehensive analyses using real benchmark instances, including randomly generated ones. This approach validated the practical effectiveness and applicability of our models. As a complement to our mathematical models, we developed a meta-heuristic localization algorithm to compare with the model  $M_2$ . The results indicate that this algorithm offers competitive solutions in terms of efficiency and accuracy and is particularly recommended for large-scale networks where resource optimization is critical. Throughout our study, we explored different controller density scenarios in networks, each representing unique conditions impacting network optimization and efficiency. Our findings show that increasing controller density tends to simplify the problem, as it enables more efficient traffic management, reducing latency and improving resource distribution. Furthermore, we introduced a sensitivity model similar to  $M_2$ , emphasizing that prioritizing connections between switches and controllers over controller-to-controller links significantly reduces CPU times. This strategic approach allows for more effective resource allocation and precise network optimization, which is particularly crucial in environments requiring scalability and efficiency.

These insights highlight the importance of considering controller density as a key factor in SDN design and management. By adjusting controller density according to specific network needs, we can simplify problem resolution and significantly enhance overall network performance and responsiveness. This adaptive and strategic approach is essential for addressing emerging challenges in modern network management and preparing networks for continued growth and increasing demands for digital services. In conclusion, this study not only advances the theoretical understanding of SDN optimization but also provides a robust framework for the practical implementation of solutions that enhance performance and efficiency in these critical digital networks. Our findings suggest that deploying the model  $M_2$  and the proposed algorithm represents a significant step toward more efficient and adaptive networks capable of meeting escalating connectivity and performance demands in enterprise and telecommunications settings.

In future work, we aim to explore the relationship between latency density and the distribution of network components, along with how this relates to service quality and

the costs associated with the deployment and maintenance of the network by service operators. It is important to address multiple issues simultaneously because there are different stakeholders with various challenges that require analysis and solutions. The goal is to find a balance that allows us to provide optimal service for users while ensuring operational feasibility conditions.

**Author Contributions:** Conceptualization, A.V., P.A., and A.D.F.; methodology, A.V., P.A., and A.D.F.; software, A.V. and P.A.; validation, A.V., P.A., and A.D.F.; formal analysis, A.V., P.A., A.D.F., and E.S.J.; investigation, A.V., P.A., A.D.F., and E.S.J.; resources, A.V., P.A., A.D.F., and E.S.J.; data curation, A.V., P.A., and A.D.F.; writing—original draft preparation, A.V. and P.A.; writing—review and editing, A.V., P.A., A.D.F., and E.S.J.; visualization, A.V., P.A., A.D.F., and E.S.J.; supervision, P.A., A.D.F., and E.S.J.; project administration, P.A. and A.D.F.; funding acquisition, P.A. and A.D.F. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors acknowledge the financial support from Projects Dicyt 062313AS, ANID/FONDECYT Iniciación No. 11230129, and the Competition for Research Regular Projects, year 2021, code LPR21-02; Universidad Tecnológica Metropolitana.

**Data Availability Statement:** The dataset is available upon request from the authors. The raw data supporting the conclusions of this article will be made available by the authors upon request.

**Acknowledgments:** The authors acknowledge the support of the Vicerrectoría de Investigación, Innovación y Creación (VRIIC) of the Universidad de Santiago de Chile.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

SDN	Software-defined network
QoS	Quality of service
IoUT	Internet of underwater things
CASM	Multi-controller balancing strategy
SQAR	Routing protocol adaptative
CAMD	Controller adaption and migration decision
DALB	Dynamic and adaptive load balancing
VLAN	virtual local area network
TSSDN	Time-sensitive software-defined networking
TSN	Time-sensitive networking
IIoT	Industrial internet of things applications
FRER	Frame replication and elimination for reliability
$G = (V, A)$	Graph with a set of nodes, $V$ , and a set of arcs, $A$
NC	Number of controllers

## References

- Ahmadi, A.; Sepehri, Z.; Gratuze, M.; Indja, M.; Jemmali, A.; Jevremovic, V.; Lamontagne, M.; Cloutier, S.; Iordanova, I.; Nerguizian, C.; et al. Wireless Network Deployment Survey. In Proceedings of the 2024 IEEE Radio and Wireless Symposium (RWS), San Antonio, TX, USA, 21–24 January 2024; pp. 143–146. [CrossRef]
- Adday, G.H.; Subramaniam, S.K.; Zukarnain, Z.A.; Samian, N. Investigating and Analyzing Simulation Tools of Wireless Sensor Networks: A Comprehensive Survey. *IEEE Access* **2024**, *12*, 22938–22977. [CrossRef]
- Goswami, B.; Kulkarni, M.; Paulose, J. A Survey on P4 Challenges in Software Defined Networks: P4 Programming. *IEEE Access* **2023**, *11*, 54373–54387. [CrossRef]
- Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual. 2023. Available online: <https://www.gurobi.com> (accessed on 5 July 2024).
- Shi, Y.; Yang, Q.; Huang, X.; Li, D.; Huang, X. An SDN-Enabled Framework for a Load-Balanced and QoS-Aware Internet of Underwater Things. *IEEE Internet Things J.* **2023**, *10*, 7824–7834. [CrossRef]
- Abdellatif, A.A.; Mohamed, A.; Erbad, A.; Guizani, M. Dynamic Network Slicing and Resource Allocation for 5G-and-Beyond Networks. In Proceedings of the 2022 IEEE Wireless Communications and Networking Conference (WCNC), Austin, TX, USA, 10–13 April 2022; pp. 262–267. [CrossRef]

7. Viveros, A.; Adasme, P.; Firoozabadi, A.D. Minimizing User Connectivity Costs and Latency Between Controllers and Switch-Controllers for Software Defined Networking. In *Mobile Web and Intelligent Information Systems, Proceedings of the MobiWIS, Marrakech, Morocco, 14–16 August 2023*; Lecture Notes in Computer Science; Younas, M., Awan, I., Grønli, T.M., Eds.; Springer: Cham, Switzerland, 2023; Volume 13977.7. [[CrossRef](#)]
8. Hlophe, M.C.; Maharaj, B.T. An SDN Controller-Based Network Slicing Scheme Using Constrained Reinforcement Learning. *IEEE Access* **2022**, *10*, 134848–134869. [[CrossRef](#)]
9. Adekoya, O.; Aneiba, A.; Patwary, M. An Improved Switch Migration Decision Algorithm for SDN Load Balancing. *IEEE Open J. Commun. Soc.* **2020**, *1*, 1602–1613. [[CrossRef](#)]
10. Prabha, C.; Goel, A.; Singh, J. A Survey on SDN Controller Evolution: A Brief Review. In Proceedings of the 2022 7th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 22–24 June 2022; pp. 569–575. [[CrossRef](#)]
11. Viveros, A.; Adasme, P.; Urrutia, E.S.J. Minimizing Latency and Number of Controllers in Software Defined Networking. In Proceedings of the 2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA), Curicó, Chile, 24–28 October 2022; pp. 1–6. [[CrossRef](#)]
12. Adasme, P.; Viveros, A.; Firoozabadi, A.D.; Soto, I. Mathematical Models for Minimizing Latency in Software-Defined Networks. In Proceedings of the MobiWIS, Rome, Italy, 22–24 August 2022; pp. 131–142.
13. Maity, I.; Misra, S.; Mandal, C. SCOPE: Cost-Efficient QoS-Aware Switch and Controller Placement in Hybrid SDN. *IEEE Syst. J.* **2022**, *16*, 4873–4880. [[CrossRef](#)]
14. Zhao, Y.; Wang, X.; He, Q.; Zhang, C.; Huang, M. PLOFR: An Online Flow Route Framework for Power Saving and Load Balance in SDN. *IEEE Syst. J.* **2021**, *15*, 526–537. [[CrossRef](#)]
15. Fortet, R. Applications de l'algèbre de boole en recherche opérationnelle. *Rev. Fr. Rech. Oper.* **1960**, *4*, 17–26.
16. Sotindjo, P.; Gbemavo, G.; Djogbe, L.; Goussi, G.; Agossou, C.M.M.; Vianou, A. Study of the Complexity of Implementation and Maintenance of an SDN Network. In Proceedings of the 2023 International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, South Africa, 16–17 November 2023; pp. 1–6.
17. Trúchly, P.; Kubica, J. Communication Networks with Multiple SDN Controllers. In Proceedings of the 2023 International Symposium ELMAR, Zadar, Croatia, 11–13 September 2023; pp. 29–32. [[CrossRef](#)]
18. Prajapati, U.; Chatterjee, B.C.; Banerjee, A. OptiGSM: Greedy-Based Load Balancing with Minimum Switch Migrations in Software-Defined Networks. *IEEE Trans. Netw. Serv. Manag.* **2024**, *21*, 2200–2210. [[CrossRef](#)]
19. Ji, L.; He, S.; Gu, C.; Shi, Z.; Chen, J. Routing and Scheduling for Low Latency and Reliability in Time-Sensitive Software-Defined IIoT. *IEEE Internet Things J.* **2024**, *11*, 12929–12940. [[CrossRef](#)]
20. Viveros, A.; Adasme, P.; Dehghan Firoozabadi, A. Optimal Topology Management for Software-Defined Networks Minimizing Latency and Using Network Slicing. *Complexity* **2024**. [[CrossRef](#)]
21. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Clifford, S. Section 24.3: Dijkstra's algorithm. In *Introduction to Algorithms*, 2nd ed.; MIT Press: Cambridge, MA, USA; McGraw-Hill: New York, NY, USA, 2001; pp. 595–601, ISBN 0-262-03293-7.
22. Nemhauser, G.L.; Wolsey, L.A. *Integer and Combinatorial Optimization*; Wiley Interscience Series in Discrete Mathematics and Optimization; Wiley: Hoboken, NJ, USA, 1988; pp. 1–763. ISBN 978-0-471-82819-8.
23. Yoo, H.S.; Yu, W.E.S. Building a QoS Testing Framework for Simulating Real-World Network Topologies in a Software-defined Networking Environment. In Proceedings of the 2022 International Conference on Engineering and Emerging Technologies (ICEET), Kuala Lumpur, Malaysia, 27–28 October 2022; pp. 1–6. [[CrossRef](#)]
24. Petale, S.; Thangaraj, J. Failure-Based Controller Placement in Software Defined Networks. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 503–516. [[CrossRef](#)]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.