QoS-Guaranteed Controller Placement in SDN

Tracy Y. 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 routers, 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 routers 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 three methods have the similar performance 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 [10]. It uses the network controller to control the routing behavior of routers or switches, so routers can focus on data forwarding. In a wide area network (WAN), it requires multiple physically dispersed controllers in the network, each of which manages and operates the routers in its local area. There are some works regarding the design and implementation of distributed systems of SDN controllers, such as ElastiCon [2], Onix [9] and HyperFlow [15].

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 routers and controllers. In a WAN, the network delay between a router to its controller becomes non-negligible and this delay affects the response time of controllers [6]. 2) load balancing among the controllers. Some controller could be overloaded while others are less busy. Therefore, it is an important research topic 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 routers to controllers is minimized. Several follow up works focused on the optimization of other performance metrics for placing k controllers in a network [12], [7], [17], [11], [5]. The work in [12] aims to minimize the communication delay between controllers. Load distribution among controllers is considered in [7]. The controller capacity is considered in [17], the network reliability is considered in [11], failure resilience is considered in [5], etc.

However, the issues of minimizing the number of controllers and the quality of service (QoS) have not been considered in the previous work. In network design, 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 routers to setup routing information of traffic flows, the response time of controllers is an important QoS parameter for network operations. The response time of a controller consists of two components: 1) the round trip of network delay between the controller and a router, 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 routers 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 [8], which is a database including publicly available network topologies. The simulation results show that the three methods have the similar performance 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.

II. Related Work

To address the controller placement problem, there are some research works considering different performance metrics such as node-to-controller latency [4], network reliability [11], load of controllers [17], inter-controller delay [7], failure resilience [5], etc.

The controller placement problem was firstly introduced by [4]. This work aims to minimize the average latency between controllers and routers. 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.

A fault tolerant controller placement was proposed in [11]. The author 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 [17]. 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 propagation delay among controllers is taken into consider in [7] for designing the control plane. A K-critical algorithm is developed to find the minimum number of controllers to satisfy a target communication between controllers and nodes, such as delay, latency, convergence time, etc.

Failure resilience is considered in [5] for placing controllers. This work concludes that whereas one controller is enough from a 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.

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 router 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 router 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 router to a controller, which is performance metric considered in our controller placement problem. Then we formulate the proposed QoS-guaranteed controller placement problem.

TABLE I
NOTATIONS

Notation	Meaning
V	set of routers
C	set of controllers
δ	upper bound of response time
r_i	the requesting rate of router i
t_i	response time of request from router i
$\bar{t}$	average response time of all routers
μ	service capacity of each controller
λ_j	traffic arrival rate of controller j
τ_j	mean service time of controller j
S_j	serving range of controller j
x_{ij}	variable indicates whether router i is assigned to controller j
y_j	variable indicates whether controller candidate j is open
a_j	variable indicates whether set S_j is selected
b_i	dual variable of router i

The network is modeled by graph $G(V, E)$, where V represents network routers, E represents the network links between nodes. Each router $i \in V$ is associated with a request demand r_i. Each router's location is also a candidate site for placing a controller. Suppose all controllers have the same processing capacity μ. Our goal is to find the minimal subset $C \in V$ to place a controller in each node of C, such that the QoS requirement is met.

The main job of a controller in SDN is to set up routing information for flows upon request from routers [16]. 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 routing information to the router to help set up the flow table for the new flow of this packet [13]. The response time of a request sent from a router is defined as the time from the moment of sending out this request until the reply is received from its controller. Given a 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 routers is within the given bound δ.

We introduce two variables, X and Y, to respectively denote the assignment of routers to controllers and the placement of controllers. For router i and controller j, $x_{ij} = 1$ if router 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 router 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 queueing model. Suppose packets arrive according to a Poisson process and the processing times of the packets are independent and identically exponentially distributed. Let λ_j denote the total arrival rate of packets at the controller j, which is represented as:

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

where r_i is the request rate of router i.

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

According to the queueing theory [3], the expected mean service time of controller j is defined as:

$$\begin{aligned} \tau_j &= \frac{1}{\mu - \lambda_j} \\ &= \frac{1}{\mu - \sum_i r_i x_{ij}} \end{aligned} \tag{2}$$

For each router i, let t_i denote the response time of its request, which can be represented as:

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

where t_{ij} denotes the network delay between router i and controller j. We assume the request and reply between the router 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 defined as:

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

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: $\bar{t} \le \delta$.*

IV. Heuristic Algorithm

Each router's location is a candidate site for placing a controller. We compute the service range of each candidate site. The service range of a candidate site include all routers whose requests have a response time less than or equal to δ from this controller candidate, which guarantees the QoS requirement. Let S_j denote the set of routers within the service range of candidate site $j \in V$. Our proposed QoS-guaranteed controller placement problem can be described as: Given a set V of n routers, a collection of subsets of V: $S = \{S_1, ..., S_n\}$, find a minimum subcollection of S that serves all routers of V. To solve this problem, we design and compare three heuristic algorithms: incremental greedy algorithm, primal-dual-based algorithm and network-partition-based algorithm.

A. Incremental Greedy Algorithm

The incremental greedy algorithm iteratively picks the largest set and removes the served routers. In each iteration, open the controller candidate j associated with the selected set S_j. Routers within the set S_j are assigned to this controller j. This iteration is repeated until every router is served by one controller.

The detail of greedy algorithm is illustrated in Algorithm 1. At the begining, we compute the service range of each candidate site. Suppose each candidate site is placed with a controller, the service range of a cadidate site include all routers whose requests have a response time less than or equal to δ from the controller at this candidate site. In each iteration, the candidate site with the largest service range is chosen to place a controller. Unserved routers within its service range are assigned to this controller. This iteration is repeated until all routers are being served.

Algorithm 1: Incremental Greedy Algorithm

```
Input: Graph $G = (V, E)$, Response time bound $\delta$;
Output: $X, Y$;
1  $R \leftarrow V$, $C \leftarrow V$ ;
2  $y_j = 0, \ \forall j \in C$ ;
3  $x_{ij} = 0, \ \forall i \in R, j \in C$ ;
4  while $R \neq \emptyset$ do
5      foreach $j \in C$ do
6          Compute the service range of candidate site $j$;
7      Find candidate site $j$ which has the largest service
       range and set $y_j = 1$;
8      $C \leftarrow C - \{j\}$ ;
9      foreach router $i$ within the service range of $j$ do
10         $x_{ij} = 1$ ;
11         $R \leftarrow R - \{i\}$ ;
12 return $X, Y$
```

B. Primal-Dual-based Algorithm

The incremental greedy algorithm does not have a performance guarantee for all kinds of inputs. It's performance often varies on different input. The primal-dual schema is used for designing approximation algorithms for integer programming problems, which provides a approximation ratio compared with the optimal solution. The basic idea is to consider the LP-relaxation of the original integer program as the primal program. We derive the dual program of the primal program. Then we settle down the complementary slackness condition of the primal program and the dual program respectively. In the primal-dual schema, 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.

To formulate the proposed controller placement problem as an integer program, let us assign a variable a_j for each set $S_j \in S$, which is allowed to be 0/1 values. The set S_j represents the set of routers within the service range of controller candidate j. a_j is set to 1 iff set S_j is selected, 0 otherwise. For each router $i \in V$, it is required that at least one of the sets containing it be selected. The controller placement problem can be formulated as:

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

The LP-relaxation of this integer program is obtained by letting the domain of variables a_j be $a_j \geq 0$, which is the primal program. We introduce a variable b_i for each router $i \in V$, we obtain the dual program.

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

The complementary slackness conditions are:

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

Define the frequency of an router to be the number of sets it is in. f denotes the frequency of the most frequent router. Set S_j is said to be tight if $\sum_{i:i \in S_j} b_i = 1$. The primal variable a_j is incremented integrally, which means to pick only tight sets in the system.

The primal-dual algorithm proceeds in iterations. In each iteration, we pick an unserved router, say i, and raise b_i until some set goes tight. Pick all tight sets and open the related controller candidates. For each tight set, assign routers within the set to the related controller. For routers 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 routers are updated. This iteration is repeated until all routers are served. This primal-dual algorithm achieves an approximation factor of f. 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 1, 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 routers $N_1, N_2, N_3, ..., N_9$ 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 routers are being served by at least one controller.

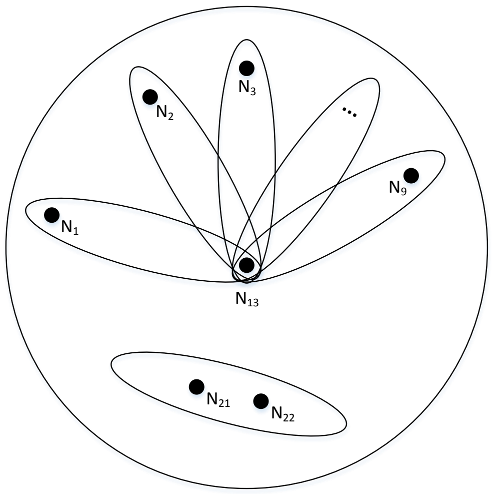

Fig. 1. An expample of applying primal-dual-based algorithm

Algorithm 2: Primal-Dual-based Algorithm

```
Input: Graph $G = (V, E)$, Response time bound $\delta$;
Output: $X, Y$;
1  foreach $j \in V$ do
2      compute the service range and $S_j$ ;
3  $S = \{S_j, \forall j \in V\}$ ;
4  $a_j = 0,\ \forall j \in S$ ;
5  $b_i = 0,\ \forall i \in V$ ;
6  while $V \neq \emptyset$ do
7      Pick an unserved router, say $i$, and set $b_i \leftarrow b_i + 1$;
       Find all sets that go tight ;
8      foreach tight $S_j$ do
9          Set $a_j = 1$ and open the controller $j$ ;
10         Assign all routers within $S_j$ to controller $j$ ;
11         $V \leftarrow V - \{i | i \in S_j\}$ ;
12 return $X, Y$
```

C. Network-Partition-based Algorithm

The above greedy algorithm and the primal-dual algorithm don't 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 which considers the load balancing among all controllers.

The basic idea of network-partition-based method is to divide the network into partitions and place controllers in each partition seperately. First, the network is divided into partitions. In each partition, the largest propagation delay between any two routers is equal to 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 partition, the load distribution among all controllers is balanced.

The detail of the network-partition-based algorithm is described in Algorithm 3. First, the network is divided into grids. Any pair of routers within each grid has a communication delay less than δ. In the network partitioning, we first divide the network into vertical strips, where each strip is left closed and right open. In each strip, the propagation delay between the node on the left boundary and the node on the right boundary is equal to δ. 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. 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 $\delta$;
  Output: $X, Y$;
1 $R \leftarrow V$, $C \leftarrow V$ ;
2 $y_j = 0, \ \forall j \in C$ ;
3 $x_{ij} = 0, \ \forall i \in R, j \in C$ ;
4 Divide the network graph into vertical strips ;
5 foreach vertical strip do
6     Divide the vertical strip into horizontal grids ;
7 foreach grid do
8     Use the incremental greedy algorithm to place
      controllers ;
9 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 simultation results and analysis.

A. Simulation Setup

The network topologies used for simulation are collected from a publicly available repository, namely the Internet Topology Zoo [8]. 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 [8] 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 router in our scenario. In addition, we consider each PoP as a candidate for hosting one controller. Each router is assigned to one controller in the topology graph G.

The simulation parameters are set as follows. According to a flow setup rate measurement test [1] 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 routers. Therefore the requesting demand of each router in our scenario is set as $r_i \in [1500, 3000]$ per second. A study [14] 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 router 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

The performance of the proposed three heuristic methods are compared using the same dataset. The simulation results are shown from Figure 2 to Figure 4. 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.

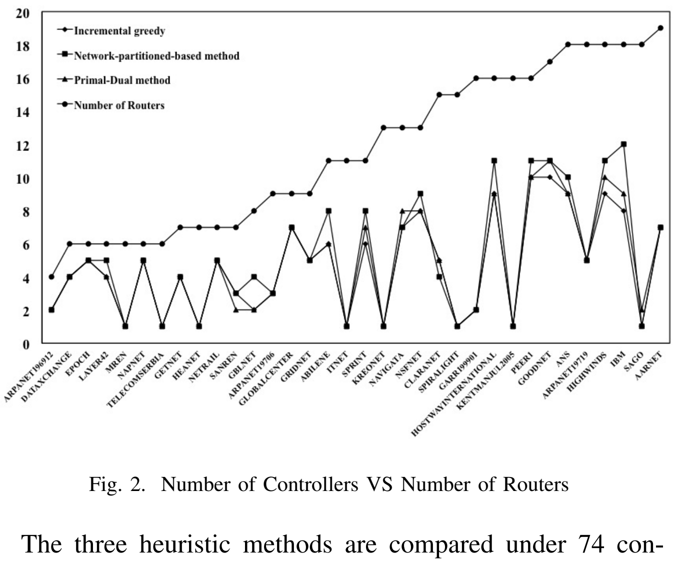

Fig. 2. Number of Controllers VS Number of Routers

The three heuristic methods are compared under 74 connected WAN topologies, simulation results are shown in two figures. Figure 2 and Figure 3 show the number of controllers output by the three methods and the total number of routers of each network topology. The number of routers for each WAN topology ranges from 4 to 74. As it is shown in Figure 2 and Figure 3, the number of controllers generated by the

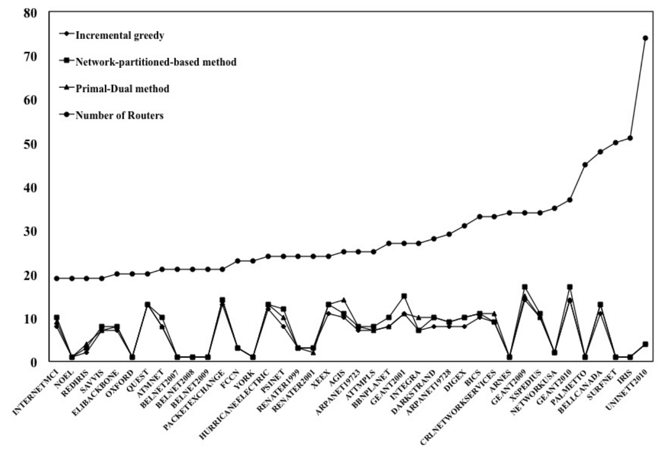

Fig. 3. Number of Controllers VS Number of Routers

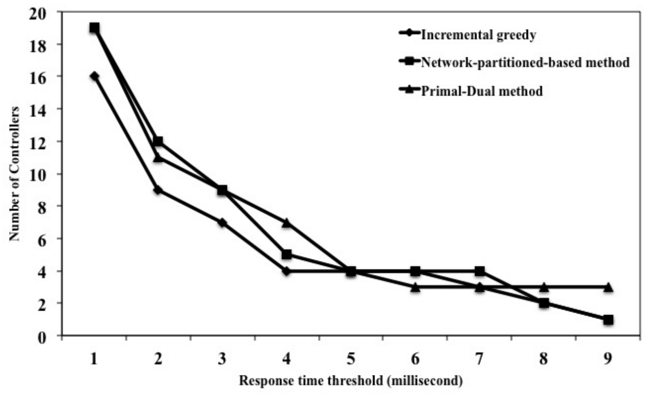

Fig. 4. Comparison results under different value of δ in one topology

three heuristic methods are very close to each other for all input topologies.

Figure 4 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 4. We can see that the performance of the three methods are very 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.

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: Given a network topology and a response time bound, how many controllers need to be instantiated, where they are deployed, and which network nodes are under control of each of the controller, in order to guarantee the response time between a node and its subscribed controller is less than the threshold. We propose three heuristic algorithms to solve this controller placement problem with the required response time constraint. The proposed heuristic methods are compared on the Internet Topology Zoo, a public network topology dataset. The simulation results demonstrate that the proposed three heuristic methods have similar performance on all input topologies.

References

[1] 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*, volume 41, pages 254–265. ACM, 2011.

[2] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an elastic distributed sdn controller. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 7–12. ACM, 2013.

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

[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*, pages 7–12. ACM, 2012.

[5] 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*, pages 1–9. IEEE, 2013.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[7] 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*, pages 1–9. IEEE, 2014.

[8] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765–1775, 2011.

[9] 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*, volume 10, pages 1–6, 2010.

[10] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. 2014.

[11] 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*, pages 31–36. ACM, 2014.

[12] 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*, pages 121–126. ACM, 2013.

[13] S. Sezer, S. Scott-Hayward, P.-K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013.

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

[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*, pages 3–3. USENIX Association, 2010.

[16] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, volume 54, 2012.

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