Controller Placement Problem resiliency evaluation in SDN-based architectures

Maurizio D'Arienzo[1], Manfredi Napolitano[1], and Simon Pietro Romano[2]

[1] Dipartimento di Scienze Politiche
Università della Campania "L.Vanvitelli" - Italy
[2] DIETI
Università di Napoli "Federico II" - Italy

Abstract. The Software-Defined Networking (SDN) paradigm does represent an effective approach aimed at enhancing the performance of core networks by introducing a clean separation between the routing plane and the forwarding plane. However, the centralized architecture of SDN networks raises resiliency concerns that are addressed by a class of algorithms falling under the Controller Placement Problem (CPP) umbrella term. Such algorithms seek the optimal placement of the SDN controller. In this paper, we evaluate the main CPP algorithms and provide an experimental analysis of their performance, as well as of their capability to dynamically adapt to network malfunctions and disconnections.

1 Introduction

Computer networks have developed exponentially over the last twenty years, reaching important levels of branching across the globe. This has allowed great coverage and consequent interconnection among users, who can nowadays benefit from a plethora of services offered by the network, whose importance is increasing over time.

The network is currently composed of heterogeneous systems and services provided by leading IT companies. The increasing complexity of current networks raises a number of interoperability issues. The capability to modify the configuration of a network system through direct intervention is impractical except in rare cases, like closed and isolated environments. Due to these problems, the process of integration of new technologies in the network is slow, leading to the phenomenon known as "ossification" [1]. This event is expressed through a high rigidity of the current infrastructure, which is poorly elastic in adapting to dynamic requirements. This rigidity is linked to the devices that compose the network itself, i.e., routers, switches and middleboxes. These elements are frequently closed systems with proprietary interfaces provided by the manufacturers and limited control over the devices. Sometimes the issue stems from the network protocols. These considerations led to the introduction of the Software Defined Network (SDN) paradigm, providing a network that can be updated with simplicity, is easy to maintain, and has high flexibility and adaptability to new services and technologies. SDN divides the control plane from the data plane, and its resiliency is of primary importance.

The resilience of a network consists of the ability to recover logical control after the detection of a failure within a specific time window [2]. In computer networks, resilience is a key factor (a failure of a few milliseconds in a high-speed connection leads to a packet loss in the order of terabytes). In traditional networks, where data and control packets are sent on the same link, control and data information incur the same damage in case of network failures. Since the control plane and the data plane have the same resilience qualities, the prior work on evaluating the resilience of a network has assumed an in-band control model. However, this model does not apply to the divided architecture of SDN. For instance, control packets in SDN networks can be transmitted on paths different from data packets (or even in separate networks). For this reason, the reliability of the control plane

DOI: 10.5121/ijcnc.2022.14507

in this type of network is not linked to that of the forwarding plane. On the other side, cutting the two planes off could render the forwarding plane utterly useless. For example, when a switch is cut off from the control plane, it is unable to receive any instructions on how to forward fresh incoming packets, thus going offline [3].

In this study, we compare the primary strategies for resolving the software-defined networking challenge of controller placement. We reproduce the execution of the algorithms on an emulated testbed, and select the controller node according to the chosen algorithm. The algorithms are executed on several network topologies. We then analyze the resiliency of the algorithms in the case of link failure. We create a dynamic critical situation through a progressive link disconnection to check the algorithm resiliency, up to a completely unrecoverable state.

In Section 2 we present state-of-the-art solutions to improve SDN resiliency. After the presentation (Section 3) of a general SDN resiliency model, in Section 4 we focus our attention on the controller placement problem (CPP). In Section 5 we survey a selection of algorithms that can determine the optimal placement of the controller, especially in large network systems. In Section 6 and Section 7 we compare the behaviour of these algorithms through a series of experiments that emulate link failures. Section 8 contains some concluding remarks.

2 Background and Related work

The requirements for improving network properties that led to the introduction of the SDN paradigm, are not easy to meet. In fact, they impose numerous structural changes that hard to implement. The main IT companies have hence focused on controlling the network in a direct way in order to provide it with the required elasticity and simplicity [4]. The Internet Engineering Task Force (IETF) took the first step towards the SDN paradigm's practical implementation in 2004 by defining a standard interface called Forwarding and Control Element Separation (ForCES) . The network structure was completely redesigned, with a sharp separation between the physical level of the control (routing) and forwarding functions. Among the first implementations of this new paradigm, the most relevant ones were the Ethane project [5] and its direct predecessor, called SANE [6]. The mentioned projects designed a logically centralized, flow-level solution for access control in enterprise networks. Ethane introduces the concept of *flow tables*, ad hoc defined data structures that are populated only by the controller based on high-level security policies. It paved the ground to the creation of OpenFlow, as the simple switch design became the basis of the original OpenFlow API.

In 2008, the first version of the OpenFlow protocol was created, which relied on the structure described by ForCES. NOX, an operating system for networks [20], was released the same year. The introduction of these protocols aroused the interest of manufacturers, who began to develop the first networks based on OpenFlow switches at both academic and prototype level. In 2011, the Open Networking Foundation (ONF) was established, officially endorsing the introduction of the SDN idea and its protocol-level implementation through OpenFlow.

In a nutshell, Software Defined Networking (SDN) is a new approach to network management. In an SDN architecture, the routing (Control Plane) and forwarding (Data Plane) processes are decoupled, the intelligence and the network status are logically centralized, and the underlying network infrastructure is transparent to applications.

The SDN paradigm structure can be divided into three layers (Planes) in order to allow for an easy understanding of the operation and the components involved: Data, Control and Application Plane.

Terminologies and components that are useful to describe the role of such layers are briefly described below:

- SDN Application: an SDN Application is a program that explicitly, directly, and programmatically communicates to the SDN controller the requests and behaviors that the network must take. This happens through the so-called North Bound Interface (NBI). In addition, applications can use an abstract view of the network to make internal decisions. Thus, an SDN application consists in an application logic and one or more NBI interfaces.
- SDN Controller: the process of translating SDN application requests to the SDN datapath and giving the application an abstract picture of the network is performed by the SDN Controller, a logically centralized device (together with relevant statistics and events). An SDN controller is composed of one or more NBI agents, a SDN control logic, and a so-called Control to Data-Plane Interface (CDPI).
- Datapath SDN: The Datapath SDN is a logical network device that provides visibility and undisputed control over its public forwarding and data processing capabilities. The underlying physical resource may be entirely or partially included in the logical representation. The datapath consists of a CDPI agent and a group of one or more traffic forwarding engines and may or may not have traffic processing functions. Engines and functions can incorporate simple forwarding between external datapath interfaces, traffic processing functions, or termination functions. Datapaths might exist on a single physical network node or across multiple ones. Similarly, a single datapath can be defined through the use of multiple network elements. The logical definition does neither prescribe nor preclude implementation details such as physical mapping, shared physical resources management, virtualization, or datapath partitioning.
- Control to Data-Plane Interface (CDPI): The CDPI is a defined interface between a controller and a datapath, that provides at least a code-level control of all forwarding operations. It also has capabilities for publishing, reporting statistics, and event notifications. A feature of SDN is the expectation that the CDPI will be implemented in an open, manufacturer-neutral, and interoperable manner.
- NorthBound Interface (NBI): NBIs are interfaces that stand between applications and controllers and generally provide abstract views of the network and enable direct expression of network behavior and requests. This can happen at any level of abstraction and through different groups of features. In this case, the SDN can express itself through a free and open implementation of the interface.

Each presented interface is implemented through a coupling with an agent, where the agent represents the lower side, i.e., the side facing the infrastructure, and the interface represents the upper side, i.e., the side facing the application. We now examine the interaction among these components through the three planes of operation.

At the bottom layer, the Data Plane is composed of network elements whose datapaths display their capabilities through the CDPI agent. On top of that there are the SDN applications within the Application Plane, which communicate their requests through NBI interface drivers. In the center, the controller translates these requests and exercises low-level controls through datapaths, while providing information to SDN applications. However, network management is required by an administrator, who is responsible for configuring the network elements, assigning SDN datapaths within the controller, and configuring the policies that define the purpose of the control or the application that uses it. This type of network can coexist with a non-SDN infrastructure, especially when migrating to a network that is fully enabled. The operation described may lead us to think

that the network is based on the use of physically centralized controllers. Actually, the description does not imply that the controller is implemented as a single device. For performance, scalability, and/or availability issues, a centralized SDN controller can be used at the logic level. This way, the controller benefits from a multitude of instances that cooperate for network control and proper application operation. With the SDN, the control plane behaves as a single network operating system logically centralized both from the point of view of resource management and its conflicts and as an abstraction of the details of low-level devices (such as electrical and/or optical transmission). In addition, there is a fourth layer, called Management Plane, which provides coverage of tasks that are more easily manageable outside of the other plans. Some examples can be the management of the economic relationships between providers and clients, the allocation of resources to the client, or the setup of physical equipment [7]. Of course, every economic entity has management bodies. In any case, communication between them goes beyond the purposes of the SDN architecture. One of the solutions is the convolution of the known management tasks from the traditional network within the CDPI. However, the description of the SDN paradigm remains very generic, leaving developers with many possibilities for implementation at the physical level. In fact, it is possible that two totally different networks interact with each other from a component/structural point of view, maintaining and exploiting SDN functionality without incurring any issues.

The Controller Placement Problem (CPP) was first introduced in 2012 [8] and it was soon clear that the CPP is NP-hard, even if a simple k-center algorithm is adopted [9]. A review of the proposed algorithms is presented in [10] and provides a taxonomy of the CPP proposals. The authors claim the importance of relying on multiple controllers to manage large networks. The CPP can determine the minimum number of controllers required to manage a system, as well as the optimal position. The algorithms are divided into two classes, namely *uncapacitated* and *capacited* controllers. As the term suggests, the former class is characterized by controllers having unlimited capacity (the opposite being true for the latter class). Another conclusion of this survey is the importance of CPP algorithms capable of managing network fractioning in the event that large networks require the division into sub-systems to reduce complexity. Many studies classify CPP on the basis of several factors or metrics related to SDN performance and QoS. The analysis in [11] analyzes factors including delay, cost, and reliability, but offers no information into resilience. In [12], the resilience is dealt with and modeled in accordance with the controller, link, or switch failure. The sudies in [13] [14] also reported from a comparison of the key ideas and improvement strategies dealing with the link failures. In [15], the resilience depends on many controllers connected to a switch in order to meet specific quality of service standards. To handle the switch traffic load, the system accounts for the switch-controller, inter-controller, and controller capacity latency requirements. The work presented in [16] offers an optimization model for installing controllers and mapping switches to controllers while ensuring full resilience against a predetermined number of controller failures, which is similar to our approach.

3 SDN resiliency model

Before delving into the resilience property of an SDN network, it is necessary to first introduce the SDN network model [17].

Let us consider a single SDN domain where switches are grouped under a single controller. The physical network can be represented as a graph, $G(V, E)$, where V is the collection of switches (or nodes), and E is the collection of network links. The link between two nodes u and v is represented as (u, v). We represent the controller as V_c (where

$V_c \in V$). Let us consider a control network composed of multiple switches and one controller, already selected to allow the transmission of control signals. The active connections are in-band, which means there are no links dedicated to controlling traffic. In addition, there is no load balancing on the controlling traffic, so all nodes will only have one single route to the controller. Hence, the controlling traffic is sent to and from the controller through a tree, where the controller is configured as the root. This structure is called *Controller Routing Tree*. Once the control network structure is defined, the switches on the network are divided in two types: protected and unprotected. To differentiate between these two types of switches, we first define a node j as upstream for a node i if j provides a path for i to the controller V_c.. Otherwise, j is a downstream node for i. Then, a switch is protected against failure if it can use an outbound backup link for the control traffic towards the controller. More specifically, a switch A is secured if and only if the network contains a switch B that satisfies these two requirements: 1) Switch B is not a downstream switch (or it is a parent for A, where parent of A denotes the set of A's upstream switches). 2) There is a link between A and B which is not part of the controller routing tree. If an output link or an upstream node of a protected switch fails, it can immediately change the path towards the controller and use the backup link to reconnect to the controller as soon as the failure is detected. Hence, the re-routing of control traffic happens with local change and without affecting the connections of the other switches.

In this type of network, the switches can not inform the downstream switches about faults. As a result, even if all downstream nodes are protected, if a switch is disconnected from the controller, all downstream nodes will also be disconnected. This means that evaluating the resilience of a network requires assigning a weight to switches that have multiple downstream nodes (generally, the assigned weight is equal to the number of downstream nodes). We define the weight of the routing tree as the sum of the weights of all unprotected nodes. This number is used to assess a network's degree of security (the lower this value, the more protected the network). For a given routing tree T, we will refer to this weight with $\Gamma(T)$.

4 Controller Placement Problem (CPP)

The Controller Placement Problem is a key issue in building a resilient and reliable network. This problem is related to the positioning of the the controller so to provide maximum protection of the network nodes, as well as a reduction in propagation delays [9]. The weight of the network can be significantly reduced with the right controller positioning, which forms the basis for selecting the routing tree method. In this section we will consider a general routing policy (in order to focus mainly on CPP), which, given a controller position, will select the primary network paths with the least number of nodes crossed in order to send control traffic. Let's see with an example how the positioning of the controller affects the resilience of the network:

As the network is a complex system, it is important to define a parameter like the centrality to define the importance of a node within a network. A node with the best centrality is the candidate to play the role of the controller [18]. The *Closeness Centrality Theorem* makes it possible to implement a resilient network. For a given graph $G(V, E)$, the closeness centrality of a node V_k can be defined as: $C_c(V_k) = \frac{1}{d'(V_k)}$, where: $d'(V_k) = \frac{1}{(n-1)} \sum_{v_k \neq v_j} d(k, j)$

The value d(k,j) is the minimum distance between k and j [19]. The equation can then be written like this: $C_c(V_k) = (n-1)/(\sum_{v_k \neq v_j} d(k, j))$

As a result, we may state that a node's proximity centrality V_k is the inverse of the average distance that separates it from other nodes in the network.

Another important factor in choosing the controller location is the latency of the network. We introduce two types of latency, which we will consider in the evaluation of the responsiveness of the network. Average-case latency is the average propagation latency for a given controller location. Given two nodes $v, sinV$ and a number of nodes $n = |V|$, the propagation latency for a controller positioning S' is as follows:

$$L_{avg} = \frac{1}{n} \sum_{v \in V} \min_{s \in S'} d(v, s) \tag{1}$$

Once more, the shortest distance between the two nodes is represented by d(v, s). The objective is to locate a site S' where the latency $L_avg(S')$ and cardinality $|S'| = k$ are as low as feasible.

An alternative approach is worst-case Latency, i.e., measuring the maximum propagation latency from the node to the controller: $L_{wc}(S') = \max_{v \in V} \min_{s \in S'} d(v, s)$ In this instance, we are also searching for the lowest $S'subseteqS$. There are of course other metrics for measuring latency. These two evaluation techniques were chosen because they consider the distance to each node, providing a more general view of the network.

5 CPP algorithms

In this section, we will analyze the main algorithms used to solve the Controller Placement Problem.

5.1 Optimized Placement Algorithm (OPA)

To search for the optimal controller location, OPA looks for the node that maximizes resilience across all possible network locations [18]. The algorithm outputs the controller position based on the entire $G(V, E)$ network topology with $|V| = n$:

Algorithm 1 Controller Placement Optimal Algorithm

procedure *Optimal_Placement(T)*

```
for each node v ∈ V do
    T = controller routing tree rooted at v
    for each node u ≠ v do
        W = 0
        if u is not protected then
            W = number of downstream nodes of u in T
        end if
        Γ(T) = Γ(T) + W
    end for
    controller_location = node v with minimum Γ(T)
end for
```

Fig. 1. Pseudo-code of the Optimized Placement Algorithm (OPA)

As presented in the pseudo code OPA implementation in Fig. 1, at line 5, the protection of a specific switch is evaluated according to the definitions presented in the previous section. The last step of the algorithm (line 10) is to seek the position that minimizes the weight of the network, thus maximizing its robustness. The nature of this algorithm entails that the optimal position in a network is always chosen (hence the name), making it the best placement algorithm in theory. However, in big networks, scanning across all potential nodes might become prohibitively expensive and time-consuming. As the network grows in size, the result of takeover process becomes less and less effective. The algorithm described in the next section is based on heuristics that allow good positioning of the controller without making major compromises.

5.2 Greedy Control Placement Algorithm

The Greedy Controller Placement Algorithm is a heuristic method to find the best candidate position for the controller. This algorithm can be conveniently executed if we represent the network graph through an adjacency matrix. This is a square binary matrix, the size N being the number of nodes. Each matrix element (i, j)can be 1if and only if there is an edge in the graph connecting vertex i to vertex j; otherwise, there is a 0. The adjacency matrix is slightly modified to include a weight $w(i, j)$ instead of a simple binary value to indicate the cost of the link (i, j).

The algorithm starts by sorting the nodes in descending order according to their degree, that is, the number of siblings a node is directly connected to. To do this, a table T of size N is created to contain the list of nodes sorted according to their degree. In the second step, each switch is inspected from the top of the table T to evaluate the number of protected nodes. The element with the highest number of protected nodes is then the candidate controller.

5.3 K-Median Algorithm

The K-Median Algorithm employs a modified version of the adjacency matrix, which contains the latency value for each link (i, j) if the link exists, otherwise it is set to 0. If such a path exists, the Floyd-Warshall algorithm can calculate it for each pair of nodes in V. This step leads to the evaluation of the average propagation delay from all the nodes towards a candidate node. Hence, it is possible to compute the average latency from each node to a candidate controller placed in S':

$$L_{min} = \frac{1}{n} \sum_{v \in V} \min_{s \in S'} d(v, s) \tag{2}$$

The position that minimizes such an average is selected for the placement of the controller.

6 Comparison between OPA and GPA

For a first comparison between OPA and GPA, a network consisting of 30 nodes has been created in Mininet, as depicted in Fig. 2

If we compute the optimal position through the Optimal Placement Algorithm, the output of the process is reported as candidate node s1. On the other hand, with the Greedy Placement Algorithm, the given output position is s2. In the first case, the controller is directly connected to 3 protected nodes, while in the second case, the controller is directly

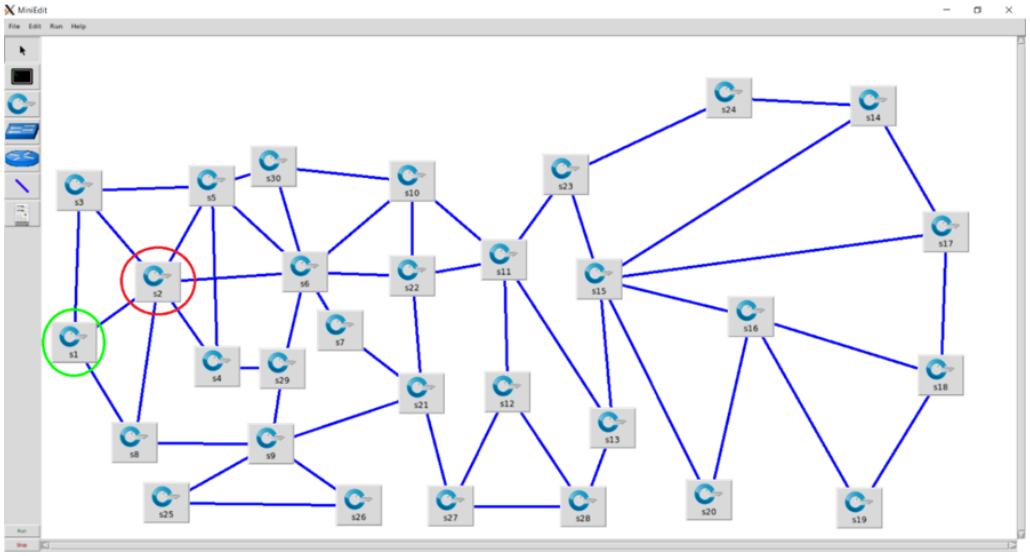

Fig. 2. The node circled in green (s1) is the selected node for the controller according to the OPA. The node circled in red (s2) is the position chosen by the GPA.

connected to 6 protected nodes. The total number of links is 52. The first test gauges how resilient the network actually is in the face of link failures. We expose the network to a sequence of disconnections to evaluate the effective communication capability among all switches. A ping connection between node 3 and node 18 is kept active throughout all the experiments. We present the results in the following graphs. In each graph, the x axis reports mainly the ICMP sequence number of the stream between node 3 and node 18. On the same axis it is reported the progressive number of disconnected links. The measured delay is reported on the y axis. For better readability, the disconnections are pointed out with red X, and the subsequent ICMP lost packets are not reported on the graphs [1] .

[1] Experimental raw data are available at https://github.com/maudarie/SDN_cpp/

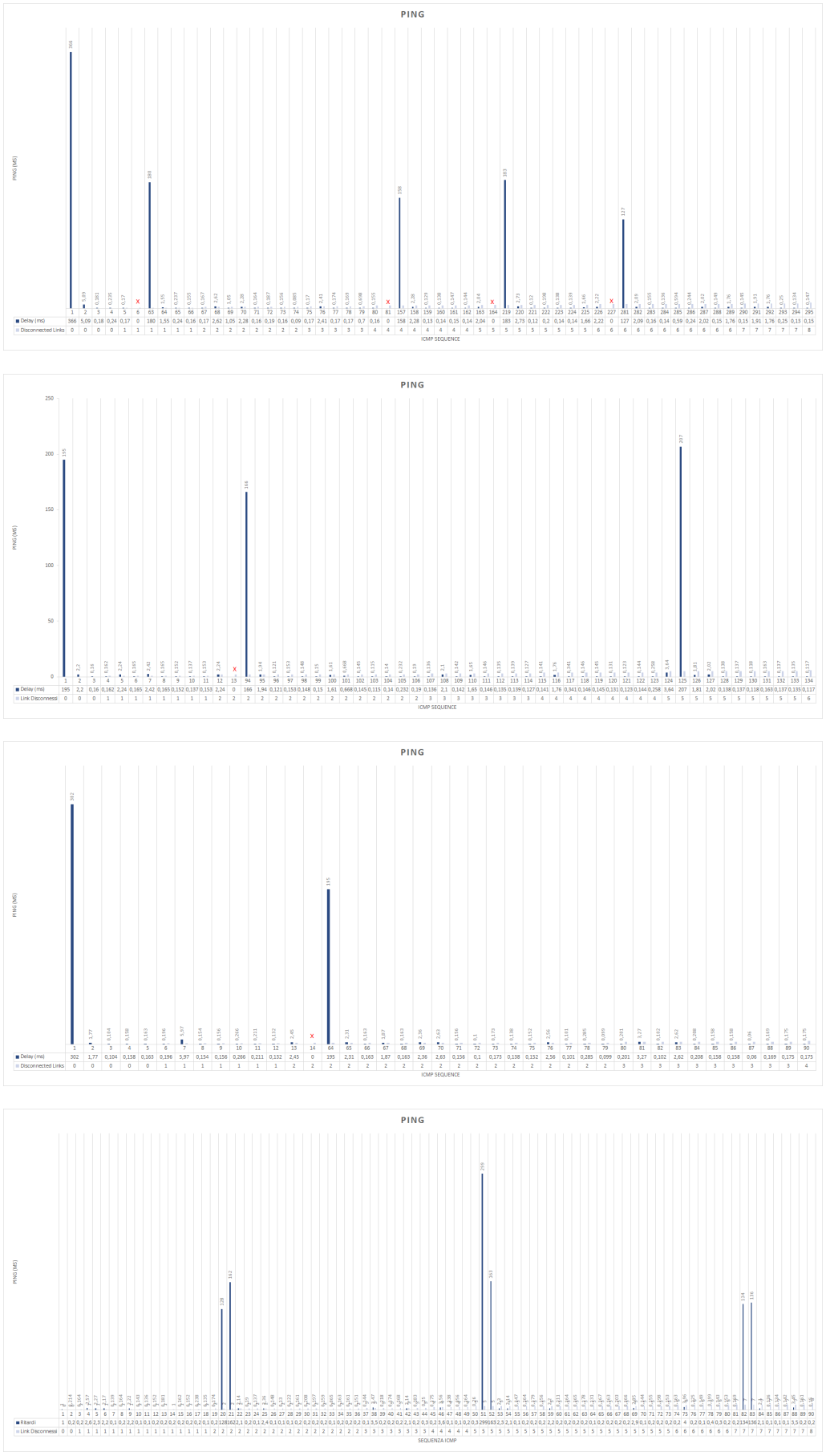

Fig. 3. Greedy Placement Algorithm resiliency test

Thus, the first ICMP packet after a disconnection is the first successful reply. Fig. 3 presents the results of four experiments, in all of which the GPA is enabled. In each of the experiments, the disconnected links are randomly selected.

Similarly, Fig. 4 presents the results of four experiments in which the OPA is enabled. Also, in this latter case, the disconnected links are randomly selected in each of the experiments.

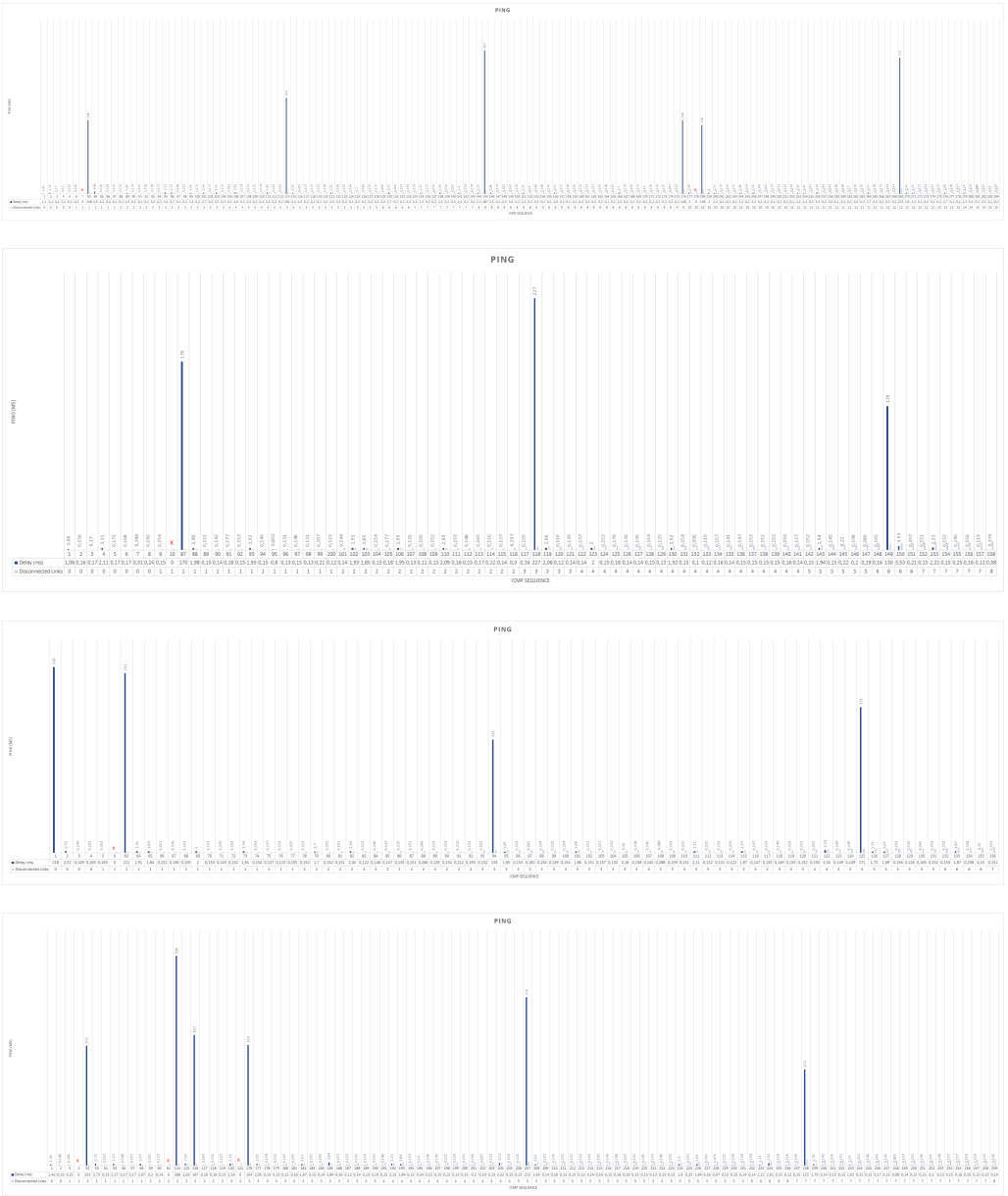

Fig. 4. OPA resiliency test

As a final result of our experiments, the maximum number of links that can be disconnected is 23 for the OPA and 18 for the GPA. Such figures ensure full operation with 44.2% and 34.6% disconnected nodes, respectively. As expected, the OPA has greater network robustness than the GPA. Greedy Placement, on the other hand, has better computational simplicity, especially when evaluating positioning in more complex networks.

7 Comparison between GPA and the K-median algorithm

We compared the implemented GPA and K-median algorithms in an emulated network topology. For implementing the K-median algorithm, an algorithm was developed that creates a single cluster containing all of the network nodes. This is done on purpose, in order to allow for a fair comparison with the Greedy Control Placement. The testing framework is based on the Mininet environment. The sample topologies are composed of Open vSwitch virtual switches, configured and managed through the OpenFlow Protocol. The CPP algorithms are compared on several network topologies configured in a mininet environment. The first topology in Fig. 5 is composed of 15 nodes and 27 connections.

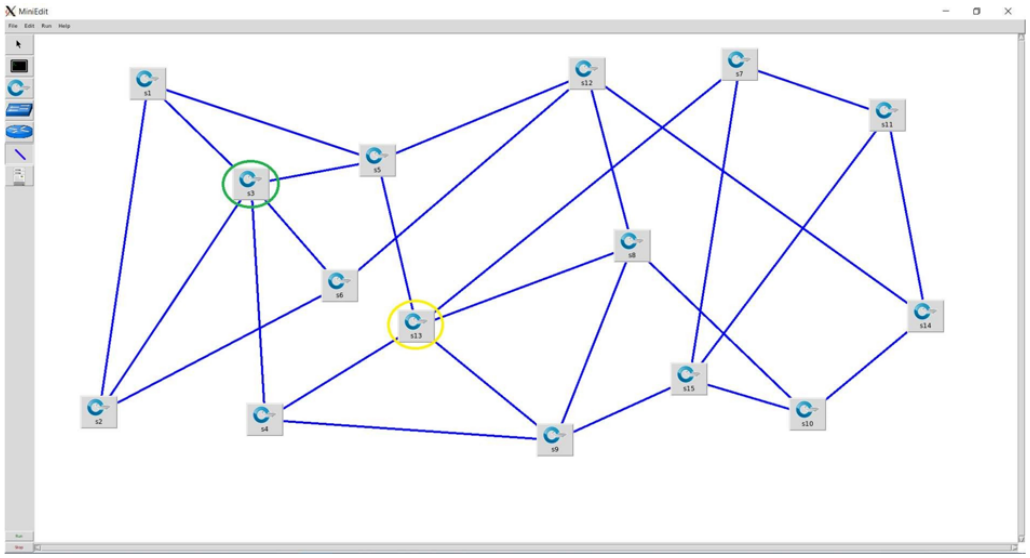

Fig. 5. The node circled in green (s3) is the controller according to GPA. The node circled in yellow (s13) is the position chosen by the K-Median algorithm

In this topology, the GPA output is reported as candidate node s3. By applying the K-Median algorithm we instead observe that the given output position is in s13. In the first case, the controller is directly connected to 4 protected nodes, while in the second case, the controller is directly connected to 3 protected nodes. The average propagation latency referred to node 3 is $2,066ms$, while that referred to node 13 is $1,733ms$.

A second topology is composed of 20 nodes and 40 connections, as depicted in Fig. 6

In this case, the outcome of both algorithms indicates the candidate node s10 for the placement of the controller. The controller is directly connected to 5 protected nodes. The average propagation latency referred to node 10 is: $1.75ms$.

The third topology in Fig. 7 is composed of 32 nodes and 63 connections.

By computing the optimal position through the Greedy Placement Algorithm, the output of the process reports the node s4 as candidate. If we apply the K-Median approach, the given output position is instead s10. In the former case, the controller is directly connected to 7 protected nodes, while in the latter, the controller has no connection with protected nodes. The average propagation latency for node $4is2.4375$ms, while for node $10itis2.0625$ms.

The fourth topology is presented in Fig. 8 and is composed of 30 nodes and 52 connections.

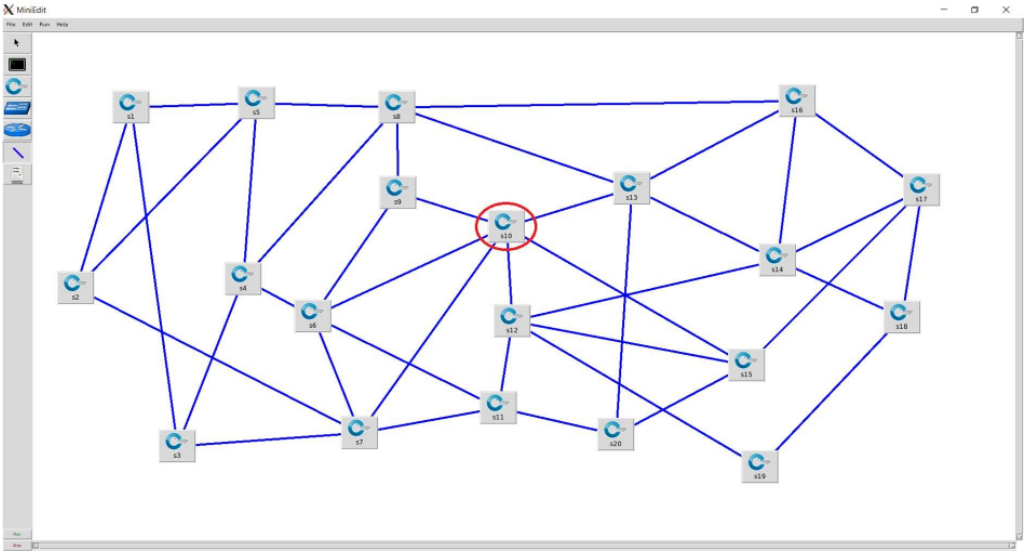

Fig. 6. The red circled node (s10) is the selected node for the controller according to both GPA and K-Median

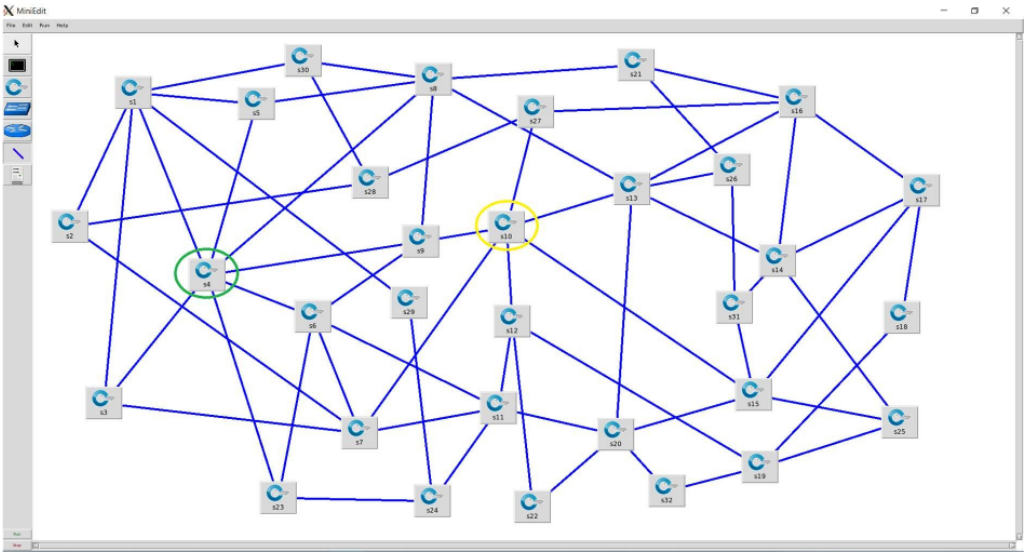

Fig. 7. The node circled in green (s4) is the selected controller node according to GPA. The one circled in yellow (s10) is instead chosen by K-Median

The optimal position reported by the Greedy Placement Algorithm is node s2. By applying the K-Median approach, the resulting position is instead node s22. In the former case, the controller is directly connected to 6 protected nodes, while in the latter case it has direct connections with 3 protected nodes. The average propagation latency referred to node 2 is $3.3ms$, while that referred to node 22 is $2,566ms$.

The fifth topology, depicted in Fig. 9, is composed of 9 nodes and 15 connections.

The optimal position computed through the Greedy Placement Algorithm is node s7. By applying the K-Median approach, we find the selected candidate is instead node s3. In the former case, the controller is directly connected to 4 protected nodes, while in the latter case it has 3 direct connections with protected nodes. The average propagation latency referred to node 7 is $1.33ms$, while that referred to node 3 is $1.22ms$.

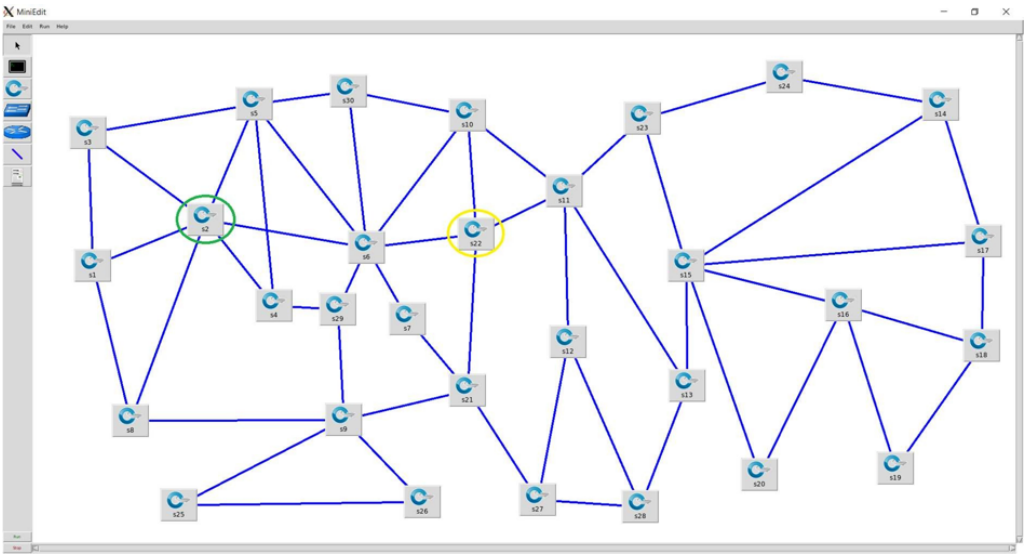

Fig. 8. The node circled in green (s2) is the selected node for the controller according to GPA. The one circled in yellow (s22) is instead chosen by K-Median

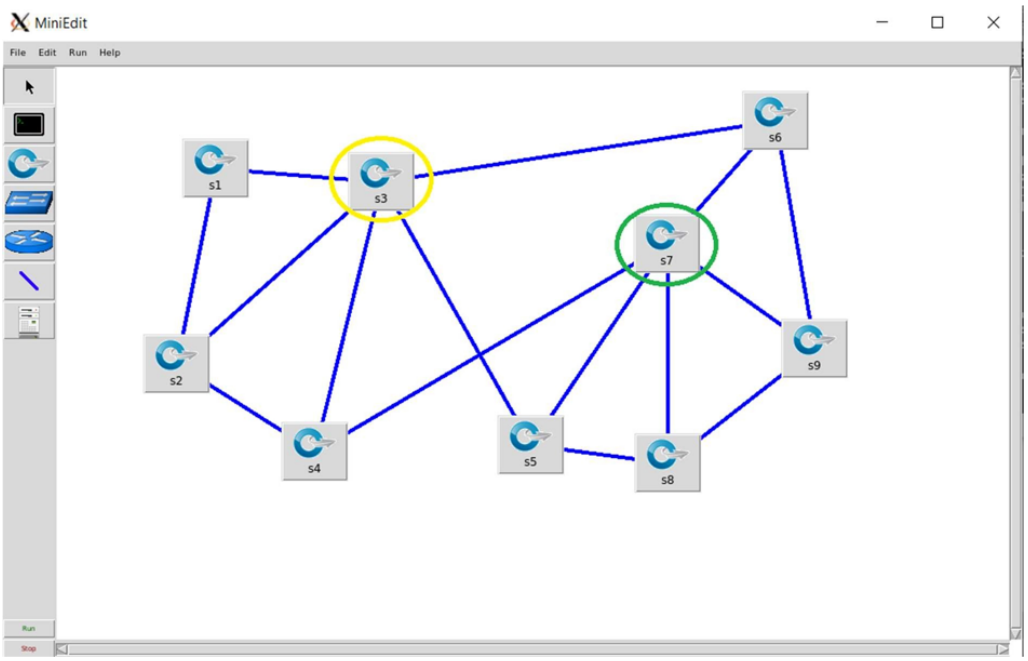

Fig. 9. The node circled in green (s7) is the selected node for the controller according to GPA. The one circled in yellow (s3) is instead chosen by K-Median

We finally provide a comparative view of two parameters of interest associated with the controllers selected by the two algorithms, namely: (i) the number of adjacent protected nodes; (ii) the average propagation latency.

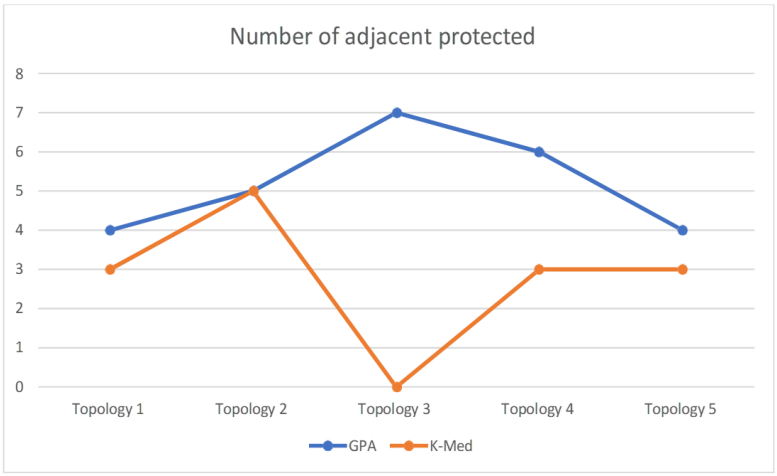

Fig. 10. Number of adjacent protected nodes of the controllers chosen by the two algorithms

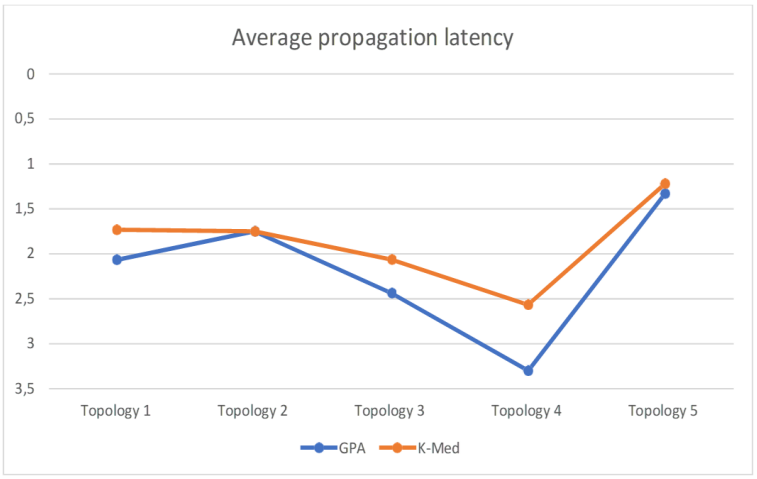

Fig. 11. Average propagation latency of the controllers chosen by the two algorithms. A low value corresponds to higher network performance.

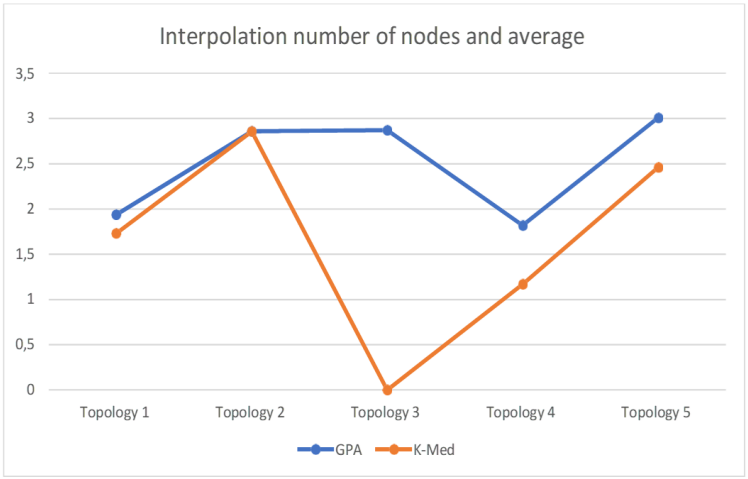

Fig. 12. Interpolation of the number of adjacent nodes and the average propagation latency. High values mean better performance and reliability for the network.

As a result of the experiments carried out, we notice that there is a considerable discrepancy between GPA and K-Median with regard to the number of protected nodes; if we analyze instead the average propagation latency, we do not find substantial differences between the two algorithms. These findings lead us to the conclusion that the Greedy Placement Algorithm is the most suitable option to locate the controller in a network where rapid response times and high speed are desired.

8 Conclusions

In this work we have evaluated the resiliency of the main algorithms for the Controller Placement Problem in SDN networks. We compared the Optimal Placement Algorithm, the Greedy Placement Algorithm and the K-Median algorithm. We tested their resiliency on emulated network topologies, by gradually increasing the number of link disconnections. From the presented experimental results carried out on several topology configurations, we were able to derive interesting considerations about the respective performance of the analyzed approaches. We were particularly interested in investigating the different reactions to the disconnection test. Based on the results of the testing campaign, we argue that, depending on parameters like the type of the network, its structure, and density, it is possible to select the algorithm that strikes an optimal balance between reliability and performance.

Conflicts of interest

The authors declare no conflict of interest.

References

1. B. A. Nunes, M. Mendonca , X.-N. Nguyen, K. Obraczka, T. Turletti, “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,” IEEE Communications Surveys & Tutorials, vol. 16, n. 4, pp. 1617-1619, 2014.
2. Sahoo, Kshira Sagar, et al, “Improving Resiliency in SDN using Routing Tree Algorithms,” IJKDB 7.1 (2017): 42-57. Web. 15 Jun. 2020. doi:10.4018/IJKDB.2017010104
3. Y. Zhang, N. Beheshti, M. Tatipamula, “On Resilience of Split-Architecture Networks,” 2011 IEEE Global Telecommunications Conference - GLOBECOM 2011, Houston, TX, USA, 2011, pp. 1-6, doi: 10.1109/GLOCOM.2011.6134496.
4. Nick Feamster, Jennifer Rexford, E. W. Zegura, “The road to SDN: an intellectual history of programmable networks,” ACM SIGCOMM Computer Communication Review Vol. 44, No. 2, April 2014, pp 87–98, doi:https://doi.org/10.1145/2602204.2602219
5. Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., Shenker, S. “Ethane: taking control of the enterprise,” In Proceedings of ACM SIGCOMM, 2007.
6. Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., Shenker, S., “SANE: a protection architecture for enterprise networks,” In Proceedings of the 15^{th} Usenix Security Symposium, 2006.
7. Kuribayashi, Shin-ichi, “Dynamic Shaping Method using SDN And NFV Paradigms,“. International journal of Computer Networks and Communications. 13. 1-14. 10.5121/ijcnc.2021.13201.
8. T. Das, V. Sridharan and M. Gurusamy, ”A Survey on Controller Placement in SDN,” in IEEE Communications Surveys and Tutorials, vol. 22, no. 1, pp. 472-503, Firstquarter 2020, doi: 10.1109/COMST.2019.2935453.
9. Heller, Brandon, Rob Sherwood, and Nick McKeown. ”The controller placement problem.” ACM SIGCOMM Computer Communication Review 42.4 (2012).
10. Singh, Ashutosh and Srivastava, Shashank, “A survey and classification of controller placement problem in SDN,” International Journal of Network Management (2018). e2018. 10.1002/nem.2018.
11. B. Isong, R. R. S. Molose, A. M. Abu-Mahfouz and N. Dladlu, ”Comprehensive Review of SDN Controller Placement Strategies,” in IEEE Access, vol. 8, pp. 170070-170092, 2020, doi: 10.1109/ACCESS.2020.3023974.

12. Dhar, M, Debnath, A, Bhattacharyya, BK, Debbarma, MK, Debbarma, S. A comprehensive study of different objectives and solutions of controller placement problem in software-defined networks. Trans Emerging Tel Tech. 2022;e4440. doi:10.1002/ett.4440
13. Sridharan V, Gurusamy M, Truong-Huu T. On multiple controller mapping in software defined networks with resilience constraints. IEEE Commun Lett. 2017;21(8):1763-1766.
14. Li H, Li P, Guo S, Nayak A. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. IEEE Trans Cloud Comput. 2014;2(4):436-447
15. Tanha M, Sajjadi D, Ruby R, Pan J. Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs. IEEE Trans Netw Serv Manag. 2018;15(3):991-1005
16. Killi BPR, Rao SV. Towards improving resilience of controller placement with minimum backup capacity in software defined networks. Comput Netw. 2019;149:102-114
17. N. Beheshti and Y. Zhang, "Fast failover for control traffic in Software-defined Networks," 2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, 2012, pp. 2665-2670, doi: 10.1109/GLOCOM.2012.6503519.
18. K. S. Sahoo, B. Sahoo, R. Dash e M. Tiwary, "Solving Multi-Controller Placement Problem in Software Defined Network," International Conference on Information Technology, pp. 188-192, 2016
19. Floyd, Robert W., "Algorithm 97: Shortest Path," Communications of the ACM. 5 (6): 345. (June 1962). doi:10.1145/367766.368168
20. N. Gude, B. Pfaff, T. Koponen, M. Casado, S. Shenker, J. Pettit e N. McKeown, "NOX: Towards an Operating System for Networks," in ACM SIGCOMM Computer Communication Review, 2008, pp. 105-110.