Profiling Software Defined Networks for Dynamic Distributed-Controller Provisioning

Diogo M. F. Mattos*†, Otto Carlos M. B. Duarte*, and Guy Pujolle†
*Grupo de Teleinformática e Automação - Universidade Federal do Rio de Janeiro (COPPE/UFRJ)
Rio de Janeiro, Brazil - Email:{diogo,otto}@gta.ufrj.br
†Laboratoire d'Informatique de Paris 6 - Sorbonne Universities, UPMC Univ Paris 06
Paris, France - Email: {Diogo.Mattos,Guy.Pujolle}@lip6.fr

***Abstract*—Software Defined Networking decouples packet-forwarding from logically centralized network control. Conceiving network control as load-aware responsive distributed system remains a key challenge. In this paper, we propose a scheme for dynamically provisioning and placing network controllers based on their usage profiles. The proposed scheme analyze the amount of calls each controller is handling at the same time, and uses a Markov chain to identify if the network load is increasing, decreasing, or stable. Our controller placement algorithm chooses the most central nodes for installing the controllers. We simulate our proposal over real network topologies for a variable network load. We compare our proposal with a centralized-controller approach and with an approach where all nodes are also controllers. Our results show that the proposal achieves an efficient controller placement and provides as many controllers as needed to respond to the demand. The proposed scheme installs 50% controllers less than the distributed approach, while keeps the minimum load over each controller.**

I. Introduction

The main concept of Software Defined Networking (SDN) is to decouple the hardware realization of the packet-forwarding from the software realization of the logically centralized control [1], [2]. The software realization of the control is usually proposed as a distributed system that provides the abstraction of a global network view [3], [4]. Distributing the SDN control enhances the availability, the performance, and the scalability of the network, as multiple coordinated nodes act as a network controller [5]. Nevertheless, changes on the network load directly imply changes on the best placement for the controllers and on the required capacity for each controller.

The variation of the network load may increase the number of requests addressed to a specific controller on the network, causing long flow setup delays, while other available controllers remain idle. It occurs because a set of nearby switches are usually controlled by the same single controller instance [6], [7], [8], clustering the network load into one single controller. Thus, an efficient controller placement on SDN has to take into account the variable load distribution on the network and has to adapt the resulting software control according to the current network conditions.

In this paper, we propose a scheme for dynamically provisioning controllers for Software Defined Networking with a distributed control plane. Our proposal creates a profile of the controller usage through sampling the number of requests each controller receives. We classify the controller behavior into one of three states: increasing, decreasing, or stable flow setup request load. The state transitions are handled as a Markov chain, which enable our scheme to calculate the most probable future state. Thus, if the network load is increasing, the proposed scheme instantiates a new controller. If it is decreasing, a controller should be shut down. A stable network load means that the current set of controllers is able to promptly respond to the demand. We also propose a greedy algorithm that installs controllers on the network according to the betweenness centrality and the estimated capacity of each node.

The controller placement is usually handled as an optimization problem that only depends on the network topology and on the resources of the nodes [9], [5], [10], [11]. The network load, however, impacts on the need of controllers, and also on the best location of each controller [6], [12]. The previous approaches for dynamically providing controllers on SDN rely on monitoring the controllers' resources while they are on operation. In turn, our proposal keeps it simpler as it only monitors the number of requests a controller receives and handles at a time. This approach is simple and hardware agnostic, since it considers just the network load and not the resource usage. Moreover, we also propose a placement algorithm, which is able to relocate the controllers to a more suitable location. We simulate our proposal over real network topologies. The results show that our proposal instantiates at most 50% of the number of controllers when compared to a fully distributed scenario, and we also decrease by 13x the average usage of a controller when compared to a centralized-control approach. Moreover, our solution achieves the same average load as a fully distributed approach.

The rest of the paper is organized as follows. The Section II presents the related work. The problem of dynamically provisioning network controllers is stated in Section III. Section IV proposes our provisioning scheme as well as our dynamic placement algorithm. The simulation results are discussed in Section V. Section VI concludes the paper.

II. Related Work

Control centralization on Software Defined Networking hampers the security, the performance, and the scalability of the network [5]. Thus, several works propose to distribute the SDN control over dedicated distributed servers that abstract the global network view. HyperFlow proposes a distributed filesystem, implemented through Publisher/Subscriber channels, in which each controller instance registers a network

This work was sponsored by CNPq, CAPES, and FAPERJ.

978–1–5090–4671–3/16/$31.00

event that is propagated to the others [4]. The Kandoo proposal hierarchally organizes the controller instances and divides the network applications into two categories, root and local [8]. The proposal argues that local applications can run next to switches, while root applications run on a root controller, top of the hierarchy. Mattos *et al.* propose the idea of a designed controller that is responsible for keeping the strong consistence of all changes over the global network view [5]. The proposal also argues that the control distribution should consider both the controller design and placement.

Heller *et al.* firstly defined the controller placement problem [9]. They argue that the suitable number of controllers and their location on the network are essential features to achieve a good performance of the network control plane. Müller *et al.* propose a controller placement algorithm that enhances the resilience of the network [10]. Zang *et al.* compare different approaches for providing resilience while placing controllers on network architectures, in which control and data plane are decoupled [11]. Lange *et al.* argues that to achieve network resilience, the controller placement should also consider other metrics and, thus, they propose Pareto-based heuristics to decide which placement better fits to a network topology [13]. Hu *et al.* introduce a reliability estimator that is a metric to evaluate the resilience of the network. They propose a linear programming approach, based on their estimator, to solve the controller placement problem [14].

Bari *et al.* introduce the idea of a dynamic controller provisioning on SDN [6]. Their main idea is to instantiate new controllers as the network load increases. The decision algorithm is based on monitoring controller resources. Similarly, Dixit *et al.* propose the idea of a controller pool, which can increase and shrink according to traffic conditions [12].

III. The Dynamic Controller Provisioning

The dynamic provisioning of controllers on SDN deals with defining how many controllers are suitable to answer the network demand for control. It worths noting that the performance of the control plane directly impacts on the data plane, as the greater is the load in network controllers, the longer is the flow setup time and the lower is the control plane throughput. Therefore, the control plane should expand and shrink according to the traffic on the network. Besides, the location of each controller should be guided by the traffic pattern on the network, as a controller that is not placed next to requesting switches also increases the flow setup time. Hence, the dynamic provisioning of controllers is composed of two sub-problems: defining the ideal number of controllers to a given network load, and placing the controllers on suitable network nodes. The two sub-problems are a generalization of the controller placement problem [9]. Nevertheless, the main constraint of the dynamic provisioning is to execute an online optimization that considers the traffic load and the current network topology as inputs. In this sense, the dynamic provisioning problem intrinsically requires networking monitoring, which is not assumed by the placement problem. Besides, the monitoring activity may focus on the controller resource usage or on the network activity.

On the one hand, monitoring controller resource usage allows providing an elastic pool of controllers, in which a new controller is added as soon as a lack of resources is detected, or a controller is removed as it detects underused controllers. The main drawback of this approach is that it is based on the controller behavior, which may introduce a late reaction to changes on the network traffic pattern. On the other hand, monitoring the network activity allows forecasting the augmentation or the reduction of new flow arrivals on the network. Moreover, provisioning controllers based on the network activity decouples the network-control decision of adding or removing a controller from infrastructure data statistics, as monitoring the resource consumption of each controller.

IV. The Proposed Dynamic Scheme

The proposed controller-provisioning scheme is divided into three main components: network usage profile, Markov chain model, and controller placement. All controllers are timely queried by a centralized orchestrator about their statistics of received and handled requests. We consider as requests all events on a controller that sign the arrival of new flows on the network [1].

A. Network Usage Profile

Each controller keeps the count of the number of requests it has received since the last time it was polled by the orchestrator. The orchestrator collects the statistics for each controller. The orchestrator keeps a sliding window data structure of size w for each controller. As a new value is added to the sliding window, the oldest value is discarded. The orchestrator polls each controller every t seconds. It is worth noting that the values of w and t are set for each network and are tuned according to the responsiveness that is expected from the scheme. Moreover, it is also important to highlight that the sliding window of one controller is not necessarily related to the others. They may not be strictly synchronized, which is not a problem because the state changes are independent and locally calculated for each controller.

B. Markov Chain Model

We consider that a controller may be into one of the three states: increasing load (S_0), stable load (S_1), or decreasing load (S_2). Hence, we consider a matrix of dimension 3×3 that describes the state changing for a controller. This matrix represents the probability of controller changing from the state S_i to the state S_j, where i represents the row index, and j, the column. We consider that a state change always occurs between two followed values in a sliding window. As each sliding window, for each controller, has w samples, thus, we calculate $w - 1$ states. Indeed, we iterate over the sliding window and if the previous value is less than the current one, we consider that the controller went to a state of increasing load, if both values are equal, stable load, and if the previous is greater than the current one, decreasing load state. Based on the state calculation, we achieve a $w - 1$ vector of states. Iterating over the state vector, we have $w-2$ state changes that are the data source for filling the Markov probability matrix.

[1] In the case of using OpenFlow as Southbound API, we consider the `PacketIn` events.

Then, we calculate the right-hand eigenvector of the Markov probability matrix,

$$\begin{bmatrix} S_0 \to S_0 & S_0 \to S_1 & S_0 \to S_2 \\ S_1 \to S_1 & S_0 \to S_1 & S_1 \to S_2 \\ S_2 \to S_2 & S_1 \to S_1 & S_2 \to S_2 \end{bmatrix} \times \begin{bmatrix} S'_0 \\ S'_1 \\ S'_2 \end{bmatrix} = \lambda \begin{bmatrix} S'_0 \\ S'_1 \\ S'_2 \end{bmatrix}, \quad (1)$$

where $S_i \to S_j$ indicates the probability of changing from state S_i to S_j, and λ is the corresponding eigenvalue. Choosing $\lambda = 1$, we have the eigenvector $\vec{P_c}$ that represents the invariant probabilities of the states S'_0, S'_1, and S'_2 of the controller c. $\vec{P_c}$ is used in the controller placement component to decide whether is needed to change the controller assignment for each switch.

C. Controller Placement

The controller placement component runs two algorithms. The first one is to decide whether to add or to remove a controller on the network. The second one decides in which available network node a new controller should be instantiated.

Let C_n be the set of all controllers currently on the network. To decide how many controllers should be started on the network, the algorithm receives as initial input the current number of controllers, $K_n = |C_n|$. Then, it calculates the new number of controllers on the network K_{n+1}:

- summing one to K_n, for each $\vec{P_c} \mid c \in C_n$, where $S'_0 > 0.5$;
- subtracting one from K_n, for each $\vec{P_c} \mid c \in C_n$, where $S'_2 > 0.5$.

After that, K_{n+1} stores the number of controllers that should be active on the network. The K_{n+1} controllers are placed on the network on the K_{n+1} nodes that have the greatest betweeness centrality on the known network topology[2] [15]. The set C_{n+1} contains the selected nodes.

The mapping procedure between switches and controllers are shown in Algorithm 1, where T is the graph of the network topology. The function $SortedListOfClosestNodes(T, c)$ returns a list containing all nodes sorted by the distance between c and the node in the topology T. Function $randomChoice(X)$ chooses a random element of the set X. It is worth to highlight that mapping switches into controllers is a NP-hard problem, because it traces back to the *facility location problem*[13]. Therefore, our algorithm is a simple heuristic that allocates first the switches to the less central nodes (lines 3-17). This behavior assures that the most central controllers still be available for the nodes that are not easy to connect with the rest of the network.

The proposed scheme estimates the capacity of each already installed controller. To do so, the number of switches controlled by a controller in the step $n+1$ is the number of switches it controlled in step n weighted by $1 - S'^c_0$. The value $1 - S'^c_0$ estimates the probability of a controller c not being overload. If the capacity estimation calculates that the currently installed controllers can respond to more than the total amount of switches, some controllers are deactivated. For calculating the number of switches a recently added controller should control, we consider that all new controllers equally share the not assigned switches (lines 19 – 21). If it still has any switch that is not assigned to any controller, we randomly chose a controller (lines 22-24). The algorithm returns the map that implements a map data structure (<key, value>), which relates the key switch with its value controller.

[2]We consider that all nodes on the network are able to run a controller.

Algorithm 1 Algorithm for mapping switches into the most suitable controller.

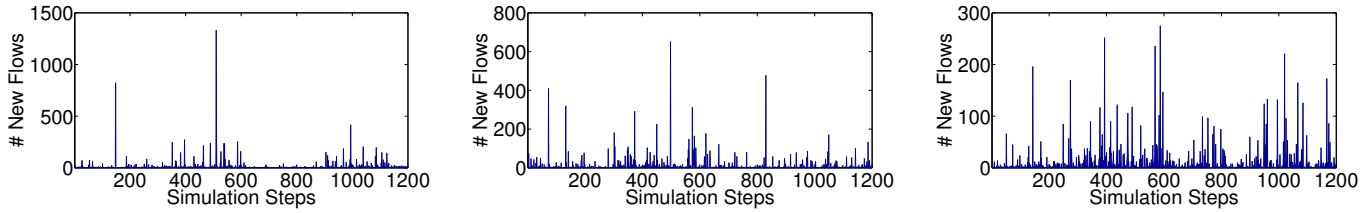
Input: C_n, C_{n+1}, and T
Output: map

```
1:  map = ∅
2:  Sort C_{n+1} on the reverse order of betweeness centrality
3:  for c ∈ C_{n+1} do
4:    if (c ∈ C_n) then
5:      k^c_{n+1} = 1 − S'_0(c) × k^c_n
6:      closestNodes = SortedListOfClosestNodes(T, c)
7:      added = 0
8:      i = 0
9:      while i < len(closestNodes) ≠ ∅ ∧ added < k^c_{n+1}
        do
10:       if (closestNodes[i] ∉ map) then
11:         map[i] = c
12:         added+ = 1
13:       end if
14:       i+ = 1
15:     end while
16:   end if
17: end for
18: for t ∈ T do
19:   if t ∉ map then
20:     map[t] = randomChoice(C_{n+1} − C_n)
21:   end if
22:   if t ∉ map then
23:     map[t] = randomChoice(C_{n+1})
24:   end if
25: end for
26: return map
```

Algorithm 1 also acts as a optimization mechanism that bans badly located controllers from the controller pool. This behavior happens because of the selection of the best-located controllers, according to our heuristic, on selecting the K_{n+1} nodes with the highest betweeness centrality. Selecting the most central nodes in the network is a simple way of fairly sharing the network load among all controller nodes [15].

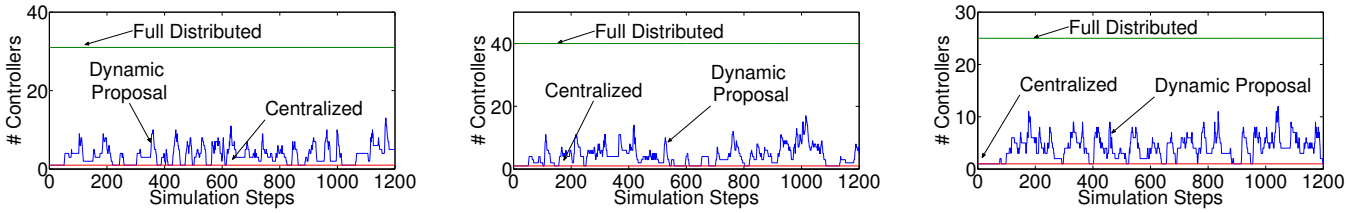
V. The Evaluation of the Proposed Scheme

The evaluation of the proposed scheme is performed by simulation. Our simulator is an extension of the SDN simulator previously developed by Mattos *et al.* [1]. The main idea of this simulator is to implement the formal SDN model proposed by Reitblatt *et al.* [16], adapting the original model into an event-driven simulation. The simulator was extended to support multiple controllers, as well as we also implemented the proposed dynamic controller provision scheme as a module of the simulator. The SDN simulator is written in Python language. We simulate three different real network topologies.



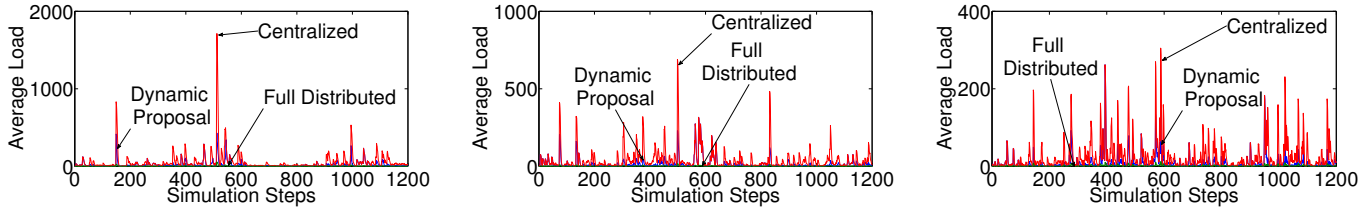
(a) RNP Topology in Brazil. (b) GEANT Topology in EU. (c) AT&T Topology in USA.

Figure 1. Network load simulation in each topology. New flows arrive at random nodes in the topology, following a log-normal distribution for the inter-arrival time ($\mu \in [6, 7]$).



(a) RNP Topology in Brazil. (b) GEANT Topology in EU. (c) AT&T Topology in USA.

Figure 2. Comparison of the instantiated number of controllers: the proposed dynamic controller-provisioning scheme with the centralized and the full distributed approaches. Our proposal instantiates more controllers as there are peak arrivals of flows on the network.



(a) RNP Topology in Brazil. (b) GEANT Topology in EU. (c) AT&T Topology in USA.

Figure 3. Comparison of the average load on the controllers: our dynamic controller provisioning scheme with the centralized and the full distributed approaches. Our proposal presents the same average load on the controllers as the full distributed approach, but our proposal uses half of the controllers.

All used topologies are available at The Internet Topology Zoo[3]. The topologies are from the RNP (National Research Network) in Brazil, with 30 nodes; the GEANT network in Europe, with 40 nodes; and the AT&T MPLS network in the U.S., with 25 nodes.

The network load was simulated as flows arriving to the network. Each flow is modeled as a stream of packets that lasts for 50 simulation steps on the network. The source and the destination of the flows are randomly chosen on the network. The inter-arrival time between new flows follows a log-normal distribution. The average of the inter-arrival distribution varies between 6 and 7 ($\mu \in [6, 7]$) and the standard deviation equals to 2 ($\sigma = 2$) [10]. The arrival of new flows happens during 1200 simulation steps and the simulation ends when all port queues of all switches are already empty. The network load for each topology is shown in Figure 1. The network load is modeled as follows. On the first 600 simulation steps, the average of inter-arrival time is set to $\mu = 6$. On the next following 300 steps, the network load decreases, as the average inter-arrival time increases, $\mu = 7$. Finally, at the last part of the simulation, the network load increases again, having the average inter-arrival time set to $\mu = 6.5$. The changing average of the flow inter-arrival time simulates the changes on the network load, such as the variation of the load that a network experiences during a day.

In the proposed scheme, the monitoring interval is defined as a simulation step and the sliding window is implemented as the last 50 measures of the controller load profile. The optimization algorithm runs after every 10 simulation steps.

Our simulations compare three network control architecture. The first control architecture is a centralized control, labeled as `Centralized`, in which the network counts with just one single controller. The second control architecture is the full distributed architecture, labeled as `Full Distributed`, in which all nodes act as both as a forwarding device and as a controller. In this architecture each node hosts its own controller. The third architecture is our proposed scheme, labeled as `Dynamic Proposal`, in which the number of controllers and the mapping between switch and controller are dynamic. The main idea of our evaluation is to compare our proposal with the two extreme cases. The centralized scenario stands for the scenario with the minimum number of controller nodes, just one. The full distributed scenario stands for the case that the average load on the controller is minimal, but the number of controllers is maximal.

Our first experiment measures the number of instantiated controllers in each topology by each control approach. It is worth to highlight that our proposed scheme varies the number of installed controllers from one up to half of the network

[3]Available at http://www.topology-zoo.org/.

nodes, shown in Figure 2. This result was the same in all topologies. When there is a utilization peak, the proposed scheme instantiates the greatest number of controllers to respond to the demand, up to 50% of the network nodes. After the utilization peak, the number of instantiated controllers is reduced up to one single controller, when the load is minimal on the network.

Our second experiment evaluates the average load on the controllers for each control approach, shown Figure 3. The results reveal that our scheme achieves a similar average load on the controllers as the full distributed control. The proposed scheme, however, reduces at up to one the number of active controllers. Comparing the case of the RNP topology, we observe on Figures 1(a), 2(a) and 3(a) that, although the proposed scheme instantiates a maximum number of 15 controllers, our proposal always keeps a minimum average controller usage, similarly to the full distributed approach. The average load achieved by our proposal is up to 13x lower than the centralized approach. Comparing the behavior of the proposed scheme on the AT&T topology, Figures 1(c), 2(c) and 3(c), it is possible to check that the number of instantiated controllers changes in function of the variation of the network load, while the proposal keeps a fair share of the load between all instantiated controllers. The GEANT network case shows the greatest number of changes on the number of installed controllers, Figures 1(b), 2(b) and 3(b). The network load is approximately the same in all topologies, as in all cases the number of arriving flows is almost the same. In the GEANT network, as it has the greatest number of nodes, it is the topology where the load is the most distributed. Therefore, our scheme is always changing the controller placement to respond to a change in the pattern of the network-traffic location.

It is worth noting that dynamic controller provisioning is not enough to ensure the fair share of the load. The results shown in Figure 3 reveal that the proposed dynamic placement heuristic finds the best location for instantiating the new controllers, as well as it also acts to optimize the location of previous controllers. If the network is initialized with a not well located controller, as the controller pool increases and shrinks, the badly located controllers are banned of the controller pool. As an effect, after some time, the chosen network controllers are just the best located controllers.

VI. Conclusion

Dynamically provisioning network controllers for Software Defined Networking on demand is important to enhance the network responsiveness. The provisioning scheme has to reason about the suitable number of controllers and the appropriate location of them according to the network load. We proposed an efficient controller-provisioning scheme that profiles the number of network requests on each controller and forecasts overloads and underloads on the network. The load forecasting is achieved by a Markov chain, which calculates the probability of a controller being in an overload or an underload state. The ideal number of switches controlled by each controller is derived from the probability of the controller being in an overload state. Moreover, we introduced a simple placement algorithm that instantiates a new controller into the available node that has the highest betweeness centrality on the network. We simulated our proposal into different real network topologies. Our results show that we achieve a reduction into the number of controllers in use, instantiating only one single controller when there is a minimal load on the network, as well as we reduce up to 13x the average controller overload when compared to a centralized controller.

References

[1] D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "Reverse update: A consistent policy update scheme for software-defined networking," *IEEE Communications Letters*, vol. 20, no. 5, pp. 886–889, May 2016.

[2] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the First workshop on Hot topics in software defined networks*, ser. HotSDN'12. Helsinki, Finland: ACM, 2012.

[3] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'14. New York, NY, USA: ACM, 2014, pp. 1–6.

[4] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3.

[5] D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "A resilient distributed controller for software defined networking," in *IEEE ICC 2016 - Next Generation Networking and Internet Symposium (ICC'16 - NGN)*, Kuala Lumpur, Malaysia, May 2016.

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

[7] X. Li, P. Djukic, and H. Zhang, "Zoning for hierarchical network optimization in software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[8] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12. New York, NY, USA: ACM, 2012, pp. 19–24.

[9] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12. New York, NY, USA: ACM, 2012, pp. 7–12.

[10] L. F. Muller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: an enhanced controller placement strategy for improving sdn survivability," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, Austin, Texas, USA, Dec. 2014.

[11] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec. 2011, pp. 1–6.

[12] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'13. New York, NY, USA: ACM, 2013, pp. 7–12.

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

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

[15] M. Andreoni Lopez, D. M. F. Mattos, and O. C. M. B. Duarte, "An elastic intrusion detection system for software networks," *Annals of Telecommunications*, pp. 1–11, 2016.

[16] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of the ACM SIGCOMM 2012*. New York, USA: ACM, 2012, pp. 323–334.