A Learning Automaton-Based Controller Placement Algorithm for Software-Defined Networks

Habib Mostafaei*, Michael Menth†, and Mohammad S. Obaidat‡§¶, Fellow of IEEE
*Department of Engineering, Roma Tre University, Italy
†Department of Computer Science, University of Tuebingen, Germany
‡ECE Department, Nazarbayev University, Astana, Kazakhstan, KASIT,
§University of Jordan, Amman, Jordan, and ¶University of Science and Technology Beijing (USTB), Beijing, China
Email: mostafae@dia.uniroma3.it, menth@uni-tuebingen.de, m.s.obaidat@ieee.org

***Abstract*—Software-defined networking (SDN) moves the control plane of network devices like switches and routers to the controller. The controller is in charge of managing the whole network through application programming interfaces (APIs). Fault tolerance in the SDN networks can be handled by leveraging multiple controllers. Placing controllers in an SDN network can be seen as facility location problem which is an NP-hard problem. In this paper, we propose a simple heuristic algorithm for controller placement in SDN networks leveraging a learning automaton (LA) approach. The proposed algorithm can place the controllers based on a predefined propagation latency between the controllers and the switches while minimizing the overall propagation latency. We perform several simulations, from the available topologies of TopologyZoo, and the results show the superiority of the proposed algorithm when compared to competing current state-of-the-art algorithms in terms of propagation latency.**

***Index Terms*—Software-Defined Networking (SDN), controller placement, propagation latency, learning automaton (LA).**

I. Introduction

The control plane of network devices is isolated from the data plane in software-defined networking (SDN). The control plane operations are performed via a controller that interacts with the data plane of devices, i.e., switches, routers, etc. via a well-known application programming interface (API) like OpenFlow [1]. The devices in the data plane act based on the roles that are determined by the control plane. Upon receiving a packet from a flow, the network device applies the rules in its flow tables to process the packet [2]. If the packet is matched by a specific rule it determines how to handle the packet. Otherwise, the devices, which are called datapath, ask the controller for the action to undertake.

In SDN networks, the task of generating and installing the rules on the forwarding devices is offloaded to the controller [3]. The missed packets based on the available rules on an OpenFlow-enabled switch are buffered or dropped until the suitable rules for them are installed by the controller.

The controller communicates with the forwarding plane devices such as switches or routers in SDN through northbound interfaces (NBIs) like OpenFlow [1]. Initially, it was supposed that there is a single controller to control the forwarding behavior of the whole network's devices. An SDN network may have more than a single controller for scalability, performance, or robustness reasons [4], [5].

One of the critical issues in an SDN-network with multiple controllers is the place of controllers. The place of each controller plays a significant role in the network because it has an impact on the propagation latency of the network. Several attempts have been done to find a solution for this optimization problem [6]. The following considerations can be taken into account in designing a controller placement algorithms like: (a) the signaling latency which determines the latency between the switches and the controllers, (b) the controller capacity specifies the amount resources, like CPU and memory, for a controller. These resources are leveraged for handling the switches. (c) the number of controllers that determine the number of switches in a large network. (d) inter-controller communication latency, which specifies the latency among the controllers that manage different switches in the network. If a controller wants to manage a switch that is controlled by another controller, interaction between the controllers is required [7]. The controller placement problem is similar to the facility location problem [8] which is known to be NP-hard. Several algorithms are proposed to improve the network performance by decreasing the propagation latency, improving the reliability, and increasing the energy-efficiency of the network [9], [10], [11], [12]. Nevertheless, none of these works consider a predefined propagation latency to place the controllers while minimizing the overall propagation latency.

In this paper we focus on the controller placement problem. It can be formulated as a cover-set problem [13] by considering the latency bound for placing the controllers [8]. It is known to be NP-hard. For that problem, we propose a novel heuristic algorithm leveraging a learning automaton (LA) [14]. It takes a network graph and a distance metric between the nodes that reflects communication latencies and selects controller positions such that the communication latencies between any node and at least one controller does not exceed a desired threshold. It also aims at minimizing the propagation latency among the controllers and the switches. It worth stating that the output of the algorithm determines the number of

†The author acknowledges the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

controllers to place in the network.

The rest of this paper is organized as follows. Sec. II reviews the state-of-the-art of controller placement in SDN-based networks. The controller placement problem is discussed in Sec. III. The basic concept for learning automaton theory is presented in Sec. IV. Sec. V proposes a new heuristic algorithm for controller placement. The performance of the proposed algorithm is studied in Sec. VI. Finally, Sec. VII concludes this paper.

II. Related work

This section aims at providing an overview of recent state-of-the-art literature on controller placement of SDN networks. The authors of [5] provide a survey on the controller placement of SDN networks. The objective of this survey is to determine the design requirements for such networks like minimizing the latency between the controller and the switches, maximizing the resiliency of the network, etc.

Heller et al. [8] introduced the controller placement problem by defining a set of performance metrics. They analyzed the effect of controller placement on several topologies like Internet2 and others taken from TopologyZoo to give a research direction for the SDN community. Several solutions like Hyperflow [15], Devoflow [16], and Onix [17] were proposed to overcome the scalability and reliability issues in SDN. Generally, the researchers have tried to solve the controller placement in large SDN networks by leveraging multiple controllers to improve the network's scalability and reliability. A K-means-based clustering approach to place controller is proposed in [9]. The basic idea of the K-means clustering algorithm is to maintain k centroids, which are used to define clusters. Following the K-means, an item or a point will be considered in a specific cluster if it is closer to that specific cluster's centroid than any others. The authors optimized the k-means algorithm for clustering to minimize the overall latency of the network. Our algorithm has a better performance than this algorithm in term of propagation latency.

A density-based clustering algorithm to place the controllers in an SDN network was proposed by the authors of [18]. After clustering a network into sub-clusters, the proposed algorithm assigns a controller for each of them. Each switch in each cluster is only connected to the controller of that specific cluster. The size of each cluster is determined based on the capacity of the dedicated controller to that cluster. The authors stated that their proposed algorithm does not fall in a local optimum, which is the drawback of many heuristic algorithms. A Pareto-based controller placement algorithm is developed in [6] to minimize the required time to place the controller. The authors have found a tradeoff between the time to place the controllers and accuracy of the algorithm. The performance of this heuristic algorithm is studied for large-scale SDN networks to examine its efficiency.

The reliability of network in placing the controllers is studied in [10]. The authors considered several parameters like the location of controllers, the number of controllers, and the nodes of the network to achieve a high reliability. The performance of the proposed solution is studied on several topologies from ToplogyZoo. Resilient controller placement is studied in [11]. The objective of this work is to determine the required number of controllers as well as their locations to reach a high reliability in the network.

The energy-aware controller placement in SDN was studied by the authors of [12]. The problem is modeled as binary integer program (BIP), which has the latency of paths and the load of controllers as constraints for minimizing the energy consumption. Modeling the problem with BIP results in high complexity. Then, a genetic algorithm is proposed to find a sub-optimal solution for the problem. The obtained results from simulations confirmed that the performance of the heuristic algorithm is close to BIP. Controller placement with the aim of minimizing the end-to-end latency and queuing latency of the controller is studied in [19]. The authors leveraged the clustering approach to split the network into subnetworks to reach the design goals.

In this paper we propose a learning automaton (LA) based heuristic algorithm to select controller locations within a network such that the communication latency from any node to the closest controller does not exceed a desired threshold. It also aims at minimizing this latency. The algorithm leverages for each node an LA which helps to decide whether a controller should be collocated with that node.

III. Problem formulation

In this section, we state the network model. We model the network as a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of switches and $\mathcal{E}$ is the set of links among the switches. Each link $e \in \mathcal{E}$ has a length. We assume that the links among the switches are bi-directional. There is a link among switches v and w if they are connected. This can be defined as follows.

$$e(v,w) = \begin{cases} 1, & \text{if there is a link between } v \text{ and } w \\ 0, & \text{otherwise} \end{cases}$$

The idea behind our algorithm is to analyze the network topology and divide it into sub-networks in such a way that the switches inside each sub-network should have a close connection to each other, while they also should have a minimal distance to the controller in the network to minimize the overall propagation latency of the network. The latency of each link can be computed as follows.

$$l_{vw} = \frac{d(v,w)}{2 \times 10^8} \tag{1}$$

where l_{vw} and $d(v,w)$ indicate the propagation latency and the distance from switch v to w, respectively. The denominator in the above equation specifies the speed of light in an optical fiber. For a sub-network like $\acute{G}$, the average propagation latency ($\overline{L}(\acute{G})$) can be computed as follows.

$$\overline{L}(\acute{G}) = \frac{1}{|\mathcal{V}'|} \sum_{v \in \acute{V}} min\ d(c,v) \tag{2}$$

where $|\mathcal{V}'|$ and c are the number of nodes in the sub-network and the selected controller for $\acute{G}$, respectively. In Eq. 2, $d(c,v)$

specifies the distance between the controller (i.e., c) and the switch.

Formal definition of the problem. Given a network with $\mathcal{V}$ switches, the considered controller placement problem is the choice of a minimum number of controllers for the network within a predefined propagation latency in such a way that the overall propagation latency of the network is minimized. If the algorithm divides the network into n sub-networks, the average propagation latency for the whole network can be computed as follows.

$$\overline{L}(\mathcal{G}) = \frac{1}{n} \sum_{\acute{G} \in \mathcal{G}} \overline{L}(\acute{G}) \tag{3}$$

Therefore, the optimization problem can be defined as follows.

$$\min \overline{L}(\mathcal{G}) \tag{4}$$

After splitting the network nodes into sub-networks, we should place a controller for each of them in such a way that the general goal of the network (i.e., minimizing the average propagation latency) is satisfied. Therefore, we should consider controller-to-switch latency.

Controller-to-switch latency. This metric for a sub-network $\acute{G}$ by considering c as the place of controller can be defined as follows.

$$\overline{L}(\acute{G}) = \frac{1}{|\acute{G}|} \sum_{v \in \acute{G}} d(c, v) \tag{5}$$

where, d(c,v) is the distance from the controller to switch in $\acute{G}$ sub-network. $|\acute{G}|$ is the number of switches in the same sub-network.

IV. Learning automaton

Learning automaton (LA) is an abstract model that has a finite set of actions. It selects a random action among its allowable actions at any time. The random environment (RE) evaluates the chosen action by the LA and provides a reinforcement signal to the chosen action. Each LA updates the probability of its actions based on the received signal from RE through a learning algorithm [14]. The LA follows an iterative approach in selecting an optimal action among its action set. Therefore, the objective of LA is to choose the optimal action.

An LA can be defined as a quintuple $LA = \{Q, A, B, F, G\}$ [20], [21], [22], [23].

- $Q = \{q_1, q_2, \ldots, q_n\}$ is the set of finite states. For example, $q(t)$ shows the state of LA at time instant t.
- $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the set of finite actions of output where $\alpha(t)$ indicates the action of LA at time instant t.
- $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$ is the set of inputs. $\beta(t)$ states the input for the LA at time instant t.
- $F : Q \times B \to Q$ maps the current state and input to the next state of LA at the instant t.
- $G : Q \times B \to A$ maps the current state and input to the next action of LA at the instant t.

The environment can be defined as a triple $E = \{A, B, C\}$ in which the descriptions for A and B are as stated above. C is the penalty probabilities that are associated with each action α and defined as follows.

$$c_n = Pr[\beta = 1 | \alpha_t = \alpha_n] \tag{6}$$

where c_n indicates the penalty value for action α_n at time t.

The learning algorithm for LA is defined as follows.

$$p(n+1) = T[p(n), \alpha(n), \beta(n)] \tag{7}$$

where $p(n)$ is the probability of action $\alpha(n)$ and $\beta(n)$ is the given response from RE to the chosen action. In Eq. 7 if T is linear, the learning algorithm is also called linear algorithm. Otherwise, it is called nonlinear algorithm.

The learning algorithm of LA can be stated as follows. If an LA chooses an action like α_i and receives the desired reinforcement signal from the RE, the probability of that action, i.e., $p_i(n)$, will be increased. While the probability of other actions will decrease. Therefore, for a desired response from the RE the action probability vector of the LA will be as follows.

$$p_j(n+1) = \begin{cases} p_j(n) + a(1 - p_j(n)) & j = i \\ (1-a)p_j(n) & \forall j, j \neq i \end{cases} \tag{8}$$

If the RE generates a negative response for the chosen action of LA, i.e., action α_i, the probability of this action, i.e., $p_i(n)$, will decrease, while the probability of other actions increases. Eq. 9 will be applied for a negative answer from the environment to update the action probability vector.

$$p_j(n+1) = \begin{cases} (1-b)p_j(n) & j = i \\ \frac{b}{r-1} + (1-b)p_j(n) & \forall j, j \neq i \end{cases} \tag{9}$$

where a and b are reward and penalty parameters, respectively. In these equations, r determines the number of action for each LA. If $a = b$ the learning algorithm is called L_{R-P} (*Reward-Penalty*), if $b \ll a$, it is called $L_{R\varepsilon P}$ (*Reward epsilon Penalty*), and if $b = 0$ the algorithm is called L_{R-I} (*Reward-Inaction*). More explanation of LA can be found in [20], [21], [22], [23].

V. LACP-Learning Automaton-based Controller Placement

This section describes how the proposed algorithm splits the network into several sub-networks to assign a proper controller to each of them. The proposed approach is an iterative algorithm which results in making sub-network (i.e., clusters) in several iterations. The LACP algorithm has two main phases, namely, *initialization* and *learning* phases. During the initialization phase, the network of LA from the network is built and each node forms its action-set. Selecting the nodes to place the controller is carried out in *learning* phase. We explain each phase in more detail.

A. Initialization Phase.

The LACP algorithm equips each node in the network topology with an LA. Therefore, the network graph constitutes the network of LA. The LA of each node has two actions to select at any time which are: being a controller (i.e. α_c) or not

(i.e. $\alpha_{\overline{c}}$). At each iteration, the LA of each node can randomly select one of these actions. At the beginning of the algorithm, the probability of each action is initialized to 0.5.

B. Learning Phase

This phase runs in several iterations and over the course of each iteration, the RE determines the suitability of each chosen action by LA of each node. This can be performed by evaluating the metric that is defined for the algorithm, namely, the average propagation latency for each sub-network.

The *learning* phase of the LACP algorithm starts by selecting a random node among the nodes of the network graph (i.e., $\mathcal{V}$). This node, which is equipped with an LA, randomly chooses an action from its action-set based on the action probability vector (i.e., lines 7-8). Then, it examines the selected action and acts as follows. If the selected action is α_c, it adds itself to $\mathcal{S}$ vector, and subsequently, adds the neighbor nodes that have a propagation latency within L_{max} to $\mathcal{W}$ vector. Note that the propagation latency parameter, i.e., L_{max}, is taken as an input for our algorithm. For example, having a value of 1 ms for this parameter states the nodes that it can be reached by a propagation latency of 1 ms from this node. This procedure finishes when the number of nodes in both vectors reaches the total number of nodes in the graph (i.e., lines 14-27). The selected controllers during each iteration is in $\mathcal{S}$ vector.

The propagation latency of nodes in the $\mathcal{S}$ vector is measured (i.e., lines 28-39). Then, this value will be compared with the threshold propagation latency value (L_{min}) for this metric. The threshold value is initialized with the highest possible value. If the latency is less than the value in the previous iteration, the RE generates a positive reinforcement signal for the chosen action of each LA in the network. The value for the RE can be computed according to Eq. 3. Consequently, this behavior of RE results in giving a reward for the selected action of each LA, and the LA of each node updates its action probability vector according to Eq. 8 (i.e., lines 40-45).

Also, the threshold value for the propagation latency will be replaced by the new threshold value (i.e, $\overline{L}_{min}$) and the best controller set (i.e., $\mathcal{C}$) until now will be replaced by $\mathcal{S}$. Otherwise, the RE generates a negative reinforcement signal, which results in giving a penalty to the selected actions by the LA of each node. In LACP algorithm, we use the L_{R-I} schema to update the action probability vector of each LA. This means that in the case of getting a penalty from the environment for the chosen actions, the probability of actions will be unchanged.

The learning procedure ends when the stop condition of the proposed algorithm meets. We define a fixed number of iterations as the stop condition. If the number of iterations reaches the threshold value (i.e., I_T), the algorithm ends. At this time, the nodes in $\mathcal{C}$ will be selected as the controllers for the network and other nodes will be placed as a switch. We use the stopping model of [24] in the LACP algorithm. Algorithm 1 shows the pseudo code of LACP to place the SDN controllers in a network with $\mathcal{V}$ nodes.

Algorithm 1 LACP Algorithm

1: **Input:**
2: The network graph $G = (\mathcal{V}, \mathcal{E})$, the desired propagation latency (L_{max}), the learning parameter a, the iteration threshold I_T
3: **Initialization**
4: $\mathcal{C} \leftarrow \mathcal{V}$ ▷ The best controller set yet.
5: $\overline{L}_{min} \leftarrow \infty$ ▷ the threshold value for latency.
6: $k \leftarrow 0$ ▷ the iteration counter
7: **for** node $v \in \mathcal{V}$ **do**
8: $\alpha_c(v) = 0.5$
9: **end for**
10: **// Learning phase**
11: **repeat**
12: $\mathcal{W} = \mathcal{V}$
13: $\mathcal{S} = \emptyset$ ▷ the selected controllers in each iteration.
14: **while** $\mathcal{W} \neq \emptyset$ **do**
15: select randomly $w \in \mathcal{W}$
16: **if** $rnd() > \alpha_c(w)$ **then**
17: // make w a controller
18: $\mathcal{W} = \mathcal{W} \setminus w$
19: $\mathcal{S} \leftarrow \mathcal{S} \bigcup w$
20: $\mathcal{X} = \mathcal{W}$
21: **end if**
22: **for** all $x \in \mathcal{X}$ **do**
23: **if** $L(w, x) \leq L_{max}$ **then**
24: $\mathcal{W} = \mathcal{W} \setminus x$
25: **end if**
26: **end for**
27: **end while**
28: // compute $\overline{L}$
29: $\overline{L} = 0$
30: **for** all $v \in \mathcal{V}$ **do**
31: tmp=∞
32: **for** all $c \in \mathcal{S}$ **do**
33: **if** $L(v, c) <$ tmp **then**
34: tmp=$L(v, c)$
35: **end if**
36: **end for**
37: $\overline{L} = \overline{L}$+tmp
38: **end for**
39: $\overline{L} = \frac{\overline{L}}{|\mathcal{V}|}$
40: **if** $\overline{L} < \overline{L}_{min}$ **then**
41: $\overline{L}_{min} = \overline{L}$
42: $\mathcal{C} \leftarrow \mathcal{S}$
43: $p_{\alpha_c}(n+1) = p_{\alpha_c}(n) + a(1 - p_{\alpha_c}(n))$
44: $p_{\alpha_{\overline{c}}}(n+1) = (1-a)p_{\alpha_{\overline{c}}}(n)$
45: **end if**
46: $k \leftarrow k + 1$
47: **until** $(k < I_T)$
48: **Output:**
49: Set of controller nodes

VI. Results

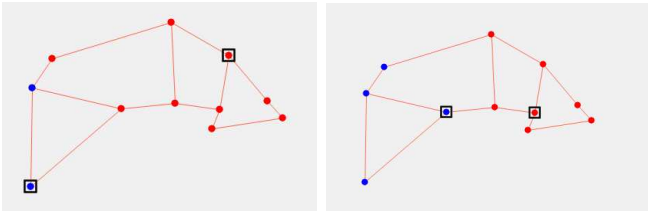
In this section, we study the performance of LACP algorithm and compare the results with K-means algorithm in [9] in terms of (a) the average switch-to-controller propagation latency and (b) the maximum switch-to-controller propagation latency. In order to have a fair comparison between the performance of LACP and K-means algorithm in [9], we use the same number of sub-networks for each topology. For example, if the number of sub-networks after applying the LACP algorithm is 5, we use this number as the input for the K-means algorithm, i.e., we use k=5. We first explain the simulation setup and then explain the obtained results.

A. Simulation Setup

We implement both algorithms in $C\#$ language by the network topologies that are taken from topologyZoo. Table. I shows the number of nodes and links in each considered network topology. In all simulation runs, we use the stop condition of [24] in our algorithm for I_T, and the initial value for the number of sub-network is set to the number of nodes in the network. We use Dijkstra algorithm to find the minimum distance between the nodes to compute the average and maximum propagation latency. To compute the maximum latency between the controllers and the switches, we measure the propagation latency of each sub-network and pick the maximum latency as the final result.

TABLE I: The topology feature of the networks.

Topology	Nodes	Links
Abilene	11	14
InternetMCI	19	45
Geant2010	37	58
Iris	51	61



(a) LACP with two controllers. (b) K-means with two controllers

(c) LACP with three controllers (d) K-means with three controllers

Fig. 1: The visualization of Abilene network for LACP and K-means algorithm of [9].

B. Performance Evaluation

We first provide a visualization of Abilene network topology for both algorithms in placing the controllers. Fig. 1a and Fig. 1b illustrate the selected controllers by two algorithms for the case of partitioning the network into two sub-networks. While Fig. 1c and Fig. 1d show the same network for three sub-networks. The main difference between the two approaches is in choosing the number of nodes for each sub-network and selecting the controller for each of them. LACP algorithm measures the propagation latency to place the controller while the K-means algorithm leverages the distance for this purpose.

Figs 2a, 2b, 2c and 2d depict the average propagation latency between the switches and the controllers for the Abilene, InternetMCI, Geant201, and Iris networks by varying the number of controllers. The general trend in these figures is that by increasing the number of deployed controllers the average and the maximum propagation latency decreases. A common reason for this fact is that by increasing the number of controllers, each controller requires taking the control of fewer switches in the network. Consequently, this task results in less propagation latency. It is clear from the figure that our algorithm reaches a better performance than that of [9] in both considered scenarios. The reasons for the results can be summarized as follows. (a) LACP algorithm finds a solution in several iterations. Doing so, the algorithm has the ability to test a different set of candidate solutions. (b) LACP looks for the nodes that are within the predefined propagation latency aiming at dividing the nodes into sub-networks to minimize the overall propagation latency.

Figs 2e, 2f, 2g and 2h illustrate the maximum propagation latency between the schemes. We see the same phenomenon for the maximum propagation latencies between the switches and the controllers for all considered networks. It can be found from the figures that LACP algorithm has a better performance than K-means algorithm in terms of maximum propagation latency.

VII. Conclusion

A learning automaton based algorithm for controller placement in SDN networks is proposed in this work. The controller placement problem is an NP-hard problem and we propose a simple heuristic algorithm based on machine learning. The main objective of our algorithm is to minimize the propagation latency among the controllers and the switches in the network. We equip each node in the network with a learning automaton. The learning automaton of each node determines the role of each node (i.e., to act either a controller or not). The proposed algorithm selects the controllers based on the goal of minimizing the propagation latency. The desired propagation latency can be set as an input for LACP algorithm, which tries to divide the nodes in the network into sub-network based on the predefined threshold value. The performance of the algorithm is studied on the topologies that are taken from TopologyZoo and the obtained results show that the LACP algorithm requires less propagation latency than K-means algorithm to place the controllers in an SDN network.

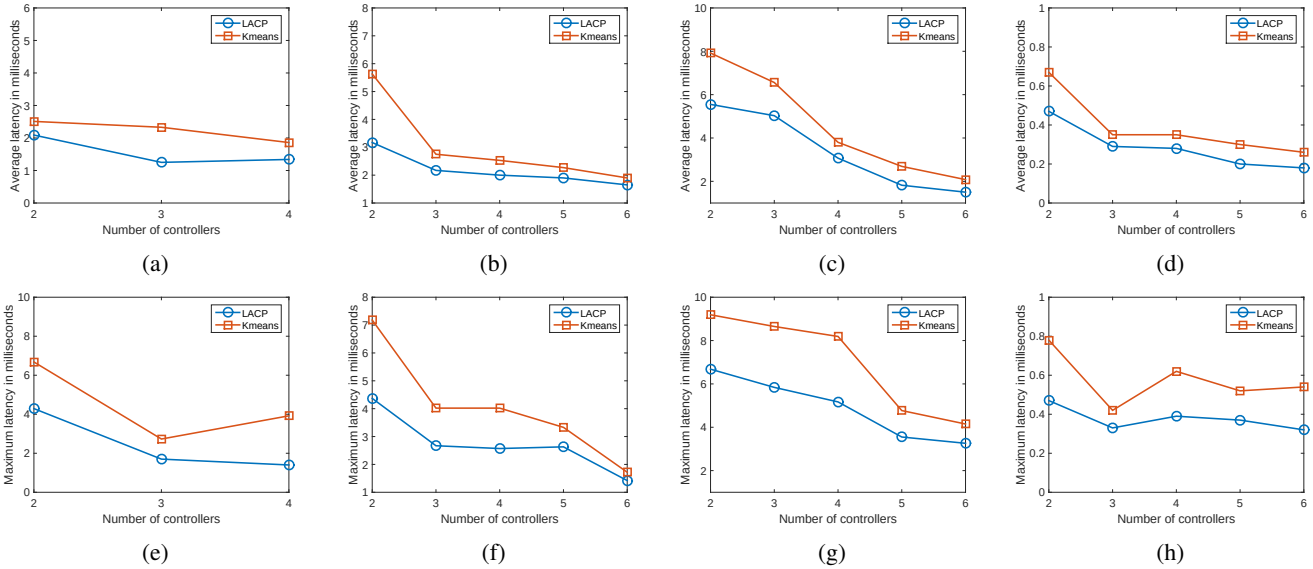


Fig. 2: The performance of both algorithms for considered topologies. a) the average propagation latency for the Abilene network. b) the average propagation latency for the InternetMCI network. c) the average propagation latency for the Geant2010 network. d) the average propagation latency for the Iris network. e) the maximum propagation latency for the Abilene network. f) the maximum propagation latency for the InternetMCI network. g) the maximum propagation latency for the Geant2010 network. h) the maximum propagation latency for the Iris network.

We intend to extend the algorithm to place the controllers by also considering the number of controllers as the input.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] W. Braun and M. Menth, "Software-defined networking using openflow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[3] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, Aug 2014.

[4] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Network*, vol. 31, no. 5, pp. 21–27, 2017.

[5] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101 – 118, 2018.

[6] 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, March 2015.

[7] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[8] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[9] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A k-means-based network partition algorithm for controller placement in software defined network," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[10] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software-defined networks," *Computer Communications*, vol. 77, pp. 41 – 51, 2016.

[11] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–7.

[12] Y. Hu, T. Luo, N. C. Beaulieu, and C. Deng, "The energy-aware controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 21, no. 4, pp. 741–744, 2017.

[13] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[14] K. S. Narendra and M. A. L. Thathachar, *Learning automata: An introduction*. Prentice Hall, 1989.

[15] 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*, 2010, pp. 3–3.

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[18] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Computer Networks*, vol. 112, pp. 24–35, 2017.

[19] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 344–355, March 2018.

[20] M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis, "learning automata: theory, paradigms, and applications," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 6, pp. 706–709, 2002.

[21] M. S. Obaidat, G. I. Papadimitriou, A. S. Pomportsis, and H. Laskaridis, "Learning automata-based bus arbitration for shared-medium atm switches," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 6, pp. 815–820, 2002.

[22] G. I. Papadimitriou, M. S. Obaidat, and A. S. Pomportsis, "On the use of learning automata in the control of broadcast networks: a methodology," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 6, pp. 781–790, 2002.

[23] M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis, "Efficient fast learning automata," *Information Sciences*, vol. 157, pp. 121–133, 2003.

[24] J. A. Torkestani and M. R. Meybodi, "Clustering the wireless ad hoc networks: A distributed learning automata approach," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 394–405, 2010.