

## An efficient and coordinated mapping algorithm in virtualized SDN networks<sup>\*</sup>

Shui-qing GONG<sup>†</sup>, Jing CHEN, Qiao-yan KANG, Qing-wei MENG, Qing-chao ZHU, Si-yi ZHAO

(College of Information and Navigation, Air Force Engineering University, Xi'an 710077, China)

<sup>†</sup>E-mail: gsq0121@126.com

Received Nov. 10, 2015; Revision accepted Feb. 16, 2016; Crosschecked June 9, 2016

**Abstract:** Software-defined networking (SDN) enables the network virtualization through SDN hypervisors to share the underlying physical SDN network among multiple logically isolated virtual SDN networks (vSDNs), each with its own controller. The vSDN embedding, which refers to mapping a number of vSDNs to the same substrate SDN network, is a key problem in the SDN virtualization environment. However, due to the distinctions of the SDN, such as the logically centralized controller and different virtualization technologies, most of the existing embedding algorithms cannot be applied directly to SDN virtualization. In this paper, we consider controller placement and virtual network embedding as a joint vSDN embedding problem, and formulate it into an integer linear programming with objectives of minimizing the embedding cost and the controller-to-switch delay for each vSDN. Moreover, we propose a novel online vSDN embedding algorithm called CO-vSDNE, which consists of a node mapping stage and a link mapping stage. In the node mapping stage, CO-vSDNE maps the controller and the virtual nodes to the substrate nodes on the basis of the controller-to-switch delay and takes into account the subsequent link mapping at the same time. In the link mapping stage, CO-vSDNE adopts the  $k$ -shortest path algorithm to map the virtual links. The evaluation results with simulation and Mininet emulation show that the proposed CO-vSDNE not only significantly increases the long-term revenue to the cost ratio and acceptance ratio while guaranteeing low average and maximum controller-to-switch delay, but also achieves good vSDN performance in terms of end-to-end delay and throughput.

**Key words:** Software-defined networking (SDN), Network virtualization, Controller placement, Virtual network embedding, Coordination

<http://dx.doi.org/10.1631/FITEE.1500387>


**CLC number:** TP393

### 1 Introduction

Software-defined networking (SDN) has emerged as a promising technology for network programmability and experiments. The main ideas of SDN include separation of the control plane from the data plane, a logically centralized controller managing the data plane, and a uniform southbound inter-

face, e.g., OpenFlow (McKeown *et al.*, 2008), between the control plane and the data plane. With such a decoupled network architecture, SDN can significantly simplify network management and enable network innovations. Network virtualization has been proposed as a fundamental ingredient of the future Internet paradigm, which abstracts the physical substrate network (SN) and allows multiple heterogeneous virtual networks (VNs) to coexist on the SN (Khan *et al.*, 2012; Wang *et al.*, 2013). Each VN is a collection of virtual nodes connected by virtual links hosted on the physical SN. Moreover, multiple VNs are isolated from each other and can provide end-to-end services for end users. The virtualization of SDN networks promises to use the merits of SDN

<sup>\*</sup> Project supported by the National Natural Science Foundation of China (Nos. 61201209 and 61401499), the Natural Science Foundation of Shaanxi Province, China (No. 2015JM6340), and the Industrial Science and Technology Project of Shaanxi Province, China (No. 2016GY-087)

 ORCID: Shui-qing GONG, <http://orcid.org/0000-0002-3657-3666>  
© Zhejiang University and Springer-Verlag Berlin Heidelberg 2016

and network virtualization, and has therefore gained considerable attention from both industry and academia in recent years.

The key component for the virtualization of SDN networks is the SDN hypervisor, which abstracts the physical SDN network into multiple isolated slices (VNs) for multiple tenants. Each slice is managed by its respective controller and can be operated independently by different tenants. Generally, existing hypervisors can be classified into two categories (Blenk *et al.*, 2016): the centralized hypervisor that consists of a single central entity, e.g., FlowVisor (Sherwood *et al.*, 2010), Advisor (Salvadori *et al.*, 2011), and VerTIGO (Corin *et al.*, 2012), and the distributed hypervisor that consists of several distributed virtualization functions, e.g., AutoSlice (Bozakov and Papadimitriou, 2012), FlowN (Drutskoy *et al.*, 2013), and NVP (Koponen *et al.*, 2014). In this study, we assume FlowVisor as the hypervisor for the virtualization of SDN networks as it is well-documented, more established, and widely used in experimental environments.

FlowVisor is a special-purpose OpenFlow controller for virtualizing and sharing SDN networks, and it sits between the control and data planes acting as the network virtualization layer. With FlowVisor, the physical SDN network is sliced in terms of switch CPU, link bandwidth, and flow tables (Sherwood *et al.*, 2010). Each slice with an OpenFlow controller has its own view of virtual topology that is specified as a list of network nodes (switches) and links. Moreover, with FlowVisor in the middle that works in a transparent manner, all OpenFlow messages between slice controllers and slice switches are intercepted and rewritten according to the slice policies that define the network resources and the slice controller allocated to each slice.

The virtualization of a given physical SDN network through FlowVisor allows multiple tenants to run distinct applications on their own slices. We can consider a slice (VN) along with its corresponding controller as a virtual SDN network (vSDN). Since multiple vSDNs share the same physical SDN network with finite resources, it is crucial to efficiently assign physical network resources to vSDN requests that specify the resource requirements, which is also known as the VN embedding problem (Fischer *et al.*, 2013) in a network virtualization en-

vironment. As one of the main challenges in network virtualization, VN embedding is known to be NP-hard (Andersen, 2002) and a number of heuristic approaches have been proposed by researchers. However, because of the distinctions brought by SDN, most of the existing embedding algorithms cannot be directly applied to the SDN virtualization environment. In particular, since each vSDN on a physical SDN network has its own controller, the controller placement problem (Heller *et al.*, 2012) should be addressed when performing VN embedding. Such a problem aims to find the optimal switch location in the physical SDN network for collocating the controller to minimize the controller-to-switch delay. As a result, the controller can communicate effectively with all the switches in the same vSDN. Moreover, the differences between virtualization technology in the SDN network environment, especially resource sharing (Sherwood *et al.*, 2010), and that in traditional networks require modifications to existing VN embedding approaches.

In this paper, we focus on the mapping techniques in virtualized SDN networks to address the challenges brought by SDN. In contrast to previous work, we consider controller placement and VN embedding as a joint vSDN embedding problem for the first time, and formulate it into a multi-objective integer linear programming (ILP) to optimize the controller-to-switch delay and the mapping cost. Due to the NP-hard nature of the ILP, we then design a novel online vSDN embedding algorithm called 'CO-vSDNE' to solve this formulation. CO-vSDNE consists of two stages: (1) the node mapping stage in which controller placement and virtual node mapping are tackled, and (2) the link mapping stage. In the node mapping stage, we first attach the controller to the substrate node with the largest controller location selection factor (CLSF), which exploits the delay information of the entire network and pre-considers the subsequent virtual node and link mapping to obtain high revenue and low controller-to-switch delay. Then we construct the virtual node mapping tree (VNMT) for each vSDN according to the resources they required, and adopt the breadth first search (BFS) strategy to map virtual nodes onto the substrate nodes with the largest node ranking (NR) value, which measures the substrate node with the local resource, controller-to-switch delay, and the number of hops of

substrate paths. In the link mapping stage, we map the virtual control links (controller-to-switch connections) and the virtual links using the  $k$ -shortest path algorithm. CO-vSDNE combines controller placement with VN embedding and introduces better coordination among controller placement, virtual node mapping, and link mapping, which enables more efficient resource utilization while guaranteeing lower controller-to-switch delay. Note that we use ‘embedding’ and ‘mapping’ interchangeably throughout this paper.

This paper presents the following major contributions:

1. To the best of our knowledge, we make the first attempt to study the online vSDN embedding problem with coordination.
2. We formulate the problem into a multi-objective ILP and design a heuristic algorithm called CO-vSDNE to minimize the controller-to-switch delay and mapping cost.
3. We conduct extensive experiments to evaluate CO-vSDNE in terms of delay, cost, revenue, throughput, etc., which demonstrates the effectiveness and efficiency of our proposed algorithm.

## 2 Related work

The VN embedding problem refers to the mapping of VN requests to specific physical nodes and paths in the SN. Due to the multiple constraints on virtual nodes and links, VN embedding has been shown to be NP-hard. Generally, VN embedding consists of two stages: virtual node mapping where virtual nodes are mapped to substrate nodes in a one-to-one manner while satisfying resource requirements of virtual nodes, and virtual link mapping where virtual links are mapped to loop-free substrate paths while satisfying the resource requirements of virtual links.

Most previous work focuses on the general embedding problem in traditional networks, and has proposed different VN embedding algorithms with specific objectives or constraints. Cheng *et al.* (2011) considered the resources and topological attributes of nodes together, and proposed an efficient embedding algorithm through topology-aware node ranking. Di *et al.* (2014) presented a reliable heuristic VN em-

bedding algorithm for efficient bandwidth sharing by using the cross and backup sharing scheme. Liu *et al.* (2015) focused on security-aware VN embedding, and proposed two heuristic algorithms based on the models of security demands in network virtualization. Su *et al.* (2014) considered the energy factor in performing VN embedding. They formulated the energy consumption models and proposed two energy-aware VN embedding algorithms. However, the distinctions of the SDN environment, such as the logically centralized controller and the different virtualization technologies, call for novel VN embedding algorithms.

There also exist a few studies that focus on VN embedding in SDN networks. Wang *et al.* (2014) studied the survivable VN embedding in virtualized SDN. They exploited optimal backup topology to survive a single link failure and proposed a survivable VN embedding algorithm. Mijumbi *et al.* (2014) studied the dynamic resource management in SDN-based virtualized networks. Zhou *et al.* (2014) proposed a multi-domain VN embedding mechanism for SDN to improve the scalability in performance. However, the aforementioned work on VN embedding in SDN networks considered neither the controller placement problem nor flow entry resource allocation in node mapping. They are similar to traditional VN embedding approaches and not applicable to the SDN virtualization environment.

The controller placement problem aims to find the best switch location to attach the controller directly, such that the control message delays from the controller to the switches are minimized. Such a problem was not widely studied in academia community. In particular, Heller *et al.* (2012) adopted the average and maximize delays between the controller and the switches as the performance metrics, and evaluated hundreds of existing network topologies through extensive simulations to find the optimal controller location in SDN networks. Hu *et al.* (2013) considered the reliability of the control traffic path and proposed four different controller placement algorithms for SDN networks. In this study, we combine controller placement with VN embedding for mapping online vSDN requests to the physical SDN network. Specifically, we introduce the coordination among controller placement, virtual node mapping, and link mapping, which helps increase the

utilization of physical network resources while keeping low controller-to-switch delay. Different from previous work (Demirci and Ammar, 2014) which performed the above-mentioned three stages in a sequential and uncoordinated manner, our work mends this gap.

### 3 System model

To design the mapping of vSDN in the SDN virtualization environment, in this section we first describe the architecture of SDN virtualization, and then provide the network model.

#### 3.1 SDN virtualization architecture

Indicated by McKeown *et al.* (2008), a typical architecture of the SDN network consists of a logically centralized controller and multiple OpenFlow (OF) switches (Fig. 1). The OF switches in the data plane communicate with the controller in the control plane via a southbound interface, i.e., OpenFlow. Moreover, the controller is responsible for assigning forwarding flow rules to switches and managing the entire network. In this study, we assume that controller-to-switch connections are deployed in in-band mode where each control path uses the existing link connections between switches in the data plane, which is more practical and cost-favorable. The architecture of SDN virtualization using FlowVisor is shown in Fig. 2. With the FlowVisor layer injected between the control plane and the data plane, multiple vSDNs can coexist on a shared underlying physical SDN infrastructure and are isolated from each other. Each vSDN consists of a managing controller and a virtual network which is composed of

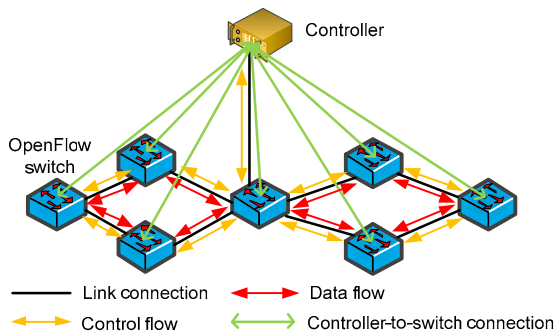


Fig. 1 Architecture of the SDN network

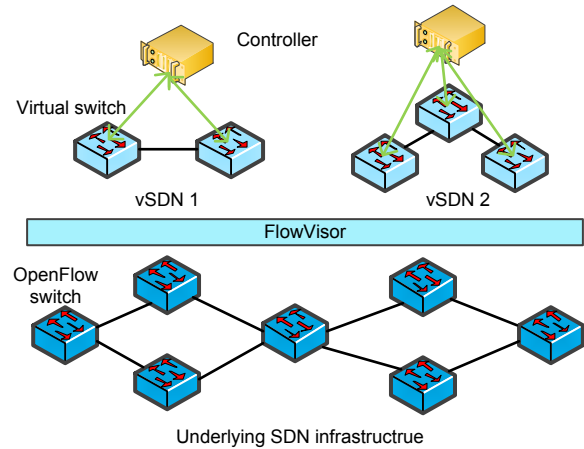


Fig. 2 Architecture of SDN virtualization

a set of virtual switch nodes and virtual links. A virtual switch node is hosted on a particular physical node, and a virtual link spans over a path in the underlying physical SDN network. Moreover, since the control traffic in vSDNs is operated in in-band mode, the controller in each vSDN is placed at the same location as any switch in a physical SDN infrastructure, and each controller-to-switch connection is viewed as a virtual control link that also spans over a path in the underlying physical SDN network.

#### 3.2 Network model

We take the underlying physical SDN infrastructure in Fig. 2 as the SN and model it by a weighted undirected network graph  $G_S=(N_S, L_S)$ , where  $N_S$  refers to the substrate node (OF switch) set with a total number of  $|N_S|$  nodes and  $L_S$  the substrate link set with a total number of  $|L_S|$  links. Since FlowVisor slices the SN in terms of multiple dimensions (Sherwood *et al.*, 2010), in this study we take the typical available switch CPU capacity and the available ternary content-addressable memory (TCAM) capacity as node attributes, and the available bandwidth and delay as link attributes, where switch CPU is used for communication message processing and TCAM for flow table processing. In addition, we denote  $P_S$  as the set of loop-free substrate paths in the SN.

Generally, a vSDN request specifies a tenant's requirements including the topology of the VN, virtual node resources, virtual link resources, etc. Similarly, we model the VN of a vSDN request as a weighted undirected network graph  $G_V=(N_V, L_V)$ ,

where  $N_V$  refers to the set of virtual nodes with a total number of  $|N_V|$  nodes and  $L_V$  the set of virtual links with a total number of  $|L_V|$  links. We express the requirements on virtual nodes and links in terms of the attributes of nodes and links in the SN. We also denote the managing controller of a vSDN request by  $n_c$  and the set of virtual control links by  $L_C$ . For each virtual control link  $l_{cv} \in L_C$ , we also consider that it requires a certain amount of bandwidth from the substrate path to which  $l_{cv}$  is mapped, to avoid control traffic congestion.

#### 4 Multi-objective optimization for joint VN embedding and the controller placement problem

When a vSDN request arrives, the SN should allocate resources to that request according to its requirements. The vSDN request will be rejected or postponed when there are not enough substrate resources available. When the vSDN request expires, the allocated substrate resources are released. The mapping of a vSDN request to the SN can be decomposed into two major components, i.e., VN embedding and controller placement. In the following, we explain each component in detail and formulate the joint VN embedding and controller placement problem as a multi-objective ILP.

##### 4.1 VN embedding

The VN embedding for a vSDN request is defined as a mapping  $M$  from  $G_V$  to a subset of  $G_S$ , such that the resource requirements of virtual nodes and links are satisfied, i.e.,  $M:G_V \rightarrow (N_S^*, P_S^*, R_S^n, R_S^l)$ , where  $N_S^* \subseteq N_S$ ,  $P_S^* \subseteq P_S$ , and  $R_S^n$  and  $R_S^l$  represent the resources of substrate nodes and links allocated to the vSDN request, respectively. As mentioned in Section 2, VN embedding can be generally decomposed into the virtual node mapping stage and the virtual link mapping stage.

###### 4.1.1 Virtual node mapping

The virtual node mapping selects suitable substrate nodes to host virtual nodes in the VN such that constraints on virtual nodes are satisfied. We define it as a mapping  $M_N: N_V \rightarrow N_S^*$  from  $N_V$  to a subset of  $N_S$ .

Since each substrate node has finite CPU and TCAM capacity, the required resources of a virtual node must not exceed the available resources of the corresponding mapped substrate node. Let  $\text{CPU}(n_s)$  and  $\text{TCAM}(n_s)$  denote the amount of available CPU and TCAM capacity of substrate node  $n_s \in N_S$ , respectively, and  $\text{CPU}(n_v)$  and  $\text{TCAM}(n_v)$  denote the amount of required CPU and TCAM capacity of virtual node  $n_v \in N_V$ , respectively. Therefore, the capacity constraints for virtual node mapping are formulated as

$$\forall n_i \in N_V, \forall n_j \in N_S: \\ x_j^i \cdot \text{CPU}(n_i) \leq \text{CPU}(n_j), \quad (1)$$

$$x_j^i \cdot \text{TCAM}(n_i) \leq \text{TCAM}(n_j), \quad (2)$$

where  $x_j^i \in \{0, 1\}$  is a binary variable indicating the mapping between a virtual node and a substrate node (i.e.,  $x_j^i = 1$  if virtual node  $n_i$  is mapped to substrate node  $n_j$ , and 0 otherwise). In virtual node mapping, each virtual node in the same VN is mapped to a different substrate node, and the corresponding constraints can be formulated as

$$\forall n_i \in N_V: \sum_{n_j \in N_S} x_j^i = 1, \quad (3)$$

$$\forall n_j \in N_S: \sum_{n_i \in N_V} x_j^i \leq 1. \quad (4)$$

Eq. (3) ensures that each virtual node is mapped to just one substrate node, and Eq. (4) ensures that virtual nodes from the same vSDN request are mapped to different substrate nodes.

###### 4.1.2 Virtual link mapping

In virtual link mapping, we assume that the SN supports only the unsplitable flow, and each virtual link is mapped to a substrate path between the corresponding substrate nodes that host the end virtual nodes of that link. Thus, virtual link mapping can also be defined as a mapping  $M_L: L_V \rightarrow P_S^*$  from  $L_V$  to a subset of  $P_S$ . Since each substrate link has finite bandwidth capacity which is shared by multiple virtual links, the total required bandwidth capacity of virtual links must not exceed the available bandwidth capacity of the corresponding mapped substrate link. Let  $\text{BW}(l_{ij})$  denote the amount of available bandwidth capacity of substrate link  $l_{ij} \in L_S$ , and  $\text{BW}(l_{uv})$  the

required bandwidth capacity of virtual link  $l_{uv} \in L_V$ . The bandwidth capacity constraint for virtual link mapping is formulated as

$$\forall l_{ij} \in L_S : \sum_{l_{uv} \in L_V} f_{ij}^{uv} \cdot \text{BW}(l_{uv}) \leq \text{BW}(l_{ij}), \quad (5)$$

where  $f_{ij}^{uv} \in \{0, 1\}$  is a binary variable indicating the mapping between a virtual link and a substrate link (i.e.,  $f_{ij}^{uv} = 1$  if the substrate path to which virtual link  $l_{uv}$  is mapped goes through the substrate link  $l_{ij}$ , and 0 otherwise). In addition, the connectivity constraints for virtual link mapping, which ensure that substrate links can be connected as a path to host a virtual link, are provided as

$$\forall n_j \in N_S, \forall l_{uv} \in L_V: \sum_{l_{ji} \in L_S} f_{ji}^{uv} - \sum_{l_{ij} \in L_S} f_{ij}^{uv} = \begin{cases} 1, & x_j^u = 1, \\ -1, & x_j^v = 1, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

#### 4.1.3 Objectives

Similar to previous work (Cheng *et al.*, 2011; Li *et al.*, 2012; Ding *et al.*, 2015), we consider the long-term average revenue, long-term average cost, and long-term revenue to cost ( $R/C$ ) ratio as the VN embedding objectives.

The revenue of embedding a VN  $G_V$  at time  $t$  refers to the total resources it demands, which can be formulated as

$$R(G_V) = \sum_{n_v \in N_V} (\text{CPU}(n_v) + \text{TCAM}(n_v)) + \sum_{l_v \in L_V} \text{BW}(l_v). \quad (7)$$

The cost of embedding a VN  $G_V$  at time  $t$  is the total substrate resources allocated to that VN, which can be formulated as

$$C(G_V) = \sum_{n_v \in N_V} (\text{CPU}(n_v) + \text{TCAM}(n_v)) + \sum_{l_{uv} \in L_V} \sum_{l_{ij} \in L_S} f_{ij}^{uv} \cdot \text{BW}(l_{uv}). \quad (8)$$

Therefore, the long-term average revenue is defined as

$$R = \lim_{T \rightarrow \infty} \frac{\sum_{t=0}^T R(G_V)}{T}. \quad (9)$$

The long-term average cost is defined as

$$C = \lim_{T \rightarrow \infty} \frac{\sum_{t=0}^T C(G_V)}{T}. \quad (10)$$

The long-term  $R/C$  ratio is defined as

$$R/C = \lim_{T \rightarrow \infty} \frac{\sum_{t=0}^T R(G_V)}{\sum_{t=0}^T C(G_V)}. \quad (11)$$

We can see that the  $R/C$  ratio refers to the resource utilization of the SN. The larger the  $R/C$  is, the higher the utilization of substrate resources is, and the more efficient the embedding algorithm is.

#### 4.2 Controller placement

The controller placement problem in in-band mode aims to find the optimal switch node location in the SN to which the controller of the vSDN is attached, such that the average controller-to-switch delay can be minimized. Let  $y_j^c \in \{0, 1\}$  be a binary variable, and  $y_j^c = 1$  if controller  $n_c$  is attached to the substrate node  $n_j$ , and 0 otherwise. Thus, the controller placement constraint is formulated as

$$\sum_{n_j \in N_S} y_j^c = 1, \quad (12)$$

which ensures that controller  $n_c$  is attached to one switch node in the SN.

Each virtual control link  $l_{cv} \in L_C$  in a vSDN request is mapped to a substrate path between the substrate node where controller  $n_c$  is collocated and the substrate node that hosts virtual node  $n_v$ . Let  $\text{BW}(l_{cv})$  denote the required bandwidth capacity of  $l_{cv}$ . The corresponding capacity constraint for virtual control link mapping is formulated as

$$\forall l_{ij} \in L_S : \sum_{l_{cv} \in L_C} g_{ij}^{cv} \cdot \text{BW}(l_{cv}) \leq \text{BW}(l_{ij}), \quad (13)$$

where  $g_{ij}^{cv} \in \{0, 1\}$  is the binary variable indicating

the mapping between a virtual control link and a substrate link (i.e.,  $g_{ij}^{cv}=1$  if virtual control link  $l_{cv}$  spans over substrate link  $l_{ij}$ , and 0 otherwise). In addition, the connectivity constraints for virtual control link mapping are provided as

$$\forall n_j \in N_S, \forall l_{cv} \in L_C: \quad \sum_{l_{ji} \in L_S} g_{ji}^{cv} - \sum_{l_{ij} \in L_S} g_{ij}^{cv} = \begin{cases} 1, & y_j^c = 1, \\ -1, & x_j^v = 1, \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

Let  $D(l_{ij})$  denote the traffic delay of substrate link  $l_{ij} \in L_S$ . Inspired by Heller *et al.* (2012), we define the average controller-to-switch delay  $D$  for a vSDN request as the ratio between the total delay of substrate links that virtual control links go over and the number of virtual control links, which is formulated as

$$D = \frac{\sum_{l_{cv} \in L_C} \sum_{l_{ij} \in L_S} g_{ij}^{cv} \cdot D(l_{ij})}{|L_C|}, \quad (15)$$

where  $|L_C|$  denotes the total number of virtual control links in  $L_C$ .

Note that the switch node to which the controller is directly attached can send its control messages without going through the SN.

### 4.3 Formulation of joint VN embedding and the controller placement problem

The formal definition of the vSDN embedding problem for joint VN embedding and controller placement in virtualized SDN networks is stated as follows. Given a substrate SDN network modeled by  $G_S=(N_S, L_S)$ , a vSDN request with a VN modeled by  $G_V=(N_V, L_V)$ , a managing controller  $n_c$ , a set of controller-to-switch connections  $L_C$ , and the corresponding resource requirements, map the vSDN request to the SN in a way that will: (1) satisfy the resource requirements of virtual nodes, virtual links, and controller-to-switch connections, (2) minimize the cost of embedding the VN for this vSDN request, and (3) minimize the average controller-to-switch delay for this vSDN request.

Therefore, with the aforementioned respective formulations for VN embedding and controller placement, we formulate the vSDN embedding problem into a multi-objective ILP:

1. Objective:

$$\begin{cases} \min C(G_V), \\ \min D. \end{cases} \quad (16)$$

2. Capacity constraints:

$$\forall n_i \in N_V, \forall n_j \in N_S: \quad \begin{cases} x_j^i \cdot \text{CPU}(n_i) \leq \text{CPU}(n_j), \\ x_j^i \cdot \text{TCAM}(n_i) \leq \text{TCAM}(n_j). \end{cases} \quad (17)$$

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

$$\sum_{l_{uv} \in L_V} f_{ij}^{uv} \cdot \text{BW}(l_{uv}) + \sum_{l_{cv} \in L_C} g_{ij}^{cv} \cdot \text{BW}(l_{cv}) \leq \text{BW}(l_{ij}). \quad (18)$$

3. Connectivity constraints:

$$\forall n_j \in N_S, \forall l_{uv} \in L_V: \quad \sum_{l_{ji} \in L_S} f_{ji}^{uv} - \sum_{l_{ij} \in L_S} f_{ij}^{uv} = \begin{cases} 1, & x_j^u = 1, \\ -1, & x_j^v = 1, \\ 0, & \text{otherwise.} \end{cases} \quad (19)$$

$\forall n_j \in N_S, \forall l_{cv} \in L_C:$

$$\sum_{l_{ji} \in L_S} g_{ji}^{cv} - \sum_{l_{ij} \in L_S} g_{ij}^{cv} = \begin{cases} 1, & y_j^c = 1, \\ -1, & x_j^v = 1, \\ 0, & \text{otherwise.} \end{cases} \quad (20)$$

4. Variable constraints:

$$\forall n_i \in N_V: \sum_{n_j \in N_S} x_j^i = 1, \quad (21)$$

$$\forall n_j \in N_S: \sum_{n_i \in N_V} x_j^i \leq 1, \quad (22)$$

$$\forall n_c: \sum_{n_j \in N_S} y_j^c = 1, \quad (23)$$

$$\forall n_i \in N_V, \forall n_j \in N_S: x_j^i \in \{0, 1\}, \quad (24)$$

$$\forall n_c, n_j \in N_S: y_j^c \in \{0, 1\}, \quad (25)$$

$$\forall l_{uv} \in L_V, \forall l_{ij} \in L_S: f_{ij}^{uv} \in \{0, 1\}, \quad (26)$$

$$\forall l_{cv} \in L_C, \forall l_{ij} \in L_S: g_{ij}^{cv} \in \{0, 1\}. \quad (27)$$

## 5 Heuristic algorithm design

Solving an ILP is known to be NP-hard (Schrijver, 1998). Although exact algorithms (e.g., branch and bound, cutting plane) can achieve optimal results, they may incur exponentially increasing running time. As a result, they cannot scale to solve large vSDN embedding problems. In this section, we propose a novel heuristic vSDN embedding algorithm, called ‘CO-vSDNE’, to achieve the trade-off between embedding performance and time complexity. More specifically, CO-vSDNE performs vSDN embedding in a way that minimizes both mapping cost and controller-to-switch delay while guaranteeing proper computing time. By treating the controller of a vSDN as a special virtual node and controller-to-switch connections as special virtual links, CO-vSDNE is presented as a two-stage algorithm: the node mapping stage that handles controller node mapping and virtual node mapping, and the link mapping stage that handles virtual control link mapping and virtual link mapping.

### 5.1 Node mapping

In this stage, we first perform controller node mapping, and then virtual node mapping.

#### 5.1.1 Controller node mapping

We have two goals to achieve in this mapping. First, we want to minimize the controller-to-switch delay by selecting the optimal substrate switch location to collocate the controller. Second, as controller node mapping affects the following virtual node mapping and link mapping, we want to perform it in a way that facilitates the subsequent mapping steps.

Towards the above two goals, in our algorithm, we attach the controller node to the substrate node with the largest controller location selection factor (CLSF). The CLSF of substrate node  $n_i$  is defined as

$$\text{CLSF}(n_i) = \sum_{n_j \in N_s} (\text{CPU}(n_j) + \text{TCAM}(n_j)) \frac{\text{bw}(n_i, n_j)}{\text{delay}(n_i, n_j)}, \quad (28)$$

where  $\text{bw}(n_i, n_j)$  is the available bandwidth along the shortest path from  $n_i$  to  $n_j$ , described as the minimum bandwidth along this shortest path, and  $\text{delay}(n_i, n_j)$  is the traffic delay from  $n_i$  to  $n_j$ , which is

calculated by the total delay along the shortest path from  $n_i$  to  $n_j$ .

Since virtual node mapping is not performed, we need to attach the controller node to the substrate node with the lowest average delay to all other substrate nodes, such that the average controller-to-switch delay can be minimized. Thus, CLSF considers the average delay from a node to all the others in the SN. Meanwhile, CLSF takes the available resource of substrate nodes and the bandwidth of substrate links into consideration, which can make subsequent virtual node mapping and link mapping easier, and increase the probability of accepting the vSDN requests.

#### 5.1.2 Virtual node mapping

In this mapping stage, the goal is to find a substrate node to host each virtual node while minimizing the delay to the controller, and satisfying their node constraints in terms of CPU and TCAM requirements. As virtual node mapping affects subsequent link mapping, we also need to pre-consider link mapping in this virtual node mapping stage.

In our algorithm, we first sort the virtual nodes according to their required resource in descending order. The required resource of a virtual node  $n_v$  is defined as

$$H(n_v) = (\text{CPU}(n_v) + \text{TCAM}(n_v)) \sum_{l_v \in L(n_v)} \text{BW}(l_v), \quad (29)$$

where  $L(n_v)$  is the set of adjacent links that connect directly to virtual node  $n_v$ , and  $\text{BW}(l_v)$  denotes the required bandwidth resource of virtual link  $l_v$ . The larger the  $H$  value of node  $n_v$  is, the more the resources it demands, and thus it is more difficult to map due to the limitation of resources in the SN.

Then, we construct the virtual node mapping tree (VNMT) for the VN in a vSDN request, which can effectively decrease the number of hops of substrate paths to which virtual links are mapped. VNMT is a mapping tree constructed according to the  $H$  values of virtual nodes and the topology of VN. Specifically, the construction of the VNMT works as follows. For a given VN, we first select the virtual node with the largest  $H$  value as the root node. Then those virtual nodes that connect directly to the root by virtual links become its children from left to right according to



descending order of their  $H$  values. Other virtual nodes in VN are constructed recursively in a similar way.

After constructing VNMT, CO-vSDNE adopts the breadth first search (BFS) strategy to map the virtual nodes. For the root node of VNMT, CO-vSDNE maps it to the substrate node with the largest  $H$  value, which represents the available resources of a substrate node. For the other virtual nodes, CO-vSDNE maps them to substrate nodes with the largest NR, which is a metric for selecting the substrate nodes in the virtual node mapping stage (see Eq. (31)). In particular, when we map a virtual node  $n_v$ , we first build a set of candidate substrate nodes  $\mathcal{Q}(n_v)$  for  $n_v$ , which consists of substrate nodes that are unmapped with any other node in the same vSDN request, and whose available node resources can satisfy the requirements of  $n_v$ , i.e.,

$$\mathcal{Q}(n_v) = \{n_s \mid \text{CPU}(n_v) \leq \text{CPU}(n_s), \text{TCAM}(n_v) \leq \text{TCAM}(n_s), n_s \in N_S\}. \quad (30)$$

Then we map virtual node  $n_v$  to candidate substrate node  $n_s$  with the largest NR value, which is defined as

$$\text{NR}(n_s) = \frac{H(n_s)}{\text{delay}(M_N(n_c), n_s) \cdot \text{hops}(M_N(f(n_v)), n_s)}, \quad (31)$$

where  $H(n_s)$  denotes the available resources of substrate node  $n_s$  and is calculated by Eq. (29),  $M_N(n_c)$  denotes the substrate node to which controller  $n_c$  is attached,  $\text{delay}(M_N(n_c), n_s)$  denotes the control message delay from  $M_N(n_c)$  to  $n_s$  in the SN.  $f(n_v)$  denotes the father node of  $n_v$  in VNMT,  $M_N(f(n_v))$  denotes the substrate node to which  $f(n_v)$  is mapped, and  $\text{hops}(M_N(f(n_v)), n_s)$  denotes the number of hops for the shortest path from  $f(n_v)$  to  $n_s$  in the SN.

The substrate node's NR is proportional to its  $H$  value, and inversely proportional to  $\text{delay}(M_N(n_c), n_s)$  and  $\text{hops}(M_N(f(n_v)), n_s)$ . The reasons why we take NR as the node selection metric are as follows: (1) The substrate node with a larger  $H$  value indicates that the node's available resource is richer, and selecting the substrate node with a larger  $H$  value helps balance the stress on substrate nodes; (2) Mapping the virtual node to the substrate node with low  $\text{delay}(M_N(n_c), n_s)$  can reduce the controller-to-switch delay; (3) As the

virtual node  $f(n_v)$  has been mapped to the SN, if  $\text{hops}(M_N(f(n_v)), n_s)$  is too large, the cost of mapping the virtual link between  $n_v$  and  $f(n_v)$  will be too large according to Eq. (8), resulting in low resource utilization of the SN. Therefore, based on VNMT and NR, the virtual node mapping algorithm can keep the mapped substrate nodes connected closely to each other, and is favorable for the following link mapping. Algorithm 1 shows the details of the node mapping algorithm.

---

#### Algorithm 1 Node mapping

---

**Input:**  $G_S=(N_S, L_S)$ ,  $G_V=(N_V, L_V)$ ,  $n_c$

**Output:** node mapping solution

/\*Controller node mapping\*/

Compute the shortest path for all node pairs  $(n_i, n_j)$ , s.t.  $n_i, n_j \in N_S$

Calculate the delay for all such node pairs

**for** each substrate node  $n_s \in N_S$  **do**

    Calculate CLSF( $n_s$ )

**end for**

Attach controller node  $n_c$  to the substrate node with the largest CLSF value

/\*Virtual node mapping\*/

**for** each virtual node  $n_v \in N_V$  **do**

    Calculate  $H(n_v)$

**end for**

Construct the VNMT for  $G_V$  according to  $H$  values of virtual nodes in descending order

Map the root node of VNMT to the substrate node with the largest  $H$  value

**for** other unmapped nodes in the VNMT **do**

    Choose the virtual node  $n_v$  using the BFS strategy, and construct the substrate candidate set  $\mathcal{Q}(n_v)$  for  $n_v$

**if**  $\mathcal{Q}(n_v) = \emptyset$  **then**

**return** NODE\_MAPPING\_FAILED

**else**

**for** each candidate node  $n_s \in \mathcal{Q}(n_v)$  **do**

            Calculate NR( $n_s$ )

**end for**

        Map  $n_v$  to the candidate node  $n_s$  with the largest NR value

**end if**

**end for**

**return** NODE\_MAPPING\_SUCCESS

---

## 5.2 Link mapping

In the link mapping stage, similar to the previous work (Cheng et al., 2011; Li et al., 2014), CO-vSDNE adopts the  $k$ -shortest path algorithm (Eppstein, 1998) to map each virtual link to a substrate path between the corresponding substrate nodes that host the end

nodes of that virtual link (Algorithm 2). Since different substrate paths to which virtual links are mapped may share the same substrate links and compete for their limited bandwidth resources, it may be difficult or even impossible to map virtual links with large bandwidth requirements due to the limitation of bandwidth resources in the SN. Therefore, virtual links with large required bandwidth should be mapped in priority. Specifically, to map a virtual control link  $l_{cv} \in L_C$ , the link mapping algorithm searches the  $k$ -shortest paths by increasing  $k$ , and stops the search if it finds a set of paths that have the same number of hops and satisfy the bandwidth constraint of  $l_{cv}$ . We then map  $l_{cv}$  to the substrate path with the lowest delay in this set. For each virtual link  $l_{uv} \in L_V$ , we also adopt the  $k$ -shortest path algorithm to map virtual links to substrate paths.

### 5.3 CO-vSDNE algorithm

In a realistic SDN virtualization scenario, vSDN requests may not always arrive one by one in regular intervals. The scenario that multiple vSDN requests arrive at the same time may occur. Thus, to be applied in a real-time scenario, our vSDN embedding algorithm is designed to be executed once in every constant time interval. This time interval depends on the permissible waiting periods of incoming vSDN requests and the processing time of the embedding work.

The detailed procedure of CO-vSDNE is shown in Algorithm 3. The node ranks for CLSF,  $H$ , and NR can be computed in polynomial time in terms of  $|G_S|$  and  $|G_V|$ , and the link mapping algorithm can also be finished in polynomial time in terms of  $|G_S|$ ,  $|G_V|$ ,  $|L_C|$ , and  $k$ . Thus, CO-vSDNE is a polynomial-time algorithm.

## 6 Performance evaluation

In this section, we first evaluate our proposed vSDN embedding algorithm in terms of controller-to-switch delay,  $R/C$  ratio, and acceptance ratio with extensive simulations. Then, to gain insights into how our proposed algorithm is influencing vSDN performance, we focus on the advantages of our algorithm in terms of end-to-end delay and throughput with Mininet (Lantz et al., 2010) emulation.

---

### Algorithm 2 Link mapping

---

**Input:**  $G_S=(N_S, L_S)$ ,  $G_V=(N_V, L_V)$ ,  $n_c$ ,  $L_C$ , node mapping solution

**Output:** link mapping solution

*/\*Virtual control link mapping\*/*

Rank the virtual control links  $l_{cv} \in L_C$  according to the required bandwidth in descending order

**for** each unmapped virtual control link  $l_{cv} \in L_C$  **do**

    Search the  $k$ -shortest paths between the selected nodes in the SN

**if** link bandwidth constraint of  $l_{cv}$  is satisfied **then**

        Map  $l_{cv}$  to the  $k$ -shortest substrate path with the lowest delay

**else**

**return** LINK\_MAPPING\_FAILED

**end if**

**end for**

*/\*Virtual link mapping\*/*

Rank the virtual links  $l_{uv} \in L_V$  according to the required bandwidth in descending order

**for** each unmapped virtual link  $l_{uv} \in L_V$  **do**

    Search the  $k$ -shortest paths between the selected nodes in the SN

**if** a path is found that can satisfy the link bandwidth constraint of  $l_{uv}$  **then**

        Map  $l_{uv}$  to this path

**else**

**return** LINK\_MAPPING\_FAILED

**end if**

**end for**

**return** LINK\_MAPPING\_SUCCESS

---



---

### Algorithm 3 CO-vSDNE

---

Sort the vSDN requests within the same time interval according to their revenues in descending order

**for** all unmapped vSDN requests **do**

    Select the vSDN request with the maximum revenue

    Map the virtual nodes and the controller of this request using the node mapping algorithm

**if** NODE\_MAPPING\_SUCCESS **then**

        Map the virtual links of this request using the link mapping algorithm

**end if**

**if** LINK\_MAPPING\_SUCCESS **then**

        Occupy the substrate resources and update the state of the substrate network

        Set the state of this request to MAPPING\_SUCCESS

**continue**

**end if**

    Set the state of this request to MAPPING\_FAILED

**end for**

---

## 6.1 Simulation environment

Similar to most previous work, we use the GT-ITM tool (Zegura *et al.*, 1996) to generate topologies of the SN and vSDN requests. The SN is configured with 100 nodes, and each pair of substrate nodes is randomly connected with a probability of 0.5. For each substrate node, the capacities of available CPU and TCAM resources are uniformly distributed between 50 and 100, respectively. For each substrate link, the delay follows a uniform distribution ranging from 5 to 20 ms, and the capacity of available bandwidth resource is also uniformly distributed between 50 and 100.

We varied the scales of vSDN requests in small, regular, and large sizes; specifically, the number of virtual nodes is uniformly distributed between 2 and 10, 10 and 20, 20 and 50, respectively. For each vSDN request of different scales, the virtual link connectivity rate of the virtual node pair is set to 0.5, and each virtual node is connected directly to the controller node by a virtual control link. All of the quality-of-service (QoS) requirements (i.e., CPU, TCAM, and bandwidth) of virtual nodes, virtual control links, and virtual links are real numbers uniformly distributed between 0 and 50. Moreover, similar to most previous work on VN embedding, we assume that the arrival of vSDN requests follows the Poisson process with an average arrival rate of 5 vSDN requests per 100 time units, and each vSDN request has an exponentially distributed lifetime with an average of 1000 time units.

As controller-to-switch delay and the mapping cost are the two objects of CO-vSDNE optimization, we use the following metrics to evaluate our algorithm: (1) the long-term average (maximum) controller-to-switch delay, which is defined as the ratio between the total average (maximum) controller-to-switch delay for each vSDN that has been mapped successfully and the number of successfully mapped vSDNs in the long run, (2) the long-term  $R/C$  ratio according to Eq. (11), and (3) the acceptance ratio of vSDN requests, defined as the ratio between the number of vSDN requests mapped successfully to the number of total arrival vSDN requests. As there are no existing algorithms that tackle the online vSDN embedding problem, we compare CO-vSDNE with its two simpler variations: the uncoordinated naive delay-minimizing vSDN embed-

ding algorithm (DM-vSDNE), and the pure cost-minimizing vSDN embedding algorithm (CM-vSDNE) (Table 1). We run our simulations under each condition for 50 000 time units to achieve a stable-state performance. Ten instances are performed for each simulation and the average values are recorded as the final result.

**Table 1 Algorithm comparison**

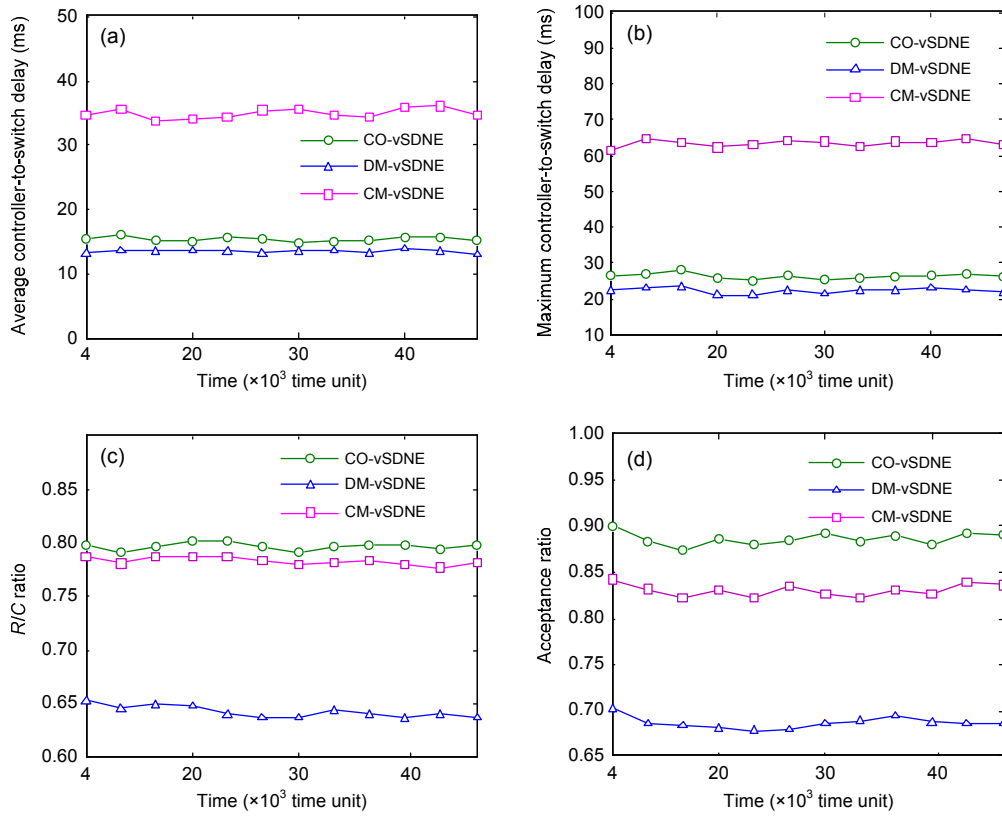
Algorithm	Description
CO-vSDNE	Our efficient and coordinated vSDN embedding algorithm
DM-vSDNE	Place the controller of vSDN randomly, and then map virtual nodes to the closest substrate nodes with sufficient node resources to minimize the controller-to-switch delay
CM-vSDNE	Place the controller of vSDN randomly, and execute the virtual node mapping algorithm of CO-vSDNE without regard to delays (i.e., cut the delay part in $NR(n_s)$ ) to minimize the mapping cost

## 6.2 Simulation results

### 6.2.1 Comparison on regular-sized vSDN scale

We first investigate the performance of our CO-vSDNE algorithm in terms of long-term average and maximum controller-to-switch delays, the long-term  $R/C$  ratio, and the acceptance ratio. The simulations are performed on regular-sized vSDN scale. The comparison results are shown in Fig. 3.

Fig. 3a shows the long-term average controller-to-switch delay for all three vSDN embedding algorithms in stable state. It can be seen that CO-vSDNE produces very close delays to DM-vSDNE, even though CO-vSDNE is trying to minimize the mapping cost, and CM-vSDNE produces obviously higher delays. This is because CO-vSDNE considers attaching the controller to the substrate node with the lowest delay compared to all other substrate nodes in controller node mapping, which makes the following virtual node mapping easier and therefore obtains low delays. The long-term maximum controller-to-switch delays shown in Fig. 3b present a similar pattern: the maximum delays produced by CO-vSDNE and DM-vSDNE are close to each other, and the maximum delays produced by CM-vSDNE are higher as expected.



**Fig. 3 Comparisons between our algorithm and others on regular-sized vSDN scales in stable state: (a) long-term average controller-to-switch delay; (b) long-term maximum controller-to-switch delay; (c) long-term  $R/C$  ratio; (d) acceptance ratio**

Fig. 3c shows the long-term  $R/C$  ratio for all three vSDN embedding algorithms in stable state. It can be seen that the  $R/C$  ratio of CO-vSDNE is a little higher than that of CM-vSDNE and obviously higher than that of DM-vSDNE. This is because: (1) CO-vSDNE pre-considers the following mapping steps in controller node mapping, which facilitates the subsequent mappings; (2) according to the definition of the  $R/C$  ratio, the cost of mapping virtual links has significant effect on the  $R/C$  ratio. Thus, CO-vSDNE achieves a higher  $R/C$  ratio because it considers reducing the number of hops of substrate paths, and tries to map the virtual nodes to the substrate nodes nearby in the virtual node mapping stage, leading to a low cost of link mapping and higher  $R/C$  ratio.

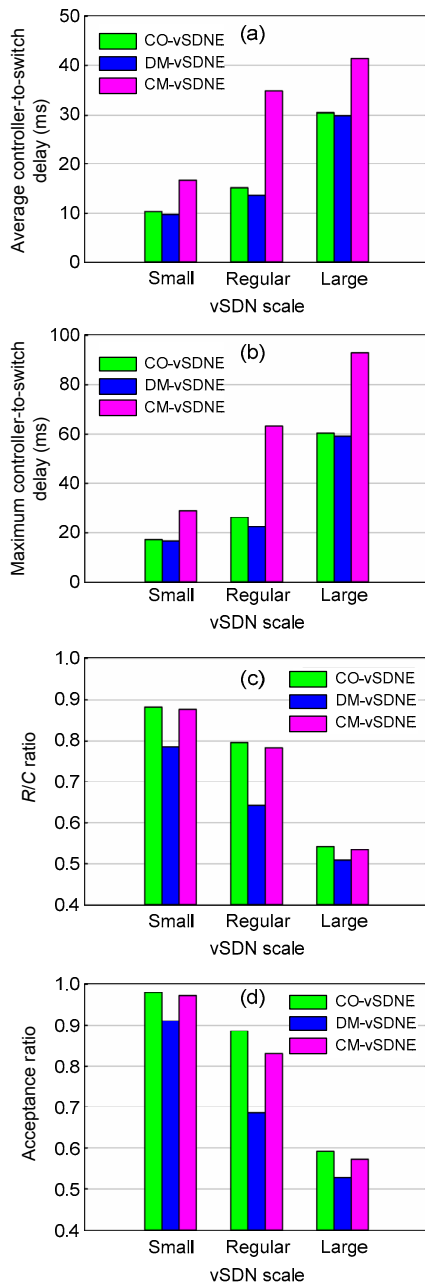
Fig. 3d shows the acceptance ratio for all three vSDN embedding algorithms in stable state. The acceptance ratio of our CO-vSDNE is larger than those of others. Moreover, the average acceptance ratio of CO-vSDNE is almost 0.887, which is the largest among all considered algorithms. This is

because CO-vSDNE coordinates the two mapping stages by pre-considering link mapping in the node mapping stage. In addition, the higher  $R/C$  ratio obtained by CO-vSDNE indicates a higher resource utilization, which further leads to accepting more vSDN requests at finite SN resources.

### 6.2.2 Impact of vSDN scales on performance

To evaluate the impact of vSDN scales on the performance of our CO-vSDNE algorithm, we generate two more different-sized vSDN requirements mentioned in Section 6.1: small-sized vSDNs with the number of virtual nodes uniformly distributed between 2 and 10, and large-sized vSDNs with the number of virtual nodes distributed between 20 and 50. The comparison results among three algorithms are shown in Fig. 4. From these results, we have the following observations.

First, as the vSDN scale increases, all three algorithms maintain the same rank in terms of the long-term average and maximum controller-to-switch



**Fig. 4** Comparisons between our algorithms and others on different-sized vSDN scales in stable state: (a) long-term average controller-to-switch delay; (b) long-term maximum controller-to-switch delay; (c) long-term  $R/C$  ratio; (d) acceptance ratio

delays, the long-term  $R/C$  ratio, and the acceptance ratio. In addition, since CO-vSDNE coordinates node mapping and link mapping when embedding a vSDN request, it always obtains a larger value than DM-vSDNE in terms of the  $R/C$  ratio and the acceptance ratio, and a smaller value than CM-vSDNE

in terms of average and maximum controller-to-switch delays.

Second, as the vSDN scale increases, from Figs. 4a and 4b we can see that the long-term average and maximum controller-to-switch delays of CO-vSDNE and DM-vSDNE are close to each other, while the relative delays of CO-vSDNE to that of CM-vSDNE decline. Specifically, for small-sized vSDN requests, CO-vSDNE saves 62.5% average delay and 66.7% maximum delay than CM-vSDNE. However, for regular-sized vSDN requests, CO-vSDNE saves 56.2% average delay and 58.5% maximum delay than CM-vSDNE. When the vSDN scale expands to large sizes, the relative average delays drop to 26.7%, and the relative maximum delays drop to 35.5%.

Third, from Figs. 4c and 4d, which illustrate the long-term  $R/C$  ratio and the acceptance ratio, we can see that as the vSDN requests scale from small to large sizes, the  $R/C$  ratio and the acceptance ratio decrease for all three algorithms. This is because with the increasing scale of vSDN requests, they require more resources (i.e., CPU, TCAM, and bandwidth) when embedded to the SN. Thus, virtual links are more likely to be assigned to longer substrate paths and consume more resources due to the limitation of bandwidth resources in the SN, leading to a decreasing  $R/C$  ratio. Besides, because of the increasing resource consumption in the SN by larger vSDN requests, the mapping of newly arriving vSDN requests may fail for lacking of substrate resources, resulting in a low acceptance ratio.

Fourth, as the vSDN scale increases, from Figs. 4c and 4d we can see that CO-vSDNE performs almost as well as CM-vSDNE in terms of the long-term  $R/C$  ratio and the acceptance ratio. The reasons are as follows: (1) Both CM-vSDNE and CO-vSDNE try to map virtual nodes to substrate nodes nearby in the virtual node mapping stage. As a result, the number of hops of substrate paths to which virtual links are mapped is reduced, leading to low cost of mapping virtual links and high  $R/C$  ratio. (2) CM-vSDNE places the controller of vSDN randomly, which might result in higher cost of mapping virtual control links and fewer available bandwidth resources for mapping virtual links compared with CO-vSDNE. However, since it maps virtual nodes according to NR without regard to delay, CM-vSDNE could reduce the

number of hops of substrate paths to which virtual links might be mapped more effectively than CO-vSDNE in the virtual node mapping stage. (3) For small-sized vSDN requests, they consume fewer substrate resources when embedded to the SN, and thus the SN has enough resources for accepting more vSDN requests, leading to a high  $R/C$  ratio and a high acceptance ratio among all three algorithms. When the vSDN scale expands to a large size, the  $R/C$  ratio and acceptance ratio for all three algorithms become low due to high resource requirements of vSDN requests and the lack of substrate resources.

### 6.3 End-to-end delay and throughput emulation

For the evaluation of end-to-end delay and throughput, we use Mininet 2.2 to emulate the operation of an SDN with multiple vSDNs placed on it. Twenty different vSDNs are embedded on six SNs sliced using FlowVisor. We set the number of substrate nodes between 50 and 100 and the number of virtual nodes for each vSDN from 5 to 20. Other settings are the same as those in Section 6.1. The end-to-end delays are measured by ‘ping’, and the throughputs are measured by executing file transfer between each pair of nodes via the Transmission Control Protocol (TCP) with the Iperf tool. We record the arithmetic means as the final results (Figs. 5 and 6).

Fig. 5 shows the average end-to-end delays for all three vSDN embedding algorithms. It can be seen that CO-vSDNE performs considerably better than CM-vSDNE, offering an average delay reduction of 48.5% compared to CM-vSDNE. This is because CO-vSDNE considers the coordination between the controller placement and the following mapping steps, and tries to map the virtual nodes to the substrate nodes nearby in the SN, resulting in low end-to-end delays. DM-vSDNE maps virtual nodes to substrate nodes which are as close to the controller as possible, so it is good at minimizing both controller-to-switch delays and end-to-end delays, and can perform similar to CO-vSDNE.

Fig. 6 shows the average throughput for all three vSDN embedding algorithms. It can be seen that the average throughput of our CO-vSDNE is close to that of CM-vSDNE and obviously larger than that of DM-vSDNE. The reason is that both CO-vSDNE and CM-vSDNE prefer to select substrate nodes with large available resources in the node mapping stage,

which helps balance the stress on the SN and leads to larger throughput. DM-vSDNE performs the worst in terms of throughput since it maps the vSDN in a confined area without any regard to node or link stress.

The emulation results indicate that our proposed vSDN embedding algorithm can improve the performance of vSDNs in a realistic network emulation environment.

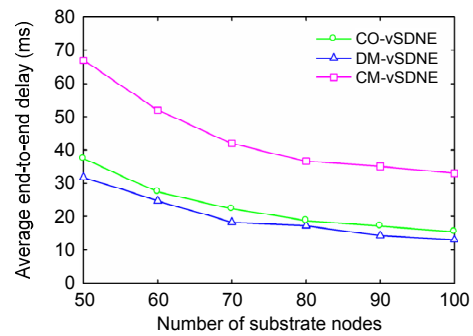


Fig. 5 Comparison of average end-to-end delay

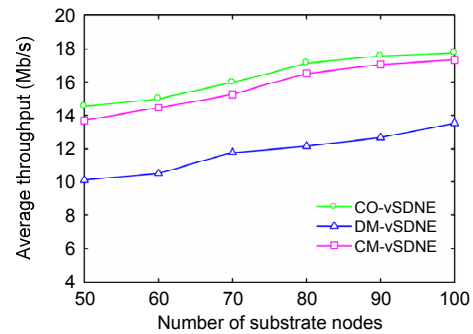


Fig. 6 Comparison of average throughput

## 7 Conclusions

VN embedding is a key problem in network virtualization. In this paper, we study the technique to perform vSDN embedding in an SDN virtualization environment. The goal of our study is to minimize the controller-to-switch delay and the mapping cost. Specifically, we first build the network model in an SDN virtualization environment and formulate the vSDN embedding problem into a multi-objective integer linear programming. To solve this formulation, we propose a novel online vSDN embedding algorithm CO-vSDNE, which performs controller placement, virtual node mapping, and link mapping in a

coordinated way. Simulation and emulation results showed that the proposed algorithm achieves good performance in terms of the average and maximum controller-to-switch delays, the  $R/C$  ratio, the acceptance ratio, the end-to-end delay, and the throughput under different-sized vSDN scales.

In the future, we plan to consider the vSDN re-configuration in our algorithm to further improve the embedding performance. Besides, we will extend our work by considering more aspects of vSDN embedding problems, e.g., energy consumption, fault tolerance, and security issues.

## References

- Andersen, D.G., 2002. Theoretical Approaches to Node Assignment. Available from <http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps> [Accessed on Sept. 20, 2010].
- Blenk, A., Basta, A., Reisslein, M., et al., 2016. Survey on network virtualization hypervisors for software defined networking. *IEEE Commun. Surv. Tutor.*, **18**(1):655-685. <http://dx.doi.org/10.1109/COMST.2015.2489183>
- Bozakov, Z., Papadimitriou, P., 2012. AutoSlice: automated and scalable slicing for software-defined networks. Proc. ACM CoNEXT Student Workshop, p.3-4. <http://dx.doi.org/10.1145/2413247.2413251>
- Cheng, X., Su, S., Zhang, Z., et al., 2011. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Comput. Commun. Rev.*, **41**(2):38-47. <http://dx.doi.org/10.1145/1971162.1971168>
- Corin, R.D., Gerola, M., Riggio, R., et al., 2012. VeRTIGO: network virtualization and beyond. European Workshop on Software Defined Networks, p.24-29. <http://dx.doi.org/10.1109/EWSDN.2012.19>
- Demirci, M., Ammar, M., 2014. Design and analysis of techniques for mapping virtual networks to software-defined network substrates. *Comput. Commun.*, **45**:1-10. <http://dx.doi.org/10.1016/j.comcom.2014.03.008>
- Di, H., Anand, V., Yu, H.F., 2014. Design of reliable virtual infrastructure with resource sharing. *Comput. Netw.*, **62**:137-151. <http://dx.doi.org/10.1016/j.bjp.2013.09.022>
- Ding, J., Huang, T., Liu, J., et al., 2015. Virtual network embedding based on real-time topological attributes. *Front. Inform. Technol. Electron. Eng.*, **16**(2):109-118. <http://dx.doi.org/10.1631/FITEE.1400147>
- Drutskey, D., Keller, E., Rexford, J., 2013. Scalable network virtualization in software-defined networks. *IEEE Internet Comput.*, **17**(2):20-27. <http://dx.doi.org/10.1109/MIC.2012.144>
- Epstein, D., 1998. Finding the  $k$  shortest paths. *SIAM J. Comput.*, **28**(2):652-673. <http://dx.doi.org/10.1137/S0097539795290477>
- Fischer, A., Botero, J.F., Till Beck, M., et al., 2013. Virtual network embedding: a survey. *IEEE Commun. Surv. Tutor.*, **15**(4):1888-1906. <http://dx.doi.org/10.1109/SURV.2013.013013.00155>
- Heller, B., Sherwood, R., McKeown, N., 2012. The controller placement problem. *ACM SIGCOMM Comput. Commun. Rev.*, **42**(4):473-478. <http://dx.doi.org/10.1145/2377677.2377767>
- Hu, Y., Wang, W., Gong, X., et al., 2013. Reliability-aware controller placement for software-defined networks. Proc. IFIP/IEEE Int. Symp. on Integrated Network Management, p.672-675.
- Khan, A., Zugenmaier, A., Jurca, D., et al., 2012. Network virtualization: a hypervisor for the Internet? *IEEE Commun. Mag.*, **50**(1):136-143. <http://dx.doi.org/10.1109/MCOM.2012.6122544>
- Koponen, T., Amidon, K., Balland, P., et al., 2014. Network virtualization in multi-tenant datacenters. USENIX Conf. on Networked System Design and Implementation, p.203-216.
- Lantz, B., Heller, B., McKeown, N., 2010. A network in a laptop: rapid prototyping for software-defined networks. Proc. 9th ACM SIGCOMM Workshop on Hot Topics in Networks, p.19:1-19:6. <http://dx.doi.org/10.1145/1868447.1868466>
- Li, X.L., Wang, H.M., Guo, C.G., et al., 2012. Topology awareness algorithm for virtual network mapping. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **13**(3):178-186. <http://dx.doi.org/10.1631/jzus.C1100282>
- Li, X.L., Wang, H.M., Ding, B., et al., 2014. Resource allocation with multi-factor node ranking in data center networks. *Fut. Gener. Comput. Syst.*, **32**:1-12. <http://dx.doi.org/10.1016/j.future.2013.09.028>
- Liu, S.H., Cai, Z.P., Xu, H., et al., 2015. Towards security-aware virtual network embedding. *Comput. Netw.*, **91**:151-163. <http://dx.doi.org/10.1016/j.comnet.2015.08.014>
- McKeown, N., Anderson, T., Balakrishnan, H., et al., 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.*, **38**(2):69-74. <http://dx.doi.org/10.1145/1355734.1355746>
- Mijumbi, R., Serrat, J., Rubio-Loyola, J., et al., 2014. Dynamic resource management in SDN-based virtualized networks. Int. Conf. on Network and Service Management, p.412-417. <http://dx.doi.org/10.1109/CNSM.2014.7014204>
- Salvadori, E., Corin, R.D., Broglio, A., et al., 2011. Generalizing virtual network topologies in OpenFlow-based networks. IEEE Global Telecommunications Conf., p.1-6. <http://dx.doi.org/10.1109/GLOCOM.2011.6134525>
- Schrijver, A., 1998. Theory of Linear and Integer Programming. Wiley, New York, USA.
- Sherwood, R., Gibb, G., Yap, K.K., et al., 2010. Can the production network be the testbed? 9th USENIX Symp. on Operating System Design and Implementation, p.1-6.
- Su, S., Zhang, Z.B., Liu, A.X., et al., 2014. Energy-aware virtual network embedding. *IEEE/ACM Trans. Netw.*, **22**(5):1607-1620. <http://dx.doi.org/10.1109/TNET.2013.2286156>

- Wang, A.J., Iyer, M., Dutta, R., *et al.*, 2013. Network virtualization: technologies, perspectives, and frontiers. *J. Lightw. Technol.*, **31**(4):523-537.  
<http://dx.doi.org/10.1109/JLT.2012.2213796>
- Wang, Z.M., Wu, J.X., Wang, Y., *et al.*, 2014. Survivable virtual network mapping using optimal backup topology in virtualized SDN. *China Commun.*, **11**(2):26-37.  
<http://dx.doi.org/10.1109/CC.2014.6821735>
- Zegura, E.W., Calvert, K.L., Bhattacharjee, S., 1996. How to model an internetwork. 15th Annual Joint Conf. of the IEEE Computer and Communications Societies, p.594-602.  
<http://dx.doi.org/10.1109/INFCOM.1996.493353>
- Zhou, B., Gao, W., Zhao, S., *et al.*, 2014. Virtual network mapping for multi-domain data plane in software-defined networks. Int. Conf. on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems, p.1-5.  
<http://dx.doi.org/10.1109/VITAE.2014.6934439>