Different QoS Constraint Virtual SDN Embedding under Multiple Controllers

Zhiyuan Zhao[1], Xiangru Meng[2], Siyuan Lu[3] and Yuze Su[2]
[1]32021 Troops
Haidian, Beijing100091 - China
[e-mail: zhaozhiyuan_0815@126.com]
[2] College of Information and Navigation, Air Force Engineering University
Xi'an, Shanxi710077 - China
[e-mail: xrmeng@126.com]
[3] Academy of military sciences
Haidian, Beijing100091 - China
[e-mail: 15668296999@163.com]
*Corresponding author: Xiangru Meng

Received April 8, 2017; revised October 11, 2017; accepted March 20, 2018; published September 30, 2018

Abstract

Software-defined networking (SDN) has emerged as a promising technology for network programmability and experiments. In this work, we focus on virtual network embedding in multiple controllers SDN network. In SDN virtualization environment, virtual SDN networks (vSDNs) operate on the shared substrate network and managed by their each controller, the placement and load of controllers affect vSDN embedding process. We consider controller placement, vSDN embedding, controller adjustment as a joint problem, together considering different quality of service (QoS) requirement for users, formulate the problem into mathematical models to minimize the average time delay of control paths, the load imbalance degree of controllers and embedding cost. We propose a heuristic method which places controllers and partitions control domains according to substrate SDN network, embeds different QoS constraint vSDN requests by corresponding algorithms, and migrates switches between control domains to realize load balance of controllers. The simulation results show that the proposed method can satisfy different QoS requirement of tenants, keep load balance between controllers, and work well in the acceptance ratio and revenue to cost ratio for vSDN embedding.

Keywords: Virtual SDN Embedding, Controller Placement, QoS, Controller Adjustment

This work was jointly supported by the National Natural Science Foundation of China (No. 61201209, 61401499)

http://doi.org/10.3837/tiis.2018.09.003 **ISSN : 1976-7277**

1. Introduction

Network virtualization has been regarded as a fundamental technology for the next generation Internet [1, 2]. By the mechanism of resource abstraction and isolation， network virtualization allows multiple virtual networks (VNs) to operate on the shared substrate network (SN) simultaneously. Software-defined networking has emerged as a promising technology for network programmability and experiments. It decouples the control plane and the data plane, and guides forwarding devices by a logically centralized controller [3]. The virtualization of SDN network is the combination of these two technologies and has gained considerable attention from both industry and academia in recent years. It provides convenient means for testing new algorithms, protocols and network architectures, helps shorten the cycles of network configuration.

The SDN virtualization platform based on transparent proxy is the central method to realize the virtualization of SDN network. FlowVisor [4], ADVisor [5], CoVisor [6] are representations. In this mode, transparent proxy sits between the control plane and data plane, and acts as the network virtualization layer. Transparent proxy slices the substrate SDN network along multiple dimensions: topology, bandwidth, switch CPU, and flow tables. Each slice has its own view of virtual topology and the associated controller. Controller defines and manages the routing policy and resources of slice. We consider a slice along with its associated controller as a virtual SDN network (vSDN).

The main work of SDN virtualization platform are slicing and embedding. Slicing identifies and isolates each vSDN from others to allow multiple vSDNs running their own applications distinctively. Embedding is to embed nodes and links of vSDN to the switches and paths of substrate SDN network on the basis of resources and topology constraints [7]. There is an improtant difference between traditional VN embedding and vSDN embedding, vSDN embedding should take the problem of controller placement into consideration.

In SDN virtualization environment, controller is usually placed at the same position of switch. Controller placement is to find the optimal switch location for controller to minimize the controller-to-switch delay. As a result, the controller can communicate effectively with all the switches, and react quickly to network events [8].

In single controller SDN architecture, controller manages the embedding and operating of all vSDNs, time delay is the main challenge to place controller. However, there is reliability and scalability problem for single controller architecture. To further improve scalability, reliability and performance of network, it is recommended to deploy multiple controllers in SDN architecture since OpenFlow (OF) protocol v1.2. In this way, multiple controllers cooperate to manage the SDN network in a physical distributed but logic-centralized form. However, the problem of placing multiple controllers is introduced [9, 10]. Time delay is a key factor to multiple controllers placement; besides, during vSDN embedding, virtual nodes of vSDNs are embedded to switches randomly due to the position and resource constraints, thus the resource consumption of switches are significantly different, and leads to load imbalance among controllers. The overload of controllers has negative effects on network stability. It is important to adjust controllers' load dynamically to avoid overload.

In this paper, we focus on designing vSDN embedding techniques in SDN environment. In contrast to previous work, we consider controller placement, vSDN embedding and controller load balance together for the first time. Besides, as the different quality of services (QoS) requirements of tenants, we provide two-level QoS for vSDN requests which require

time delay constraint of both controller-to-switch connection and virtual link. The problem is described as different QoS constraint vSDN embedding under multiple controllers.

To solve the problem, we formulate the controller placement and adjustment problem into a multi-objective nonlinear integer program (NLIP) to optimize the average time delay of controller-to-controller, the average time delay of controller-to-switch, and the load imbalance degree of controllers. We formulate different QoS constraint vSDN embedding problem into an integer linear program (ILP) to optimize the embedding cost.

Due to the NP-hard nature of the problem, we design a heuristic method. Our method consists of three aspects: the Controller Placement method based on Immune optimization Algorithm (IACP), vSDN embedding algorithm, and the Controller adaptive Adjustment algorithm based on Threshold (TCA). In our method, firstly, controllers are placed and control domains are partitioned by IACP, according to substrate SDN network and the number of controllers. Secondly, once vSDN request arrives, it will be embedded by corresponding algorithm according to its QoS requirement. Thirdly, once controller is overload after embedding, switches of overload controller will be migrated to other control domain by TCA. The method coordinates the relationship between controller placement, vSDN embedding and controller adjustment. Simulation result shows that our method satisfies different QoS requirement of tenants, works well in the acceptance ratio and revenue to cost ratio for vSDN embedding, and realizes controller load balance.

In summary, the main contributions of this paper can be summarized as follows: (1) To the best of our knowledge, we make the first attempt to study the vSDN embedding problem under multiple controllers. (2) We formulate the controller placement and adjustment problem into a NLIP formulation, formulate different QoS constraint vSDN embedding problem into an ILP separately, and design a heuristic method to solve the problem. (3) We evaluate the performance in terms of acceptance ratio, revenue to cost ratio of vSDN embedding, load imbalance degree of controllers. Simulation results demonstrate the effectiveness of the proposed method, and we analyze the effects of each aspect of method on the performance.

The rest of paper is organized as: we discuss the related works in section 2. Section 3 gives the architectures of OpenFlow based multiple controllers SDN network, SDN virtualization, and the vSDN embedding model. Section 4 describes the NLIP and ILP formulations. Section 5 presents our method. Section 6 describes simulation results and analysis. The paper is concluded in section 7.

2. Related Work

There are three kinds of existing work related to our work: the controller placement, controller load balance in multiple control domains and VN embedding.

The controller placement problem is a pre-planning problem of SDN, it was first proposed in [8], where authors solved how many and where to place controllers. They solve the problem by minimizing average latency and maximum latency from switches to controllers. Yao et al. in [9] consider both propagation delay and transmission delay, formulate the controller placement as an optimization problem. They present two algorithms based on greedy and Dijkstra algorithms to solve the problem. Hu et al. in [10] present a metric to characterize the reliability of SDN control network, and develop several heuristic placement algorithms. However, the methods mentioned above are all static placement without considering the change of network traffic, which lead to load imbalance among controllers.

Dynamic switch migration scheme can help load balance for controllers and optimize network management. Bariin et al. in [11] formulate the optimal controller provision problem as an integer linear problem and propose two heuristic algorithms. Dixit et al. in [12] propose a detailed migration mechanism to assign switches to controller dynamically. Yao et al. in [13] define a controller placement metric considering the switch degree and the delay from switches to controller, propose a dynamic switch migration algorithm to adapt to the flow dynamics, and realize controller load balance in multiple SDN domains. These works are designed for SDN architecture but not for SDN virtualization architecture, in which vSDN embedding and its effects on controller's load should be taken into consideration, and novel adjustment method need to be developed for realizing load balance of controllers.

There are many previous works focus on VN embedding problem in traditional network, and many VN embedding algorithms are proposed with different objectives or constraints [14-18]. Cui et al. in [16] introduce the node connection-degree based on virtual topology connection feature, it helps increase the utilization efficiency of substrate network. Ding et al. in [17] introduce betweenness centrality to sort virtual nodes, introduce correlation properties between substrate nodes to coordinate the process of node embedding and link embedding. Liao et al. in [18] consider topology attributes of substrate and virtual networks through multiple characteristics to better coordinate node and link embedding. As the distinctions of SDN environment, these method cannot be directly applied to the SDN virtualization environment, and novel embedding method need more efforts.

There are also a few studies focus on VN embedding in SDN network [19-22]. Mijumbi et al. in [20] consider the load balance of nodes and links together, and propose a flow migration method based on real-time network state, which dynamically manage the node and link resources in SDN virtualization environment. Mehmet et al. in [21] tackle virtual node and link embedding, and controller placement together, develop techniques to perform embedding with two goals: balancing the load on the substrate network and minimizing controller-to-switch delays. Gong et al. in [22] propose an online vSDN embedding algorithm, which embeds the controller and the virtual nodes to the substrate nodes at the same time by considering both of controller-to-switch delay and link embedding, then virtual links are embedded by k-shortest path algorithm. However, these method are all based on single controller architecture.

In multiple controllers SDN architecture, the control plane is distributed, and the architecture includes two categories: the flat control architecture [23-25], in which all the controllers are in equal status; the hierarchical control architecture [26], in which the control plane are layered as root controller and leaf controller. Tootoonchian et al. in [23] design and realize HyperFlow, which is a distributed and event-based OpenFlow controller. HyperFlow allows to place multiple controllers in network, offers extendibility and keeps the centralization of network control logic by sharing the coincident network view between all controllers. Hassas et al. in [26] propose Kandoo, which is two-layer controller architecture. In Kandoo, bottom controllers are isolated from each other and have no idea about the network view, they are all connect to the top controller. The top controller is logically centralized and maintain the global network view. By this way, bottom controllers handle local events and screen local messages to the top, which reduces the cost of top controller.

In this study, we combine controller placement, controller adjustment and vSDN embedding together for embedding vSDN in multiple controllers SDN network.

3. System Model

3.1 Problem Description

Fig. 1 shows the multiple controllers SDN network architecture based on OpenFlow, which consists of forwarding plane and control plane. OF switches in forwarding plane maintain their own flow table structures, manage and forward packets according to flow tables. Controllers in control plane compute and assign forwarding flow rules to OF switches via the southbound interface, i.e., OpenFlow. Moreover, the controllers are responsible for network management, including the resource scheduling of forwarding plane, network topology maintenance and real-time update of network status.

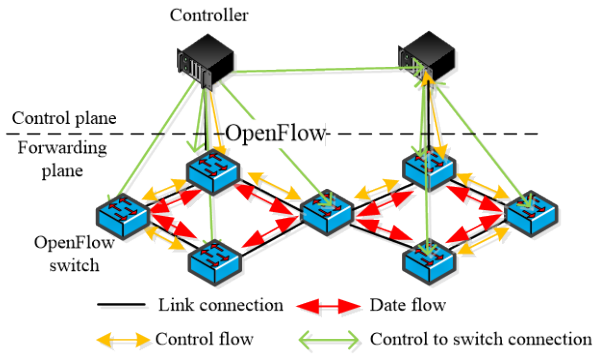


Fig. 1. OpenFlow based multiple controllers SDN architecture

We take FlowVisor as an example to illustrate the SDN virtualization based on transparent proxy and vSDN network embedding, as shown in **Fig. 2**. The proxy FlowVisor transmits all the control and status messages, it interjects between the forwarding plane and control plane. vSDNs coexist on the shared substrate SDN network and are isolated from each other. Each vSDN owns its managing controller and the set of virtual nodes and links. A virtual node is hosted on a particular OF switch, and a virtual link spans over a path in the underlying substrate SDN network. In **Fig. 2**, virtual nodes of vSDN1 are embedded to A, B and C, virtual links are embedded to AB and BC, the controller- to-switch connection, i.e., the red dashed lines are pre-assigned in SDN network.

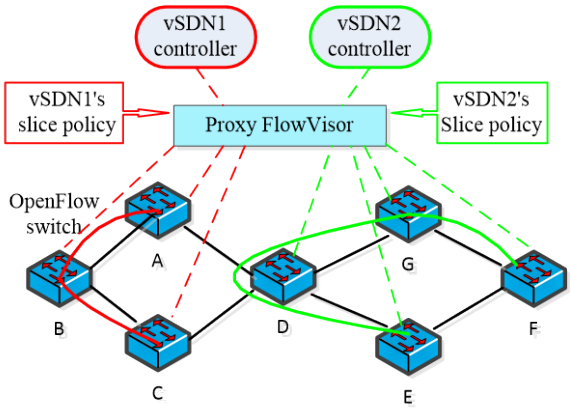


Fig. 2. Architecture of SDN virtualization

3.2 Network Model

Substrate SDN network We model the multiple controllers substrate SDN network by a weighted undirected graph $\boldsymbol{G}_S = (N_S, N_C, L_S, L_C)$, where N_S, N_C, L_S, L_C represent the sets of substrate node (OF switch), controller, substrate link and control path respectively, the control path includes controller-to-switch connection and controller-to-controller connection.

For each substrate node $n_s \in N_s$, we take the available CPU, available ternary content-addressable memory (TCAM) capacity and position as its attributes, which are denoted by $cpu(n_s)$, $tcam(n_s)$ and $loc(n_s)$; TCAM is used for flow table storage and processing, $loc(n_s) = (x_s, y_s)$ is a two-dimensional coordinate. For each controller $n_c \in N_C$, we take control domain and load as its attributes and denote them by $CA(n_c)$ and $Atcam(n_c)$, $CA(n_c)$ includes the OF switches that are managed by n_c, $Atcam(n_c) = \sum_{n_s \in CA(n_c)} contcam(n_s)$, where $contcam(n_s)$ denotes the TCAM consumption of n_s. For each substrate link $l_s \in L_S$, its attributes are available bandwidth and time delay, which are denoted by $bw(l_s)$ and $dl(l_s)$. For each control path $l_c \in L_C$, we take time delay as its attribute and denote it by $dl(l_c)$.

Different QoS constraint vSDN request The request is modeled as a weighted undirected graph $\boldsymbol{G}_{vSDN} = (N_v, L_v, QoS_v, T_v)$, where N_v and L_v represent the sets of virtual node and link.

For each virtual node $n_v \in N_v$, we take the requirement of CPU, TCAM, position and position constraint as its attributes and denote them by $cpu(n_v)$, $tcam(n_v)$, $loc(n_v)$ and $D(n_v)$ respectively. For each virtual link $l_v \in L_v$, we take the requirement of bandwidth as its attribute and denote it as $bw(l_v)$. QoS_v represents the QoS requirement of $\boldsymbol{G}_v$, in our work, we set QoS_2 requirement as to satisfy the time delay constraint of both control path and virtual link. T_v represents survival time of $\boldsymbol{G}_v$.

Different QoS constraint vSDN embedding The embedding is defined as an embedding action M from G_{vSDN} to a subset of G_s, the embedding should meet the resource and QoS requirement of vSDN request, it is denoted as

$$M : G_v \rightarrow (N_s^{sub}, L_s^{sub}, R_N, R_L) \tag{1}$$

where $N_s^{sub} \in N_s$, $L_s^{sub} \in L_s$, R_N and R_L represent resources of switches and links that allocated to vSDN request.

Different QoS constraint vSDN embedding under multiple controllers The problem consists of three aspects: controller placement, vSDN embedding, and controller adaptive adjustment for load balance.

In our work, the controller placement is defined as: given the substrate network and the number of controllers, where should the controllers go and how to assign OF switches to each controller for best assignment. The vSDN embedding problem has been described above. Controllers are immobility in substrate SDN network once they are placed, and OF switches are assigned to different control domains. During vSDN embedding, the TCAM consumption of OF switches changes, which leads to the load change of controllers. Controller adaptive adjustment is defined as: how to adjust the management relationship between controllers and switches dynamically, to realize load balance of controllers.

4. Mathematical Model

4.1 Objectives

The main goal of different QoS constraint vSDN embedding is to make full use of substrate network resources, accept more vSDN requests while satisfy their QoS requests, increase revenue and reduce cost. The optimization objectives include:

Average time delay of control path

The time delay of controller-to-switch connection affects the response speed of controller to events of substrate SDN network. The time delay of controller-to-controller connection affects the information synchronization between controllers. The average time delay of control path is defined as

$$T_{mean} = \frac{1}{|L_C|}\sum_{l_c \in L_C} dl(l_c) = \frac{1}{|L_C|}\left(\sum_{l_{cs} \in L_{CS}} dl(l_{cs}) + \sum_{l_{cc} \in L_{CC}} dl(l_{cc})\right) \tag{2}$$

where $|L_C|$ is the number of control path, L_{CS} and L_{CC} represent the control path set of controller-to-switch and controller-to-controller respectively, $L_C = L_{CS} \bigcup L_{CC}$, l_{cs} and l_{cc} represent two kinds control path respectively.

The acceptance ratio

The acceptance ratio is defined as

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T} vSDNR_{map}(t)}{\sum_{t=0}^{T} vSDNR(t)} \tag{3}$$

where $vSDNR(t)$ is the number of vSDN requests that arrive at time t, $vSDNR_{map}(t)$ is the number of vSDN requests that have been successfully embedded at time t.

The revenue to cost ratio (R/C)

Revenue of G_{vSDN} at time t is defined as

$$R(G_{vSDN}, t) = \gamma \cdot \left(\sum_{n_v \in N_v} cpu(n_v) + \alpha_1 \sum_{n_v \in N_v} tcam(n_v) + \beta_1 \sum_{l_v \in L_v} bw(l_v)\right) \tag{4}$$

where γ is the revenue weight of different QoS constraint G_{vSDN} . In our work, we set $\gamma = 1.2$ for QoS_2 and $\gamma = 1$ for QoS_1 . α_1 and β_1 are weighting coefficient to balance the relative revenues from TCAM, bandwidth and CPU, we set $\alpha_1 = 1$, $\beta_1 = 1$.

Cost of G_{vSDN} at time t is defined as

$$Cost(G_{vSDN}, t) = \sum_{n_v \in N_v} cpu(n_v) + \alpha_2 \left(\sum_{n_v \in N_V} tcam(n_v) + \sum_{n_s \in P_{L_v}} tcam(n_s)\right) + \beta_2 \sum_{l_v \in L_v} hops(l_v) \cdot bw(l_v) \tag{5}$$

where P_{L_v} denotes the total substrate paths of G_{vSDN} . $\sum_{n_s \in P_{L_v}} tcam(n_s)$ is the TCAM consumption of substrate nodes that paths span. $hops(l_v)$ is the total hop counts of path that l_v embed to. α_2 and β_2 are weighting coefficient to balance the relative costs from TCAM, bandwidth and CPU, we set $\alpha_2 = 1$, $\beta_2 = 1$.

So the R/C is defined as

$$R/C=\lim_{T\to\infty}\frac{\sum_{t=0}^{T}\sum_{G_{vSDN}\in vSDN_{map}(t)}R(G_{vSDN},t)}{\sum_{t=0}^{T}\sum_{G_{vSDN}\in vSDN_{map}(t)}Cost(G_{vSDN},t)} \tag{6}$$

The load imbalance degree of controllers

We define the load imbalance degree of controllers as the variance of each controller's load relative to the average load, as illustrated in **Eq. 7**.

$$D=\left(\frac{\sum_{n_c}^{|N_C|}(Atcam(n_c)-\overline{Atcam(n_c)})^2}{|N_C|}\right)^{\frac{1}{2}} \tag{7}$$

where $\overline{Atcam(n_c)}$ is the average load of controllers.

4.2 Mathematical Model

In this section, we formulate the controller placement and adjustment problem into a multi-objective Nonlinear Integer Program as follows.

Objectives:

$$\min \quad T_{mean} \tag{8}$$

$$\min \quad D \tag{9}$$

Multiple controller placement constraints:

$$\sum_{j\in N_S} y_j=|N_C| \tag{10}$$

$$\sum_{j\in N_C} x_{ij}=1;\forall i|y_i=0,i\in N_S \tag{11}$$

$$\sum_{i,j\in V_c} x_{ij}=|N_C|\frac{|N_C|-1}{2} \tag{12}$$

$$dl(l_{cs})\le t_{cs},\quad \forall l_{cs}\in L_{CS} \tag{13}$$

$$dl(l_{cc})\le t_{cc},\quad \forall l_{cc}\in L_{CC} \tag{14}$$

$$y_i,x_{ij}\in\{0,1\};\ \forall i\in N_S,j\in N_S \tag{15}$$

The objectives of the NLIP try to minimize the average time delay of control path, and minimize the load imbalance degree of controllers, as is shown in **Eqs. 8 and 9**. **Constraint (10)** denotes that there are $|N_C|$ nodes in N_S are selected to place controllers. **Constraint (11)** denotes that for one node there is only one controller-to-switch connection access to control plane; **constraint (12)** denotes the number of controller-to-controller connections. **Constraints (13)** and **(14)** are time delay constraint of two type control paths. In **constraint (15)**, $y_i=1$ denotes a controller is placed in the position of node i, otherwise $y_i=0$; $x_{ij}=1$ denotes $y_i=1$ and node i is assigned to controller j, or $y_i=y_j=1$ and controller i is connect to controller j, otherwise $x_{ij}=0$.

We formulate the problem of different QoS constraint vSDN embedding into an Integer Linear Program as follows.

Objective:

$$\min \quad Cost\left(G_{vSDN}\right) \tag{16}$$

Node embedding constraints:

$\forall n_i \in N_v, \forall n_j \in N_S$:

$$x_j^i \cdot cpu(n_i) \le cpu(n_j) \tag{17}$$

$$x_j^i \cdot tcam(n_i) \le tcam(n_j) \tag{18}$$

$$x_j^i \cdot dis(loc(n_i), loc(n_j)) \le D(n_i) \tag{19}$$

$$\sum_{n_j \in N_S} x_j^i = 1, \quad \sum_{n_i \in N_v} x_j^i \le 1 \tag{20}$$

Link embedding constraints:

$\forall l_{ij} \in L_S$:

$$\sum_{l_{uw} \in L_V} f_{ij}^{uw} \cdot bw(l_{uw}) \le bw(l_{ij}) \tag{21}$$

$\forall n_j \in N_S, l_{uw} \in L_V$:

$$\sum_{l_{ji} \in L_S} f_{ji}^{uw} - \sum_{l_{ij} \in L_S} f_{ij}^{uw} = \begin{cases} 1, & x_j^u = 1 \\ -1, & x_j^w = 1 \\ 0, & otherwise \end{cases} \tag{22}$$

QoS constraint:

QoS_2 :

$$dl(l_{vc}) \le t_{cs1}, \quad \forall l_{vc} \in G_{vSDN} \tag{23}$$

$$dl(l_v) \le t_p, \quad \forall l_v \in L_v \tag{24}$$

Variable domain constraint:

$$\forall n_i \in N_v, \forall n_j \in N_S : \quad x_j^i \in \{0,1\} \tag{25}$$

$$\forall n_j \in N_S, l_{uw} \in L_V : \quad f_{ji}^{uw} \in \{0,1\} \tag{26}$$

The objective of the LIP tries to minimize the cost of vSDN embedding, as is shown in **Eq. 16**. **Constraints (17), (18)** and **(19)** denote the constraints of CPU, TCAM and position respectively. **Constraint (20)** ensures that each virtual node in vSDN request must be embedded to just one substrate nodes. **Constraint (21)** is the bandwidth constraints. **Constraint (22)** is the connectivity constraint. **Constraints (23)** and **(24)** are time delay of control path and virtual link for QoS_2 request, as for QoS_1 request, there is no time delay constraint. **Constraints (25)** and **(26)** denote the binary domain for the variables x_j^i and f_{ji}^{uw}.

5 Heuristic Method Design

The NLIP model and the ILP model are both NP-hard problem, thus, different QoS constraint vSDN embedding under multiple controllers is NP-hard, too. In this section, a heuristic method is proposed to solve the problem.

Firstly, by using the Controller Placement method based on Immune optimization Algorithm, controllers are placed and control domains are partitioned according to substrate SDN network and the number of controllers. Secondly, when vSDN requests arrive, each vSDN request is embedded by corresponding algorithm according to its QoS requirement. Thirdly, once vSDN is successfully embedded and substrate network resources are consumed, whether the load of controllers exceed threshold or not will be judged. If there is controller be

overload, OF switches from overload control domain will migrate to lightly load control domain. Which switches to select and where to go are calculated by Controller adaptive Adjustment algorithm based on Threshold.

5.1 Controller Placement Method Based on Immune Optimization Algorithm

Immune optimization algorithm is introduced into the IACP method to resolve controller placement problem. Immune optimization algorithm applies whole search strategy and emphasizes information exchange between the colony. The operation of IACP cycles the process of initial antibody population generating, evaluation criterion calculation, individual information exchange among population, and new antibody population generating. Then an optimal solution can be obtained after cycling.

Fig. 3 illustrates the process of IACP, and the specific steps are as follows.

1) Network information initialization, including the substrate SDN network information and the set of controllers.

2) Initial antibody population generating. The feasible solution of controller placement problem is expressed as an antibody through encoding, and the initial antibody population are generated randomly in solution space.

In our work, the antibody population is represented by An, and its number is $|An|$. $\boldsymbol{X}_i$ represents antibody, indicating a scheme of controller placement. The length of $\boldsymbol{X}_i$ is $|N_C|$, which is the number of controllers. We set $\boldsymbol{X}_i$ as

$$\boldsymbol{X}_i = [x_{i1},\ x_{i2},\ \ldots\ x_{i|N_C|}],\ x_{ij} \in \{1,2,\ldots,|N_S|\} \tag{27}$$

where x_{ij} is the serial number of a switch. x_{ij} represents controller n_j is placed at the position of the switch n_i. For $\boldsymbol{X}_i$, the set of control paths is denoted by $L_{X_iC} = L_{X_iCC} \cup L_{X_iCS}$. For $\forall l_{cc} \in L_{X_iCC}$, l_{cc} denotes the controller-to-controller connection, and it is the shortest time delay path between controllers in $\boldsymbol{X}_i$. For $\forall l_{cs} \in L_{X_iCS}$, $l_{cs} = \arg\min_{n_c \in N_C} dl(n_s, n_c)$, in which l_{cs} denotes the controller-to-switch connection of n_s. If the shortest time delay path from n_s to controller n_c is shortest in all the paths from n_s to all controllers, then l_{cs} is the shortest time delay path from n_s to controller n_c, and n_s is belonged to the control domain of n_c, i.e. $n_s \in CA(n_c)$.

For $\forall l_{cc} \in L_{X_iCC}$, $\forall l_{cs} \in L_{X_iCS}$, whether the antibody satisfies **Eqs. 13** and **14** is judged, new antibody will be generated if the constraints are not satisfied for $\boldsymbol{X}_i$.

3) Calculating the fitness value A_{X_i}, the antibody affinity S_{X_i,X_j} and the concentration C_{X_i} of X_i, according to **Eqs. (28, 29, 30)**.

$$A_{X_i} = \frac{1}{T_{mean}} = \frac{|L_C|}{\sum_{l_c \in L_{X_iC}} dl(l_c)} \tag{28}$$

$$S_{X_i,X_j} = \frac{k_{X_i,X_j}}{|N_C|} \tag{29}$$

$$C_{X_i} = \frac{1}{|An|} \sum_{X_j \in An} S_{X_i,X_j},\quad S_{X_i,X_j} = \begin{cases} 1, & S_{X_i,X_j} > \omega \\ 0, & otherwise \end{cases} \tag{30}$$

where k_{X_i,X_j} denotes the number of the same elements in X_i and X_j, ω is threshold.

4) Calculating the reproduction probability constant of antibody, we denote it as $P(X_i)$.

$$P(X_i)=\varepsilon\frac{A_{X_i}}{\sum A_{X_i}}+(1-\varepsilon)\frac{C_{X_i}}{\sum C_{X_i}} \tag{31}$$

where ε is a constant.

5) The crossover, selection and mutation operation of antibody, and new antibody population is produced, then return to step 2) to circle.

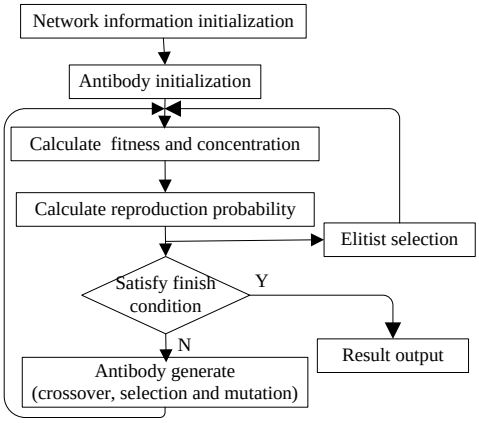


Fig. 3. Process of IACP

5.2 Different QoS Constraint vSDN Embedding Algorithm

In our work, for QoS_2, we design two algorithms: the vSDN embedding algorithm for cost optimization (CO-vSDNE) and vSDN embedding algorithm for time delay optimization (DO-vSDNE). For QoS_1, we design the vSDN embedding algorithm for minimum cost (MC-vSDNE).

Three algorithms are all two-step embedding algorithm that embeds virtual nodes first, and then embeds virtual links to paths in substrate SDN network. The difference is that: CO-vSDNE sastifies time delay constraint for QoS_2, and focuses on minimizing embedding cost. DO-vSDNE sastifies time delay constraint for QoS_2, and focuses on minimizing time delay of virtual links. MC-vSDNE doesn't consider time delay constraint, and focuses on minimizing embedding cost. The difference of the three algorithms in embedding process including: candidate substrate nodes selecing in node embedding process, and shortest path calculating in link embedding process.

1) Node embedding

In node embedding stage, the virtual nodes are sorted first, then for each virtual node to be embedded, its candicate substrate nodes are sorted and selected.

To sort virtual nodes, we define function R to calculate resource requirement of virtual nodes.

$$R(n_v)=(cpu(n_v)+tcam(n_v))*\sum_{l_v\in L(n_v)}bw(l_v) \tag{32}$$

where $L(n_v)$ denotes the set of adjacent links of n_v.

The virtual nodes sorting process is as follows: firstly, calculating R of all virtual nodes, virtual node with maximum R is selected as the root node to run Breadth First Search (BFS) algorithm. Secondly, the rest virtual nodes are divided into sets $\Omega^1_{n_v}$, $\Omega^2_{n_v}$,…, $\Omega^n_{n_v}$, where $\Omega^i_{n_v}$ represent the set in which nodes are i hop counts from root node. Thirdly, sorting nodes in

descending order in each set according to R, and then the last embedding sequence of virtual nodes is build.

For each virtual node to be embedded, its candidate substrate nodes are sorted by considering their resources and connectivity, as **Eq. 33**.

$$NF(n_s) = \frac{R(n_s)}{Dis(n_s) + \mu} \tag{33}$$

where $R(n_s)$ is calculated by **Eq. 31**, μ is a small positive number to prevent the dividend being zero, $Dis(n_s)$ is the distance parameter of n_s. Virtual node will be embedded to the substrate node with max NF.

For CO-vSDNE, the process of calculating $Dis(n_s)$ is as follows: firstly, for virtual node n_{vi} that to be embedded in turn, substrate node which satisfies **constraints (13, 17, 18, 19)** is selected to its candicate substrate node set, and the set is denoted as $Can(n_{vi})$. Secondly, another set is generated with substrate nodes which are embedded by the virtual nodes directly connected to n_{vi}, and the set is denoted as $Embed(n_{vi}) = \{n_s \mid n_v \uparrow n_s, hops(n_v, n_{vi}) = 1\}$, where $n_v \uparrow n_s$ denote that n_v is embedded to n_s, $hops(n_v, n_{vi}) = 1$ denote that n_{vi} is directly connected to n_v in vSDN request. Thirdly, the connectivity parameter of each candidate substrate node is defined as

$$Dis(n_s) = \sum_{n_k \in Embed(n_{vi})} hops(n_s, n_k),\ n_s \in Can(n_{vi}) \tag{34}$$

For DO-vSDNE, its connectivity parameter is calculated according to time delay, as **Eq. 35**.

$$Dis(n_s) = \sum_{n_k \in Embed(n_{vi})} dl(l_{sk}),\ n_s \in Can(n_{vi}) \tag{35}$$

For MC-vSDNE, substrate nodes that satisfy **constraints (17, 18, 19)** are selected as candicate substrate node set of n_{vi}**,** i.e. the time delay is not considered, and other parts of node embedding process are same with CO-vSDNE.

The details of node embedding algorithm is shown in Algorithm 1.

Algorithm 1 Node Embedding Algorithm

Input: Substrate network G_s,Virtual network request G_{vSDN}

Output: Node embedding M^N

1. **for** each virtual node $n_v \in N_v$
2. Calculate $R(n_v)$
3. **end for**
4. Take n_v with max R as root node, run BFS, divide the remaining nodes into sets $\Omega^1_{n_v}, \Omega^2_{n_v}, \ldots, \Omega^n_{n_v}$
5. Sort nodes in $\Omega^i_{n_v}$ in descending order according to their $R(n_v)$
6. Record the virtual nodes embedding sequence into *VirtualNodeList*
7. **for** each n_v in *VirtualNodeList* **do**
8. Generate candidate node set $Can(n_{vi})$
9. **if** $Can(n_{vi})$ is empty
10. **Return** NODE_EMBEDDING_FAILED
11. **else** Generate the embedded substrate node set $Embed(n_{vi})$

12. **for** each n_s in $Can(n_{vi})$
13. Calculate $NF(n_s)$
14. **end for**
15. Embed n_v to n_s with max NF, namely $M^N(n_v) = n_s$
16. **end if**
17. **end for**
18. **return** NODE_EMBEDDING_SUCCESS

2) Link embedding

In link embedding stage, we adopt shortest path algorithm to embed virtual links to substrate paths, as shown in Algorithm 2. The difference is that: CO-vSDNM takes hop least path which sastifies the time delay constraint as the result of link embedding. TO-vSDNM takes time least path which sastifies the time delay constraint as the result of link embedding. MC-vSDNM takes the hop least path without considering time delay as the result of link embedding.

Algorithm 2 Link Embedding Algorithm

Input: Substrate network G_s ,Virtual network request G_v ,Node embedding M^N

Output: Link Embedding M^L

1. **for** each virtual link $l_{uv} \in L_v$ to be embedded **do**
2. Search the shortest path between node u and v in substrate, record it as P_{uv}
3. **if** request is QoS_2 & $dl(P_{uv}) > t_{ss}$ then
4. return LINK_MAPPING_FAILED
5. **else** Embed l_{uv} to P_{uv}, namely $M^L(l_{uv}) = p$
6. **end if**
7. **end for**
8. **return** LINK_EMBEDDING_SUCCESS

5.3 Controller Adaptive Adjustment Algorithm Based on Threshold

We design the TCA algorithm to migrate switches from overload controller domain to lightly load controller domain. The process is as follows.

1) Beginning time of adjustment: once a vSDN request is successfully embedded, if there is $n_c \in N_c$ whose $Atcam(n_c) > \eta$, the first adjustment circle starts, η is a pre-set threshold.

2) The control domain to move out: $CA(n_c) = \arg\max_{n_c \in N_C}(Atcam(n_c))$, i.e. , control domain with heaviest load, then its switches will be moved out.

3) Number of switches to be moved out: the heaviest and lightest load controllers are denoted by $n_{c,\max}$ and $n_{c,\min}$ respectively, then $\Delta_{TCAM} = [Atcam(n_{c,\max}) - Atcam(n_{c,\min})]/2$ is calculated. We migrate switches of $n_{c,\max}$ until the TCAM consumption of all migration switches is more than Δ_{TCAM}.

4) The control domain to move in, the migration switches and the constraint for switch moving: switches from the heaviest load control domain $CA(n_{c,\max})$ are moved to the lightest load control domain $CA(n_{c,\min})$. Firstly, Switches in $CA(n_{c,\max})$ are sorting according to their

time delay to $n_{c,\min}$, switch that meets $dl(n_s, n_{c,\min}) \le t_{cs}$ is moved to $CA(n_{c,\min})$ in turn until the TCAM consumption of migration switches is more than Δ_{TCAM}.

Secondly, if there is no more switch in $CA(n_{c,\max})$ meets $dl(n_s, n_{c,\min}) \le t_{cs}$ and the TCAM consumption of all migration switches is still less than Δ_{TCAM}, the second lightest load controller $n_{c,\min 2}$ is selected to move switches in. Residual switches in $CA(n_{c,\max})$ are sorting according to their time delay to $n_{c,\min 2}$, and circle the migration process until the condition of step 3) is satisfied.

5) The time to end: once the first adjustment circle is finish, if there is $n_c \in N_c$, whose $Atcam(n_c) > \eta$, then start the second adjustment circle. If the total counts of adjustment circle is more than $|N_C|/2$, then the adjustment process is finished. By this way, the condition of repeatedly adjustment is avoided.

We design the controller adaptive adjustment algorithm by considering: firstly,the adjustment is triggered by threshold rather than periodic, which is more flexible and timely. Secondly, controller whose load exceeds threshold will migrate rather than execute load balance in all controllers, which help reduce the number of migration switches, and maintain network stability. Thirdly, migration switches are selected according to their time delay to target controller rather than their load, the position of selected switches are near to the boundary of control domain, which is conducive to maintain the continuation of control domain. The details of TCA algorithm are shown in Algorithm 3.

Algorithm 3 TCA Algorithm

Input: Control domain $CA(n_c)$, Control load $Atcam(n_c)$

Output: Control domain $CA_n(n_c)$

1. $count = 1$, $k1 = 1$
2. **while** $\exists n_c \in N_c$, $Atcam(n_c) > \eta$ & $count < |N_C|/2$
3. Sort all control domains in ascending order according to their $Atcam(n_c)$ as $CA(n_{c,1})$, $CA(n_{c,2})$, …, $CA(n_{c,k})$, calculat Δ_{TCAM}
4. **while** $\Delta 1 < \Delta_{TCAM}$ & $k1 < |N_c| - 1$
5. Sort switches in $CA(n_{c,k})$ according to their time delay to $n_{c,1}$
6. Migrate $n_s \in CA(n_{c,k})$ that sastify $dl(n_s, n_{c,k1}) \le t_{cs}$ to $CA(n_{k1})$ in turn, calculate $\Delta 1 = \sum contcam(n_s)$
7. $k1 = k1 + 1$
8. **end while**
9. $count = count + 1$
10. **end while**

6. Performance Evaluation and Analysis

6.1 Simulation Environment

A simulator using Matlab to evaluate the performance of our method was developed. Substrate SDN network is composed by 100 nodes and about 500 links. The positions of nodes follow a

uniform distribution in the scope of $L \times L = 1000 \times 1000$. The initial CPU, TCAM of substrate nodes and bandwidth of substrate links are real numbers following a uniform distribution between 50 and 100. Time delay of substrate links is calculated by $dl(l_s) = 3d(l_s)/2c$, where $d(l_s)$ denotes the length of l_s, we set $c = 3e5$. Number of controllers is 5, the threshold to trigger controller adjustment is set as $\eta = 500$.

The number of virtual nodes in each vSDN request is uniformly distributed between 5 and 15. Positions of nodes follow a uniform distribution in the scope of $L \times L = 1000 \times 1000$ and all position constraints are set as 300. The required CPU and bandwidth are real numbers uniformly distributed between 10 and 30. We set the required TCAM of each virtual node as $tcam(n_v) = |N_v| - 1$, the TCAM consumption of substrate nodes that virtual links span is set as $tcam(n_s) = 2$. The vSDN requests arrive by the Possion distribution with the rate of 10 per 100 time units, and the lifetime follows an exponentially distribution with the mean of 200 time units. Simulations were run 3000 time units to reach a stable state, which contain about 300 vSDN requests, of which QoS_2 requests account for 1/3, i.e, there is about 100 QoS_2 requests.

The parameters that we use in our simulations are summarized in **Table 1.**

Table 1. Parameters in simulations

	Parameters	**Values**
$\boldsymbol{G}_S$	Number of substrate nodes	100
	Positions of substrate nodes	a uniform distribution in the scope of $L \times L = 1000 \times 1000$
	Number of substrate links	about 500
	CPU and TCAM of substrate node bandwidth of substrate link	A uniform distribution from 50 to 100
	Time delay of substrate link	$dl(l_s) = 3d(l_s)/2c$
N_C	Number of controllers	5
$\boldsymbol{G}_{vSDN}$	Number of virtual nodes	A uniform distribution from 5 to 15
	Positions of virtual nodes	a uniform distribution in the scope of $L \times L = 1000 \times 1000$
	Position constraints of virtual nodes	$D(n_{vi}) = 300$
	CPU of virtual node bandwidth of virtual link	A uniform distribution from10 to 30
	TCAM of virtual node	$tcam(n_v) = \|N_v\| - 1$ for virtual node $tcam(n_s) = 2$ for substrate nodes that virtual links span
vSDN events	Arrival rate of VN requests	10 per 100 time units
	Lifetime of VN request	200 time units exponentially distribution
η	threshold to trigger	500

Based on the parameters setting above, the time delay of substrate links is distributed in the scope of [1.9049e-4, 1.2e-3], the time delay of shortest path between each substrate node

pairs is distributed in the scope of [1.9049e-4, 6.5e-3]. In our work, we set time delay constraint of controller-to-controller and controller-to-switch as $t_{cc} = 0.002$ and $t_{cs} = 0.003$ respectively. For QoS_2, the time delay constraints for controller-to-switch and virtual link are set as $t_{cs1} = 0.0015$ and $t_p = 0.004$ respectively. The parameters setting of the four time delay constraint is based on that, the ratios of shortest path time delay between each substrate node pairs which sastify $dl(P_s) < 0.002$, $dl(P_s) < 0.003$, $dl(P_s) < 0.0015$ and $dl(P_s) < 0.004$ are about 32%, 54%, 22% and 80% seperately. Based on that, we evaluate our method for embedding different QoS constraint vSDN under multiple controllers.

To avoid the disturbance of random factors to the experimental results, each simulation is carried out for 10 times, and the average value was recorded as the final results.

Fig. 4 illustrates the flow chart of simulation. Modules of controller placement, vSDN embedding and controller adjustment are designed in Section 5. Simulation inputs including: Substrate SDN network, set of controllers, vSDN sequence (the set of vSDN requests), vSDN events which record the arrive time, survival time and end time of all vSDN requests. At beginning, controllers are placed and control domains are partitioned according to $\boldsymbol{G}_S$ and N_C, then simulation starts.

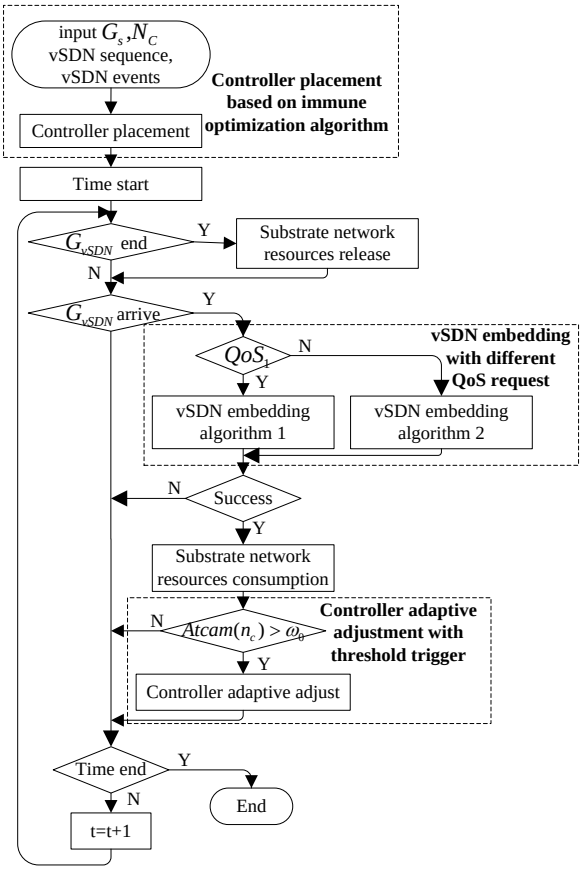


Fig. 4. Flow chart of simulation

We set average time delay of control path, load balance and embedding cost as the optimal objects. As there are no existing algorithms that tackle the problem, we compare our

method with its three variations to evaluate performance. The method is consist of three aspects, we set BM1 as the baseline method which places controllers by IACP, embeds QoS_2 request by CO-vSDNM, embeds QoS_1 request by MC-vSDNM, and adjusts controllers by TCA. BM2 embeds QoS_2 request by CO-vSDNM, other steps are the same with BM1. We set comparison method CM1 which embeds all vSDN requests by TF-SVNM [16] without considering the time delay constraint; CM2 places controllers by a random way which is set to evaluate the effect of controller placement on embedding. The comparison of the four methods is listed in **Table 2**.

Table 2. Methods comparison

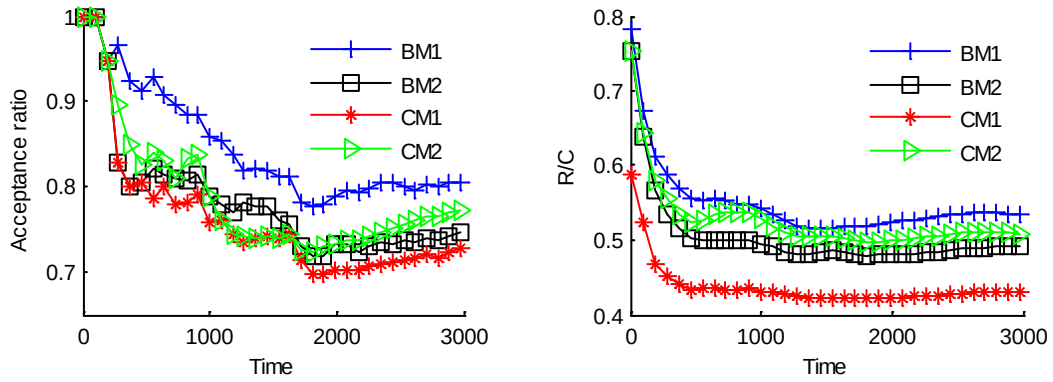
Methods	Description			
	Controller placement	QoS_2 request embedding	QoS_1 request embedding	Controller adjustment
BM1	IACP	CO-vSDNE	MC-vSDNE	TCA
BM2	IACP	DO-vSDNE	MC-vSDNE	TCA
CM1	IACP	TF-vSDNE	TF-vSDNE	TCA
CM2	Random placement	CO-vSDNE	MC-vSDNE	TCA

6.2 Simulation Results

1） Acceptance ratio and R/C of vSDN requests

Fig. 5 (a) illustrates the acceptance ratios of four methods. The acceptance ratio of BM1 (about 80%) is better than others, follow is CM2 (about 76%), then are BM2 (about 74%) and CM1 (about 72%). The reason for the better performance of BM1 than CM2 is that the controllers are placed more reasonable, and it is easier to meet the time delay constraint by BM1. Performance of BM1 is better than BM2, which shows that the strategy of cost optimization is better than time delay optimization for QoS_2 request embedding.

Fig. 5 (b) illustrates the R/C of four methods. The R/C of BM1 (about 0.53) is best, the follow is CM2 (about 0.51), then are BM2 (about 0.49) and CM1 (about 0.42). The better performance of BM1 is due to BM1 embeds two kinds vSDN requests all by cost optimization embedding algorithms, which reduce the cost. The comparison result shows that BM1 has the best combination of strategies in controller placement, vSDN embedding and controller adjustment. BM1 has better performance than others in acceptance ratio and R/C.



(a) Acceptance ratio (b) R/C

Fig. 5. Comparison between four method

2) Success number of different QoS constraint vSDN requests

Table 3 shows the success number of different QoS constraint vSDN requests. In one experiment, the numbers of vSDN requests with QoS_2 and QoS_1 are 92 and 211 seperately. Obviously, BM1 embeds the most QoS_2 requests, its number is 56. This is because the restriction to embed QoS_2 request is more strict. BM1 embedding QoS_2 requests with less cost while satisfies time delay constraint, which is conducive for the embedding of later QoS_2 requests. CM1 embeds all vSDN requests with minimum time delay of virtual links, which leads to the heavy load of key substrate links, and makes more difficult for later QoS_2 requests embedding. On the other hand, the results of QoS_1 request embedding of the four methods are similar. This is because QoS_1 request has no time delay constraint, the embedding is influenced only by the resource richness of substrate network.

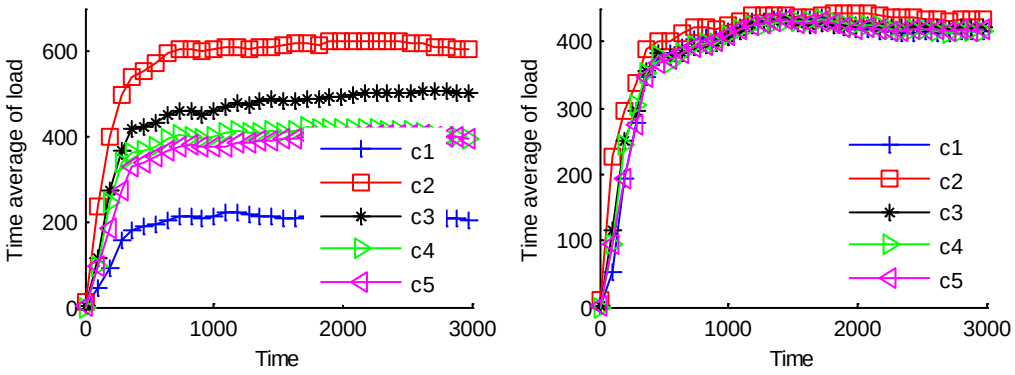
Table 3. Comparison of success number

	BM1	**BM2**	**CM1**	**CM2**
Total number	244	224	218	232
QoS_2 requests number	56	36	27	43
QoS_1 requests number	188	188	191	189

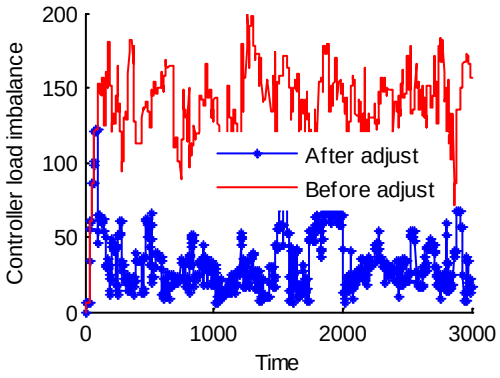
By comparing the performance in **Fig. 5** and **Table 3**, simulation results show that our method do well in the problem of different QoS constraint vSDN network embedding under multiple controllers. The performance comparison of BM1 and CM2 shows that controller placement affects the performance of vSDN embedding. Better placement helps sastify the time delay constraint. The performance comparison of BM1 and CM1 shows that links with short time delay in substrate SDN network are important for high QoS constraint vSDN request. The strategy of cost optimization reduces the occupancy of key substrate link while satisfies the time delay constraint, which helps the embedding for high QoS constraint vSDN requests.

3) Load imbalance degree of controllers

Fig. 6 shows the effect of TCA on controllers' load. **Figs. 6 (a)** and **(b)** illustrate the time average of controller load before and after adjustment. **Fig. 6 (c)** illustrates the load imbalance degree of controllers at each time before and after adjustment. **Fig. 6 (a)** shows that loads of controllers are great inequality before adjustment. This is because controllers manage different switches, virtual nodes of vSDN requests are embedded to different switches and consume different amount of TCAM, all these lead to load imbalance of controllers. Then **Fig. 6 (b)** shows that the time average of load of the five controllers is very close. TCA improves load balance between controllers after adjustment. **Fig. 6 (c)** also shows that the load imbalance degree of controllers is apparently decreased after adjustment. Simulation results show that the TCA algorithm can improve the load balance of controllers and avoid overload.



(a) Time average of load before adjustment (b) Time average of load after adjustment



(c) Comparison of controller load imbalance

Fig. 6. Effect of controller adjustment on load balance

7. Conclusion

In this paper, we study the problem of different QoS constraint vSDN embedding under multiple controllers. We establish the mathematical models both of the controller placement and adjustment problem, and the vSDN embedding problem. We propose a controller placement method based on immune optimization algorithm, design the node embedding algorithm and link embedding algorithm for different QoS constraint vSDN embedding, design the controller adaptive adjustment algorithm based on threhold to improve load balance of controllers. We analyze the performance of the method through simulation. Simulation results show that our method optimizes the placement of controllers, improves load balance, and satisfies user's requirement for different QoS constraint vSDN service. The proposed method achieves good performance in terms of acceptance ratio and R/C for vSDN embedding.

However, there are also some problems appeared in our work. Firstly, the way we abstract and apply the TCAM resource is simple, which need more efforts to insight to the mechanism of SDN. Secondly, we place the controllers to optimize the time delay of control paths according to the information of controller number and substrate SDN network. There are many more application scenes to explore other controller placement methods. Thirdly, we define function R to calculate resource requirement of virtual nodes and resource richness of substrate nodes, the function is too simple to describe the relationship between resources of

CPU, TCAM, and bandwidth. It needs more efforts to explore how to make the best of all kinds of resources. In the future work, we will extend our work to solve the problem mentioned above. Besides, we plan to consider the vSDN reconfiguration in our algorithm to further improve the embedding performance.

References

[1] A. Wang, M. Iyen, R. Dutta, et al. "Network virtualization: technologies, perspectives, and frontiers," *Journal of Lightwave Technology*, vol.31, no.4, pp.523-537, August, 2012. Article (CrossRef Link)

[2] T. Anderson, L. Peterson, S. Shenker, et al. "Overcoming the Internet impasse through virtualization," *Computer*, vol.38, no.4, pp. 34-41, May, 2005. Article (CrossRef Link)

[3] C. Zhang, Y. Cui, H. Tang, et al. "State-of-the-Art Survey on Software-Defined Networking (SDN)," *Journal of Software*, vol.26, no.1, pp. 62-81, May, 2015. Article (CrossRef Link)

[4] X. Yin, S. Huang, S. Wang, et al. "Software defined virtualization platform based on double-FlowVisors in multiple domain networks," in *Proc. of 8th International Conference on Communications and Networking in China*. Beijing, China, pp. 776-780, August 14, 2013. Article (CrossRef Link)

[5] E. Salvadori, R. Doriguzzi, A. Broglio, et al. "Generalizing virtual network topologies in OpenFlow-based networks," in *Proc. of 54th Annual IEEE Global Telecommunications Conference*. Houston, USA, pp. 1-6, December 5-9, 2011. Article (CrossRef Link)

[6] X. Jin, J. Rexford, D. Walker. "Incremental update for a compositional SDN hypervisor," in *Proc. of the Third Workshop on Hot Topics in Software Defined Networking*, Chicago, USA, pp. 187-192, August 22, 2014. Article (CrossRef Link)

[7] J. Liu, T. Huang, C. Zhang, et al. "Research on network virtualization slicing mechanism in SDN-based testbeds," *Journal on Communications*, vol. 37, no. 4, pp. 2016083-1-13, April, 2016. Article (CrossRef Link)

[8] B. Heeler, R. Sherwood, N. Mckeown, et al. "The controller placement problem," in *Proc. of the First Workshop on Hot Topics in Software Defined Networks*, New York, America, pp.7-12, August 13, 2012. Article (CrossRef Link)

[9] L. Yao, Y. Chen, F. Song, et al. "Delay-aware Controller Placement for Fast Response in Software-defined Network," *Journal of Electronics & Information Technology*, vol. 36, no. 12, pp. 2802-2808, December, 2014. Article (CrossRef Link)

[10] Y. Hu, W. Wang, X. Gong, et al. "Reliability-aware Controller Placement for Software-Defined Networks," in *Proc. of 2013 IFIP/IEEE International Symposium on Integrated Network Management*, Ghent, Belgium, pp. 672-675, May 27-31, 2013.Article (CrossRef Link)

[11] M. Bari, A. Roy, S. Chowdhury, et al. "Dynamic Controller Provisioning in Software Defined Networks," in *Proc. of 9th International Conference on Network and Service Management*, University of Zrich, Switzerland, pp.18-25, October 14-18, 2013. Article (CrossRef Link)

[12] A. Dixit, F. Hao, S. Mukherjee, et al. "Towards an elastic distributed SDN controller," in *Proc. of ACM SIGCOMM*, HongKong, China, pp.7-12, August 12-16, 2013.Article (CrossRef Link)

[13] L. Yao, P. Hong, W. Zhang, et al. "Controller Placement and Flow based Dynamic Management Problem towards SDN," in *Proc. of 2015 IEEE International Conference on Communication Workshop*, Hefei, China, pp.363-368, June 8, 2015. Article (CrossRef Link)

[14] L. Wang, H. Qu and J. Zhao. "A strategy of controller placement in software defined networks using binary paticle swarm optimization," *Journal of Xi'an Jiaotong University*, vol. 49, no. 6, pp.67-71, June, 2015.Article (CrossRef Link)

[15] S. Nashid, A. Reaz, R. Shihabur, et al. "Connectivity-aware Virtual Network Embedding," in *Proc. of 2016 IFIP Networking Conference and Workshops*, Vienna, Austria, pp.45-54, May 17-19, 2016. Article (CrossRef Link)

[16] H. Cui, W. Gao and J. Liu, "A virtual network embedding algorithm based on virtual topology connection feature," in *Proc. of 16th International Symposium on Wireless Personal Multimedia Communications*, Atlantic City, USA, pp. 1-5, June 24-27, 2013. Article (CrossRef Link)

[17] J. Ding, T. Huang, J. Liu, Y. Liu, "Virtual network embedding based on real-time topological attributes," *Frontiers of Information Technology & Electronic Engineering*, vol.16, no.2, pp.109-118, February, 2015. Article (CrossRef Link)

[18] J. Liao, M. Feng, T. Li, J. Wang and S. Qing, "Topology-aware virtual network embedding using multiple characteristics," *KSII Transactions on Internet and Information Systems*, vol. 8, no. 1, pp. 145-164, January, 2014. Article (CrossRef Link)

[19] Z. Wang, J. Wu, Y. Wang, et al. "Survivable virtual network mapping using optimal backup topology in virtualized SDN," *China Communications*, vol. 11, no. 2, pp. 26-37, February, 2014. Article (CrossRef Link)

[20] R. Mijumbi, J. Serrat, J. Rubio, et al. "Dynamic resource management in SDN-based virtualized networks," in *Proc. of the 10th International Conference on Network and Service Management*, Rio deJaneiro, Brazil, pp.412-417, November 17-21, 2014. Article (CrossRef Link)

[21] D. Mehmet and A. Mostafa. "Design and analysis of techniques for mapping virtual networks to software-defined network substrates," *Computer Communications*, vol.45, no.1, pp.1-10, June, 2014. Article (CrossRef Link)

[22] S. Gong, J. Chen, S. Zhao, et al. "An Efficient and Coordinated Mapping Algorithm in Virtualized SDN Networks," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 7, pp. 701-716, July, 2016.Article (CrossRef Link)

[23] A. Tootoonchian, Y. Ganjali. "HyperFlow: A distributed control plane for OpenFlow," in *Proc. of the 2010 internet network management conference on Research on enterprise networking*, USENIX Association, pp. 13-23, 2010.Article (CrossRef Link)

[24] C. Macapuna, C. Rothenberg, et al. "In- packet bloom filter based data center networking with distributed openflow controllers," in *Proc. of GLOBECOM Workshops, 2010 IEEE*, pp. 584-588, 2010. Article (CrossRef Link)

[25] A. Tam, K. Xi, et al. "Use of devolved controllers in data center networks," in *Proc. of Computer Communications Workshops*, pp. 596-601, 2011.Article (CrossRef Link)

[26] Y. Hassas, Y. Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proc. of the first workshop on Hot topics in software defined networks*, pp. 19-24, 2012. Article (CrossRef Link)

Zhiyuan Zhao is a Ph.D. candidate in Computer Application Technology from Air Force Engineering University, China. His research interests include virtual network embedding and software defined networking.

Xiangru Meng is a professor in Communication and Information System from Air Force Engineering University, China. His research interests include next generation Internet, cloud computing and software defined networking.

Siyuan Lu is a Ph.D. candidate in Computer Application Technology from Academy of military sciences, China. His research interests include network virtualiztion and network survivability.

Yuze Su is a Ph.D. candidate in Computer Application Technology from Air Force Engineering University, China. His research interests include network virtualiztion and cloud computing.