Hindawi
Journal of Control Science and Engineering
Volume 2022, Article ID 1047898, 12 pages
https://doi.org/10.1155/2022/1047898

Research Article

GravCPA: Controller Placement Algorithm Based on Traffic Gravitation in SDN

Chenhui Wang,[1,2] **Hong Ni**,[1,2] **and Lei Liu**[1,2]

[1]*National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China*
[2]*University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Lei Liu; liul@dsp.ac.cn

Received 21 December 2021; Revised 10 February 2022; Accepted 4 March 2022; Published 27 March 2022

Academic Editor: Daniel Morinigo-Sotelo

Software-defined network separates the control plane and the data plane, making the network more flexible. With the expansion of the network scale, one centralized controller cannot meet the latency needs of large-scale networks. Therefore, it is necessary to use multicontroller architecture, which has some problems with the controller placement. In this article, we take both the average latency and the worst latency between switch and controller into consideration and make a multi-objective optimization model. An improved label propagation algorithm based on traffic gravitation is proposed to solve the subdomain division problem, and a heuristic method is for subdomain controller placement. The simulation experiments show the effectiveness of the proposed algorithm and the time complexity guarantee for large-scale networks.

1. Introduction

Software-defined network (SDN) provides extremely flexible and customizable network services by separating the control plane and the data plane. However, with the expansion of network scale and data traffic, centralized architectures cannot fulfill the needs of efficiency, scalability, and availability [1]. In that case, there should be several distributed synchronous controllers in the SDN network. It has some problems that need to be solved, like how many controllers are needed in an SDN network, where should they locate, and what is the mapping of the controllers and the switches. Heller et al. [2] proposed the controller placement problem (CPP) for the first time and established a mathematical model for it, and the experiments showed that adding controllers for most networks would reduce the latency, but the benefits would decrease with the increase of controllers.

Most research regards CPP as a clustering problem, in which case the switches are regarded as the nodes with community attributes. However, CPP and traditional community detection [3] are not exactly the same. The nodes in the latter have strong connections with neighbors but weak connections with distant nodes, which is different from the forwarding devices in the network. Although the two network nodes are far apart (cannot share a controller), a large amount of data traffic can be exchanged between them. Therefore, in addition to considering the nature of the network topology itself, it is also necessary to increase the consideration of traffic demands.

The CPP has many different optimization objectives [4], such as network responsiveness, fault tolerance, resilience, and QoS. We focus on the responsiveness of the control plane, including the average and the worst latency between switch and controller, and solve the CPP in two steps. The first step is to solve the problem of controller number and mapping, by dividing the network topology into multiple fully connected subdomains through an improved label propagation algorithm (LPA) that uses the abstract gravitation between nodes as a value function. In the second step, we find the best controller placement position in each subdomain, using the gravitational force of nodes to the controller and a heuristic algorithm based on open searching. Our main contributions are as follows:

(i) A model that considers both the average latency and the worst latency is designed for the CPP, and a two-step heuristic algorithm is proposed to get the approximate optimal solution

(ii) We define the traffic gravitation using the traffic demands of the network and propose a subdomain division algorithm based on the LPA to determine the number of controllers and the mapping

(iii) By defining the space gravitation between nodes and controllers, a controller placement algorithm is proposed to determine the specific location of the controller in the subdomain

(iv) Experiments show that the proposed placement algorithm is effective and has less time complexity compared with some excellent algorithms

The rest of this article is organized as follows: Section 2 surveys the related work, showing the present research methods and their insufficiency. Section 3 optimizes the model of CPP with both average latency and worst latency and shows a brief of LPA. A two-step heuristic algorithm for the CPP is proposed in Section 4. Section 5 shows comparative simulations between the proposed algorithm and others. Finally, there are the future work in Section 6 and the conclusion in Section 7.

2. Related Work

There has been some research on the CPP, and some progress has also been made. The most common method is to optimize the proposed mathematical model through (integer) linear programming (ILP). Yao et al. [5] consider the controller's capacity for the first time, which relies on the k-center algorithm but adds capacity constraints. Finally, the problem is solved by integer linear programming. Simulation experiments show that the strategy can effectively reduce the number of controllers and the load of the busiest controller. Sallahi et al. [6] take the deployment cost as the optimization goal to determine the optimal number and the placement of controllers. The research uses linear programming to build the model, which is solved by a linear solver. Although this method is effective, it is very time-consuming and only suitable for static calculations in small networks. He et al. [7] select the forwarding devices in the network topology as an alternative location and complete the construction of a linear optimization model for the end-to-end data flow establishment time. They convert the problem into a mixed-integer linear optimization problem and adopt a linear optimizer, Gurobi [8], to solve it. It can be seen that the linear programming method has the advantages of strong versatility and optimization. But no matter what the optimization goal is, it requires a lot of prior knowledge, and its computational complexity is too high to be accepted in the network solution for dynamical adjusting in time.

The heuristic algorithm based on modularity is a common community detection method, which is also suitable for solving many CPP problems. Fan et al. [9] establish an optimization model considering both control latency and reliability, which uses an improved Louvain algorithm [10] based on modularity to calculate subdomains. Then, they use particle swarm optimization (PSO) to further calculate the placement of the controller in the subdomain. In order to solve the problem of resolution limit and subdomain disconnection caused by the Louvain algorithm, Traag et al. [11] propose a Leiden algorithm, which is based on the subdomain calculated by the Louvain algorithm and performs an internal subdivision. It combines the nodes with a certain probability and forms further-subdivided subdomains. Experiments show that the Leiden algorithm has more efficient subdomain partitioning capabilities while avoiding the problem of resolution limit and subdomain disconnection, and it is still applicable in the weighted graph. Chen et al. [12] adopt the same research steps. First, the Louvain algorithm based on modularity is used to calculate the subdomain, and then, the nodes with the smallest average and the smallest worst latency are found in the subdomains as the placement of the controller. A modularity-based heuristic algorithm is a common community detection algorithm, which has good results in community discovery. However, the CPP is not exactly the same as ordinary community detection, because of the traffic demands between network nodes. So when this type of algorithm is applied to the real network topologies, the performance is slightly unsatisfactory.

Another common method of community detection is the label propagation algorithm (LPA) [13], which is also widely used in CPP problems. CLPA is an algorithm for controller load balancing and network stability proposed by Liu et al. [14]. The network stability is abstracted from the reciprocal of the number of hops from the forwarding device to the controller. CLPA divides the network into different subdomains by using LPA and then uses the k-median algorithm [15] to calculate the placement of the controller in each subdomain. The simulation experiments show that its computational complexity and delay performance are higher than LPA.

Table 1 is a brief overview of the current and the proposed algorithms, where the SC-avg and SC-wst, respectively, represent the average and the worst latency between the switch and the controller, and MILP represents mixed-integer linear programming. And the remaining CPP research involves many optimization methods and goals, such as NSGA-II [16] based on the genetic algorithm and effective Pareto algorithm [17] based on the Nash bargaining model. Most CPP research sets the constant (or infrequently changing) attribute value of the nodes as the optimization goals, but the combination of the immutability of network topology and the variability of traffic demands is actually one of the difficulties of CPP [18]. For this reason, this article converts the characteristics of traffic demands into the attributes of network nodes as much as possible and proposes a controller placement algorithm that focuses on the influence of traffic, so that the final strategy is more in line with the real network topology.

Table 1: An overview of current controller placement approaches and the proposed approach.

Author	Objective(s)	Method	Tool(s)	Evaluation
Yao et al. [5]	The SC-wst	LP	Capacitated K-center	Fewer number of controllers, but is complex for large-size networks
Sallahi et al. [6]	Network cost	ILP	Linear solver	High complexity, only for the small size networks
He et al. [7]	Flow setting time	MILP	Gurobi [8]	Moderate complexity
Fan et al. [9]	Control latency and reliability	Modularity	Louvain and PSO [10]	Low complexity, but has subdomain disconnection problem
Traag et al. [11]	Control latency	Modularity	Leiden	Low complexity, fix the problem of the Louvain algorithm
Chen et al. [12]	The SC-avg or SC-wst	Modularity	Louvain, traverse search	Moderate complexity, well-done for subdomain division
Liu et al. [14]	Load balancing and stability	LPA	K-median	Much low complexity, but is bad for subdomain division with traffic
The proposed	The SC-avg and SC-wst	LPA	Heuristic algorithm	Much low complexity and is good for both SC-avg and SC-wst

3. Optimization Model

3.1. Problem Description. The SDN infrastructure is shown in Figure 1, and CPP is to calculate the optimal number, location, and mapping of controllers in the control plane when the data plane and related network parameters are known. As shown in Figure 1, the entire topology deploys two controllers, located at SW2 and SW4, respectively. The control domain of CTL A is \{SW1, SW2, SW3\}, and the control domain of CTL B is \{SW4, SW5\}.

3.2. Model and Symbol. This article uses the models of graph theory to abstract the optimization model. For a given network topology, the data layer is $G = (V, E)$, where $V = \{v_i\}_n$ is the set of network nodes, $|V| = n$ is the total number of nodes, and $E = \{e_{ij}\}_{n*n}$ is the set of network links. When $e_{ij} = 0$, there is no direct physical link between nodes v_i and v_j. The control layer is $C = \{v_i\}_k$, where k is the total number of controllers.

For a simple description, it is necessary to describe the mapping of switches and controllers using a specific data structure. First of all, the entire network topology is divided into several subdomains according to the control domain, presented as a dictionary structure subDomain = $\{l_i: \{v_0, v_1 \ldots v_{n_i}\}\}, i \in [0, k)$, where l_i is the label (unique identifier) of i subdomain and n_i is the number of nodes in i subdomain. We have

(1) Each switch only has one controller, that is,

$$\cup_{i=0}^{k-1} s_i = V, \tag{1}$$

$$s_i \cap s_j = \phi, \quad \forall i, j \in [0, k) \text{ and } i \neq j, \tag{2}$$

where $s_i = \text{subDomain}[i]$ is the node set of i subdomain. Equation (1) means that all nodes are divided into subdomains, and equation (2) means that the subdomains do not overlap each other.

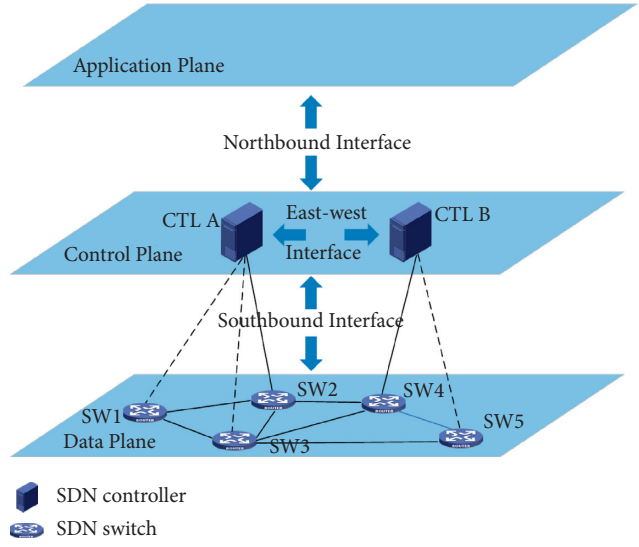

Figure 1: The infrastructure of SDN.

(2) Each subdomain only has one controller, and the controllers of all subdomains are different from each other. In this article, unless otherwise specified, the label of the subdomain and the subscript of the controller are considered to have a one-to-one correspondence, shown in the following equation:

$$s_i \leftrightarrow l_i \leftrightarrow c_i. \tag{3}$$

Then, we use traffic matrix $M = \{m_{ij}\}_{n*n}$ to describe the end-to-end traffic demands between nodes in network G, where m_{ij} is the total traffic that node v_i needs to send to node v_j in the initial state. Define $T_{fwd} = \{tf_i\}_n$ and $T_{snd} = \{ts_{ij}\}_{n*n}$, where tf_i is the total traffic actually forwarded by the node v_i and ts_{ij} is the total traffic actually required to be sent from v_i to v_j, shown as in the following equations:

$$tf_i = \sum_{m_{sd} \in M} m_{sd} \cdot \delta_{sd}(i), \quad \forall i \in [0, n), \tag{4}$$

$$ts_{ij} = \sum_{m_{sj} \in M} m_{sj} \cdot \delta_{sj}(i), \quad \forall i, j \in [0, n), \tag{5}$$

$$\delta_{sd}(i) = \begin{cases} 1, & \text{if } v_i \in \text{Path}[v_s][v_d], \\ 0, & \text{else}, \end{cases} \tag{6}$$

where $\text{Path}[v_s][v_d]$ is the routing path from v_s to v_d.

Finally, summarize the optimization model of controller placement. We focus on the response latency between switches and controllers, in which the average latency (represented by SC-avg) can reflect the basic performance of propagation in the SDN network and the worst latency (represented by SC-wst) can reflect the performance under strict constraints [19]. The optimization problem for optimizing these two goals can be obtained as

$$\min \alpha L_{avg} + \beta L_{wst}, \tag{7}$$

$$L_{\text{avg}} = \frac{1}{n} \sum_{v \in V} d(v, c_i) | v \in s_i, \tag{8}$$

$$L_{wst} = \max_{v \in V} d(v, c_i) | v \in s_i, \tag{9}$$

$$\text{s.t. } \alpha + \beta = 1, \tag{10}$$

$$\text{len}(\text{subDomain}) = k \quad \text{satisfy equation (1)(2)(3)}, \tag{11}$$

where $d(v, c_i) | v \in s_i$ is the latency from the node v to the controller c_i of its subdomain. Equations (8) and (9) are the calculation formulas for the average and the worst latency. Equation (10) is the weight constraint, and equation (11) is the controller number constraint.

3.3. Brief of LPA. Since the subdomain division algorithm is based on LPA, in order to ensure the continuity of the article, it is necessary to briefly introduce LPA. LPA is originally applied to solve the problem of community detection. Because of its simple idea and approximately linear time complexity; meanwhile, the result does not depend on the initial solution, and it is widely used in various fields. The main process is

(1) Initialize the network; each node is regarded as an independent community and marked with a globally unique label

(2) Randomly select some nodes and update the node label to the label with the largest value of the evaluation function among its neighbors

(3) Iterate the second step until there are no more changes to the label

When LPA converges, nodes with the same label belong to the same community. The limitations of LPA are dichotomous oscillation, unstable results of multiple calculations, and resolution limitations [11]. These problems are solved in Section 4.1 using some restrictions.

4. Solution of the Problem

For the optimization goal of Section 3.2, a controller placement algorithm is proposed, which aims at minimizing the average latency and the worst latency between switches and controllers as much as possible in a large-scale network. As analyzed above, CPP needs to be divided into two parts, which are the subdomain division of the network and the placement of the controller in each subdomain. Therefore, algorithms are proposed to solve these two subproblems based on the node's traffic gravitation and space gravitation, respectively. The flow diagram of the whole algorithm is shown in Figure 2, and the details are introduced in Sections 4.1 and 4.2.

4.1. Subdomain Division Algorithm. In community detection, there are some indicators to measure the importance of nodes, such as degree centrality [20], betweenness centrality [21], and LeaderRank value [22]. However, these indicators cannot fully measure the importance of nodes in the field of communication networks, in which one of the main differences is the traffic demands. We use T_{fwd}, the total traffic forwarded by nodes, to present the importance of nodes. Furthermore, inspired by the betweenness centrality, the out-degree $d_{\text{out}}(v)$ is selected to be the penalty factor. Combining the results of several rounds of simulation experiments, the importance of node I_v is defined in the following equation:

$$I_v = \frac{tf_v}{\sqrt{d_{\text{out}}(v)}}, \quad \forall v \in V. \tag{12}$$

Influenced by traffic demand, we believe that there is a mutual attraction relationship between nodes. The more traffic that needs to be transmitted between two nodes, the greater the possibility of belonging to the same subdomain. Therefore, through analogy with the concept of gravity, the traffic T_{snd} actually transmitted between nodes is defined as the quality of the nodes, and the length hops_{ij} of the routing path between two nodes is defined as the distance. Therefore, the traffic gravitation $F_{traf}(v_i, v_j)$ between nodes is defined in the following equation:

$$F_{traf}(v_i, v_j) = \frac{ts_{ij}}{\text{hops}_{ij}^2}, \quad \forall v_i, v_j \in V. \tag{13}$$

Section 3.3 has briefly introduced LPA, which is improved in this section to get better performance. In the label propagation process of LPA, the update selection of nodes is random, which may increase the convergence time exponentially in the worst case. So we consider sacrificing a certain amount of randomness to greatly shorten the convergence time. When initializing the network topology, the traffic matrix is used to calculate the importance of each node, and the initial update queue is formed in descending order. In each round of updates, the node whose label

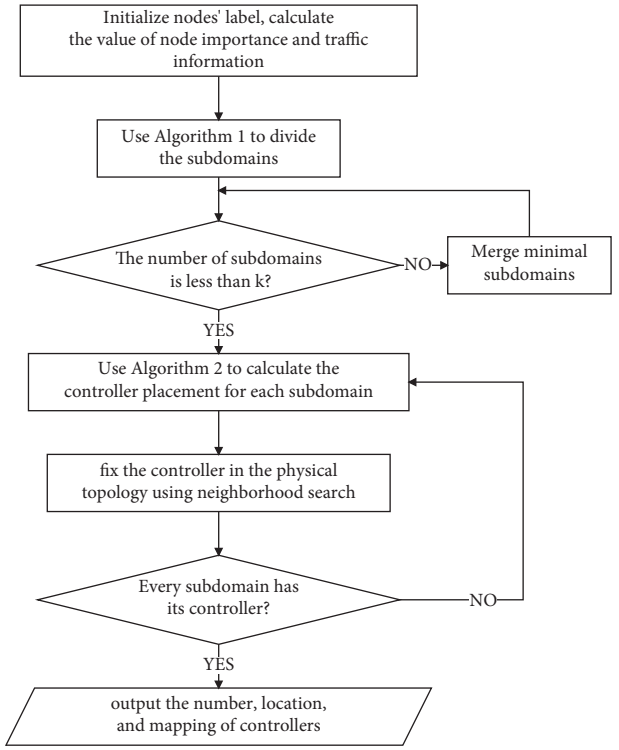

Figure 2: The flow diagram of the placement algorithm.

changes and its neighbors join the update queue again according to the node's importance, to wait for the next update round. This node selection strategy can effectively reduce the number of nodes that need to be processed during each update round in a large-scale network, thereby greatly reducing the algorithm convergence time. The complete process of the subdomain division algorithm (gravSDA) is described in detail through Algorithm 1, and Table 2 shows the meaning of the symbols used in gravSDA.

Algorithm 1, gravSDA, improves on LPA, in which the initialization phase completes the initialization of the node label l_i, the calculation of T_{fwd} and T_{snd}, and the initialization of the update queue Q_{update}. The entire iterative process of label propagation is completed by lines 1 to 11. Lines 3 to 10 are the core of gravSDA, which uses the traffic gravitation among neighboring nodes to update the label of the node; lines 7 and 10 indicate that the update queue is updated to the union of the nodes with the label change and its neighbors, and it is sorted in descending order according to I_v. In the iterative process, the maximum number of iterations maxIter is added to prevent the dichotomous oscillation that may occur. It should be noted that if the number of subdomains is larger than k in line 15, we will combine the smallest subdomain to its neighbor until there are just k subdomains. The flow diagram of gravSDA is shown in Figure 3.

GravSDA uses the network traffic matrix and the traffic gravitation between nodes as the standard for label update, so it is more in line with the requirements of subdomain division in the communication network topology compared with other graph theories and clustering algorithms. Its effectiveness for the real network is experimentally verified in Section 5.3. Furthermore, gravSDA improves the order and range of label updating on the basis of LPA, which greatly reduces the algorithm convergence time, and it is verified in Section 5.4.

4.2. Controller Placement Algorithm in Subdomain. With the results of Section 4.1, it can further study the placement of controllers in each subdomain. Since there is no strict correspondence between the data flow and the control flow [23], the traffic gravitation model is not suitable for controllers. Thus, we assume the following:

(1) All new data flows trigger packet-in message from the switch to the controller, and the total amount of transmitted data is equal

(2) The soft and hard timeout time of all flow tables is constant

(3) Although in-band communication is used between the switch and the controller, sufficient bandwidth is reserved under any circumstances to transmit control messages

Under the above assumptions, the latency of control messages is only determined by the length of the routing path, and the importance of the node depends on its out-in degree in a subdomain. Since the synchronization latency between the controllers needs to be considered, when the subdomain s_i is analyzed separately, the remaining subdomains should be regarded as virtual nodes to analyze the impact on s_i.

Inspired by the idea of force balance, CPP can be analogous to a scene where one particle is balanced by force and is fixed in a force field. First, select an initial position for the controller c_i, which can be any position in the space area. And then, it is affected by the gravitational force $\overrightarrow{F_g}$ of both nodes in the domain and virtual nodes outside the domain. The force $\overrightarrow{F_g}$ of the node to the controller is similar to the node's demand for the controller. When the controller is far away, the node "eagerly" wants the controller to be closer, so the calculation formula should be more similar to Hooke's law. We propose the calculation formula in the following equation:

$$\overrightarrow{F_g}(v_i, c, e) = \begin{cases} 0, & \text{if } \operatorname{dis}(v_i, c) < e, \\ \omega \cdot (\operatorname{dis}(v_i, c) - e), & \text{else}, \end{cases} \tag{14}$$

where e is the original length, that is, when the distance between nodes and controllers is less than e, the gravitational force is ignored, $\operatorname{dis}(v_i, c)$ is the Euclidean distance between node v_i and controller c, and ω is the coefficient of elasticity.

Finally, some special processing is added to ensure the time complexity and effectiveness of the controller placement algorithm (gravCPA). When selecting the initial position, choose the center of the smallest covering circle that contains all nodes in a subdomain, which can be obtained with the Elzinga–Hearn algorithm [24] in $O(n)$ time complexity. The controller may oscillate in the space force field, so every time in the iteration, the penalty factor φ is used to reduce the moving step length of the controller to ensure

Input: $G(V, E), M$, maxIter, k
Output: subDomain = $\{l_i: \{v_0, v_1 \ldots v_{n_i}\}\}$
Initialize: subDomain = $\{v_i$: set for v_i in $V\}$
T_{fwd}, T_{snd} calculate values using equations (4) and (5)
Q_{update} = sorted (V, key = lambda x $I_v(x)$)
Process
1 **while** is change and iterTimes < maxIter **do**
2 iterTimes+ = 1, $Q_{\text{new}} = \phi$
3 **for** node v_i in Q_{update} **do**
4 l_{new} = findMaxNeighbors (G, T_{snd}, v_i)
5 **if** $l_i \neq l_{\text{new}}$ **then**
6 update l_i and is change
7 Q_{new}.add $(v_i \cup G.\text{neighbours}(v_i))$
8 **end if**
9 **end for**
10 Q_{update} = sorted (Q_{new}, key = lambda x $I_v(x)$)
11 **end while**
12 **for** node v_i in V **do**
13 subDomain $[l_i]$.add (v_i)
14 **end for**
15 **return** subDomain
Function 1 findMaxNeighbors $(G, T_{snd}, v_i) \longrightarrow$ label:
16 res = $[-1, G.\text{Nodes}(-1)]$
17 **for** node v_j in G.neighbours (v_i) **do**
18 res = $[F_{\text{traf}}(v_i, v_j), v_j]$if $F_{\text{traf}}(v_i, v_j) > \text{res}[0]$
19 **end for**
20 **return** res[1].label if res[0] $\neq -1$ else v_i.label

Algorithm 1: Subdomain Division Algorithm (gravSDA).

Table 2: Symbols of Algorithm 1.

Symbol	Description
$Q_{\text{update}}/Q_{\text{new}}$	Node queue to be updated in this round of iteration/ next round
isChange	Flag indicated whether the label has changed in this iteration
G.nodes (i)	Construct an instance of the node numbered i in the network G

final convergence. Due to the open search, the final controller may place outside the network topology. In order to make gravCPA more general, it needs to search again in its surroundings to find an alternative placement in the topology. It is described in Algorithm 2 in detail, and Table 3 is a description of the symbols used in gravCPA.

The idea of Algorithm 2 is based on the space gravitation $\overrightarrow{F}_g$ of the node to the controller, which is firstly finding the optimal controller placement through open search and finally fixing the placement in the network topology. The open search process is lines 2 to 8, in which the resultant force in intradomain $\overrightarrow{F}_{\text{int}}$ is calculated by equation (14) and the resultant force extra-domain $\overrightarrow{F}_{\text{ext}}$ only considers the unit force formed by the relative positions. Each iteration penalizes the step length to ensure that the algorithm eventually converges. The surrounding search process is the 9th to 13th line, in which the placement is found through traversal in some subdomain nodes with suitable distance tolerance. The flow diagram of Algorithm 2 is shown in Figure 4.

5. Simulation

The performance and convergence time of gravCPA are simulated and evaluated on the x86 platform in this section. The operating system of the simulation experiment is Ubuntu 16.04.3 LTS, the processor model is Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40 GHz, and the physical memory is 16 GB. And in this section, we define the sum of SC-avg latency and SC-wst latency as the integrated latency for a simple description.

5.1. Influence of the Number of Controllers. Except for a few algorithms that depend on the initial conditions, other CPP algorithms can obtain the optimal number of controllers when running. In fact, the number of controllers cannot be increased without any upper limit. Based on this consideration, we limit the number of controllers and compare the proposed gravCPA with CTR [7], LDN [11], and CLPA [14]. The metrics are SC-avg latency and SC-wst latency in the network. Particularly, when the limited number is greater than the calculated number, the latter is selected.

We use Python 3.8.0 and NetworkX components for simulation and use a randomly generated LFR benchmark network that is fully connected. After 200 times repeated experiments, the average results are shown in Figure 5. The parameters in this experiment are shown in Table 4, where PLD means power law distribution.

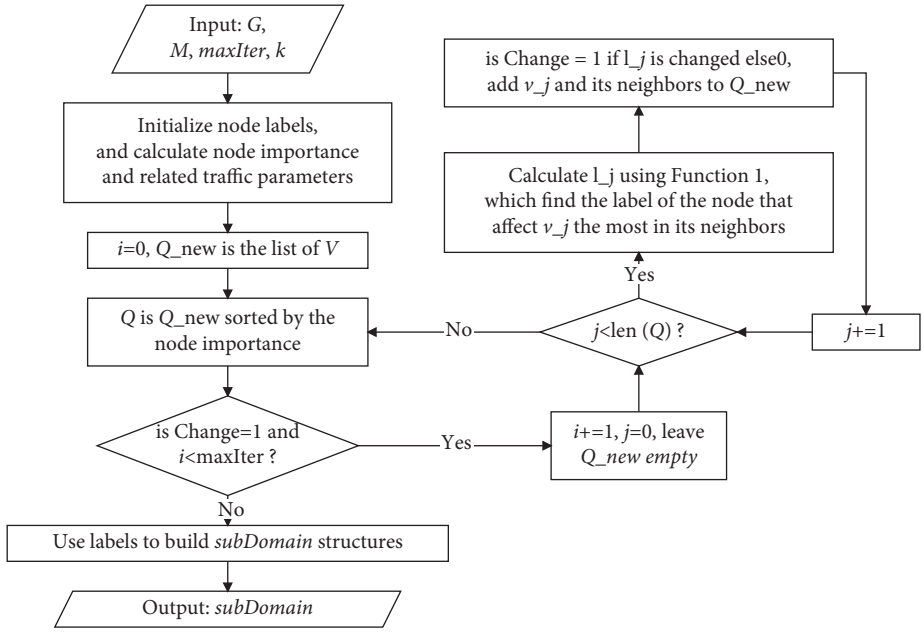

Figure 3: The flow diagram of Algorithm 1.

Input: $G(V, E)$, subDomain, $\varphi, \varepsilon, \delta$
Output: dictionary structure: $ctls = \{l_i: v\}$
Initialize: $ctls = \{l_i: \text{EH}(s_i) \text{ for } l_i, s_i \text{ in subDomain}\}$
Process

for s_i in subDomain **do**
  $\text{loc}_i = ctls[l_i]$
  while STEP $> \varepsilon$ and loc_i change **do**
    $\overrightarrow{F}_{\text{int}} = \text{sum}(\overrightarrow{F_g}(v_j, \text{loc}_i, e) \text{for } v_j \text{ in } s_i)$
    $\overrightarrow{F}_{\text{ext}} = \text{sum}\left(\begin{array}{c} (v_j.x - \text{loc}_i.x, v_j.y - \text{loc}_i.y)/\sqrt{v_j.x - \text{loc}_i.x^2 + v_j.y - \text{loc}_i.y^2} \\ \text{for } v_j \text{ in } V_{\text{ext}}(s_i) \end{array} \right)$
    $\text{loc}_i \leftarrow \text{loc}_i + (\overrightarrow{F}_{\text{int}} + \overrightarrow{F}_{\text{ext}}) \cdot \text{STEP}$
    STEP $* = \varphi$
  end while
  $v_{\text{res}} \leftarrow$ find the closest node to loc_i
  $D = \text{dis}(v_{\text{res}}, \text{loc}_i)$
  for v_j in \{v for v in s_i if $\text{dis}(v, \text{loc}_i) < D + \delta$\} **do**
    $v_{\text{res}} = v_j$ if calOptFunc $(v_j) <$ calOptFunc (v_{res})
  end for
  $ctls[l_i] = v_{\text{res}}$
end for
return $ctls$

Function 2 calOptFunc $(v) \longrightarrow$ int:
  res$\leftarrow$calculate the value of equation (7)
  return res

Algorithm 2: Controller Placement Algorithm (gravCPA).

Figure 5 shows the response latency curve of the network control plane when the number of controllers changes under gravCPA and three comparative algorithms. Figure 5(a) shows the change of SC-avg latency, and it can be seen that gravCPA has better performance on SC-avg than other CPP algorithms. When the number of controllers $k = 5$, gravCPA's SC-avg is 233.86 μs, which is 21.1% lower than CLPA's 296.52 μs and 32.6% lower than CTR's 347.1 μs; when $1 \le k \le 6$, the SC-avg of gravCPA is lower than LDN, but it is a little worse when $k > 6$. Combined with Figure 6, it can be seen that 95% of k is less than 8.63 and 80% is less than 7.03 for gravCPA,

Table 3: Symbols of Algorithm 2.

Symbol	Description
φ	Factor for STEP penalty
ε	Iteration accuracy for STEP
δ	Factor for distance tolerance
EH (s_i)	Function that calculates the minimum covering circle's center of s_i using Elzinga–Hearn [24]
STEP	Step length of controller movement
$\overrightarrow{F}_{\text{int}}/\overrightarrow{F}_{\text{ext}}$	Resultant force of intra-/extra-domain nodes
$V_{\text{est}}(s_i)$	Set of subdomain abstract nodes except s_i

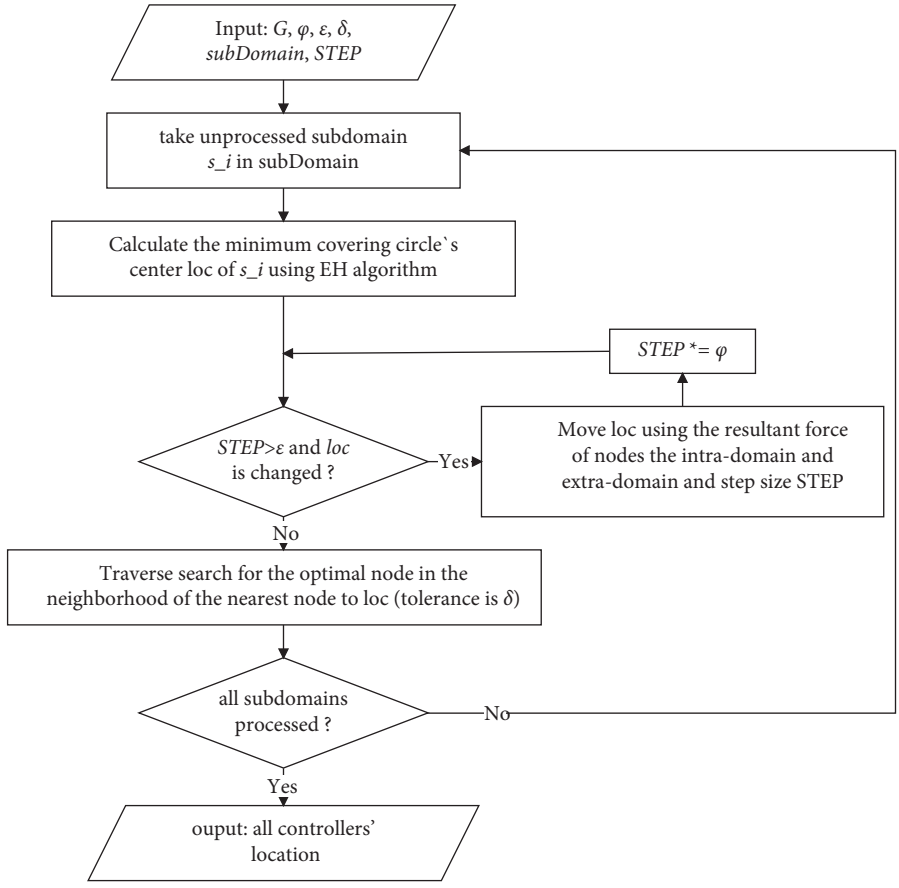

Figure 4: The flow diagram of Algorithm 2.

while these two values for LDN are 10.25 and 9.24, respectively. In another word, the controller's number of gravCPA is not more than 7 in most cases. When k increases in the gravCPA experiment, the number of controllers does not increase actually so that the SC-avg latency does not decrease. In summary, gravCPA can obtain a better average latency with a smaller number of controllers.

Figures 5(b) and 5(c) show the curve of the worst latency and the integrated latency, respectively. It can be seen that these two parameters of gravCPA are better than other algorithms. Furthermore, based on the numerical error of 4.56%, gravCPA converges when $k = 7$, and at this time, the integrated latency 867.62 μs decreases by 7.0%, 8.1%, and 16.8%, respectively, compared with LDN's 932.68 μs, CLPA's 943.82 μs, and CTR's 1042.6 μs.

5.2. Cumulative Distribution Function of Latency. Although each CPP algorithm wants to make an optimal placement for every network as much as possible, its performance varies with the network parameters and situation. We use a randomly generated network topology with full connection to verify the versatility, where the number of nodes is $n \in [12, 200]$ and the number of links is $m \in [n-1, n*(n-1)/2]$. The cumulative distribution function of the integrated latency of gravCPA is compared with CTR, LDN, and CLPA.

We also use Python 3.8.0 and NetworkX components for simulation and use Seaborn components to generate the cumulative distribution function curve from the results of 200 repeated experiments. The results are shown in Figure 7.

Figure 7 shows the cumulative distribution function of the integrated latency of four CPP algorithms in a

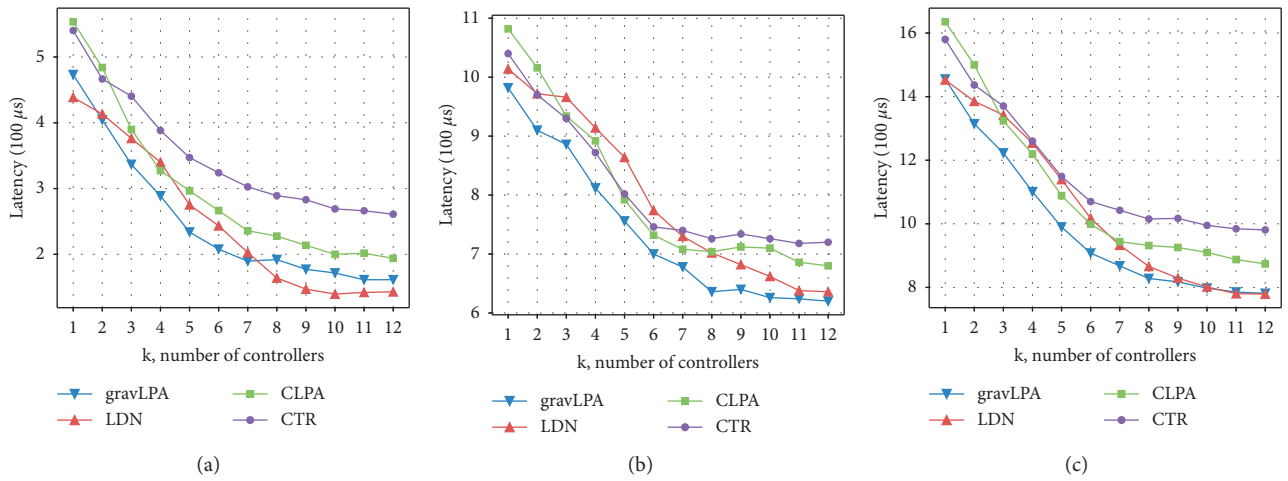

Figure 5: Response latency curve of controllers' number k.

Table 4: Simulation parameters settings.

Parameter	Value	Description
n	100	Number of nodes
$d_{\min}$	2	Minimum degree of nodes
τ_1	3	PLD index for the degree
τ_2	1.5	PLD index for the community size
μ	0.1	Fraction of intercommunity edges
$\alpha, \varphi, \varepsilon, \delta$	0.5, 0.99, 10^{-6}, 3 units	Parameters of gravCPA

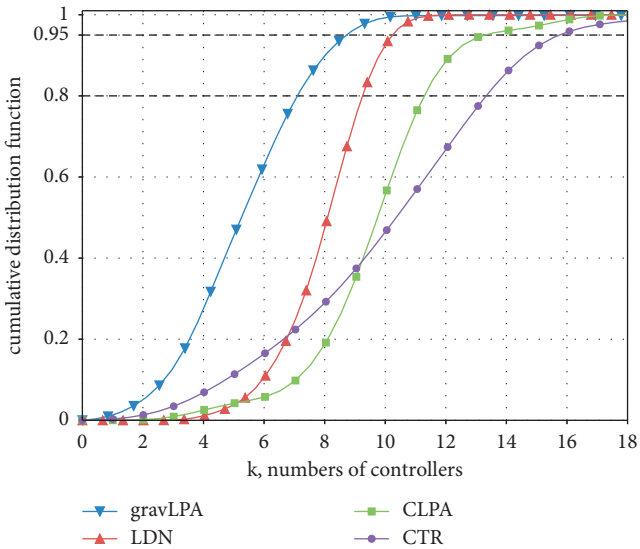

Figure 6: The cumulative distribution function curve of controllers' number without k-limitation.

randomly generated network. When $k = 3$, as shown in Figure 7(a), the integrated latency performance of gravCPA is significantly better than the other three algorithms. Compared with CLPA that has the second best performance, when the confidence levels are 95% and 80%, the integrated latency of gravCPA decreases by 26.2% (from 1789.11 μs to 1320.58 μs) and 18.8% (from 1420.98 μs to 1153.25 μs), respectively.

Comparing Figures 7(b) and 7(c), it can be seen that when $k > 6$, the integrated delay of gravCPA no longer decreases with the increase of k. Relatively, the other three algorithms have different improvements and the performance of LDN is improved the most. However, when $k = 12$, the integrated latency of gravCPA (1084.46 μs) is just slightly larger than LDN (972.9 μs), which is about 111.4% of LDN.

In summary, it is concluded that gravCPA can still achieve better performance of integrated latency with a small number of controllers when faced with the randomly generated network.

5.3. Performance in Real Networks. Sections 5.1 and 5.2 show the latency performance of the four algorithms in the LFR benchmark network and the randomly generated network. Since the traffic demands between nodes in the real network are not exactly the same, it is more complicated for research. In this section, we use the real network topologies and traffic demands [25] to compare the latency performance of each algorithm.

We use the real topologies in Table 5 simulated by Python 3.8.0 and NetworkX components. Assume that the first data packet of each flow triggers the packet-in message, and the transmission latency of the control message is the sum of each hop latency from the node to the subdomain controller, where the hop latency is equal to the reciprocal of one-tenth of its bandwidth. The experimental results are shown in Figure 8.

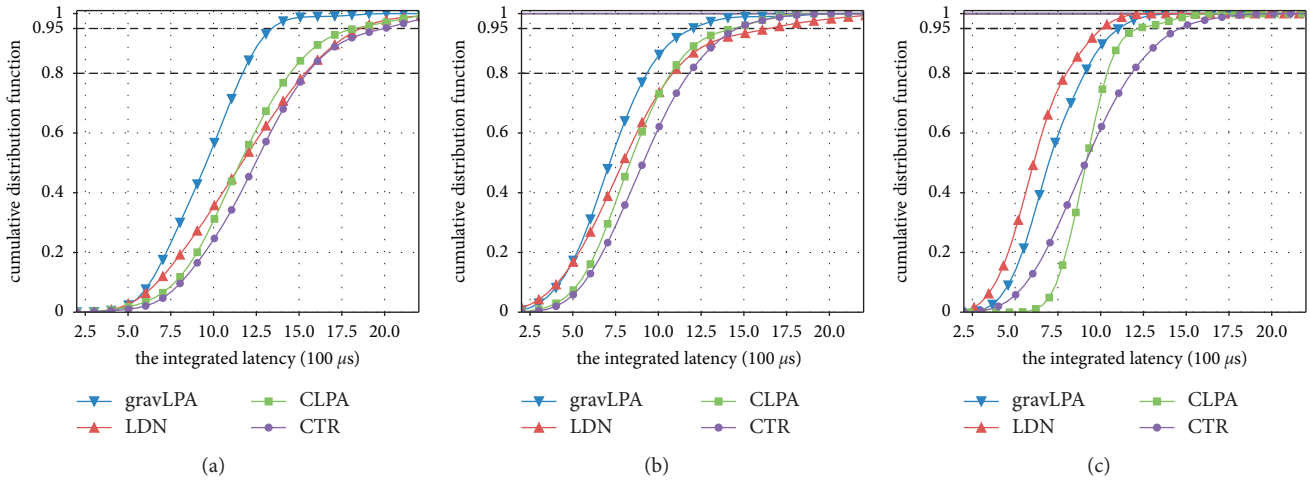

Figure 7: The cumulative distribution function curve of the integrated latency.

Table 5: Topologies in the experiment.

Topology	Nodes	Links	Demand pairs
Abilene	12	15	132
Cost266	37	57	1332

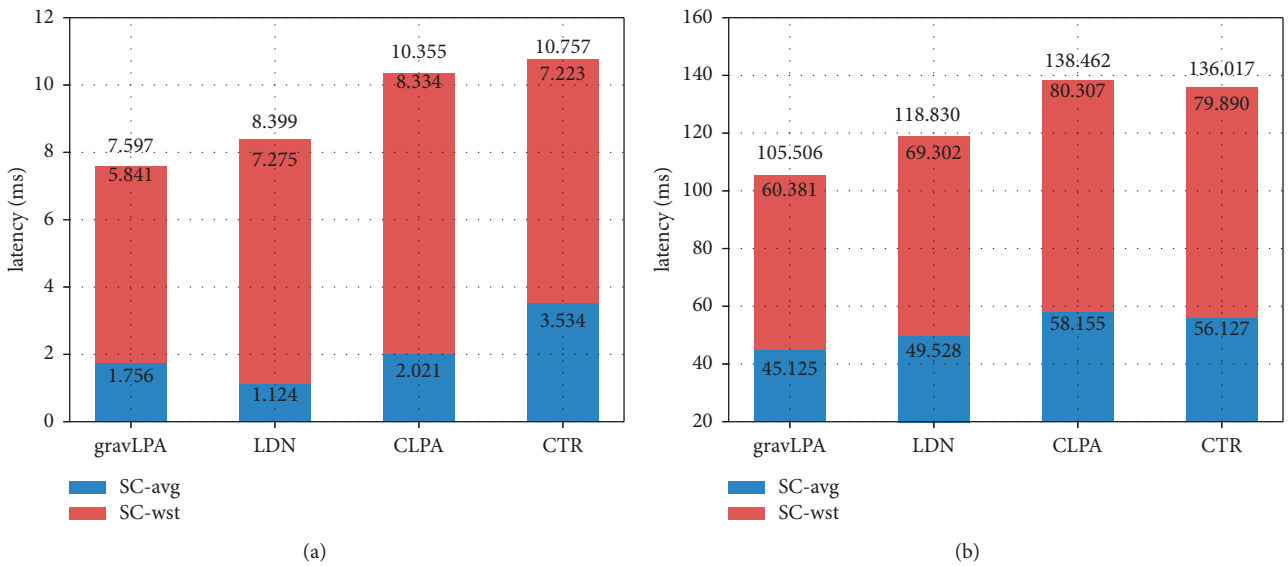

Figure 8: The latency of CPP algorithms in real networks.

Figure 8(a) shows the simulation result of the Abilene network with 12 nodes, which is representative of the small-scale network. It can be seen that although the SC-avg latency of gravCPA is slightly larger than LDN, the remaining parameters are all better than the other three algorithms. The integrated latency of gravCPA is only 7.597 ms, compared with 8.399 ms of LDN, 10.355 ms of CLPA, and 10.757 ms of CTR, which are reduced by 9.54%, 26.63%, and 39.37%, respectively. In a word, the gravCPA based on traffic gravitation has a stronger optimization performance when considering the real traffic demands.

Figure 8(b) shows the simulation result of the Cost266 network with 37 nodes, which is a medium-scale network. Compared with Alibene, the traffic demands in large- and medium-scale networks are more complex, and better performance cannot be obtained without considering the impact of traffic. When gravCPA restricts the number of controllers in Cost266, the integrated latency is 60.381 ms. Compared with LDN's 69.302 ms, CLPA's 80.307 ms, and

Table 6: Convergence time (ms) of four algorithms.

Nodes number	10	50	100	500	1,000	5,000	10,000
gravCPA	0	1.137	3.794	17.784	35.453	929.28	2070.218
LDN	0	1.099	5.191	15.399	52.202	1170.729	4725.17
CLPA	0	2.592	14.956	71.583	194.544	3616.096	TLE
CTR	0	1.027	16.74	202.514	9152.028	TLE	TLE

CTR's 79.89 ms, it decreases by 12.87%, 24.81%, and 24.42%, respectively.

5.4. Convergence Time. Although this article does not focus on the dynamic placement, time complexity will seriously affect the universality of the algorithm faced with large-scale network topologies. Thus, we increase the scale of the network to compare the convergence time of each algorithm. The results are shown in Table 6, which cannot be shown in a line or bar figure because of the serious nonlinear growth of the value. It should be noticed that the convergence time includes the time of subdomain division and controller placement, and the TLE in Table 6 means the code is the time limit (5 minutes) exceeded.

Although there may be errors in code optimization, the experimental results are in line with expectations. GravCPA and LDN are based on LPA and Louvain algorithms, respectively, which improve the iterative operations and have good time complexity. However, CLPA does not consider calculation acceleration, while CTR is an algorithm based on linear programming. It is difficult for the two to complete the calculation in a short time with a larger network.

6. Future Work

6.1. Flow Types. In traffic engineering, data flow is usually divided into two types, elephant flow (bulk data transfer), and mice flow (short-lived data exchange) [26]. These two types have completely different effects on the control messages between the switch and the controller. The former has huge data transmission but only exchanges little control messages, while the latter is totally opposite. In this article, the placement strategy is only based on the size of the traffic demands, the performance will decrease when facing extreme conditions. We consider completing the controller deployment strategy based on the predicted flow type in the next step.

6.2. Stability. We have noticed that both LPA and gravitation models have problems with oscillations and nonunique results. Although we have added a lot of assumptions and constraints, there is still a problem that the placement strategy results are not unique in specific or complex scenarios. And this part will become the direction of follow-up research.

7. Conclusion

In this article, we focus on the CPP in the SDN multicontroller architecture. Different from many research, we optimize the average latency and the worst latency using the data plane traffic demands. For the subdomain division problem, the traffic gravitation is defined and an improved LPA is designed accordingly. On the other hand, for the subdomain controller placement, we use the open search for the first and then use the traversal search in the surrounding of the first step's result to decide which placement is located in the topology. The comparative experiment proves the effectiveness of the proposed algorithm, which can achieve lower average latency and worst latency with a smaller number of controllers, and it also has a certain time complexity guarantee.

Data Availability

The part of experimental data used to support the findings of this study is included within the article. The rest of the experimental data and code used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was funded by the Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project no. XDC02070100).

References

[1] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on sdn architectures with multiple controllers," *Journal of Computer Networks and Communications*, vol. 2016, 8 pages, Article ID 9396525, 2016.

[2] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM-Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[3] B. S Khan and A. N. Muaz, "Network community detection: a review and visual survey," 2017, https://arxiv.org/abs/1708.00977.

[4] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in SDN," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 472–503, 2019.

[5] 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, 2014.

[6] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 19, no. 1, pp. 30–33, 2014.

[7] H. Mu, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in SDN: evaluation for dynamic flows," in *Proceedings of the 2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2017.

[8] G. Gurobi, "Optimizer reference manual," 2015, http://www.gurobi.com.

[9] Z. Fan, J. Yao, X. Yang, Z. Wang, and X. Wan, "A multi-controller placement strategy based on delay and reliability optimization in sdn," in *Proceedings of the 2019 28th Wireless and Optical Communications Conference (WOCC)*, pp. 1–5, IEEE, Beijing, China, May 2019.

[10] D. B. Vincent, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 10, Article ID P10008, 2008.

[11] V. A Traag, L Waltman, and N. J Van Eck, "From louvain to leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, pp. 5233–5312, 2019.

[12] W. Chen, C. Chen, X. Jiang, and L. Liu, "Multi-controller placement towards sdn based on louvain heuristic algorithm," *IEEE Access*, vol. 6, pp. 49486–49497, 2018.

[13] U. N Raghavan, R Albert, and S Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, no. 3, Article ID 036106, 2007.

[14] B. Liu, B. Wang, and X. Xi, "Heuristics for sdn controller deployment using community detection algorithm," in *Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 253–258, IEEE, Beijing, August 2016.

[15] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on Computing*, vol. 33, no. 3, pp. 544–562, 2004.

[16] F. Bannour, S. Souihi, and A. Mellouk, "Scalability and reliability aware sdn controller placement strategies," in *Proceedings of the 2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–4, IEEE, Tokyo, Japan, November 2017.

[17] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of sdn controllers," in *Proceedings of the 2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Kuala Lumpur, Malaysia, May 2016.

[18] M. Priyadarsini and P. Bera, "Software defined networking architecture, traffic management, security, and placement: a survey," *Computer Networks*, vol. 192, Article ID 108047, 2021.

[19] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24290–24307, 2019.

[20] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[21] L. C Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977.

[22] L Lü, Y. C Zhang, C. H Yeung, and T Zhou, "Leaders in social networks, the delicious case," *PLoS One*, vol. 6, no. 6, Article ID e21202, 2011.

[23] Q. Yang, H. Deng, and L. Wang, "A cache strategy based on control of traffic load in the information-centric networking," *Journal of Network New Media*, vol. 10, no. 05, pp. 17–22, 2021.

[24] E. Jack and D. W Hearn, "Geometrical solutions for some minimax location problems," *Transportation Science*, vol. 6, no. 4, pp. 379–394, 1972.

[25] Snd lib, "A library of test instances," 2006, http://sndlib.zib.de/home.action.

[26] W. Wang, Y. Sun, K. Zheng, M. Kaalifar Ali, D. Li, and Z. Li, "Freeway: adaptively isolating the elephant and mice flows on different transmission paths," in *Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols*, pp. 362–367, IEEE, Raleigh, NC, USA, October 2014.