IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 17, 2018, accepted August 24, 2018, date of publication September 6, 2018, date of current version September 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2867931

Multi-Controller Placement Towards SDN Based on Louvain Heuristic Algorithm

WEN CHEN, CONG CHEN, XUEQIN JIANG, (Member, IEEE), AND LEIJIE LIU
School of Information Science and Technology, Donghua University, Shanghai 201620, China
Corresponding author: Cong Chen (congchen@mail.dhu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61501108.

ABSTRACT The separation of the forwarding and control planes of software-defined networking brings a lot of flexibility to network management. However, with the increase in network capacity, network structure becomes more and more complicated. The controller placement problem in a large-scale network is still a hard nut to crack because of high complexity and difficulty in the tradeoff between performance. In this paper, a novel approach named community detection controller deployment is proposed. With the aid of theory of complex network analysis, the network topology of the controller to be deployed is regarded as a network composed of multiple communities, and then a suitable position is selected in each community to place the controller, which is capable of avoiding the complexity of global deployment. In order to balance the number of switches managed by the controller in each community, the scale constraint factor is introduced into Louvain heuristic algorithm to limit the number of nodes in each community and balance the differences in the number of nodes among different communities. Distinct from the existing clustering-based approaches, this method can independently identify partitions with community attributes according to the network structure without manual intervention. On the other hand, it can adjust the number of nodes within communities on demand to achieve topological equilibrium partition. Experiments are performed on real network topologies, and corresponding results show that the proposed method is more suitable for networks with plenty of nodes and can effectively balance the controllers' load while keeping latency at a lower level.

INDEX TERMS SDN, network structure, controller placement, scale constraint factor, topological equilibrium partition.

I. INTRODUCTION

IP network as the underlying network that carries various future infinite data services such as Internet of Things, cloud computing, VR, AI, etc., its flexibility, scalability and support for services require the network to complete its own evolution as soon as possible. As one of the emerging network technologies, Software-Defined Networking (SDN) has the capability to evidently improve network performance and intelligently manage network services. Therefore, it can be one of the solutions to the above needs. The prominent feature of this novel paradigm is that decoupling the control plane from the forwarding device, data plane can focus on performing basic functionalities such as packet forwarding in high speed, while the logically centralized control plane is in charge of issuing control command through control protocol such as OpenFlow [1]. This innovation on the network greatly enhances the reconfiguration of the network.

The centralized controller-based architecture in SDN can simplify control plane design. A single controller, however, has limited capability to handle network events and easily becomes a bottleneck in reliability and scalability. A solution to this problem is adopting multiple controllers in the control plane [2]–[4]. For example, when SDN is deployed in a data center with a large number of network devices, administrators can use controller cluster with software definition function to configure the network devices. With multiple controllers, an important concern is the controller deployment, which is first mentioned in [5]. For data center scenarios, devices are centralized and deployment is relatively easy. For scenarios where the locations of devices are not concentrated, how to determine the optimal number and appropriate location for controller is a challenging problem. Fig. 1 shows a large-scale network with four regions. Each region composed of enterprise, school, government and data center,

is geographically separated and has a complex network structure like the network infrastructure plane in Fig. 1. If the controller is deployed directly from the global scope, it is difficult to achieve network requirements for specific services. Thus, deploying controllers in this scenarios would be tricky.

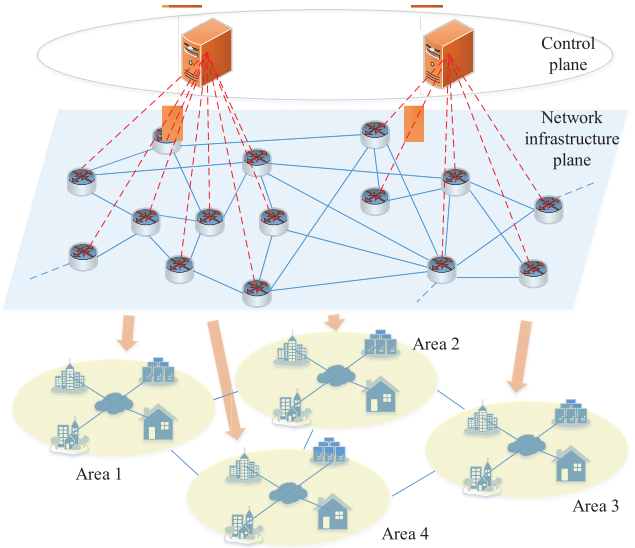


FIGURE 1. A large-scale SDN-enabled network with four regions and complicated structure.

In recent years, research and practice around SDN have been in full swing. Controller Placement Problem (CPP) is regarded as the most important issue which can directly affect the entire network performance. Thus it is also one of the hot topics in SDN research fields. The study of CPP mainly focuses on two aspects: the performance optimization and the deployment model. From the view of performance optimization, many network performance metrics such as propagation latency [5], controller capacity [6], node failure [7], deployment cost [8], [9], energy saving [10], have been taken into consideration. When multiple performance metrics are considered, the corresponding problem is generally converted into integer programming problem, and then the controller position satisfying the conditions is found in the global scope. From the deployment model, clustering methods are generally used in network topology to form several clusters and determine their cluster centers at the same time, i.e., the controller locations [11]–[13].

Although many related studies on controller placement have been pursued, there are still some details that have not been taken into account. For example, some parameters such as cluster number and density threshold need to be given when using clustering methods. However, these manual interventions will affect the accuracy of clustering. In addition, with the explosive growth of network capacity, the network structure becomes so complicated that the number of controllers needed in large-scale network can not be specified in advance. Considering that the network structure itself can be served as a clue to controller deployment, in this paper, we study the placement of multiple controllers in complex networks, and our contributions are summarized as follows:

- The controller deployment is combined with the theory of complex network analysis, and the whole network is partitioned by identifying the network structure with community attributes in large-scale networks, as shown four regions in Fig. 1. Louvain heuristic algorithm based on modularity (LHA) [14] is used in this paper, which can determine the appropriate partition according to the network topology completely independently, rather than prespecifying the number of partitions and other parameters, thus ensuring the accuracy of partitioning.
- In addition, the existing deployment schemes based on clustering often ignore the size of cluster when clustering the nodes, this is likely to lead to the uneven division. In view of this, we introduce the scale constraint factors on the basis of LHA, it is able to limit the size of sub-network and balance the number of nodes between different sub-networks. Then the entire network is divided into multiple communities with moderate scale and balanced number of nodes, which improves the load balancing performance.
- The experiments are performed on real world topologies, and the results show that our approach is suitable for achieving the balanced partition of large-scale networks and can obtain a good trade-off between latency and controller's load.

The remainder of this paper is organized as follows. In Section II, the related work is described in details. In Section III, we formulate this problem mathematically. Next, the related algorithm is introduced and the deployment process is elaborated in section IV. Section V shows the simulation and presents the performance analysis. Finally, the conclusion is given in Section VI.

II. RELATED WORK

The separation of data plane and control plane in SDN structure impel operators to consider the design of control plane when deploying SDN. As a critical issue in the field of SDN, controller deployment naturally arouses the interest of researchers. According to the existing literature, CPP has been studied from several perspectives, such as network performance, deployment scenarios and so on, and most of which focus on network performance optimization. The following is an overview of the works related to controller deployment.

Heller *et al.* [5] initiate the research on CPP and emphasize the importance to minimize average-case and worst-case propagation latency from switch to controller. They regard this issue as a facility location problem, then K-median and K-center are adopted to solve this problem. However, this approach is limited to specific topologies and does not take into account the capacity constraints of the controller, which is not applicable in real network. The capacitated controller placement problem (CCPP) in [6] considers the additional load of controller. To solve the CCPP, Yao *et al.* [6] propose an advanced capacitated K-center algorithm to search the best

placement solutions. In [15] and [16], K-critical strategy is used to achieve the minimum number of controllers as compared with methods in [5]. Apart from minimizing the number of controllers, Wang *et al.* [11] investigate the placement problem from the perspective of latency minimization and an optimized K-means algorithm is presented. They cluster nodes by improving the selection of initial centroid positions, which effectively reduce the maximum latency between centroid and their nodes. Similar to the above approaches, both spectral clustering [13], [17] and density-based clustering [18] are also used for controller placement to optimize the propagation latency. As far as deployment algorithms are concerned, they have one thing in common: the number of clusters or controllers need to be prescribed while studying the details of deployment. However, how many controllers required is unknown in advance. Furthermore, determining the number of clusters by enumeration is not feasible especially in large-scale networks with a huge amount of devices.

In addition to the performance metrics described above, several other factors have been taken into account. To strengthen network robustness, Killi and Rao [7] propose a novel model that planning ahead for controller failures to avoid drastic increase in latency and disconnections. Sallahi and St-Hilaire [8], [9] model the CPP as integer linear program (ILP) to solve how to build and expand the networks at minimum cost. Hu *et al.* [10] address the controller placement from a standpoint of energy consumption, and they can achieve energy saving by adjusting the state of the control link.

However, the research on controller deployment is not confined to pure SDN network, but also in other scenarios. Ksentini *et al.* [19] and Abdel-Rahman *et al.* [20] propose the SDN-based wireless network architecture. The CPP is modeled as ILP and solved by heuristic algorithm. Liu *et al.* [21] discuss the joint placement problem for satellite gateways and controllers in SDN-enabled 5G-satellite integrated network. To obtain the maximum average reliability under the given latency constraint, simulated annealing and clustering hybrid algorithm is developed and can achieve the near optimal reliability with much smaller running time. In [22], the CPP is in the background of virtual environment. Different from the usual way, the deployment method in [22] is embedding the virtual control graph over the physical substrate, then they propose an extensible ILP model to address the corresponding problem. In addition, for deployment problem in [17], [18], and [23], the central idea of them is to partition the large-scale network into smaller domains by using clustering algorithm. It should be mentioned that when deploying controllers in large-scale networks, clustering directly affects network performance. These clustering algorithms either need to specify the number of clusters, or need to set other parameters that impact the clustering effect, which will result in unstable partitions. Furthermore, due to the deployment complexity, some of the models described above are not suitable for large-scale networks with complicated structures.

Aiming at the problems of instability in the process of obtaining sub-networks and high complexity of global deployment, we propose a strategy named community detection controller placement (CDCD) in large-scale networks. To the best of our knowledge, it is the first to use the community detection in CPP. In our approach, the entire network can be split into multiple communities independently according to the network structure. In addition, in order to obtain balanced communities, we limit the size of the resulting partition by introducing scale constraint factors for nodes. The work done can be summarized as follows: firstly, we divide the topology into several sub-networks, then we place the controller in each sub-network to obtain optimal latency. The partition performance of the proposed approach have been validated on real network topologies.

III. CONTROLLER DEPLOYMENT MODEL AND FORMULATION

The mathematical model of the CPP is presented in this section. For an SDN-enabled network, the main network elements include controllers, switches and links. Thus, the network can be modeled by an undirected graph $G = (V, E)$, where V is the set of switches and E is the set of physical links between the switches. The deployment method used in this paper is in-band[1] deployment. Assuming that the number of controllers to be deployed throughout the SDN network is k, then let $\mathbb{C} = \{c_i | i = 1, 2, \ldots, k\}$ be the set of controllers. $\varphi(c)$ $(c \in \mathbb{C})$ denotes the mapping function that connects a single controller with a set of switches. Then the whole network G is managed by k controllers such that

$$\varphi(c_i) \neq \emptyset, \quad \forall\, c_i \in \mathbb{C}, \tag{1}$$

$$\varphi(c_i) \cap \varphi(c_j) = \emptyset, \quad \forall\, i \neq j, c_i, c_j \in \mathbb{C}, \tag{2}$$

$$\cup_{i=1}^{k} \varphi(c_i) = V, \quad \forall\, c_i \in \mathbb{C}. \tag{3}$$

The above formulas mean that each sub-network must contain at least one node, each node can be allocated to only one sub-network and all sub-networks need to cover the whole network, respectively.

In large-scale networks, communication timeliness is an important factor in network performance. In our work, the network is divided into multiple sub-networks and latency between controller and switch is used as the key performance metric. The latency model in this paper can be given by the following formula, which is similar to [18],

$$L_{avg}(\varphi(c)) = \frac{1}{|\varphi(c)|} \sum_{s \in \varphi(c)} dijk(s, c), c \in \varphi(c), \tag{4}$$

$$L_{max}(\varphi(c)) = \max_{s \in \varphi(c)} dijk(s, c), c \in \varphi(c). \tag{5}$$

Equation (4) means the average latency between switches and controllers, equation (5) means the worst-case latency between switches and controllers, where $dijk(s, c)$ is the *dijkstra* shortest path distance, corresponds to the latency

[1]The so-called in-band mode means that the controller is placed at the location of node, and the control link shares the data link.

from the switch s to the associated controller c, $c \in \varphi(c)$ denotes that controller c is placed at the position of one of the switches, $|\varphi(c)|$ is the number of switches controlled by c. Meanwhile, in each sub-network, it should be ensured that the control traffic at any time does not exceed the load capacity of the associated controller [6], which is denoted by

$$\sum_{n\in\varphi(c)} l(n) \le L(c), \quad \forall\, c \in \mathbb{C}. \tag{6}$$

In addition, based on (6) we take into account the load balancing performance among all sub-networks, which can be intuitively described as

$$\left| \sum_{m\in\varphi(c_i)} l(m) - \sum_{n\in\varphi(c_j)} l(n) \right| \le \varepsilon, \quad \forall\, c_i, c_j \in \mathbb{C}, \tag{7}$$

where $L(c)$ represents the maximum capacity of controller c, $l(n)$ represents the traffic required to control a switch n, and ε indicates the maximum difference in total load between controllers. According to (7), the load is more balanced when ε is smaller and vice versa. Therefore, the main objective in this paper is to minimize the latency between controllers and switches in each control domain under the constraints (6) and (7),

$$\min_{c\in\varphi(c)} L_{avg}(\varphi(c)), \quad \forall\, c \in \mathbb{C}, \tag{8}$$

$$\min_{c\in\varphi(c)} L_{max}(\varphi(c)), \quad \forall\, c \in \mathbb{C},$$

$$s.t.\ constraints\ (6)\ and\ (7). \tag{9}$$

IV. NETWORK PARTITION AND LOCATION SELECTION

In this paper, we propose a scheme named community detection controller deployment (CDCD). The controller deployment consists of two parts. One is the network partitioning, that is, community detection, and the other is the selection of controller position. Louvain heuristic algorithm (LHA) is leveraged to split the large-scale SDN network into several sub-networks with community attributes. Nodes within the same domain are closely connected and the connection between nodes within different sub-networks is sparse. We assume that only one controller is deployed in each sub-network, and the communication between controller and switch adopts the in-band mode for information exchange. Then determining the appropriate location in each sub-network to place the controller.

A. LOUVAIN HEURISTIC ALGORITHM

For the completeness of narrative, we need to introduce the algorithm as our background. Given a network $G(\mathbf{V}, \mathbf{E})$ with node set $\mathbf{V}$ and edge set $\mathbf{E}$, the community detection is to partition the network into disjoint communities $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$, such that $\mathbf{V} = \mathbf{c}_1 \cup \mathbf{c}_2 \cup \ldots \cup \mathbf{c}_k$ and that $\mathbf{c}_i \cap \mathbf{c}_j = \emptyset$ for any distinct i and j. Edge $e\,(u, v) \in \mathbf{E}$ has weight $w\,(u, v)$ between node u and node v. The modularity Q [24] is used to measure the quality of partitioning

$$Q = \sum_{\mathbf{c}\in\mathbf{C}} \left[\frac{\sum_{in}^{\mathbf{c}}}{2m} - \frac{\left(\sum_{tot}^{\mathbf{c}}\right)^2}{4m^2} \right], \tag{10}$$

where $\sum_{in}^{\mathbf{c}}$ is sum of the internal edge weights of community $\mathbf{c}$,[2] calculated as $\sum w\,(u, v)\,, \forall u, v \in \mathbf{c}$ and $e\,(u, v) \in \mathbf{E}$, $\sum_{tot}^{\mathbf{c}}$ is sum of the edge weights connected to nodes in community $\mathbf{c}$, calculated as $\sum w\,(u, v)\,, \{u, v\} \cap \mathbf{c} \neq \emptyset$ and $e\,(u, v) \in \mathbf{E}$, m is sum of all edge weights, calculated as $\sum w\,(u, v)\,, e\,(u, v) \in \mathbf{E}$.

The Louvain heuristic algorithm [14] greedily maximizes the modularity gain $\Delta Q_{u\to\mathbf{c}}$ by moving a node u into a community $\mathbf{c}$ of its neighbor nodes. The gain can be calculated by the following equation

$$\Delta Q_{u\to\mathbf{c}} = \frac{w_{u\to\mathbf{c}}}{m} - \frac{\sum_{tot}^{\mathbf{c}} \times w(u)}{2m^2}, \tag{11}$$

where $\sum_{in}^{\mathbf{c}}$, $\sum_{tot}^{\mathbf{c}}$ and m have been defined earlier, $w_{u\to\mathbf{c}} = \sum_{v\in\mathbf{c}} w(u, v)$ is sum of the weights from node u to nodes in community $\mathbf{c}$, $w(u)$ is sum of edge weights connected to node u. This algorithm consists of two stages. The task of the first stage is to incorporate all nodes into the most suitable community according to modularity gains. It should be noted that the current communities formed at this time are not stable. In the second stage, each community generated in the first stage is aggregated into a super node to build a new network, and the weights between super nodes are also updated accordingly. Then performing these two processes on new network until the community structure is stable. When all the community structures reach stability, the modularity obtained is maximum, and the structures of entire network is the most obvious at this time.

In addition, the greater the weight between two nodes, the more likely that these two nodes will be merged into same community. And what's more, in reality, if two entities are closer, the connection between them tends to be relatively tighter. In our work, in order to maximize the tightness of community, we define the weight function for corresponding edge by

$$w(u, v) = 1 - \frac{d_{u,v}}{d_{max}}, \tag{12}$$

where $d_{u,v}$ represents that the distance between node u and node v, d_{max} represents the maximum distance among all node pairs. That is, the distance between two nodes is mapped to a range from 0 to 1, which can effectively reduce the communication latency between nodes within the same sub-network.

B. NETWORK PARTITION BASED ON IMPROVED LOUVAIN HEURISTIC ALGORITHM

The community detection problem is to find a division scheme that can produce maximum modularity. Compared with those graph-based partitioning methods that need to

[2]Distinct from the symbol c.

give the number of partitions in advance, community detection does not need to specify this parameter. Indeed, it is impossible to accurately determine the number of partitions in a large-scale network. Specifically, graph-based partitioning methods may ignore the internal structural compactness in order to obtain the target number of partitions. Instead, community detection ensures that the resulting partitions have good structures. However, the size of sub-networks obtained by LHA varies greatly due to the unevenness of actual network structure. In other words, there may be a situation where there are too many nodes in some communities and too few nodes in other communities. Suppose all switches need to send flow-request messages to controller at a certain moment (which is regarded as the worst-case scenario), it may lead the controller to be overloaded if there are too many switches in a sub-network. In this subsection, we will try to eliminate the overload problem of controller under worst-case scenario by introducing scale constraint factor, that is, limiting the number of nodes in each community in the process of forming communities, and ensuring that the differences between communities are within an acceptable range.

Fig. 2 shows the flowchart for obtaining a balanced community set in controller deployment. The process can be described as the following steps:

1) Initialize the corresponding community for each node in the topology, the number of initial communities is the same as the number of nodes, and update the community's scale parameters.
2) For each node v, try to allocate node v to community where each of its neighbor nodes is located, calculate the pre-allocation and post-allocation modularity gain ΔQ, and find out the neighbor node with the maximum ΔQ. If max $\Delta Q > 0$ and scale constraint condition is satisfied, then node v is assigned to community where the neighbor node that maximizes ΔQ is located, otherwise it remains unchanged.
3) Repeat step 2) until the communities to which all nodes belong no longer change.
4) Rebuild network topology. All nodes in the same community are compressed into a super node. The edges that exist between two communities are converted into a new edge, the weight of new edge is the sum of weights between nodes in corresponding two communities, and the influence coefficient of super node on community scale is the sum of influence coefficient of all nodes in original community.
5) Back to step 1), and repeat these steps until the total modularity of entire topology no longer changes. That is, the total modularity gets the maximum value and a balanced community set is obtained.

In the above process, step 2) is the key to achieving balanced partitioning. In order to limit the size of sub-networks, we introduce the scale constraint factors η and β, which correspond to the controller's maximum load capacity and load balancing coefficient, respectively. At the same time, we also consider the influence coefficient of each node on

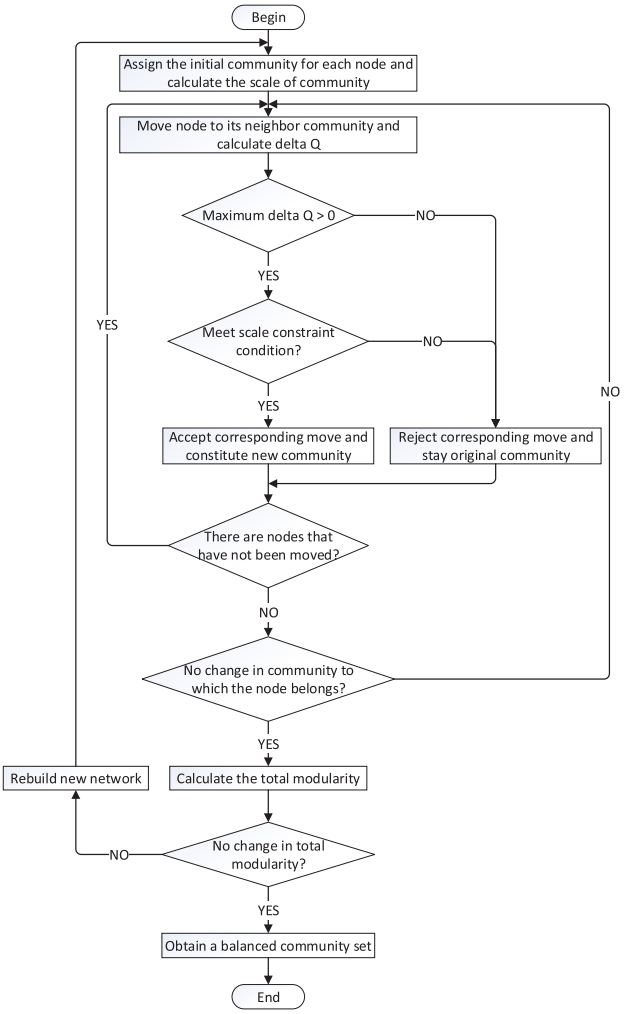


FIGURE 2. Flowchart for obtaining balanced partition.

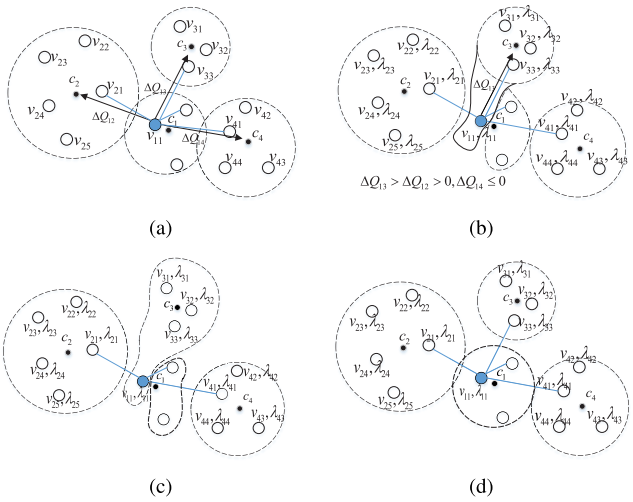


FIGURE 3. Process of finding target community for a node. (a) Initial community. (b) Search candidate community. (c) Incorporate into new community. (d) Retain in initial community.

community scale, λ, which corresponds to the control traffic required by a switch. Fig. 3 illustrates the process of moving a node to a community. The solid circle in this figure repre-

sents the node, the dashed circle represents the community, the black dot represents the centroid of community, and the colored point represents the node to be moved. Only the edges incident to node v_{11} are shown in Fig. 3. In Fig. 3(a), node v_{11} has three neighbor communities $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$, and the modularity gains corresponding to three movements are calculated separately, i.e. ΔQ_{12}, ΔQ_{13}, ΔQ_{14}. In this example, we assume that $\Delta Q_{13} > \Delta Q_{12} > 0, \Delta Q_{14} \leq 0$, that is, community $\mathbf{c}_3$ will be the candidate community for node v_{11}, as depicted in Fig. 3(b). Next considering the influence coefficient to determine the final community of node v_{11}. We set the influence coefficient for each node and give it by format $\{v_{ij}, \lambda_{ij}\}$ in the figure. For ease of description, we assume that constraint factor η is large enough, namely, to ignore it here, and we mainly consider β as well as community $\mathbf{c}_3, \mathbf{c}_4$. If $\lambda_{11} + \lambda_{31} + \ldots + \lambda_{33} - (\lambda_{41} + \lambda_{42} + \ldots + \lambda_{44}) \leq \beta$, v_{11} will be a member of community $\mathbf{c}_3$. Otherwise, v_{11} remains in community $\mathbf{c}_1$. As a result there are two situations, as shown in Fig. 3(c) and Fig. 3(d) respectively. The complete pseudo code for obtaining balanced partitions is given in **Algorithm 1**.

C. LOCATION SELECTION

After the above partitioning process, several communities have been obtained. As a part of controller deployment, the selection of location for controller will be described in this subsection. We select the propagation latency between controller and switch as our performance metric when determining the position of controller. Taking Fig. 4 as an example to describe the process of location selection in detail. We consider the selection in two scenarios, one for minimization of the average latency, and the other for minimization of the maximum latency.

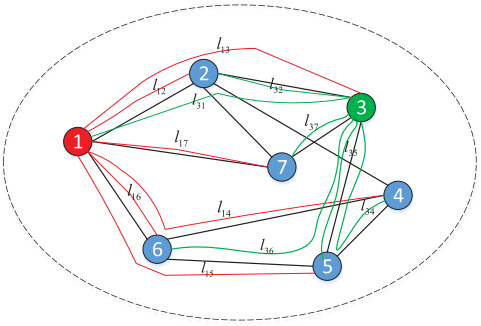


FIGURE 4. Optimal position selection of controller in a sub-network.

As presented in Fig. 4, there is a community that consists of seven nodes. To be intuitive, we identify nodes and lines in this topology using three colors, respectively. The default color of edge between nodes is black, the red line and the green line represent the *dijkstra* shortest path from node 1 and node 3 to remaining nodes, respectively. l_{ij} denotes the latency corresponding to the shortest path from node i to node j. In order to give a general description of the node selection process, we only consider node 1 and node 3 as candidate

Algorithm 1 Improved LHA: Balanced Community Detection With Scale Constraint Factors

Input: $G(\mathbf{V}, \mathbf{E})$: the weighted network; β, η: the scale constraint factors; λ: the influence coefficient of each node on community scale;

Output: Balanced partitions $\mathbf{C}$ with best modularity Q_{best};

while *True* **do**
  $\mathbf{C} \leftarrow \{\{u\}\}, u \in \mathbf{V}$;
  $\sum_{load}^{\mathbf{c}} \leftarrow \sum_{u \in \mathbf{c}} \lambda_u, \mathbf{L} \leftarrow \{\sum_{load}^{\mathbf{c}}\}, \forall \mathbf{c} \in \mathbf{C}$;
  Calculate $\sum_{in}^{\mathbf{c}}$ *and* $\sum_{tot}^{\mathbf{c}}$;
  while *community changes* **do**
    for $u \in \mathbf{V}$ **do**
      //search for the best community
      $\mathbf{c}_{can} \leftarrow \arg \max_{\forall \mathbf{c}_n \in \text{neighbor communities}} \Delta Q_{u \to \mathbf{c}_n}$;
      $\mathbf{L}_{tmp} \leftarrow \mathbf{L} \setminus \{\sum_{load}^{\mathbf{c}}, \sum_{load}^{\mathbf{c}_{can}}\}$;
      if $\Delta Q_{u \to \mathbf{c}_{can}} > 0$ **then**
        //calculate the temp $\sum_{load}$
        $\sum_{load\_tmp}^{\mathbf{c}_{can}} \leftarrow \sum_{load}^{\mathbf{c}_{can}} + \lambda_u$;
        $\sum_{load\_tmp}^{\mathbf{c}} \leftarrow \sum_{load}^{\mathbf{c}} - \lambda_u$;
        $\mathbf{L}_{tmp} \leftarrow \mathbf{L}_{tmp} \cup \{\sum_{load\_tmp}^{\mathbf{c}_{can}}, \sum_{load\_tmp}^{\mathbf{c}}\}$;
        $index_{blc} \leftarrow |x - y|, \forall x, y \in \mathbf{L}_{tmp}$;
        if $\sum_{load\_tmp}^{\mathbf{c}_{can}} \leq \eta$ && $index_{blc} \leq \beta$ **then**
          //update $\sum_{in}, \sum_{tot}, \sum_{load}$ *of* $\mathbf{c}_{can}$
          $\sum_{in}^{\mathbf{c}_{can}} \leftarrow \sum_{in}^{\mathbf{c}_{can}} + w_{u \to \mathbf{c}_{can}}$;
          $\sum_{tot}^{\mathbf{c}_{can}} \leftarrow \sum_{tot}^{\mathbf{c}_{can}} + w(u)$;
          $\sum_{load}^{\mathbf{c}_{can}} \leftarrow \sum_{load\_tmp}^{\mathbf{c}_{can}}$;
          //update $\sum_{in}, \sum_{tot}, \sum_{load}$ *of* $\mathbf{c}$
          $\sum_{in}^{\mathbf{c}} \leftarrow \sum_{in}^{\mathbf{c}} - w_{u \to \mathbf{c}}$;
          $\sum_{tot}^{\mathbf{c}} \leftarrow \sum_{tot}^{\mathbf{c}} - w(u)$;
          $\sum_{load}^{\mathbf{c}} \leftarrow \sum_{load\_tmp}^{\mathbf{c}}$;
          //move u to candidate community
          $\mathbf{c}_{can} \leftarrow \mathbf{c}_{can} \cup \{u\}$; $\mathbf{c} \leftarrow \mathbf{c} \setminus \{u\}$;
        end if
      end if
    end for
  end while
  $Q \leftarrow 0$; $Q_{best} \leftarrow 0$;
  Update total modularity Q;
  if $Q - Q_{best} \leq 0$ **then**
    break;
  end if
  $Q_{best} \leftarrow Q$;
  print $\mathbf{C}$ *and* Q_{best};
  //Rebuild network;
  $\mathbf{V}_{new} \leftarrow \mathbf{C}$;
  $\mathbf{E}_{new} \leftarrow \{e(\mathbf{c}, \mathbf{c}')\}, \exists e(u, v) \in \mathbf{E}, u \in \mathbf{c}, v \in \mathbf{c}'$;
  $w(\mathbf{c}, \mathbf{c}') \leftarrow \sum_{e(u,v) \in \mathbf{E}, u \in \mathbf{c}, v \in \mathbf{c}'} w(u, v)$; $\lambda_{\mathbf{c}} \leftarrow \sum_{load}^{\mathbf{c}}$;
  $\mathbf{V} \leftarrow \mathbf{V}_{new}$; $\mathbf{E} \leftarrow \mathbf{E}_{new}$;
end while

positions of controller. For the case of minimum average latency, the total propagation latency can be expressed as $l_{11} + l_{12} + \ldots + l_{17}$ when controller is placed at candidate

position 1. When at position 3, the latency is represented as $l_{31} + l_{32} + \ldots + l_{37}$. Assuming that the former is less than the later, position 1 will be the most appropriate deployment location. For the case of minimum maximum latency, l_{14} is the maximum latency when controller is deployed at position 1 and the maximum latency is l_{31} when deployed at position 3. Due to $l_{14} > l_{13} = l_{31}$, the best deployment location in this case is position 3. By traversing all nodes in each community, the locations of controllers to be deployed in the entire network will be found. And **Algorithm 2** and **Algorithm 3** show pseudo code for location selection in two deployment scenarios, respectively.

Algorithm 2 Location Selection Based on Minimum Average Latency

Input: **C**: topology partitions;
Output: $\mathbb{C}$: controller position set;

$\mathbb{C} \leftarrow \{\}$;
for $\mathbf{c} \in \mathbf{C}$ **do**
  $set \leftarrow \{\}$;
  for $u \in \mathbf{c}$ **do**
    $dist^{u}_{dijk} \leftarrow 0$;
    for $v \in \mathbf{c}$ **do**
      $dist^{u}_{dijk} \leftarrow dist^{u}_{dijk} + dijk(u, v)$;
    end for
    $set \leftarrow set \cup \{\frac{dist^{u}_{dijk}}{|\mathbf{c}|}\}$;
  end for
  $position \leftarrow \arg\min_{u, t \in set} t$;
  $\mathbb{C} \leftarrow \mathbb{C} \cup \{position\}$;
end for

Algorithm 3 Location Selection Based on Minimum Maximum Latency

Input: **C**: topology partitions;
Output: $\mathbb{C}$: controller position set;

$\mathbb{C} \leftarrow \{\}$;
for $\mathbf{c} \in \mathbf{C}$ **do**
  $set1 \leftarrow \{\}$;
  for $u \in \mathbf{c}$ **do**
    $set2 \leftarrow \{\}$;
    for $v \in \mathbf{c}$ **do**
      $set2 \leftarrow set2 \cup \{dijk(u, v)\}$;
    end for
    $set1 \leftarrow set1 \cup \{\max_{t \in set2} t\}$;
  end for
  $position \leftarrow \arg\min_{u, t' \in set1} t'$;
  $\mathbb{C} \leftarrow \mathbb{C} \cup \{position\}$;
end for

D. DEPLOYMENT COMPLEXITY

The controller deployment scheme based on community detection described above, including community detection in network topology and selection of controller location in each community. Compared with the global deployment, the proposed scheme divides the entire deployment into two separate sub-processes, and it only needs to consider placing the controller in a local scope. Here we briefly analyze the deployment complexity. As far as community detection is concerned, the number of communities is drastically reduced in the first few iterations. For a network with k nodes, the method appears to run in time $\mathcal{O}(k \log k)$. Ideally we can obtain m strictly balanced communities, that is, each community contains the same number of nodes, denoted as n, i.e. $n = k/m$. At this point, the complexity of location selection is $\mathcal{O}(m \cdot n^2)$. Therefore, we can obtain the total complexity of our deployment scheme, i.e. $\mathcal{O}((mn)\log(mn)) + \mathcal{O}(m \cdot n^2)$. Instead, that of global deployment can reach $\mathcal{O}((m \cdot n)^2)$. More importantly, it can provide a stable and accurate partition.

V. PERFORMANCE EVALUATION AND ANALYSIS

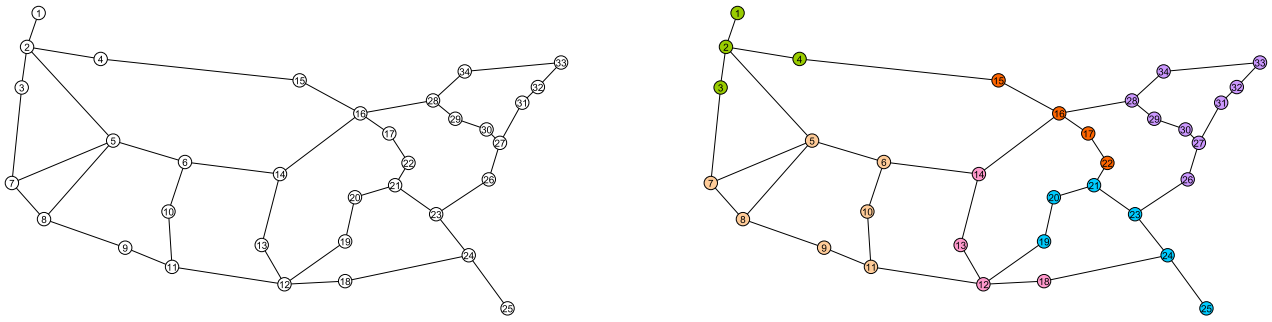
In this section, to evaluate the performance of the proposed method, the Internet2 OS3E topology [25] that is widely adopted in other literatures, is still used in this paper. Besides, we also select four topologies with different scales from the SND-lib [26], so as to evaluate the performance of the proposed algorithm on different network scales. The characteristic of topologies is listed in Table 1, the length of links is calculated by Haversine formula instead of Euclidean distance. Let the propagation speed be two-thirds of light speed. All simulations are implemented through Python programming language.

TABLE 1. Characteristic of Experiment Topologies.

Network topology	The number of nodes	The number of links
Polska	12	18
Atlanta	15	22
OS3E	34	42
Germany50	50	88
Ta2	65	108

A. ANALYSIS ON NETWORK PARTITION

First, we take Internet2 OS3E as an example to illustrate the process of network partition based on original algorithm LHA. Fig. 5(a) shows its topology structure with 34 nodes and 42 edges. Each node represents a switch and each line between nodes represents the network link. According to [14], the order of nodes will affect the output of the algorithm to some extent, but its influence on modularity can be ignored, while it will only affect the computation time. Thus, in our work, we generate a random sequence for all nodes every time when executing the algorithm. We let 500 times be one round and then go for 20 rounds. The corresponding modularity distribution is shown in Fig. 6. It can be seen from this figure that the modularity distribution of 20 rounds is similar and the values are roughly distributed between 0.59 and 0.62. Note that in each round, the number of possible values is much less than 500 due to the fact that different orders may result in the same partition. Moreover, the minimum value



(a) (b)

FIGURE 5. Demonstration of the network partition on OS3E. (a) Original topology. (b) Partition result with the maximum *Q*.

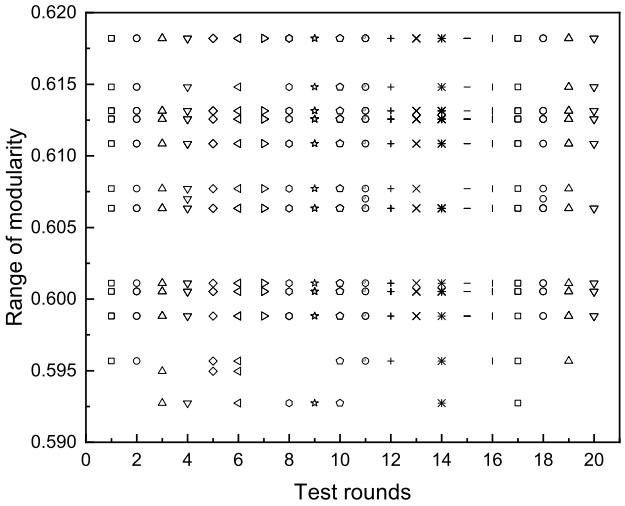


FIGURE 6. Distribution of modularity in 20 rounds.

of modularity is not exactly the same in these 20 rounds, but the maximum value is exactly the same, which equals to 0.6182070368504007. The greater the modularity, the better the effect of partitioning. We select the partition result with the maximum modularity. The corresponding topology partition is shown in Fig. 5(b). There are six sub-networks in this figure, nodes with the same color belong to the same sub-network: {3, 4, 1, 2}, {11, 8, 10, 9, 5, 7, 6}, {13, 12, 18, 14}, {22, 15, 16, 17}, {19, 23, 24, 25, 20, 21}, {32, 26, 29, 28, 33, 30, 31, 34, 27}.

As previously mentioned, the modularity reflects the quality of the network partition results, we apply this partitioning algorithm to other four topologies with different scale in Table 1. As shown in Table 2, the modularity generally scales with the number of nodes in the network. That is to say, to a certain extent, this algorithm can achieve stable and better partition effect for large-scale network with plenty of nodes.

B. ANALYSIS ON LOAD BALANCE

In this subsection, we evaluate the load balancing performance achieved by the improved LHA (ILHA). For the sake

TABLE 2. Quality of Partition.

Network topology	Quality(*Approximation*)
Polska	0.360
Atlanta	0.402
OS3E	0.618
Germany50	0.607
Ta2	0.665

of simplicity, we assume that all controllers have the same load capability and the control traffic required by each switch are the same as well. Therefore, the scale constraint factors β, η and the influence coefficient of each node on community scale λ, can be transformed into the number of network devices in each community. Then we let $\lambda = 1$, β and η are the constants to be given. In the process of dividing the network, the number of switches in each sub-network should not exceed the number of switches that the associated controller can hold. The load balancing index of the algorithm is given through α as following formula

$$\alpha = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left(N_i - \frac{|\mathbf{V}|}{K}\right)^2}, \tag{13}$$

where K denotes the number of sub-networks, N_i denotes the number of nodes in sub-network i and $|\mathbf{V}|$ denotes the total number of nodes in the entire network. The smaller the α, the better the load balancing effect.

We compare the situation with the topological equilibrium partition against the situation that topological equilibrium partition is not considered. For these five topologies we selected, the number of sub-networks obtained by using LHA is 3, 5, 6, 7, 7 respectively. Among of them, Ta2 has a sub-network with 15 nodes and this is the largest sub-network in our partitioning. Since the capacity of controller is known in actual deployment, we consider two cases of capacity in our experiments, one of which is large enough, and the other is of general capacity. In view of this, we firstly let $\eta = 15$ (or more) so as to ensure that controller has sufficient capacity for the selected topologies. When $\beta = 5$, only the number of sub-networks of Ta2 has changed compared with LHA,

from 7 to 8. Besides, the number of nodes in the largest sub-network reduces from 15 to 10. For Germany50, although the total number of its sub-networks remains unchanged, the number of nodes within each sub-network has changed. Specifically, the number of nodes in all sub-network was initially between 4 and 11, but in this case between 4 and 9. When $\beta = 3$, Polska and Atlanta remain unchanged compared with $\beta = 5$, but other three topologies have changed over the number of nodes within each sub-network or the total number of sub-networks.

Next we let $\eta = 8$. In this case, the number of nodes in some sub-networks is greater than 8, while in some sub-networks the number of nodes is less than 8. When $\beta = 5$, OS3E, Germany50, and Ta2 have changed compared with LHA. Although the number of sub-networks in OS3E has not changed, the number of nodes in each sub-network has changed. For example, the number of nodes in 6 sub-networks changes from 4, 4, 4, 6, 7, 9 to 4, 5, 6, 6, 6, 7. As for Germany50 and Ta2, the number of their sub-networks has increased. When $\beta = 3$, only the number of sub-networks of Ta2 has changed compared with $\beta = 5$, the remaining four topologies remain unchanged.

The effect of topology partitioning on load balancing performance is presented in Fig. 7 and Fig. 8. And it shows that the improved algorithm ILHA can effectively balance the load among all sub-networks. In Fig. 7, for Polska and Atlanta, the balancing index remains the same over three cases, and the same situation appears in Fig. 8. The main reason is that the number of nodes in entire network is too few that the result obtained without considering the topological equilibrium partition is already relatively balanced. However, in Fig. 7, when β changes from 5 to 3, the corresponding balancing index of the last three topologies changes from 1.89 to 0.94, 1.96 to 0.83, 1.17 to 1.15, respectively. What we can also observe is that when $\beta = 5$, ILHA achieves the same

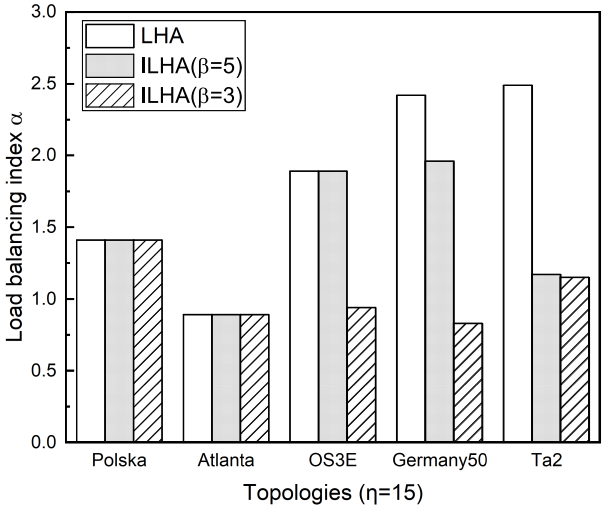


FIGURE 7. Comparison of load balancing performance over five networks ($\eta = 15$).

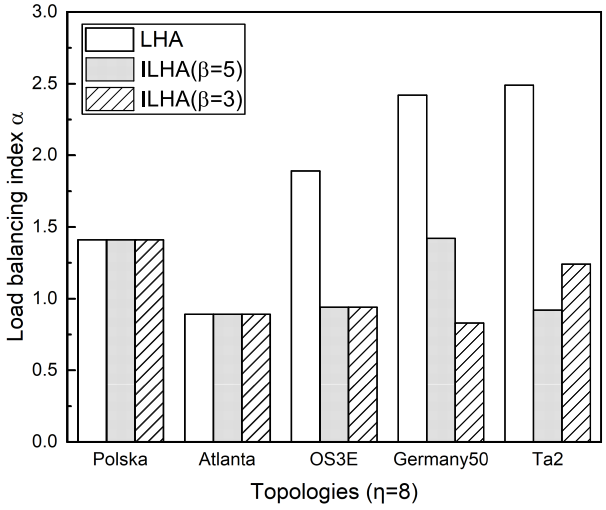


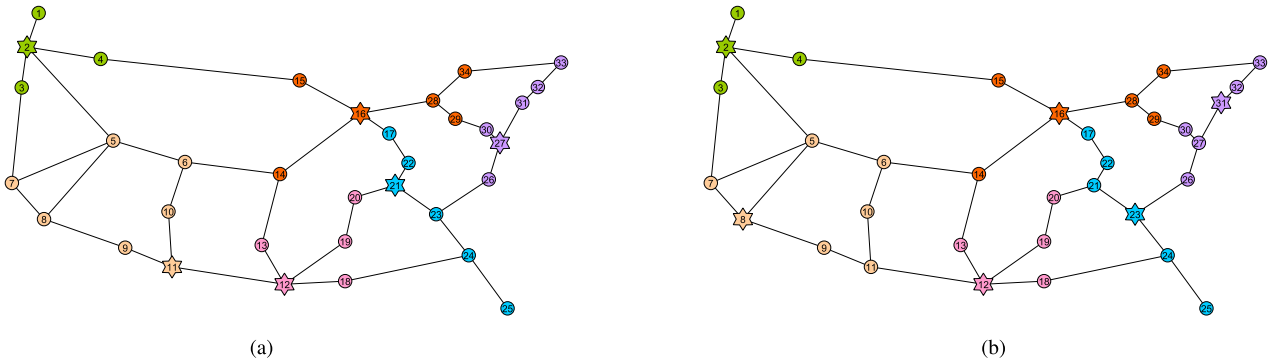
FIGURE 8. Comparison of load balancing performance over five networks ($\eta = 8$).

balancing index as the original algorithm LHA on OS3E. The reason is that the constraints are too loose and have no impact on such topologies. In Fig. 8, because of the enhanced constraints ($\eta = 8$), the ILHA gets a smaller balancing index on OS3E when $\beta = 5$, which is different from Fig. 7. On the contrary, the balancing index of $\beta = 3$ is the same as that of $\beta = 5$. In addition, there is another difference between Fig. 8 and Fig. 7. For Ta2 in Fig. 8, the balancing index changes from 0.92 to 1.24 when β changes from 5 to 3. The reason may be that the constraints are strong, which results in too many sub-networks and leads to the decline in the balancing performance of the entire network. It is worth noting that the load balancing index is still smaller than 2.49 obtained by LHA.

In general, the improved algorithm can effectively adjust the number of nodes in each sub-network when partitioning the large-scale network topologies, so as to ensure the load balancing performance for the whole network.

C. ANALYSIS ON LATENCY PERFORMANCE

We evaluate the latency performance on OS3E topology by comparing our approach with K-means proposed in [11] and K-median mentioned in [12] and [13], respectively. We choose the partition result at $\beta = 3$ as our research objective, then we deploy the controller in each sub-network to minimize the average latency and maximum latency between controller and switch. Fig. 9 shows the corresponding placement under two cases. Nodes with the same color belong to the same sub-network and stars indicate the location where the controllers are placed. We can get the minimum average latency between switches and associated controllers when the controllers are deployed at {2, 11, 12, 16, 21, 27}. Compared with this deployment scheme, there are changes on the placement when minimum worst-case latency is obtained. The corresponding deployment scenario becomes to {2, 8, 12, 16, 23, 31}.



(a) (b)

FIGURE 9. Controller placement on OS3E topology. (a) Placement with minimizing average-case latency. (b) Placement with minimizing worst-case latency.

According to [11] and [13], K-median is suitable for minimizing average propagation latency, while K-means is for minimizing the maximum latency between controllers and switches. What these two approaches have in common is that they both need to initialize the locations of controllers. Fig. 10 shows the horizontal step diagram of average latency and maximum latency obtained by performing K-median and K-means 100 times on OS3E topology. It shows that when $k = 6$, the latency achieved by K-median or K-means is less than that of $k = 5$. That is, as more controllers are deployed across the entire network, the latency between controller and switch decreases accordingly. Specifically, the average latency in Fig. 10(a) fluctuates from 3.107 ms to 4.897 ms at $k = 6$. And when $k = 5$, the average latency is in the range from 3.460 ms to 5.166 ms. When 6 controllers are deployed, about 65% of deployment latency using K-median is less than 3.7 ms. In Fig. 10(b), the maximum latency using K-means at $k = 5$ and $k = 6$ ranges from 5.798 ms to 9.836 ms and 5.770 ms to 10.019 ms, respectively. For $k = 6$, the proportion of latency between 6.5 ms and 7.5 ms is approximately 50%. The reason for this uneven distribution is that the position of central node of the cluster needs to be determined in each execution, which leads to an unstable clustering.

Since the number of partitions obtained by using both ILHA and LHA is 6, we compare the situation in which the network is divided into 6 sub-networks. For a given topology, our method is stable for the partition result and the latency is a definite value. This is the reason why there are two horizontal lines in Fig. 10. The red line represents the latency performance after equalization, while the blue line represents the latency performance without equalization. The latency for red line in Fig. 10(a) is 3.068 ms, which is better than optimal latency 3.107 ms ($k = 6$) and is higher than 2.993 ms that corresponds to the blue line. Despite the slight increment in average latency compared with LHA, ILHA ensures the load balancing performance.

On the contrary, the latency performance shown in Fig. 10(b) is not as good as that in Fig. 10(a). Because the sub-network {11, 8, 10, 9, 5, 7, 6} has not been changed

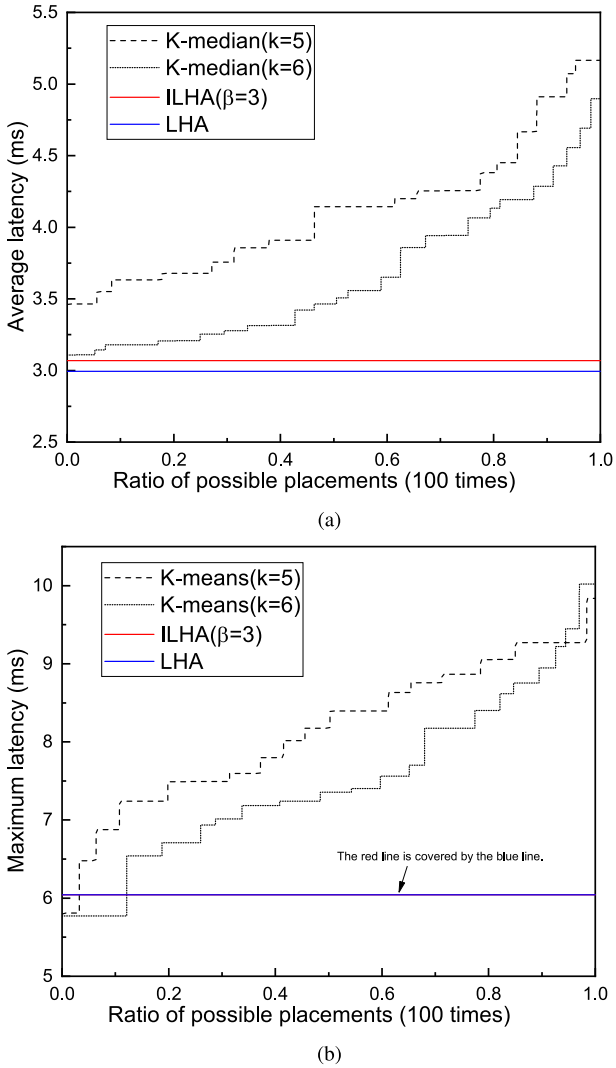


FIGURE 10. Comparison of latency performance. (a) Average latency. (b) Maximum latency.

before and after the use of topological equilibrium partitioning algorithm, the controller is still deployed at the location of node 8 when minimum maximum latency is obtained in

this sub-network. In addition, this maximum latency in {11, 8, 10, 9, 5, 7, 6} represents the maximum latency in the entire network. And the maximum latency equals to 6.044 ms, which is higher than optimal latency 5.770 ms obtained by K-means ($k = 6$). But it is better than about 85% of deployment compared with K-means ($k = 6$). Overall, our deployment approach is able to maintain latency by a lower level while ensuring a relatively balanced load among all sub-networks, which is feasible for the trade-off between performance.

VI. CONCLUSION

As SDN network architecture is widely integrated with existing networks and future networks, network operators need to consider the controller placement problem when building networks. In this paper, we try to tackle the problem through mining the community structure in large-scale network topology. Based on the theory of complex network analysis, we propose a new placement strategy named community detection controller deployment (CDCD). Considering that the controller has limited control capabilities, it is necessary to limit the number of forwarding devices in each management domain. To this end, the scale constraint factor is introduced into CDCD to limit the size of community and minimize the differences between communities. The results show that our deployment approach is suitable for large-scale network and can achieve load balancing among different management domains. Meanwhile, it is able to maintain latency at a lower level in terms of the trade-off between latency and controller's load. Although it can provide a stable and effective controller deployment, many factors still need to be considered in actual deployment, such as minimizing deployment costs within community and maximizing network resilience among communities, which also contributes to our future research goals.

REFERENCES

[1] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[2] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. Usenix Conf. Oper. Syst. Design Implement.*, 2010, pp. 1–6.

[3] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 19–24.

[4] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, pp. 1–6.

[5] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. ACM SIGCOMM HoTSDN*, 2012, pp. 7–12.

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

[7] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 514–527, Sep. 2017.

[8] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, Jan. 2015.

[9] A. Sallahi and M. St-Hilaire, "Expansion model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 274–277, Feb. 2017.

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

[11] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A K-means-based network partition algorithm for controller placement in software defined network," in *Proc. IEEE Int. Conf. Commun.*, May 2016, pp. 1–6.

[12] J. Zhao, H. Qu, J. Zhao, Z. Luan, and Y. Guo, "Towards controller placement problem for software-defined network using affinity propagation," *Electron. Lett.*, vol. 53, no. 14, pp. 928–929, Jun. 2017.

[13] K. S. Sahoo, B. Sahoo, R. Dash, and M. Tiwary, "Solving multi-controller placement problem in software defined network," in *Proc. Int. Conf. Inf. Technol.*, 2016, pp. 188–192.

[14] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech., Theory Exp.*, vol. 2008, no. 10, p. P10008, 2008.

[15] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "Defining a network management architecture," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, pp. 1–3.

[16] Y. Jiménez, C. Cervelló-Pastor and A. J. García, "On the controller placement for designing a distributed SDN control layer," in *Proc. IFIP Netw. Conf.*, 2014, pp. 1–9.

[17] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The SDN controller placement problem for WAN," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Oct. 2014, pp. 220–224.

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

[19] A. Ksentini, M. Bagaa, and T. Taleb, "On using SDN in 5G: The controller placement problem," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[20] M. J. Abdel-Rahman, E. A. Mazied, A. MacKenzie, S. Midkiff, M. R. Rizk, and M. El-Nainay, "On stochastic controller placement in software-defined wireless networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2017, pp. 1–6.

[21] J. Liu, Y. Shi, L. Zhao, Y. Cao, W. Sun, and N. Kato, "Joint placement of controllers and gateways in SDN-enabled 5G-satellite integrated network," *IEEE J. Sel. Area Commun.*, vol. 36, no. 2, pp. 221–232, Feb. 2018.

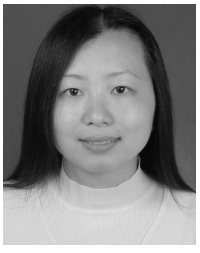
[22] I. Vaishnavi and W. Y. Poe, "Virtualized control plane placement problem: Provisioning the control paths and architectures," in *Proc. IEEE Conf. Comput. Commun. Workshops*, May 2017, pp. 695–700.

[23] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 1, pp. 344–355, Mar. 2018.

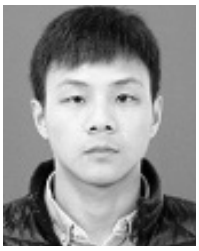
[24] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 70, no. 6, p. 066111, 2004.

[25] *Internet2 Open Science, Scholarship and Services Exchange*. Accessed: Dec. 17, 2017. [Online]. Available: http://www.internet2.edu/network/ose/

[26] *SNDlib: Library of Test Instance for Survivable Fixed Telecommunication Network Design*. Accessed: Dec. 17, 2017. [Online]. Available: http://sndlib.zib.de/home.action

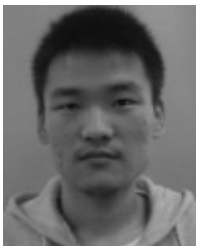


WEN CHEN received the Ph.D. degree from the Computer Science and Engineering Department, Shanghai Jiao Tong University, in 2006. She is currently an Associate Professor with the College of Information Science and Technology, Donghua University, where she is also a member with the Engineering Research Center of Digitized Textile and Fashion Technology, Ministry of Education. She has authored or co-authored over 50 technical papers. Her research interests are resource management, optimization techniques, and feedback control techniques and their applications in wireless communications.



CONG CHEN received the B.S. degree in communication engineering from the Wuhan Institute of Technology. He is currently pursuing the M.S. degree in information and communication engineering with Donghua University, Shanghai, China. His research interests include software-defined networking and its applications in wireless networks.

XUEQIN JIANG received the M.S. and Ph.D. degrees in electronics engineering from Chonbuk National University, Jeonju, South Korea. He is currently an Associate Professor with the School of Information Science and Technology, Donghua University, Shanghai, China. He has authored or co-authored over 70 technical papers and several book chapters. His main research interests include LDPC codes, physical-layer security, and wireless communications.



LEIJIE LIU received the B.S. degree in electronic information science and technology from Zhejiang Sci-Tech University. He is currently pursuing the M.S. degree in control science and engineering with Donghua University, Shanghai, China. His research interests include software-defined networking and Internet of Things.