On the controller placement for designing a distributed SDN control layer

Yury Jiménez, Cristina Cervelló-Pastor, and Aurelio J. García

Abstract—**The software-defined network (SDN) advocates a centralized network control, where a controller manages a network from a global view of the network. Large SDN networks may consist of multiple controllers or controller domains that distribute the network management between them, where each controller has a logically centralized but physically distributed vision of the network. In this context, a key challenge faced by providers is to define a scalable control network that exploits the benefits of SDN when used in conjunction with efficient management strategies. Most of the control layer models proposed are not concerned with controller scalability, because they assume that commercial controllers are scalable in terms of capacity (quantity of flows processed per second). However, it has been demonstrated that overloads and long propagation delays among controllers and controllers-switches can lead to a long response time of the controllers, affecting their ability to respond to network events in a very short time and reducing the reliability of communication.**

In this work we define the principles for designing a scalable control layer for SDN, and show the desired control layer characteristics that optimize the management of the network. We address these principles from the perspective of the controller placement problem. For this purpose we improve and evaluate our previous approach, the algorithm called k-Critical. K-Critical discovers the minimum number of controllers and their location to create a robust control topology that deals robustly with failures and balances the load among the selected controllers. The results demonstrate the effectiveness of our solution by comparing it with other controller placement solutions.

Index Terms—**Software-defined network, controller scalability, control layer, controller placement problem.**

I. Introduction

NETWORK architectures such as Software-Defined Networking (SDN) in which the control plane is decoupled from the data plane have been growing in popularity. Basically, the rationale behind this is to provide a more structured software environment for developing network-wide abstractions while simplifying the data plane. SDN makes networking functions available as programmable resources, via a logically centralized controller, which manages and operates a network (switches) from a global view of the network.

The authors are with the Department of Telematics Engineering, Universitat Politècnica de Catalunya (UPC), Esteve Terradas, 7, 08860, Castelldefels, Spain, e-mails: {yury.jimenez;cristina;aurelioj.garcia}@entel.upc.edu.

This work has been supported by the Government of Catalonia through a predoctoral FI scholarship and the Government of Spain through project TEC2010-20527-C02-01.

ISBN 978-3-901882-58-6 © 2014 IFIP

In the SDN architecture, the controller has knowledge of the network topology created by the forwarding devices that it manages, and is responsible for configuring the forwarding state on those devices through the control channel. This managed state is a set of forwarding tables that provide a mapping between packet header fields and actions to execute on matching packets. This abstraction enables the controller to easily enforce sophisticated flow-based traffic management policies in the network.

Although decoupling control functions provides network flexibility, this also brings scalability concerns [1] [2] [3]. Some concerns in SDN are related to the incremental load on controllers and the controller communication to handle the consistency of the network-state information among them. These problems are not any different from those faced in the design of traditional distributed networks. With the current trend towards implementing SDN in large-scale networks, the control layer composed of geographically separated controllers with shared computational resources must satisfy the performance of some metrics and must also be robust to failures.

As SDN is unable to directly control the whole underlying physical link resources, several aspects of the traditional network operation can affect its performance. For instance, failures in the underlying resources may lead to switches being disconnected from the control layer, and controllers located in a dense area of the network (with high connectivity) can become the bottleneck in the network operation. Thus, consideration is required of the technical and operational characteristics of the physical network to design an efficient and robust control layer.

In this work, some principles for building a scalable, robust and balanced control layer are described from the perspective of the controller placement problem. We introduce k-Critical [4], an algorithm that finds the minimum number of controllers to satisfy a target communication between controller and nodes, such as delay, latency, convergence time, etc.

In addition, we demonstrate that a good selection of controllers may not only balance the load among them, but may also reduce the data loss in the control layer. As a result, the selected controllers can efficiently distribute the management duties between them to improve scalability of the management process, where each controller operates on its own abstract view of the network.

The remainder of the paper is organized as follows. In Section II, we introduce some approaches that have been proposed to alleviate the scalability problem in SDN. In Section III, the principles for building a scalable control layer are presented. In Section IV, some approaches for solving the controller placement problem are described. In Section V, k-Critical is explained in detail. Numerical results of the methods presented, including our proposal are shown in Section VI. Finally, conclusion and future work are presented in Section VII.

II. Related work

The controller scalability is related to its capacity to handle a determined number of flows per second and its response time to events, such as updating information on controllers, response to a request, etc. These not only depend on the capacity of the controllers for storage and processing, but also of their placement in the network. In this section, some approaches that have been proposed to alleviate the scalability problem in SDN are described.

A. Logical Mechanisms

In order to address the limitations of controller scalability and network-state consistency some strategies have been considered in the controller software. For instance, in [2] the authors propose a distributed system that runs on a control platform called Onix. This distributed platform handles the distribution and collection of information from switches and distributes controls appropriately among various controllers. In this approach, the applications are implemented by reading and writing over a data structure that stores the network state, called Network Information Base (NIB). Onix proposes three strategies that can be used to improve the controller scalability, which are partition, aggregation, and consistency and durability. Partition allows control applications to divide the workload among controller instances, keeping only a subset of the NIB in memory. Aggregation allows the exposure of one subset of elements as an instance of another Onix instance. Finally, consistency and durability are ensured by using control applications for managing the network state. In Onix, the NIBs operate asynchronously, and no ordering or latency guarantees are given.

HyperFlow [5] is a control application that works with an event propagation system. This application publishes events that change the network state, and other controllers replay the event to update the network state. In this way, the information is kept consistent among the controllers. Both the approaches of HyperFlow and DIFANE [6] introduce new functionalities in switches to suppress frequent events and to reduce the load on the controllers. Adding new primitives to switches is undesirable because it breaks the general principles of Software-Defined Networks (SDNs).

In order to limit the overload on the control layer in [7] some events are managed by local controllers that are located near the forwarding devices. For this purpose, the authors present a distributed control plane composed of two levels of hierarchy controllers. The bottom layer (local controllers) is a group of controllers with no interconnection, and no knowledge of the network-wide state, while the top layer (root controller) is a logically centralized controller that maintains the network-wide state.

These approaches consider network strategies implicit in the controller structure to improve the controller scalability. In general, these solutions consider that the control topology is defined by geographical proximity among controllers and switches.

B. System Design

The importance of controller placement is well established [1] [8] [9] [10]. In [8] the authors show the implications of changing the number of controllers for the latency between switches and controllers, and [9] develops different algorithms to make placement decisions to maximize the reliability of SDN. In [10] the authors describe the implications of the controllers view of the local network and discuss the design of a distributed control plane. In [11] it is demonstrated that the switch-over time depends on the latencies between networking devices and the controllers. In [12] the authors demonstrate that long propagation delays among controllers limit the network convergence time, and affect the controller ability to respond to network events in a minimal time.

In general, most of these approaches ignore i) the control layer topology, ii) how many controllers are required to satisfy network requirements and iii) the management distribution among controllers.

III. Principles for designing the Control layer

The control layer in SDNs may take any topology, including that of a star, where a single controller manages the network; a hierarchical architecture where controllers are connected creating a mesh network; or even a ring composed of a set of controllers that are managed using a distributed hash table. Each one of these control layer topologies has implicit limitations. However, regardless of the topology, there are some general aspects, such as the distance both among controller-switches and among controllers, as well as the workload on each controller, that affect the ability of the controllers to respond to network events [9].

A. Control layer

Here we consider the control layer as a virtual overlay network that is constructed on top of the underlying physical network, where every node forwards queries to the controller through its control path. In this context, we address a

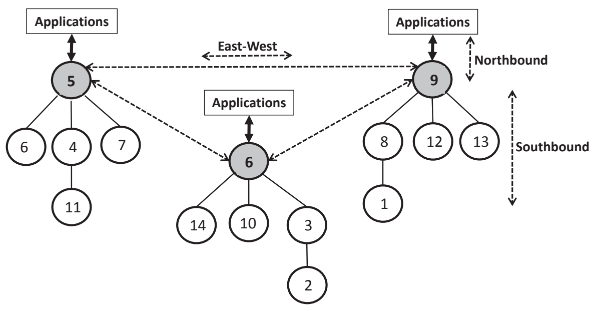


Fig. 1. Representation of the control layer from the NSFNet topology.

distributed control layer which is composed of a set of trees rooted at controllers, such as in Fig. 1. Thus, each controller in the control layer manages a set of nodes that make up the tree. The paths between the controller and each node in the tree are the control paths used to exchange control information. Some protocols to coordinate the operation among switches-controllers and also among controllers are required in SDN. The protocol between the controller and switches is referred to as southbound; northbound refers to the communication among controller and the application. The east-west protocol coordinates the communication among the SDN controllers.

In this context, several aspects must be considered in the control layer design, such as: how many controllers are required, and where these controllers should be located in the network in order to minimize the communication time in the control layer and balance the load among controllers.

B. Load Distribution among controllers

One of the key functions of an SDN controller is to establish flows. As such, the performance metrics associated with an SDN controller are the flow setup time and the number of flows per second that the controller can setup. Flows can be configured proactively or reactively. In a proactive flow setup, the switches know in advance what to do with the flow received. In contrast, the reactive setup occurs when the switch receives a flow that does not match the flow table entries. In this case, the switch has to forward the flow to the controller for processing. The time associated with the reactive flow setup is the sum of the time it takes to send the packet from the switch to the controller, the processing time in the controller and the time it takes to send the configuration message back to the switch. Assuming that the controller is not a bottleneck and that it has information about the service required, the total setup time can only be affected by the distance between switch-controller. In [12], it has been demonstrated that a long propagation delay limits the network convergence time, and affects the controller ability to respond to network events in a minimal time. In this context, a good controller placement is essential in order to i) distribute the load among controllers to avoid the controller becoming congested and ii) minimize the communication time between controller and switches. By restricting the communication time among the switches and the controller defined as $\mathcal{D}_{req}$, the response time may be reduced and load distribution among controllers may be balanced.

C. Connectivity among controllers

Large SDN networks operate on a global network view that is logically centralized. However, control layer state and logic must be physically distributed in order to achieve responsiveness, reliability and scalability goals [13]. The process of configuring changes over each controller imposes overload and instability on the network.

In [13], the authors describe the impact of the physical distribution of the control plane for the performance and coordination of a control application logic. Regardless of the strategies used to manage the network state consistency, the connectivity among controllers determines the maximum time required to update information among them. It is called the window of inconsistency in [12]. This is a factor of delay bounded by the farthest controllers in the network and the load on controllers.

In this context, we consider that the communication delay time between the controllers i and j defined as $\mathcal{D}_{(i,j)}$ must be a required constraint, $\mathcal{D}_{cont}$. Consequently, the farthest controllers in the network i and j satisfy the condition $\mathcal{D}_{(i,j)} \leq \mathcal{D}_{cont}$.

IV. Controller Placement

In this section we briefly introduce two algorithms used in clustering, k-Center and k-Median, which were adapted in [4] to find the number of controllers required to satisfy a delay time, $\mathcal{D}_{req}$. In these algorithms, it is considered that any node in the network can be replaced by a controller. Hence, the objective is to find the best controller placement.

Consider that the physical network is modeled by a graph denoted by the tuple $G = (V, E, C)$, where $V = \{1, \ldots, \mathcal{N}\}$ is the set of vertices, E is the set of edges, and $C = \{C_1, \ldots, C_k\}$ defines the set of controllers, where $C \subseteq V$. All solutions presented below require information about distances between the $\mathcal{N}$ network nodes, $d(i, j)$, in order to compute the shortest paths (SP) between them.

A. k-Median problem

This solution finds the controller placement C_k from the set of all possible controllers C that minimize the average propagation latencies between nodes and controllers, $d(v, C_k)$, using the Eq. (1).

$$L_{avg} = \frac{1}{\mathcal{N}} \sum_{v \in V} d(v, C_k) \tag{1}$$

K-Median starts by identifying the shortest link $d(i,j)$ in the network, selecting node k as a candidate to become a controller, $k = \{i,j\}$, which minimizes the average delay L_{avg} to the rest of the network. All nodes v for which $d(k,v) \leq \mathcal{D}_{req}$ are joined to the selected node k, creating a cluster. The next step is to check if a node exists in the cluster with better average delay than the node k, that satisfies the $\mathcal{D}_{req}$ for all nodes in the cluster. If a node $n \neq k$ exists that minimizes the L_{avg}, it is selected as a controller and the cluster is updated. Otherwise, the node k is selected as a controller. After that, the algorithm finds the nearest node to the cluster, repeating the process. This algorithm finishes when all nodes are included in a cluster.

B. k-Center problem

This approach finds the controllers so as to minimize the maximum distance of the nodes v to their closest controller C_k, defined by (Eq. (2)).

$$L_{k-center} = \max_{v \in V} \min_{C_k \in C} d(v, C_k) \tag{2}$$

At the beginning, this solution selects randomly a node k in the network, creating a cluster with all nodes for which $d(k,v) \leq \mathcal{D}_{req}$. If there exists a node n in the cluster that minimizes the delay to the nodes, it is selected as a controller and the cluster is updated. Otherwise, the node k is selected as a controller. After that, the algorithm finds the furthest node on average to the cluster, repeating the process until all nodes are included in a cluster.

V. K-Critical

In this section, the algorithm to solve the controller placement problem, k-Critical, is described in detail. This algorithm: i) minimizes the number of controllers required to cover a whole network and, ii) selects the controllers that allow homogeneous and robust management trees to be built. Here, we consider that a tree topology is robust if the loss of management data due to node or link failures is minimized. Moreover, it is homogeneous if the number of nodes along the different branches is balanced. The resulting control layer is a set of trees that satisfy the network requirements, in which the root nodes are the controllers. Before describing our algorithm, the idea behind tree robustness is explained.

A. Robust Tree

Network robustness is a property that will come from paths, which ensure that communication remains in spite of network failures, thus minimizing any data loss. Our objective is to focus on selecting the controllers that allow a robust control layer to be built.

Fig. 2 shows the resulting management trees obtained from the NSFNet topology. These trees are built from controllers found by k-Center, k-Critical and k-Median respectively. If we consider the worst case when a failure occurs in the network, which is when the adjacent link to the controller with highest number of nodes in the branch fails, we found that tree (a) loses information from 6 nodes, tree (b) from 4 nodes, and tree (c) from 7 nodes. So, as we can see in Fig. 2, the shortest and load balanced paths reduce the loss probability and bottlenecks in the tree.

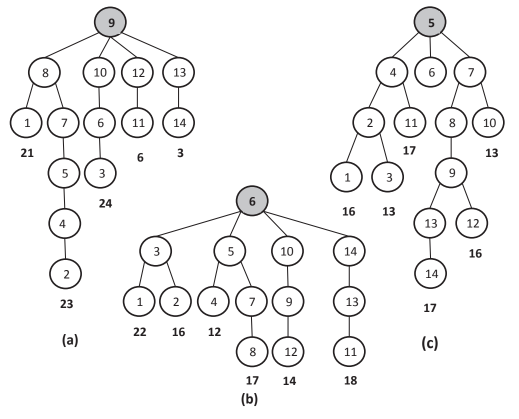


Fig. 2. Trees created from selected controllers for $\mathcal{D}_{req} = 24$ms. The number at the end of each branch is the delay in it.

B. K-Critical algorithm

In Algorithm 1, k-Critical is defined. The selection of the controller placement depends on the network requirements and the physical network characteristics. In order to select the best controller placement, the nodes characteristics are evaluated with respect to the network topology through function θ. This function evaluates the potential controller placement, which are called candidate nodes. The nodes that are candidates to become a controller are the nodes that, due to their location, satisfy the delay $\mathcal{D}_{req}$ to nodes that are not already managed by a controller. In this way, we guarantee that the branches of the resulting trees will have a maximum delay between nodes and the controller, limited by $\mathcal{D}_{req}$.

In the function θ, Eq. (3), the coefficient γ weighs the node connectivity and the path weight for each candidate node, defined as $\gamma = \frac{\mathcal{L}_h}{\mathcal{L}_{max}}$, where $0 \leq \gamma \leq 1$. It involves $\mathcal{L}_h$, defined as the maximum path length measured in number of hops for the candidate node, and $\mathcal{L}_{max}$, defined as the maximum path length found among the candidate nodes. Note that γ takes the lowest value for the candidate node with smallest diameter, and 1 for the candidates nodes with the largest diameter.

$$\theta = \gamma \times (\text{node connectivity}) + (1-\gamma) \times (\text{path weight}) \tag{3}$$

In Eq. 3 node connectivity relates the degree of the candidate node to the node distribution in its branches in i hops. The path weight relates the maximum delay value that the candidate node reaches when it is the root, with respect to the $\mathcal{D}_{req}$.

Considering the physical network diameter allows the selection of the candidate nodes with the best relation between the delay and the node distribution in its branches. The diameter of the control layer should be as small as possible given that longer paths mean longer transmission times and greater load on links. By evaluating the characteristics of traditional spanning trees (e.g., shortest path, fewest hops and minimum weight), it follows that shortest branches with optimal connectivity reduce the impact of the node failures, as shown in Fig. 2.

C. Load Distribution among controllers

In this approach, we discover the number of controllers that are necessary, given that both applications and networks have specific requirements (e.g., latency, delay, convergence time, etc) and that satisfying these requirements depend on both the number of controllers and their location in the network. In Algorithm 1 the information about distances is used to discover if one node alone is able to manage the whole network satisfying the required delay, $\mathcal{D}_{req}$. If this is not possible, a critical node is selected, $\mathcal{C}_n$. The critical node is the furthest on average from the rest of the nodes in the network. In consequence, the critical node restricts the controller selection. This criterion minimizes the number of controllers, and guarantees that all nodes in the network are managed by the nearest controller. Thus, the management of the whole network is guaranteed while satisfying the required delay $\mathcal{D}(v, \mathcal{C}_k) \leq \mathcal{D}_{req}$, where $\mathcal{D}(v, \mathcal{C}_k)$ is the delay from a controller $\mathcal{C}_k$, $\forall \mathcal{C}_k \in \mathcal{C}$, to a node v. It also limits the number of nodes to be managed for each controller.

The function θ is defined as follows:

$$\theta = \gamma \times \frac{\mathcal{D}_g}{N - \hbar_i} + (1 - \gamma) \times \frac{\mathcal{D}_{max}(v, \mathcal{C}_k)}{\mathcal{D}_{req}} \tag{4}$$

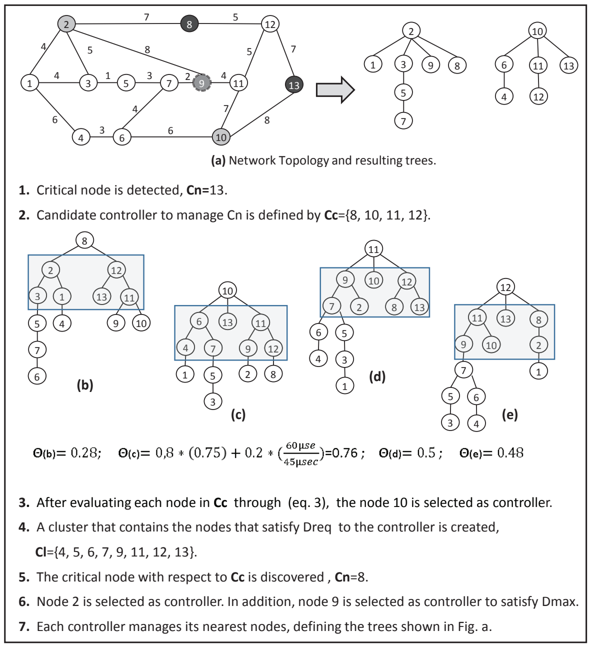
The first term of the θ function involves the number of nodes in the network, N; the number of nodes that a candidate node has covered at i hops, $\hbar_i$; and the node degree, $\mathcal{D}_g$. The second term of this function evaluates the maximum delay found from a node to the controller with respect to $\mathcal{D}_{req}$.

Consider the candidate nodes in Fig. 2 and assume that all these have the same diameter $\mathcal{L}_{max}$, so that the second term of θ function is annulled. By evaluating the θ function for these nodes, where N=14 y $\hbar_i = 2$, the following values were obtained: $\theta_a = \frac{4}{15-9} = 0.66$, $\theta_b = \frac{4}{15-10} = 0.8$, and $\theta_c = \frac{3}{15-7} = 0.37$. Note that even though the nodes 9 and 6 have the same degree, the node distribution in the tree created at node 6 is better and the highest function value is obtained. In this way, the controller placement with the best relation between the degree and the node distribution in its branches is discovered.

Algorithm 1 Controller selection algorithm k-Critical Node

```
Require: $(\mathcal{N} \times \mathcal{N})$ SP Delay Matrix. $\mathcal{D}_{req} \leftarrow$ Req. Delay
  $C \leftarrow$ Set of candidate nodes that satisfy $\mathcal{D}_{req}$
  if $\mathcal{C} \neq \emptyset$ then
    for each node $n \in \mathcal{C}$ to become a controller do
      Evaluate $\theta$
    end for
    Select the node $n \in C$ with the highest value in $\theta$; $\mathcal{C} = n$
  else
    while there are nodes not belonging to the cluster do
      $\mathcal{C}_n \leftarrow$ Find the furthest node to the Cluster
      $\mathcal{C}_k \leftarrow$ Find the set of candidates nodes to manage $\mathcal{C}_n$
      for each candidate node $n \in \mathcal{C}_k$ do
        Evaluate $\theta$
      end for
      Select the node $n$ with the highest value in $\theta$
      Cluster$_k \leftarrow$ Find from $(\mathcal{N} \times \mathcal{N})$ the nodes $v$ that satisfy
      $d(v, n) \leq \mathcal{D}_{req}$, where $v \notin$ Cluster
      Cluster $\leftarrow$ Cluster $\cup$ Cluster$_k$, $\mathcal{C} = \mathcal{C} \cup n$, Cluster$_k \leftarrow \emptyset$,
      $\mathcal{C}_n \leftarrow \emptyset$
    end while
  end if
```

We seek to construct trees that are wide near the controllers and narrow further away from them, where paths have as few hops as possible in order to reduce the data loss when nodes or links fail. The rationale is that failures in shorter paths result in a lower data loss, and nodes with higher degree provide a better distribution of nodes on branches (see Fig. 2.b). For this purpose, the θ function in Eq. (4) weighs the tree-depth of a node in the network in terms of hop count and the delay path by means of γ. An example is shown in Fig. 3.



1. Critical node is detected, **Cn**=13.
2. Candidate controller to manage Cn is defined by **Cc**={8, 10, 11, 12}.

Θ(b)= 0.28; Θ(c)= $0{,}8 * (0.75) + 0.2 * (\frac{60\mu se}{45\mu sec})$=0.76 ; Θ(d)= 0.5 ; Θ(e)= 0.48

3. After evaluating each node in **Cc** through (eq. 3), the node 10 is selected as controller.
4. A cluster that contains the nodes that satisfy Dreq to the controller is created, **Cl**={4, 5, 6, 7, 9, 11, 12, 13}.
5. The critical node with respect to **Cc** is discovered , **Cn**=8.
6. Node 2 is selected as controller. In addition, node 9 is selected as controller to satisfy Dmax.
7. Each controller manages its nearest nodes, defining the trees shown in Fig. a.

Fig. 3. Discovering the controllers in network (a) using k-Critical.

D. Connectivity among controllers

The set of controllers found by Algorithm 1 guarantees that every node is managed by at least one controller and that the maximum delay between them is no higher than $\mathcal{D}_{req}$. However, the delay among each pair of controllers, $\mathcal{D}_{cont}$, is not guaranteed. We consider that a minimum delay in the control layer must be guaranteed, that is, $\mathcal{D}_{req} + \mathcal{D}_{cont} \leq \mathcal{D}_{max}$. This delay, $\mathcal{D}_{max}$ may be defined based on the requirement of the applications or services to be provided by the network.

Algorithm (2) checks if the distance among the set of selected controllers defined as $C = \{C_1, ...C_k\}$ is satisfied. Otherwise, it finds the required controllers that satisfy the given constraint in the control layer.

To minimize the number of controllers to be added, the critical controller or the furthest controller on average to the set of controllers in C, must be identified. After that, the candidate nodes that can become controllers and that also minimize the distance to the other controllers are identified and evaluated by means of θ, selecting as controller the node with the highest value. This process continues until $\mathcal{D}_{cont}$ is satisfied for each pair of controllers.

In order to restrict the $\mathcal{D}_{cont}$ requirement, an initial condition was defined in Algorithm (2). This condition limits the $\mathcal{D}_{cont}$ in the control layer to $\frac{D_{max}}{k+1}$, where $\mathcal{D}_{max}$ is the maximum delay in the network. This condition, together with the criterion used to select the controllers, guarantees that the minimum number of controllers is added.

The idea behind this approach is to find intermediate controllers that allow network management information to be distributed to the controllers in C satisfying $\mathcal{D}_{req}$. Besides guaranteeing a $\mathcal{D}_{cont}$ in the control layer, adding intermediate controllers also creates a hierarchical layer. These controllers are the upper controller that maintain the network state managed by the lower ones, and therefore are responsible for updating network changes in the control layer satisfying $\mathcal{D}_{cont}$.

Algorithm 2 Controller selection

```
Require: $(\mathcal{N} \times \mathcal{N})$ SP Delay Matrix. $\mathcal{D}_{cont} \leftarrow$ Delay required
    among controllers. $\mathcal{D}_{max} \leftarrow$ Maximum delay in network.
    $C \leftarrow$ controllers previously selected $k \leftarrow$ Number of controllers
    in $C$.
if $\mathcal{D}_{cont} > \frac{D_{max}}{k+1}$ then
    $\mathcal{D}_{cont} = \frac{D_{max}}{k+1}$
end if
while $\mathcal{D}(C_i, C_j) \geq \mathcal{D}_{cont}$ do
    $\mathcal{C}_n \leftarrow$ Find the furthest node in the Controller cluster
    $\mathcal{C}_k \leftarrow$ Find the candidates nodes to manage $\mathcal{C}_n$ where $C_k \notin$ C
    and minimize the delay to $C$
    for each candidate node $n \in \mathcal{C}_k$ do
        Evaluate $\theta$
    end for
    Select the node $n$ with the highest value in $\theta$
    $C_{up} \leftarrow C \cup C_k$, $\mathcal{C}_n \leftarrow \emptyset$
end while
```

VI. Analysis and Results

In this section we show the implications of the controller selection in the control topology performance in terms of delay, data loss, node management distribution among controllers and average tree-depth. For this purpose, we randomly generated three network categories: sparse, medium and dense networks. These networks were generated using the software described in [14]. Then we evaluated the performance and robustness of the trees built at the controllers selected and for each one of the presented solutions for each one of the network categories. Nodes in sparse networks have between 1 and 10 neighbors; in medium networks between 20 and 30 neighbors, and in dense networks between 40 and 50 neighbors. For each category we generated 100 networks which consist of 100 nodes with the edge distance uniformly distributed between 1 and 10 km.

The controllers were found for different delay values in the range of microseconds. The results presented in this section are the average results obtained for each network category evaluated. MATLAB was used to evaluate the algorithms and their performance.

In Fig. 4, 5 and 6, the number of controllers found for each defined $\mathcal{D}_{req}$ in each network category is shown.

In order to compare the effect that the controllers selected have over control topology, a quantitative analysis to determine the number of optimal controllers was applied. Based on the maximum delay time reached for each network category, we found that the optimal number of controllers is the one that has the best cost/benefit relation. In other words, it is the maximum number of controllers for which adding another controller results in negligible delay reduction.

For sparse networks (Fig. 4), a significant delay range –from $23\mu s$ to $60\mu s$– is covered when 5 controllers are considered. This is the same number of controllers for all methods. The benefit of using 5 controllers is a considerable delay reduction in comparison with using fewer controllers. The maximum delay ($60\mu s$) is reduced by more than a third. We consider that this is an appreciable delay reduction, given that this network category has a low connectivity. Consequently, 5 controllers are considered as optimal.

Fig. 5 shows the number of controllers found for a specific delay range in medium connectivity networks. As can be seen, the maximum delay reached ($35\mu s$) is significantly lower than for sparse networks (approximately a half). This is a logical result, because networks with high connectivity have a small diameter. For this network category we found that the delay decreases when from 1 to 3 controllers are considered. But just considering 3 controllers halved the maximum delay time ($17\mu s$) for all methods. Therefore, we consider that the optimal number of controllers to manage

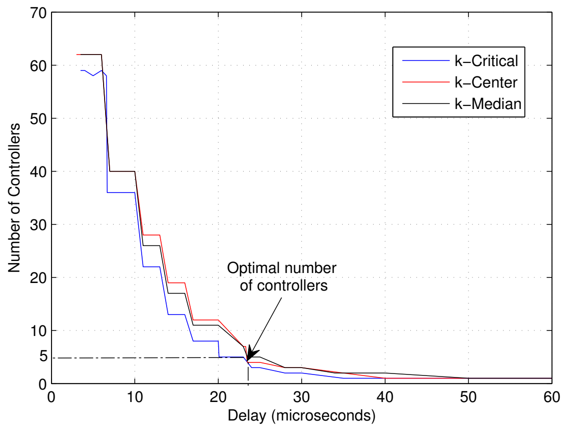


Fig. 4. Number of controllers for all possible delay ranges in generated networks with sparse connectivity .

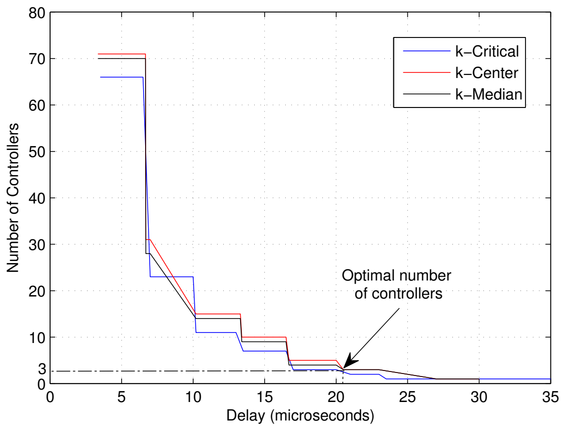


Fig. 5. Number of controllers for all possible delay ranges in generated networks with medium connectivity.

this network category is 3.

As was to be expected, for dense networks the maximum delay in a tree is reduced ($27\mu s$) (Fig. 6) in comparison with the maximum delay for the medium network category. Fig. 6 shows that there are two points for which the delay reduction is significant, i.e., 1 and 5 controllers. As can be seen (Fig. 6) when only one controller manages the network, trees with a $\mathcal{D}_{req}$ between $20\mu s$ and $26\mu s$ can be built. Note that the delay reduction is negligible when 2, 3 or 4 controllers are considered compared with using just 1. A significant delay reduction is reached when 5 controllers manage the network, which reduces the maximum delay by half. In order to select the optimal number of controllers for dense networks, we have taken as a reference the delay reached for the optimal number of controllers selected for medium and sparse network categories, $17\mu s$ and $23\mu s$, respectively. If we consider this delay interval as optimal ($17\mu s$ to $23\mu s$), we can say that just 1 controller is a good selection in terms of delay for dense networks, since trees with $20\mu s$ can be built. Nevertheless, using just 1 controller implies that it has to handle the information from all the network, which may be inefficient. We consider that this is not a problem, since some approaches may be considered for processing information efficiently (e.g., parallel processing) when just 1 controller has to manage the whole network.

The presented results indicate that using more controllers than the optimal number or using a poor controller location, reduces slightly the delay compared with the cost of adding more controllers.

A. Expected data loss

We consider that in tree topology networks the failure of a node is equivalent to the failure of the link to its parent. This is true for every node in the tree except the controllers, which we assume will not fail. This is because

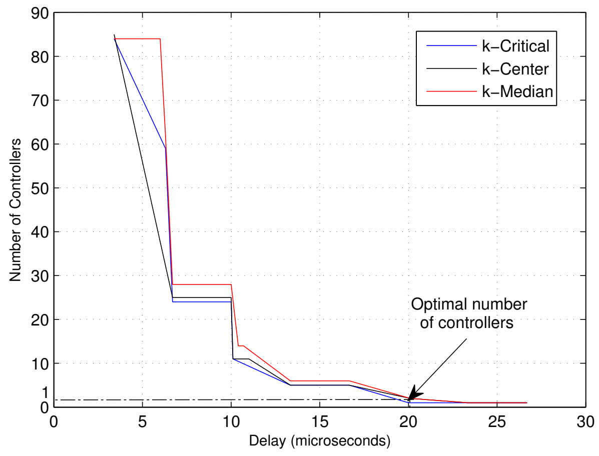


Fig. 6. Number of controllers for all possible delay ranges in generated networks with high connectivity.

independently of the network topology, if the controller fails, the control over the managed nodes is lost while the control is reassigned. Note that the expected data loss d_i for a node i depends on the number of nodes n_i that are rooted at it, and on the probability that the node i fails. If we consider that every node in the tree topology T_k has the same loss probability, the expected data loss d_i can be calculated by Eq. (5):

$$d_i = \frac{1}{N-1} \sum_{i \in V(T_k)} n_i, \tag{5}$$

where N is the number of nodes in the network and $V(T_k)$ are the nodes in the tree T_k.

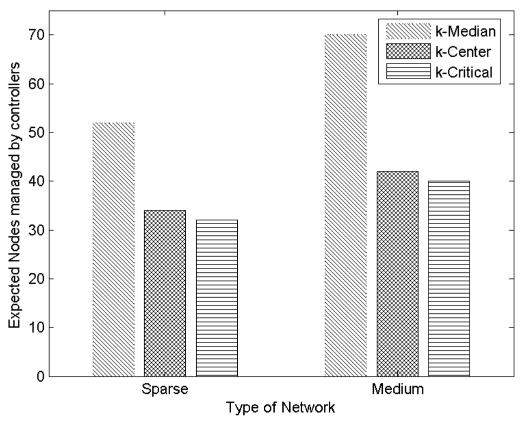


Fig. 7. Average nodes managed by controller.

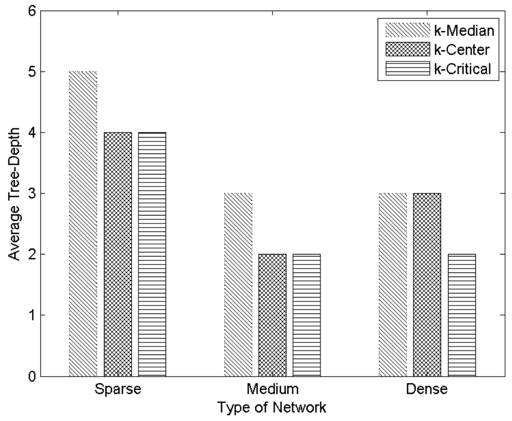


Fig. 8. Average depth on control topology.

B. Analysis

In this section we compare the characteristics of the tree topologies built from controllers selected for each one of the presented methods. This analysis is done for the optimal number of controllers found for each network category. The maximum number of nodes managed by the controllers selected from each method is shown in Fig. 7. The maximum number of nodes managed by controllers are averaged over the 100 random graphs in the respective categories. Fig. 7 shows that for sparse and medium connectivity networks the controllers selected by k-Critical have better node distribution than the other solutions. The results obtained for dense networks is not shown, because just 1 controller manages the whole network for all cases.

Fig. 8 shows the average tree-depth obtained from controllers selected for each method. We observe from Fig. 7 and Fig. 8 that the node distribution among controllers affects the tree-depth. For instance, the trees created from the controllers found by k-Median have the worst node distribution, and as a consequence have the longest paths. For the dense network category, when only 1 controller is required, k-Critical has the best node distribution among branches, and consequently the resulting trees are the shortest ones.

Fig. 9 shows the expected data loss on trees built from the selected controllers. For each network category, the data loss parameter is computed using Eq. (5). From the obtained results we can see clearly the relation among node distribution, tree-depth and data loss. The shortest trees have the lowest data loss, which is the case for trees created from the controllers selected by k-Critical. On the other hand, the longest trees have the highest data loss, as shown in the case of k-Median in Fig. 9.

Fig. 10 shows the maximum delay of the trees obtained for each network category. As shown, k-Critical improves the robustness for networks with high connectivity but, as a consequence, the delay of the branch cannot be improved in comparison with other methods. However, it is $\leq \mathcal{D}_{req}$. Thus, k-Critical improves the tree robustness at the expense of delay.

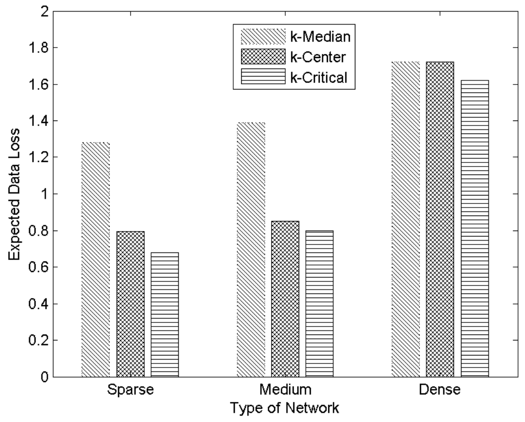


Fig. 9. Expected data loss on randomly generated networks.

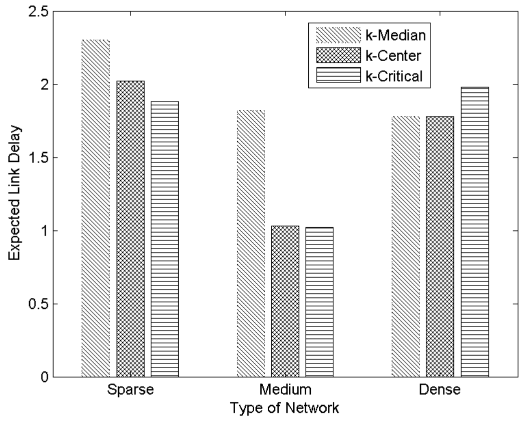


Fig. 10. Expected link delay on control topology.

Even though the trees created from controller placement by k-Center and k-Critical have similar characteristics in terms of tree-depth and nodes managed by controllers, the resulting trees built at controllers selected by k-Critical reduces considerably the data loss for all the network categories, as can be seen in Fig. 9. K-Center has good results on average, but not as good as k-Critical. This is because k-Center does not have any initial criteria to select the potential candidate nodes to become controller.

As shown, k-Critical makes the best controller selection in general. The good results obtained using k-Critical are due to the criterion to select the first controller. This selection depends on the worst node in the network in terms of delay, the critical node. On the other hand, k-Median does not obtain a good performance because the initial criteria to select the controller is not efficient.

The computational complexity for each one of the controller placement solutions evaluated is $O(nk)$. In general, finding a controller requires checking the shortest distances of the n nodes in the network. Therefore, the computational complexity for finding one controller is defined by $O(n)$. This process has to be repeated until k controllers are discovered.

VII. Conclusions

Previous work has shown the implications of the placement controller problem in some network parameters. However, these studies have not taken into consideration the load distribution among controllers, nor have they focused on the control network topology built from controllers. In this work we have compared some controller placement solutions and have evaluated the implications of controller selection in the resulting control layer. We have demonstrated that the number of controllers selected, as well as their location, determine the network performance. As can be seen from the results obtained, a poor controller selection can affect considerably the control network robustness,which affects the network operation (e.g., long recovery time after failures). On the other hand, using more than the optimal number of controllers can be inefficient and costly, because the delay improvement is negligible. From the results obtained, we can see that the optimal number of controllers depends on the physical network characteristics and the network requirements. We have found that the criterion used to select the first controller is crucial in defining the control layer topology and therefore its performance. A good controller selection allows the network to balance load and respond to events in the shortest time possible.

We show that k-Critical selects the controllers by constructing a robust control layer in response to network disturbances. This is because the physical characteristics of the network are taken into account when selecting the controllers. This solution can also be used to find the best controller location when a network needs to be extended or the load needs to be reallocated. We consider that a scalable control layer design combined with efficient management strategies can exploit the benefits of SDN.

Turning our attention to future work, we are interested in defining how the tree topology is built from the selected controllers taking into account several network performance metrics. This is because selecting the shortest paths between controller-nodes is not the best option for improving the load and the robustness of the control layer. In addition, a mechanism for load migration among the controllers must be defined.

References

[1] S. Yeganeh, and Y. Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications", *proceedings of the first workshop on Hot topics in software defined networks ACM SIGCOMM 2010 conference (HotSDN '12)*, pp. 19-24, 2010.

[2] T. Koponen, M. Casado, N. Gude, and et al. "Onix: A Distributed Control Platform for Large-scale Production Networks,", *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI'10)*, USENIX 2010, pp.1-6, 2010.

[3] M. Reitblatt, N.Forter, J. Rexford and et al. "Consistent updates for software-defined networks: change you can believe in!", *The 10th ACM Workshop on Hot Topics in Networks (HotNets)*, ACM New York, ISBN:978-1-4503-1059-8, USA, Nov. 2011.

[4] Y. Jiménez, C. Cervelló-Pastor, and J. García. "Defining a Network Management Architecture", *the 21st IEEE International Conference on Network Protocols proceedings, PhD Forum*, Germany, October 7-11, 2013.

[5] A. Tootoonchian and Y. Ganjali. "Hyperflow: a distributed control plane for openflow", *Proceedings of the 2010 internet network management conference on Research on enterprise networking (INM)*, pages 3, 2010.

[6] M. Yu, M. J. Freedman, and J. Wang. "Scalable flow-based networking with DIFANE", *proceedings of the ACM SIGCOMM 2010 conference*, pp. 351-362, 2010.

[7] S. Yeganeh, A. Tootoonchian, and Y. Ganjali. "On Scalability Software-Defined Networking,", *IEEE Communication Magazine*, vol.51 , Issue 2, pp. 136 - 141, ISSN 0163-6804, Feb. 013.

[8] B. Heller, R. Sherwood, N. McKeown. "The Control Placement Problem", *The first workshop on Hot topics in Software Defined Networks (HotSDN '12).*, ACM New York, ISBN: 978-1-4503-1477-0, USA 2012.

[9] Y. Hu, W. Wang, X. Gong and et al. "On placement of controllers in software-defined networks", *The Journal of China Universities of Post and Telecommunications*, vol. 19, no 2, pp. 92-97, Oct. 2012.

[10] S. Schmid and J. Suomela. "Exploiting Locality in Distributed SDN Control", *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Hong Kong, China, Aug. 2013.

[11] K. Nguyen, Q. Tran Minh, and S. Yamada "A Software-Defined Networking approach for Disaster-Resilient WANs", *22nd International Conference on Computer Communications and Networks (ICCCN).*, ICCCN2013, ISBN: 978-1-4673-5774-6, Nassau, Bahamas, 2013.

[12] A. Tootoonchian, Y. Ganjali, M. Casado, and et al. "On controller Performance in Software-Defined Networks,", In USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE). April. 2012.

[13] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann "Logically centralized?: state distribution trade-offs in software defined networks", *Proceedings of the first workshop on Hot topics in software defined networks (HotSDN '12).*, ACM New York, pp. 1-6. ISBN: 978-1-4503-1477-0, New York, USA, 2012.

[14] M. Bastian, S. Heymann, and M. Jacomy "Gephi: an open source software for exploring and manipulating networks", *Proceedings of International AAAI Conference on Weblogs and Social Media (ICWSM).*, pp. 361-362. California, USA, 2009.