QoS-Guaranteed Controller Placement in SDN

Tracy Yingying Cheng, Mengqing Wang, Xiaohua Jia
City University of Hong Kong
Email: {tracy.cheng, mengq.wang}@my.cityu.edu.hk, csjia@cityu.edu.hk

***Abstract*—The controller placement problem tries to place k controllers in a given network to optimize performance metrics such as propagation latency, load distribution, network reliability and failure resilience. However, the quality of service (QoS) is always a primary concern of the network operators in the placement of SDN controllers. Since the SDN controllers are responsible to provide services for switches, the response time of controllers is an important QoS parameter of network operators. In this paper, we introduce the QoS-Guaranteed Controller Placement problem: Given a network topology and a response time bound, how many controllers are needed, where to place them, and which switches are assigned to each of the controller. We propose three heuristic algorithms: incremental greedy algorithm, primal-dual-based algorithm and network-partition-based algorithm. The proposed algorithms are tested and compared on the Internet Topology Zoo, a dataset of public available network topologies. Simulation results show that the proposed incremental greedy method slightly outperforms the other two methods on all input topologies.**

***Index Terms*—Software-Defined Networks, Controller Placement, QoS Guaranteed.**

I. Introduction

The software-defined network (SDN) aims to decouple the control plane from the data plane. It uses the network controller to control the routing behavior of switches or switches, so switches can focus on data forwarding. In a large-scale network, it requires multiple physically dispersed controllers in the network, each of which manages and operates the switches in its local area. There are some works regarding the design and implementation of distributed systems of SDN controllers, such as ElastiCon [1], Onix [2] and HyperFlow [3].

In the design of distributed control plane, the location of controllers has significant impact on the network performance. There are two major factors that affect the performance of controllers: 1) the network distance between switches and controllers. In a large-scale network, the network delay between a switch to its controller becomes non-negligible and this delay affects the response time of controllers. 2) load balancing among the controllers. Some controller could be overloaded while others are less busy. Therefore, it is important to place controllers properly in a network.

The controller placement problem has been studied in recent years. Heller et al. first introduced the controller placement problem in [4]. The problem is to place k controllers in a given network such that the propagation delay between switches to controllers is minimized. Several follow up works focused on the optimization of other performance metrics for placing k controllers in a network [5], [6], [7], [8], [9]. The work in [5] aims to minimize the communication delay between controllers. Load distribution among controllers is considered in [6]. The controller capacity is considered in [7], the network reliability is considered in [8], failure resilience is considered in [9], etc.

However, the issues of guaranteeing quality of service (QoS) have not been considered in the previous work of controller placement. In network design, QoS is always a primary concern of the network operators in the placement of SDN controllers. QoS specification requires network elements to deliver a guaranteed service (guaranteed delay and bandwidth) in the network [10]. Since the SDN controllers are responsible to provide services for switches to setup routing information of traffic flows, the response time of controllers is an important QoS parameter for network services. The response time of a controller consists of two components: 1) the round trip of network delay between the controller and a switch, and 2) the service time of the controller, which depends on the service capacity and the work load of the controller.

In this paper, we study the QoS guaranteed controller placement problem, which is to place the minimum number of controllers in the network such that response time of controllers can meet a given delay bound. That is, given a delay bound, we need to determine: a) how many controllers are needed, b) where to place them, c) which switches are assigned to each of the controllers. This problem is more complicated than the traditional controller placement problems that were studied in the literatures. This is because the response time considered in our problem includes the service time of the controllers, which depends on the workload of the controllers. The traditional optimization methods for k-median or k-center problems are not applicable to our problem.

For the proposed QoS-guaranteed controller placement problem, we propose three heuristic algorithms: incremental greedy algorithm, primal-dual-based algorithm and network-partition-based algorithm. Extensive simulations have been done to compare the performance of the three heuristic algorithms on the same dataset. The simulations are performed on the Internet Topology Zoo [11], which is a database including publicly available network topologies. Simulation results show that the proposed incremental greedy method slightly outperforms the other two methods on all input topologies.

The remining of the paper is organized as follows: Section II presents the related work on the controller placement problem in SDN. Section III gives the system model and problem formulation. Section IV introduces our solutions. Section V shows the simulation and related analysis. Finally conclusion is given in VI.

978-1-4799-5952-5/15/$31.00 ©2015 IEEE

II. Related Work

The controller placement problem was firstly introduced by [4]. This work aims to minimize the average latency between controllers and switches. It is pointed out that the average latency and the maximal latency cannot be optimized at the same time, and a K-means greedy method was employed to find the optimal placement that minimizes the average latency and k-center greedy method was proposed to minimize the maximal latency.

The communication time and load distribution among controllers are taken into consider in [6] for designing the control plane. This paper introduces network robustness, a property that minimizes data loss on the communication paths between controller and its switches in spite of network failures. A K-critical algorithm is developed to find the minimum number of controllers that allow a robust control layer to be built.

Failure resilience is considered in [9] for placing controllers. This work concludes that whereas one controller is enough from latency point-of-view, more controllers are needed to meet resilience requirement. They also consider inter-controller latency, load balancing between controllers, and trade-off considerations between latency and failure resilience. A framework for resilience pareto-based optimal controller placement is proposed that provides network operators with all pareto-optimal requirements.

A fault tolerant controller placement was proposed in [8]. The paper formulated the fault tolerant controller problem considering the network reliability constraint. A heuristic algorithm was proposed to computes placements with the required reliability. This work concludes that each node is required to connect to 2 or 3 controllers, which typically provide more than five nines reliability.

The load of controllers is taken into account for designing controller placement in [7]. The controller problem was extended to a capacitated problem where there is an upper bound for the load of each controller. The author proposed a greedy K-center algorithm to solve the problem.

The previous works on the controller placement problem focus on placing a given number of controllers to optimize a given performance metric such as propagation delay, controller load distribution, network reliability, etc. However, minimizing the number of controllers has not been considered and the QoS has not been guaranteed in the previous work. The response time of the request sent from a switch to its controller is an important parameter of QoS. Therefore, we propose the QoS-guaranteed controller placement problem to determine: 1) the minimal number of controllers needed; 2) where to place them; 3) which switch is under controller of each of them.

III. System Model and Problem Formulation

In this section, we first formulate the response time of a request from a switch to a controller, which is the performance metric considered in our controller placement problem. Then we formulate the proposed QoS-guaranteed controller placement problem.

TABLE I: Notations

Notation	Definition
V	set of switches
C	set of controllers
δ	upper bound of response time
r_i	the requesting rate of switch i
t_i	response time of request from switch i
$\bar{t}$	average response time of all switches
μ	service capacity of each controller
λ_j	traffic arrival rate of controller j
τ_j	mean service time of controller j
S_j	QoS service range of controller j
x_{ij}	variable indicates whether switch i is assigned to controller j
$t_{i,j}$	network delay between switch i and controller j
y_j	variable indicates whether controller candidate j is open
a_i	dual variable of switch i

The network is modeled by graph $G(V, E)$, where V represents network switches, E represents the network links between nodes. Each link is associated with a network delay. Each switch $i \in V$ is associated with a request demand r_i. Each switch's location is also a candidate site for placing a controller. Suppose all controllers have the same processing capacity μ. Our goal is to place minimal number of controllers meeting the QoS requirement. Next we introduce the QoS requirement.

The main job of a controller in SDN is to set up forwarding path for flows upon request from switches [12]. The flow setup process works as follows: When the first packet of a new flow arrives at an OpenFlow switch, its header information is extracted and then matched against the flow table entries. If there is no match found for the packet, the packet is forwarded to the controller for processing. The controller will send the forwarding information to the switch to help set up the flow table for the new flow of this packet. The response time of a request sent from a switch is defined as the time from the moment of sending out this request until the reply is received from its controller. Given an upper bound δ of the response time, our task is to place the minimum number of controllers in V such that the average response time for all switches is within the given bound δ.

We introduce two variables, X and Y, to respectively denote the assignment of switches to controllers and the placement of controllers. Initially, each node in V has a switch. Suppose each node can also be placed with a controller. For switch $i \in V$ and controller $j \in V$, $x_{ij} = 1$ if switch i is assigned to controller j and $x_{ij} = 0$ otherwise; $y_j = 1$ if there is a controller placed at node j and $y_j = 0$ otherwise. The response time of a flow setup request from a switch has two components: 1) the round trip time from the switch to its controller, and 2) the service time of the controller for the request. We first take a look at the service time of a controller.

Each controller can be modeled as a M/M/1 queuing model [13], a canonical single server queuing model. Suppose packets arrive according to a Poisson process and the processing times of the packets are independent and identically exponentially distributed [14]. Let S_j denote the set of switches served by controller j. λ_j denotes the total arrival rate of packets at the

controller j, which can be calculated as follows:

$$\lambda_j = \sum_{\forall i \in s_j} r_i x_{ij}, \tag{1}$$

where r_i is the request rate of switch i.

The packets arrived at a controller are processed according to First-Come-First-Serve order. Each controller $j \in V$can only accept number of clients within its processing capacity μ, therefore it is required that: $\lambda_j < \mu$.

According to the queuing theory [15], the expected mean service time τ_j of controller j can be calculated as follows:

$$\tau_j = \frac{1}{\mu - \lambda_j} = \frac{1}{\mu - \sum_i r_i x_{ij}}. \tag{2}$$

For each switch i, let t_i denote the response time of its request, which is calculated as follows:

$$t_i = 2t_{ij} + \tau_j = 2t_{ij} + \frac{1}{\mu - \sum_i r_i x_{ij}}, \tag{3}$$

where t_{ij} denotes the network delay between switch i and controller j. We assume the request and reply between the switch and the controller always take the shortest path in the network. t_{ij} is the sum of the link delay of the shortest path between nodes i and j in the network.

Let $\bar{t}$ denote the mean response time, which is calculated as follows:

$$\bar{t} = \frac{1}{|V|} \sum_{i \in V} t_i \tag{4}$$

In this paper, we define the QoS-guaranteed controller placement problem as follows:

Definition 1. *Given a network topology G, we are asked to place the minimal number of controllers subject to the response time constraint:* $\bar{t} \leq \delta$.

IV. Heuristic Algorithms

To solve the proposed QoS-guaranteed controller placement problem, we propose three heuristic algorithms: incremental greedy algorithm, primal-dual-based algorithm and network-partition-based algorithm. The incremental greedy algorithm iteratively opens the controller candidate that can serve the largest number of switches until all switches are served. We also design a primal-dual-based algorithm to solve the proposed problem. The above two algorithms do not have a global view of the network. Therefore, we develop a network-partition-based algorithm that first divides the network into equal partitions and places controllers.

A. Incremental Greedy Algorithm

The incremental greedy algorithm iteratively opens the controller candidate that can serve the maximal number of switches. S_j denotes the set of switches served by controller j. Each switch in S_j has average response time less than the upper bound δ from controller j. As more switches are added in S_j, i.e., more switches are served by controller j, the average response time of all switches $\bar{t}$ in S_j increases. $\bar{t}$ in S_j is affected by the service time of controller j. S_j is determined by iteratively adding in a switch and checking the response time constraint. Initially, S_j is an empty set. In each iteration, the switch with the minimal network delay from controller j (not belong to S_j) is added into S_j. As more switches are added in S_j, more switches are served by controller j and the service time of the controller increases. Therefore $\bar{t}$ in S_j increases. We stop adding switches to S_j until $\bar{t}$ in S_j reaches the upper bound δ.

The detail of greedy algorithm is illustrated in Algorithm 1. X and Y denote the assignment of switches to controllers and the placement of controllers respectively. Suppose each switch's location is a candidate site for placing a controller. In each iteration, we first determine the set of switches served by each controller candidate, as it is shown in line 5-10. Then the controller candidate j serving the maximal set of switches is chosen to open. Switches within S_j are assigned to controller j. Then the served switches and opened controllers are removed from the candidates. This iteration is repeated until every switch is served by a controller.

Algorithm 1: Incremental Greedy Algorithm

Input: Graph $G = (V, E)$, Response time bound δ;
Output: X, Y;

1 $R \leftarrow V$, $C \leftarrow V$; // R denotes the set of remaining unserved switches, C denotes the set of controller candidate sites ;
2 $y_j = 0, x_{ij} = 0,\ \forall i \in R, j \in C$;
3 **while** $R \neq \emptyset$ **do**
4 **foreach** $j \in C$ **do**
5 $S_j \leftarrow \emptyset$;
6 **while** $\bar{t}_i \leq \delta, \forall i \in S_j$ **do**
7 Find the switch i' closest to j, $i' \notin S_j$;
8 $S_j \leftarrow S_j + \{i'\}$;
9 Find candidate site j with the maximal S_j, set $y_j = 1$;
10 $C \leftarrow C - \{j\}$;
11 **foreach** *switch* i *within* S_j **do**
12 $x_{ij} = 1$;
13 $R \leftarrow R - \{i\}$;
14 **return** X, Y

B. Primal-Dual-based Algorithm

The incremental greedy algorithm above is a simple method to solve the controller placement problem. It does not have any theoretical performance guarantee to the generated results. The primal-dual method has wide application on finding the near-optimal solution to integer programs with a theoretical performance guarantee.

The proposed QoS guaranteed controller placement problem is a combinatorial optimization problem that can be formulated as integer program. The primal-dual method finds the near-optimal solution to the integer program by using linear relax-

ation and the duality of the original program. The performance guarantee of this method has been theoretically proved.

First we formulate the QoS-guaranteed controller placement problem into an integer program. Given a set V of n controller candidates, let S denote the collection of QoS service ranges for all controller candidates: $S = \{S_1, ..., S_n\}$, where S_j is the set of switches within the QoS service range of controller j. Our goal is to find the minimal sub-collection of S that serves all switches. For each switch $i \in V$, it is required that at least one of the sets containing it is selected. The controller placement problem can be formulated as:

$$\begin{aligned} \text{minimize} \quad & \sum_{S_j \in S} y_j \\ \text{subject to} \quad & \sum_{S_j : i \in S_j} y_j \geq 1, \qquad \forall i \in V \\ & y_j \in \{0, 1\}, \qquad \forall j \in V \end{aligned} \tag{5}$$

where y_j is set to 1 iff controller j is selected, 0 otherwise.

In the primal-dual method, a pair of primal program and dual program is derived from the original integer program. Then the complementary slackness conditions of the primal program and the dual program are generated respectively. We start with initial feasible solutions to the primal and dual programs, it iteratively starts satisfying complementary slackness conditions. The current primal solution is used to determine the improvement to the dual solution, and vice versa. This algorithm is ended when a primal feasible solution is obtained and all complementary slackness conditions are satisfied. During the iterations, the primal is always modified integrally, so that eventually we get an integral solution.

The primal program is obtained by letting the domain of variables y_j be $y_j \geq 0$. To derive the dual program, we introduce a dual variable a_i for each switch $i \in V$. The dual program is derived as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in V} a_i \\ \text{subject to} \quad & \sum_{i : i \in S_j} a_i \leq 1, \qquad \forall S_j \in S \\ & a_i \geq 0, \qquad \forall i \in V \end{aligned} \tag{6}$$

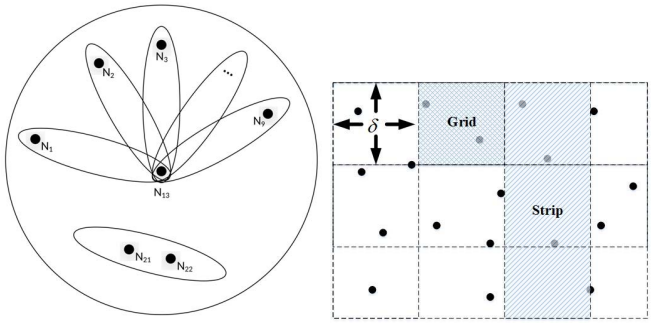
The complementary slackness conditions are described as follows, which are derived based on standard procedure:

$$\begin{aligned} \text{Primal condition:} \quad & y_j \neq 0 \Rightarrow \sum_{i : i \in s_j} a_i = 1, \ \forall S_j \in S \\ \text{Dual condition:} \quad & a_i \neq 0 \Rightarrow \sum_{S_j : i \in s_j} y_j \leq F, \ \forall i \in V \end{aligned} \tag{7}$$

where F denotes the largest number of sets that one switch is included in. S_j is said to be tight if $\sum_{i : i \in S_j} a_i = 1$. The initial values of $\{y_j, \forall j \in V\}$ and $\{a_i, \forall i \in V\}$ are set to 0. The variable y_j is incremented integrally, which means to pick only tight sets in the algorithm.

The primal-dual algorithm proceeds in iterations. In each iteration, we pick an unserved switch, say i, and raise a_i until some set goes tight. Pick all tight sets and open the corresponding controller candidates. For each tight set, assign switches within the set to the controller. For switches covered by multiple tight sets, each of them can randomly choose a tight set and connect to its controller. Then the unselected sets and unserved switches are updated. This iteration is repeated until all switches are served. The detail of this primal-dual algorithm is illustrated in Algorithm 2.

To help illustrate this primal-dual-based controller placement algorithm, we give an example. In Figure 1a, there are some sets $\{N_1, N_{13}\}$, $\{N_2, N_{13}\}$, $\{N_3, N_{13}\}$, ... $\{N_9, N_{13}\}$, $\{N_{21}, N_{22}\}$. Each of these sets represents a service range of a controller candidate. Suppose the algorithm raises b_{13} in the first iteration. When b_{13} is raised to 1, all sets including N_{13} go tight. They are all picked in the system, thus switches $N_1, N_2, N_3, ..., N_9$ are served by the controllers of the tight sets. In the second iteration, b_{21} is raised to 1 and the set $\{N_{21}, N_{22}\}$ goes tight. Pick the set $\{N_{21}, N_{22}\}$ into the system and all switches are being served by at least one controller.



(a) Primal-dual-based algorithm (b) Network-partition algorithm

Fig. 1: example

Algorithm 2: Primal-Dual-based Algorithm

Input: Graph $G = (V, E)$, Response time bound δ;
Output: X, Y;

foreach $j \in V$ **do**
  Compute the switch set S_j served by candidate j ;
$S = \{S_j, \forall j \in V\}$;
$y_j = 0, \ \forall j \in V$;
$a_i = 0, \ \forall i \in V$;
while $V \neq \emptyset$ **do**
  Pick an unserved switch, say i, and set $a_i \leftarrow a_i + 1$;
  Find all sets that go tight ;
  foreach *tight* S_j **do**
    Set $y_j = 1$ and open the controller j ;
    Assign all switches within S_j to controller j ;
    $V \leftarrow V - \{i | i \in S_j\}$;
return X, Y

C. Network-Partition-based Algorithm

The above incremental greedy algorithm and the primal-dual-based algorithm do not have a global view of the whole network. They create unbalanced placement and unbalanced load among controllers. Next we design a network-partition-based algorithm that balances the load among all controllers.

The basic idea is to divide the network into partitions and place controllers in each partition separately. First, the network is divided into partitions. In each partition, the largest propagation delay between any two switches is within the bound δ. In each partition, we use the incremental greedy method to place controllers. Since we limit the service range of each controller to one of equally divided partitions, the load distribution among all controllers is more balanced than the greedy method and the primal-dual method.

The detail of the network-partition-based algorithm is described in Algorithm 3. First, the network is divided into grids. Any pair of switches within each grid has a communication delay less than δ. In the network partitioning, we first divide the network into vertical strip-areas, as it is illustrated in Figure 1b. In each strip-area, the propagation delay between the node on the left boundary and the node on the right boundary is equal to δ. The nodes located at the boundary belong to the left strip-area. Then each strip is divided into grids. In each grid, the propagation delay between the node on the top boundary and the node on the bottom boundary is equal to δ. The nodes located at the boundary belong to the top grid. After partitioning the network, we use the greedy method to place controllers in each partition.

Algorithm 3: Network-Partition-based Algorithm

Input: Graph $G = (V, E)$, Response time bound δ;
Output: X, Y;

$R \leftarrow V$, $C \leftarrow V$;
$y_j = 0,\ \forall j \in C$;
$x_{ij} = 0,\ \forall i \in R, j \in C$;
Divide the network graph into vertical strips ;
foreach *vertical strip* **do**
    Divide the vertical strip into horizontal grids ;
foreach *grid* **do**
    Use Algorithm 1 to place controllers in this grid ;
return X, Y

V. Simulation and Analysis

The proposed three heuristic methods are tested on the Internet Topology Zoo, a public dataset of network topologies. We first introduce the setup of the simulation, and then give the simulation results and analysis.

A. Simulation Setup

The network topologies used for simulation are collected from a publicly available repository, namely the Internet Topology Zoo [11]. The Internet Topology Zoo is a store of wide area network topologies created from the information that network operators makes public. From the set of networks reported in [11] we exclude all disconnected topologies, which leaves us with 74 topologies at the Point-of-Presence (PoP) level. In a network topology, every PoP corresponds to one network switch in our scenario. In addition, we consider each PoP as a candidate for hosting one controller. Each switch is assigned to one controller in the topology graph G. The proposed algorithms are implemented in Python program to test the input dataset.

The simulation parameters are set as follows. According to a flow setup rate measurement test [16] on the HP ProCurve 5406zl switch, the results indicates that the switch completes roughly 275 flow setups per second. In the Internet Topology Zoo, a PoP may consist of a number of switches. Therefore the requesting demand of each PoP in our scenario is set as $r_i \in [1500, 3000]$ per second. A study [17] shows that a popular network controller (NOX) handles around 30k flow requests per second while maintaining a sub-10ms flow install time. Therefore, the processing rate of each controller in our problem is set as $u_j = 30000$ per second.

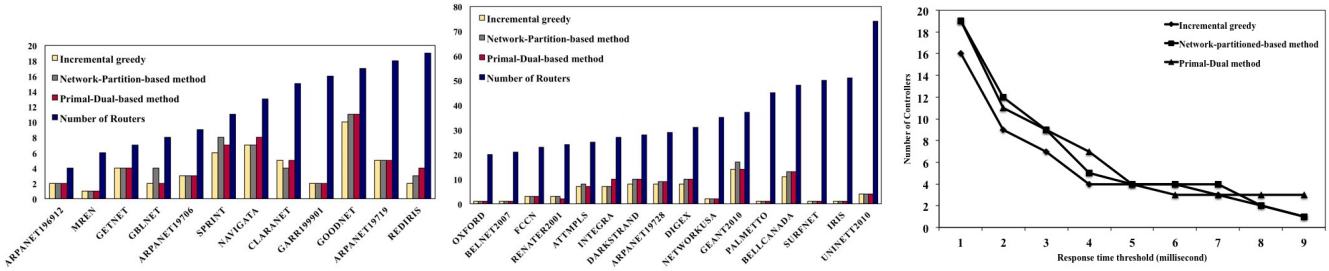
The response time of the request sent from a switch to its controller consists of three components: the propagation time from the node to the controller (uplink), the service time (sojourn time) of the controller, the propagation time from the controller to the node (downlink). Let T_G denote the largest propagation delay for the network topology G. The value of the response time bound δ depends on the network topology. Since each input network topology is different from each other, the value of δ is also different for each topology. The response time upper bound δ in our problem is set as $\delta \in [2T_G+5, 2T_G+10]$ ms. For each pair of network topology G and response time bound δ, we run ten times for each algorithm to obtain the final results.

B. Simulation Results and Analysis

The performance of the proposed three heuristic methods is compared using the same dataset. The simulation results are shown from Figure 2a to Figure 2c. For each network topology G, we first compute a proper response time bound δ based on the network topology, then get the minimal number of controllers output by the three heuristic algorithms.

The three heuristic methods are compared under 74 network topologies. Figure 2a and Figure 2b show the number of controllers output by the three methods and the total number of switches of some representative network topologies. The number of switches for each WAN topology ranges from 4 to 74. As it is shown in Figure 2a and Figure 2b, the number of controllers generated by the three heuristic methods are very close to each other for all input topologies.

Figure 2c shows the performance of the proposed three methods under various value of δ in one topology. We choose the network topology named UNINETT2010, which has 74 nodes. For this network topology, we compute 9 different value of δ: 1 ms to 9 ms, as it is shown in the X-axis of Figure 2c. We can see that the performance of the three methods are very



(a) Number of Controllers VS Number of switches

(b) Number of Controllers VS Number of switches

(c) Comparison results under different value of δ in one topology

Fig. 2: Simulation Results and Analysis

close to each other under various value of δ. From the above three simulation results, we can conclude that the proposed three heuristic methods have the similar performance.

Algorithm	Average controllers
Incremental greedy	4
Primal-dual	7
Network partition	5

TABLE II: Comparison of Proposed Algorithms

For the average performance, the average number of controllers generated by each algorithm is computed among all 74 input topologies. Table II shows the average performance of each algorithm. The incremental greedy algorithm slightly outperforms the other two algorithms for all input topologies. For the response time, the incremental greedy algorithm and the primal-dual algorithm iteratively add in one controller until all switches have been served. Thus the number of iterations (or response time) is equal to the number of controllers generated by the algorithm. On the contrary, the network partition based algorithm first divides the network into partitions. Then each partition is placed with controllers simultaneously. Since a network partition is far smaller than the whole network, the network partition algorithm costs less response time than the other two algorithms.

VI. CONCLUSION

In previous works for controller placement, much concern has been focused on placing k controllers to optimize performance metrics such as propagation latency, load distribution, network reliability. However, in the actual applications, the number of controllers is unknown beforehand. Moreover, the quality of service (QoS) is always a primary concern of the network operators in the placement of SDN controllers. For this reason, we introduce the QoS-Guaranteed Controller Placement problem. Three heuristic algorithms are proposed to solve this controller placement problem with the QoS requirement. The proposed heuristic methods are compared on the Internet Topology Zoo. The simulation results demonstrate that the proposed incremental greedy method slightly outperforms the other two methods.

REFERENCES

[1] 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.

[2] 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.

[3] 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*. USENIX Association, 2010, pp. 3–3.

[4] 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.

[5] S. Schmid and J. Suomela, "Exploiting locality in distributed sdn control," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 121–126.

[6] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed sdn control layer," in *Networking Conference, 2014 IFIP*. IEEE, 2014, pp. 1–9.

[7] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," 2014.

[8] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 31–36.

[9] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in sdn-based core networks," in *Teletraffic Congress (ITC), 2013 25th International*. IEEE, 2013, pp. 1–9.

[10] S. Shenker, "Specification of guaranteed quality of service," 1997.

[11] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765–1775, 2011.

[12] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[13] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 248–260, 1975.

[14] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *Communications Magazine, IEEE*, vol. 32, no. 3, pp. 70–81, 1994.

[15] D. Gross, *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[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] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter." in *HotNets*, 2009.