Proactive and Hitless vSDN Reconfiguration to Balance Substrate TCAM Utilization: From Algorithm Design to System Prototype

Sicheng Zhao, Deyun Li, Kai Han, and Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—The combination of network virtualization and software-defined networking (SDN) enables an infrastructure provider (InP) to create software-defined virtual networks (vSDNs) over a shared substrate network (SNT), for supporting new network services more timely and cost-effectively. Meanwhile, as both the services and traffic in the Internet are becoming more and more dynamic, how to properly maintain vSDNs in a dynamic network environment exhibits increasing importance but still has not been fully explored. In this work, we conduct a study on how to realize proactive and hitless vSDN reconfiguration to balance the utilization of ternary content-addressable memory (TCAM) in a dynamic SNT. Specifically, we consider both algorithm design and system prototyping. From the algorithmic perspective, we try to solve the problems of "what to reconfigure" and "how to reconfigure". A selection algorithm is designed to proactively choose the virtual switches (vSWs) that should be migrated to other substrate switches (S-SWs) for balancing TCAM utilization, *i.e.*, solving "what to reconfigure". Then, for the problem of "how to reconfigure", *i.e.*, where to re-map the selected vSWs and the virtual links (VLs) connecting to them, we formulate a mixed integer linear programming (MILP) model to solve it exactly, and design two heuristics to improve time efficiency. Next, we move to the system part, implement the proposed algorithms in our protocol-oblivious forwarding (POF) enabled network virtualization hypervisor (NVH) system, and conduct experiments to demonstrate proactive and hitless vSDN reconfiguration. The experimental results indicate that our proposal does make vSDN reconfiguration transparent to the vSDNs' virtual controllers (vCs) and proactive, and when reconfiguring a vSDN with live traffic, it achieves hitless operations without traffic disruption.**

***Index Terms*—Network virtualization, Software-defined networking (SDN), Virtual network migration, Protocol-oblivious forwarding (POF), Hitless reconfiguration.**

I. Introduction

OVER the past decade, network virtualization technologies have attracted intensive research interests [1–4] and been considered as an effective solution to address the ossification of current Internet infrastructure. Network virtualization makes the conventional Internet service providers (ISPs) evolve as infrastructure providers (InPs) and service providers (SPs). Specifically, an InP owns a substrate network (SNT), collects virtual network (VNT) requests from SPs, build logically-isolated VNTs through virtualizing its substrate resources accordingly, and leases the VNTs to SPs to satisfy their requests [5]. Then, the SPs do not need to worry about the efforts of building and maintaining physical networks (*e.g.*, the tasks considered in [6, 7]), and thus can deliver new network services in a short time-to-market and cost-effective manner [8]. Meanwhile, the InP can effectively improve its resource usage and get rid of the hassles of running various services by itself. Hence, a "win-win" situation can be achieved.

In addition to network virtualization, software-defined networking (SDN) [9, 10] is another important innovation that enables SPs to provision new services better and more timely. SDN separates the control and data planes of a network and implements centralized network control and management (NC&M) to enhance programmability and application-awareness [11–13]. As a famous SDN implementation, OpenFlow [14] defines the communications between the control and data planes, and by abstracting the behaviors of data plane as protocol-dependent flow-entries, it facilitates programmable packet processing based on the "match-and-action" principle. Recently, the technical advances on programming protocol independent processors (P4) [15] and protocol-oblivious forwarding (POF) [16] decouple network protocols from packet processing and realize protocol-independent data plane to fully unleash its programmability for future-proof SDN. Therefore, it would be desired to provision software-defined VNTs (*i.e.*, virtual software-defined networks (vSDNs)) to SPs [17–19].

The rational combination of network virtualization and SDN (*i.e.*, efficiently creating vSDNs over an InP's SNT) would require both a virtual network embedding (VNE) algorithm [20] and a network virtualization hypervisor (NVH) system [21]. To build a vSDN, the VNE algorithm allocates the flow-entry space on substrate switches (S-SWs) to virtual switches (vSWs) (*i.e.*, the node mapping) and assigns the bandwidth capacity on substrate links (SLs) to virtual links (VLs) (*i.e.*, the link mapping). Note that, the flow-entry space concerned in the node mapping is essentially the ternary content-addressable memory (TCAM), which is usually limited in each S-SW due to its expense and power consumption [22], and the vSDN consumes TCAM on the S-SWs along the paths that carry its VLs [19, 23]. The VNE algorithm's result is implemented by the NVH, which also bridges the communication between the used S-SWs and the virtual controller (vC) of the vSDN and facilitates two-way translation of control messages [17].

Although both VNE algorithms and NVH systems have already been studied intensively in literature [17, 24], the problem of how to realize proactive vSDN reconfiguration with a NVH system for addressing the dynamics in the SNT has not been fully explored yet. Note that, both the principle

S. Zhao, D. Li, K. Han, and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received on July 31, 2018.

of network virtualization (*i.e.*, creating and leasing vSDNs on demand) and the SPs' application traffic can bring in dynamics into the InP's SNT and make the optimal VNE results implemented by the NVH system sub-optimal. More importantly, as the TCAM on each S-SW is limited and vSWs' requirements on flow-entry space are usually time-varying and unpredictable, it would not be reasonable or practical to allocate TCAM resources to vSDNs in the fixed and isolated way. Specifically, as a type of scarce resources in S-SWs, TCAM is generally 400 times more expensive [25] and 100 times more power-consuming [26] than the RAM-based storage. A more reasonable approach is to let the vSDNs share the TCAM on each S-SW for storing the flow-entries on their vSWs in a statistical multiplexing manner [27]. Note that, different from the TCAM allocation, dedicated bandwidth, which is a different type of substrate resources on the SLs, is assigned to each vSDN. Hence, the vSDN's quality-of-service (QoS) is ensured as long as there is enough TCAM on the S-SWs that carry its vSWs. However, the dilemma is that the vSWs mapped onto a same S-SW have to compete for flow-entry space when TCAM becomes insufficient. This would let the vSWs flood *PacketIn* messages to their vCs, and seriously impact the operations of both the vSDNs and NVH system.

Apparently, in order to properly maintain vSDNs in a dynamic network environment, an InP needs to consider vSDN reconfiguration to balance the TCAM utilization in its SNT from time to time [27]. Although this idea is straightforward, realizing it effectively is still challenging from both algorithmic and systemic aspects. First of all, we need a time-efficient algorithm that can intelligently solve the problems of when and how to reconfigure vSDNs and can be integrated in an NVH easily. Note that, the vSDN reconfiguration should be triggered proactively, *i.e.*, before a TCAM depletion actually happens. Next, the system design and implementation should properly address the following two challenges. Firstly, the vSDN reconfiguration should be solely handled by the NVH and made transparent to the vSDNs' vCs, for simplifying SPs' NC&M tasks and minimizing reconfiguration latency. Secondly, the vSDN reconfiguration should be "hitless", which means that each vSDN should be reconfigured with live traffic flowing and the traffic should not be disrupted during and after reconfiguration. Otherwise, the vSDN reconfiguration may cause noticeable packet loss and degrade the vSDNs' services to its flows. However, to the best of our knowledge, these algorithmic and systemic challenges have not been considered jointly in previous studies on vSDN reconfiguration.

In this work, we conduct a relatively comprehensive study on how to realize proactive and hitless vSDN reconfiguration in dynamic network environment. We first design an algorithm to proactively select the vSWs that should be migrated to other S-SWs for balancing the TCAM utilization in an SNT. Then, we try to tackle the problem of how to reconfigure, *i.e.*, where to re-map the selected vSWs and the VLs connecting to them. A mixed integer linear programming (MILP) model is formulated to solve the problem exactly, and we also design two heuristics to improve time efficiency. Next, we modify the design of our POF-based NVH (PVX) [21] and implement the proposed algorithms in it to facilitate proactive and hitless vSDN reconfiguration. Finally, the experimental results demonstrate that our system prototype does make the vSDN reconfiguration transparent to the vSDNs' vCs and proactive, and when reconfiguring a vSDN with live application traffic, it achieves hitless operations without traffic disruption.

The rest of this paper is organized as follows. We survey the related work briefly in Section II. The problem description of vSDN reconfiguration in dynamic network environment is presented in Section III. Section IV discusses the algorithm design, while the proposed algorithms' performance is evaluated in Section V with extensive numerical simulations. In Section VI, we describe our system design and implementation and perform experiments to demonstrate the effectiveness of our system prototype. Finally, Section VII summarizes the paper.

II. Related Work

Previously, the problem of VNE has already been studied in [2–4, 28, 29] for various networks, and a comprehensive survey on the existing VNE algorithms can be found in [24]. However, these studies did not discuss how to implement the designed VNE algorithms in a practical NVH system. NVH realizes the creation of vSDNs over a shared SNT, and many investigations have been dedicated to developing an effective NVH system [17]. In NVH's early stage, FlowVisor [30] was developed to build OpenFlow-enabled vSDNs with the constraint that the vSDNs should take the same topology as that of the SNT. Then, people designed OVX [31] as a new version of NVH, which extended the functionalities of FlowVisor and removed the restriction on vSDN topology. More recently, we programmed our PVX [18, 21] based on OVX to realize protocol-independent vSDNs, by leveraging POF [10]. In our latest study [8], we design and implement a physically distributed but logically centralized NVH system based on ONOS [32] to realize highly-available and scalable vSDN slicing. In addition to the studies on NVH itself, people have also utilized NVH systems to realize various network functions [33–36]. Nevertheless, none of these NVH systems supports proactive and hitless vSDN reconfiguration.

Network reconfiguration schemes have been considered to resolve various issues in dynamic network environments [37, 38]. However, vSDN reconfiguration is different from and more complicated than the reconfiguration scenarios considered before. This is because remapping a vSW to a new S-SW changes not only the node mapping of the vSW but also the link mapping of all the VLs that connect to it, while a VL consumes TCAM on both the S-SWs that two of its vSWs are embedded on and the S-SWs that are intermediate nodes on its substrate path. The study in [39] considered how to reconfigure VNTs in a dynamic network environment, but it was purely algorithmic and did not address vSDNs or the TCAM utilization of them in the network model. Although a system for realizing SDN migration has been demonstrated in [40], network virtualization was not involved. Based on a different network virtualization scenario from the one considered in this work, the authors of [41, 42] have implemented two systems to address vSDN reconfiguration. Specifically, they virtualized the substrate resources used to

build vSWs and created an independent software-based switch to instantiate each vSW, *i.e.*, vSWs of different vSDNs would never share the same S-SWs. Pisa *et al.* [43] tried to realize vSDN reconfiguration with the involvement of vSDNs' vCs. As a vC should only handle the NC&M tasks of its vSDN and should not know the resource utilization in the SNT, it can hardly determine when and how to reconfigure its vSDN.

In [27], we presented our system design to achieve hitless vSDN reconfiguration for avoiding TCAM depletion and showed some preliminary results. Nevertheless, the algorithm that can solve the problems of what and how to reconfigure vSDNs is still absent, and the experimental demonstrations were based on pre-coded scenarios. In this work, we extend the NVH system developed in [27] by designing the algorithm for determining what and how to reconfigure vSDNs and implementing it for prototyping, and thus vSDN reconfiguration is addressed in a more comprehensive manner.

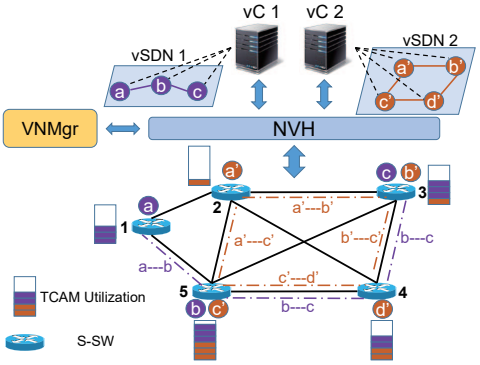

Fig. 1. Network architecture for realizing vSDN creation and reconfiguration.

III. Problem Description

In this section, we define the network model and describe the problem of vSDN reconfiguration. Since abbreviations are frequently used in this paper, we list all the major but not well-known abbreviations in Table I for the readers' convenience.

TABLE I
Major Abbreviations

Abbrev.	Full Name	Abbrev.	Full Name
POF	Protocol-oblivious forwarding	SP	Service provider
TCAM	Ternary content-addressable memory	SNT	Substrate network
VNE	Virtual network embedding	S-SW	Substrate switch
vSDN	Virtual software-defined network	SL	Substrate link
NVH	Network virtualization hypervisor	VNT	Virtual network
PVX	POF-based NVH	VL	Virtual link
VNMgr	Virtual network manager	vC	Virtual controller
NF-R	Node-first remapping algorithm	vSW	Virtual switch
LF-R	Link-first remapping algorithm		

A. Network Model

Fig. 1 shows the network architecture considered in this work for vSDN creation and reconfiguration. The SNT is built with S-SWs, each of which contains a fixed volume of TCAM. With network virtualization, the NVH system can create multiple vSDNs over the SNT. Here, the NVH system consists of an NVH and a virtual network manager (VNMgr). To create a vSDN, the VNMgr first calculates its VNE scheme based on the SNT's status and then instructs the NVH to implement the scheme. Next, the vC of the vSDN gets connected to the S-SWs that carry its vSWs through the NVH. During network operation, the vC can install, update and remove flow-entries on the vSWs for managing the traffic in its vSDN, and the related control messages get translated by the NVH before reaching the S-SWs, according to the mapping relation. Meanwhile, the VNMgr monitors the SNT's status (*i.e.*, substrate TCAM utilization) proactively, and when necessary, it would invoke a vSDN reconfiguration to balance TCAM utilization[1]. Note that, in our design, the NVH system handles the vSDN reconfiguration by itself and makes the operation completely transparent to the vCs.

The topology of the SNT can be modeled as an undirected graph $G_s(V_s, E_s)$, where V_s and E_s represent the sets of S-SWs and SLs, respectively. There are two types of substrate resources in the SNT, which are the TCAM resources on S-SWs and bandwidth resources on SLs. Hence, we denote the TCAM capacity of S-SW $v_s \in V_s$ as T_{v_s} in terms of flow-entries, while the bandwidth capacity of SL $(u_s, v_s) \in E_s$ is $B_{(u_s,v_s)}$. Since a vSDN reconfiguration is triggered when the SNT is operational, we denote the used TCAM and bandwidth as t_{v_s} and $b_{(u_s,v_s)}$ for S-SW v_s and SL (u_s, v_s), respectively.

The topology of a vSDN can also be modeled as an undirected graph, *i.e.*, $G_r(V_r, E_r)$, where V_r and E_r are the sets of vSWs and VLs, respectively. The TCAM usage of vSW $v_r \in V_r$ is denoted as t_{v_r}, while the bandwidth usage of VL $(u_r, v_r) \in E_r$ is $b_{(u_r,v_r)}$. Note that, in addition to the S-SWs on which its vSWs are embedded, the vSDN also consumes TCAM resources on the S-SWs that are the intermediate nodes on the substrate paths carrying its VLs. For instance, in Fig. 1, *VL* b-c gets embedded on *Substrate Path* 5-4-3, where *S-SWs* 5 and 3 are two end nodes and *S-SW* 4 is an intermediate node. As the traffic going through *VL* b-c needs to be forwarded correctly on *S-SW* 4, the vC has to install corresponding flow-entries in it. Meanwhile, since all the packets that belong to a vSDN get forwarded in the same way on such an intermediate node, one of its VLs only consumes two flow-entries in each intermediate node, *i.e.*, for two-way communication in the VL.

B. vSDN Reconfiguration for Balancing TCAM Utilization

When the SNT is operational with multiple vSDNs, each vSDN can have dynamic traffic, which will make the TCAM utilization in each S-SW change over time. This will in turn cause unbalanced TCAM utilization on the S-SWs, *e.g.*, as shown in Fig. 1. To avoid the unbalanced TCAM utilization getting worse and eventually leading to TCAM depletion, we introduce periodical SNT maintenance based on vSDN reconfiguration. Specifically, the idea is to proactively monitor the SNT in a periodical manner and invoke vSDN reconfiguration when necessary to balance the TCAM utilization in it. Fig. 2 provides an intuitive example on the vSDN reconfiguration. In

[1]Here, the vSDN reconfiguration would not try to balance the bandwidth utilization in the SNT. This is because balancing bandwidth usage is related to the operation of the vSDNs, and thus it should be handled and could be more effectively achieved by the vCs performing traffic engineering.

Fig. 1, before the vSDN reconfiguration, the TCAM utilization on *S-SW* 5 is significantly higher than that on other S-SWs, which is because *S-SW* 5 carries two busy vSWs, *i.e.*, *vSWs* b and c' in *vSDNs* 1 and 2, respectively. Then, we decide to migrate *vSW* b from *S-SW* 5 to *S-SW* 2 to balance the substrate TCAM utilization. Hence, for *vSDN* 1, both the node mapping and the link mapping that are related to *vSW* b need to be reconfigured (as shown in Fig. 2), and the TCAM utilization gets more balanced after the vSDN reconfiguration.

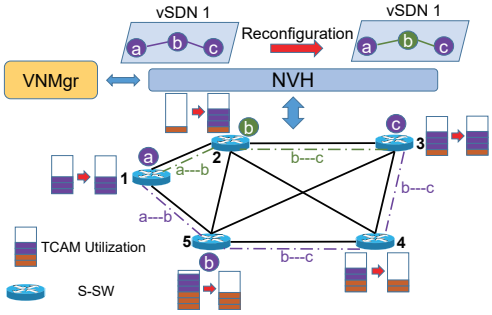

Fig. 2. Example on vSDN reconfiguration to balance TCAM utilization.

As explained above, a vSDN reconfiguration should first select suitable vSWs to reconfigure and then find appropriate new S-SWs to re-map them on. In other words, its algorithms solve the problems of what and how to reconfigure vSDNs.

IV. Algorithm Design

In this section, we design the algorithms for vSDN reconfiguration. Specifically, we first describe the overall procedure of vSDN reconfiguration, and then discuss how to select vSWs to reconfigure and how to re-map the selected vSWs in details.

A. Overall Procedure

Algorithm 1 shows the overall procedure of periodical SNT maintenance based on vSDN reconfiguration. When an SNT maintenance starts, we choose the vSWs that should be reconfigured to balance the TCAM utilization in the SNT with a selection algorithm, and store the selected vSWs in set V_R^s (*Lines* 2-3). Then, *Line* 4 uses a remapping algorithm to get the new node mapping schemes of the vSWs in V_R^s, and the vSWs are re-mapped in *Line* 5. Note that, the vSDN reconfiguration not only re-maps the vSWs in V_R^s onto their new S-SWs but also migrates all the VLs that connect to them. Finally, the algorithm waits for the next maintenance time (*Line* 6). Here, the system decides when to perform maintenance (*i.e.*, vSDN reconfiguration) based on the TCAM utilization in the SNT. Specifically, we define two thresholds, *i.e.*, one sets an upper-limit on the TCAM utilization in each S-SW and the other is the maximum degree of TCAM unbalance in the SNT. Then, if the system finds that either of these two thresholds has been exceeded, it will trigger a vSDN reconfiguration. Note that, the vSDN reconfiguration can only address the situations in which the TCAM utilization in the SNT is highly unbalanced while the SNT's overall TCAM capacity is sufficient for the vSDNs. Otherwise, if the whole SNT is heavily loaded and the TCAM utilization on most of the S-SWs is about to exceed the threshold, the vSDN reconfiguration would not be helpful. Hence, for the heavily loaded situations, the NVH system should try other options, *e.g.*, blocking vSDN requests. The detailed procedures of the selection and remapping algorithms will be discussed in the following subsections.

Algorithm 1: Overall Procedure of SNT Maintenance

```
1 while the SNT is operational do
2     choose vSWs to reconfigure with a selection
      algorithm;
3     store the selected vSWs in set V_R^s;
4     determine new node mapping schemes of vSWs
      in V_R^s with a remapping algorithm;
5     re-map vSWs in V_R^s accordingly and migrate all
      the affected VLs;
6     wait for the next maintenance time;
7 end
```

B. Selection Algorithm

Before describing the procedure of our selection algorithm, we define the average TCAM utilization in the SNT as

$$\bar{t} = \frac{1}{|V_s|} \cdot \sum_{v_s \in V_s} t_{v_s}, \tag{1}$$

where $|V_s|$ means the number of S-SWs in the SNT and t_{v_s} represents the TCAM utilization on S-SW $v_s \in V_s$ before vSDN reconfiguration. Here, for simplicity, we assume that all the S-SWs have the same TCAM capacity. Hence, our algorithms will balance substrate TCAM utilization based on actual values. However, this would not prevent our algorithms from being applied to situations in which the S-SWs' TCAM capacities are different, since they can be generalized by normalizing the TCAM utilization as a relative ratio. Then, based on $\bar{t}$, we design a two-step selection algorithm to find the most "critical" vSWs to reconfigure for balancing the substrate TCAM utilization, *i.e.*, *Algorithm* 2.

In *Algorithm* 2, *Lines* 1-2 are for the initialization. The for-loop that covers *Lines* 3-17 is the first step that selects all the vSWs, which can be re-mapped for balancing TCAM utilization. Specifically, as shown in *Line* 4, we only check the S-SWs whose TCAM utilizations are larger than the average value $\bar{t}$. We use tt_{v_s} to store the TCAM utilization on *S-SW* v_s after the vSDN reconfiguration has been done, thus we initialize it as $tt_{v_s} = t_{v_s}$ (*Line* 5). In *Line* 5, we also initialize a temporary variable as $t = t_{v_s}$, and empty two vSW sets $V_r^{v_s}$ and V_r^t. *Line* 6 uses V_r^t to store the vSWs that are embedded on *S-SW* v_s, while $V_r^{v_s}$ is used later to store the vSWs that could be mitigated away from v_s. In the while-loop covering *Lines* 7-14, each iteration tries to select a vSW to migrate away from v_s such that the resulting TCAM utilization on v_s (*i.e.*, t) would become closer to the average value $\bar{t}$, until we have $t \leq \bar{t}$. Here, the counter n is used to record the number of vSWs that get selected in the first step (*Line* 13).

In *Lines* 18-24, we use another for-loop to accomplish the second step of vSW selection. Here, to control the complexity

of vSDN reconfiguration, we introduce a selection ratio $\gamma \in (0, 1]$ to determine how many vSWs that will be eventually chosen for reconfiguration. Specifically, we will choose the $\lceil \gamma \cdot n \rceil$ most critical vSWs to include in set V_R^s. *Line* 19 finds the S-SW v_s^s whose TCAM utilization $tt_{v_s^s}$ is currently the maximum. Then, in *Lines* 20-23, we select the first vSW in $V_r^{v_s^s}$ to insert in V_R^s and update $V_r^{v_s^s}$ and $tt_{v_s^s}$ accordingly. *Lines* 19-23 are repeated until $\lceil \gamma \cdot n \rceil$ vSWs have been selected. Finally, in *Line* 25, *Algorithm* 2 outputs the selected vSWs for reconfiguration in V_R^s. The time complexity of *Algorithm* 2 is $O(|V_s| \cdot |V_R|^2)$, where V_R is the set of vSWs in all the vSDNs.

Algorithm 2: Choose vSWs to Reconfigure

1 $n = 0$, $V_R^s = \emptyset$;
2 calculate average TCAM utilization $\bar{t}$ with Eq. (1);
3 **for** *each S-SW* $v_s \in V_s$ **do**
4 **if** $t_{v_s} > \bar{t}$ **then**
5 $tt_{v_s} = t_{v_s}$, $t = t_{v_s}$, $V_r^{v_s} = \emptyset$, $V_r^t = \emptyset$;
6 store the vSWs embedded on v_s in set V_r^t;
7 **while** $t > \bar{t}$ **do**
8 $v_r^s = \underset{v_r \in V_r^t}{\operatorname{argmin}} [|(t - t_{v_r}) - \bar{t}|]$;
9 $t = t - t_{v_r^s}$;
10 **if** $t \geq \bar{t}$ *or* $|(t + t_{v_s}) - \bar{t}| > |t - \bar{t}|$ **then**
11 insert v_r^s in set $V_r^{v_s}$;
12 remove v_r^s from set V_r^t;
13 $n = n + 1$;
14 **end**
15 **end**
16 **end**
17 **end**
18 **for** $i = 1$ *to* $\lceil \gamma \cdot n \rceil$ **do**
19 $v_s^s = \underset{v_s \in V_s}{\operatorname{argmax}}(tt_{v_s})$;
20 select the first vSW v_r in $V_r^{v_s^s}$;
21 insert v_r in set V_R^s;
22 remove v_r from set $V_r^{v_s^s}$;
23 $tt_{v_s^s} = tt_{v_s^s} - t_{v_r}$;
24 **end**
25 **return**(V_R^s);

C. Remapping Algorithms

With the chosen vSWs from the selection algorithm, we need a remapping algorithm to determine their new node mapping schemes for balancing TCAM utilization. In the following, we first formulate a mixed linear programming (MILP) model to solve this problem exactly, and then design two heuristics for high time efficiency.

1) MILP Model: The MILP model to find the remapping schemes of vSWs in V_R^s is formulated as follows.

Notations:

- V_s: set of S-SWs in the SNT.
- E_s: set of SLs in the SNT.
- $G_s(V_s, E_s)$: topology of the SNT.
- T_{v_s}: TCAM capacity of S-SW $v_s \in V_s$.
- $B_{(u_s,v_s)}$: bandwidth capacity of SL $(u_s, v_s) \in E_s$.
- t_{v_s}: TCAM utilization on S-SW $v_s \in V_s$ before vSDN reconfiguration.
- $b_{(u_s,v_s)}$: bandwidth utilization on SL $(u_s, v_s) \in E_s$ before vSDN reconfiguration.
- V_R^s: set of the vSWs that are chosen for reconfiguration.
- R: set of vSDNs, each of which contains vSWs in V_R^s.
- V_r: set of vSWs in a vSDN $r \in R$.
- E_r: set of VLs in a vSDN $r \in R$.
- $G_r(V_r, E_r)$: topology of a vSDN $r \in R$.
- $b_{(u_r,v_r)}$: bandwidth usage of VL $(u_r, v_r) \in E_r$.
- t_{v_r}: TCAM usage of vSW $v_r \in V_r$.
- $\bar{t}$: average TCAM utilization in the SNT before vSDN reconfiguration (calculated with Eq. (1)).
- $\widetilde{\delta}_{v_s}^{v_r}$: boolean that equals 1 if vSW v_r is embedded on S-SW v_s before vSDN reconfiguration, and 0 otherwise.
- $\widetilde{\rho}_{(u_s,v_s)}^{(u_r,v_r)}$: boolean that equals 1 if VL (u_r, v_r) is embedded on SL (u_s, v_s) before reconfiguration, and 0 otherwise.
- $\widetilde{\omega}_{v_s}^{(u_r,v_r)}$: boolean that equals 1 if S-SW v_s is an intermediate node on the substrate path that carries VL (u_r, v_r) before vSDN reconfiguration, and 0 otherwise.
- m_{v_r}: boolean that equals 1 if vSW v_r is in V_R^s for reconfiguration, and 0 otherwise.
- $m_{(u_r,v_r)}$: boolean that equals 1 if VL (u_r, v_r) needs to be reconfigured, and 0 otherwise.

Variables:

- $\delta_{v_s}^{v_r}$: boolean variable that equals 1 if vSW v_r gets embedded on S-SW v_s after reconfiguration, and 0 otherwise.
- $\rho_{(u_s,v_s)}^{(u_r,v_r)}$: boolean variable that equals 1 if VL (u_r, v_r) gets embedded on SL (u_s, v_s) after reconfiguration, and 0 otherwise.
- $\omega_{v_s}^{(u_r,v_r)}$: boolean variable that equals 1 if S-SW v_s is an intermediate node on the substrate path that carries VL (u_r, v_r) after reconfiguration, and 0 otherwise.
- c_{v_s}: TCAM utilization of vSWs on S-SW v_s after reconfiguration.
- $c_{\max}$: the maximum TCAM utilization of vSWs on an S-SW after reconfiguration.
- $c_{\min}$: the minimum TCAM utilization of vSWs on an S-SW after reconfiguration.

Objective:

The objective of vSDN reconfiguration is to balance TCAM utilization in the SNT. Hence, we define a metric to measure the balance degree of TCAM utilization in the SNT.

$$\tilde{c} = c_{\max} - c_{\min}. \tag{2}$$

Meanwhile, we hope to point out that vSDN reconfiguration may result in more TCAM utilization in the SNT. This is because in addition to the S-SWs on which its end vSWs are embedded, a VL also consumes TCAM resources on the S-SWs that are the intermediate nodes on its substrate path. Therefore, if the vSDN reconfiguration re-maps a VL to a substrate path with more hop-count, its TCAM consumption will increase (*i.e.*, two more flow-entries for each additional intermediate S-SW). Note that, for each VL, the number of flow-entries used on such an intermediate S-SW is independent of the number of flows going through it. This is because the

intermediate S-SW should forward all the flows on the VL to the same output port regardless of their source and destination addresses. In our system, this is achieved by installing two flow-entries in each intermediate S-SW, which match to the *Tenant ID* and *Link ID* fields that we insert in packet headers for realizing network virtualization [27]. The remapping algorithm should also try to avoid increasing substrate TCAM utilization significantly, while the difference in total TCAM utilization before and after vSDN reconfiguration is

$$\tilde{t} = \sum_{r \in R} \sum_{v_r \in V_r} \left[\omega_{v_s}^{(u_r, v_r)} - \widetilde{\omega}_{v_s}^{(u_r, v_r)} \right]. \tag{3}$$

Finally, we define the optimization objective as

$$\text{Minimize} \quad (\alpha \cdot \tilde{c} + \beta \cdot \tilde{t}), \tag{4}$$

where α and β are the weights to balance the importance of the two terms. We set $\alpha \gg \beta$ to ensure that the primary objective is to minimize $\tilde{c}$ for balancing TCAM utilization in the SNT.

Constraints:

- *Node Mapping Constraints:*

$$\sum_{v_s \in V_s} \delta_{v_s}^{v_r} = 1, \quad \forall v_r \in V_r, \ \forall r \in R. \tag{5}$$

Eq. (5) ensures that each vSW still gets mapped onto a single S-SW after reconfiguration.

$$\sum_{v_r \in V_r} \delta_{v_s}^{v_r} \leq 1, \quad \forall v_s \in V_s, \ \forall r \in R. \tag{6}$$

Eq. (6) ensures that different vSWs in a vSDN still get mapped onto different S-SWs after reconfiguration.

$$\begin{aligned} &\widetilde{\delta}_{v_s}^{v_r} \cdot (1 - m_{v_r}) - \delta_{v_s}^{v_r} \cdot (1 - m_{v_r}) = 0, \\ &\forall v_r \in V_r, \ \forall v_s \in V_s, \ r \in R. \end{aligned} \tag{7}$$

Eq. (7) ensures that in a reconfigured vSDN, each vSW that does not need to be reconfigured still gets mapped onto the same S-SW.

- *Link Mapping Constraints:*

$$\begin{aligned} &\sum_{\{v_s : (v_s, u_s) \in E_s\}} \rho_{(u_s, v_s)}^{(u_r, v_r)} - \sum_{\{v_s : (v_s, u_s) \in E_s\}} \rho_{(v_s, u_s)}^{(u_r, v_r)} \\ &= \delta_{u_s}^{u_r} - \delta_{u_s}^{v_r}, \quad \forall u_s \in V_s, \ \forall (u_r, v_r) \in E_r, \ \forall r \in R. \end{aligned} \tag{8}$$

Eq. (8) ensures that in a reconfigured vSDN, each VL still gets embedded on a single substrate path.

$$\begin{aligned} &\rho_{(u_s, v_s)}^{(u_r, v_r)} = \rho_{(v_s, u_s)}^{(u_r, v_r)}, \quad \forall (u_s, v_s) \in E_s, \\ &\forall (u_r, v_r) \in E_r, \ \forall r \in R. \end{aligned} \tag{9}$$

Eq. (9) ensures that the flows in both directions on the same VL traverse only one substrate path.

$$\begin{aligned} &\left[\widetilde{\rho}_{(u_s, v_s)}^{(u_r, v_r)} - \rho_{(u_s, v_s)}^{(u_r, v_r)} \right] \cdot \left[1 - m_{(u_r, v_r)} \right] = 0, \\ &\forall (u_r, v_r) \in E_r, \ \forall (u_s, v_s) \in E_s, \ r \in R. \end{aligned} \tag{10}$$

Eq. (10) ensures that in a reconfigured vSDN, each VL that does not need to be reconfigured still gets embedded on the same SL.

- *Intermediate Node Constraints:*

$$\begin{aligned} &\sum_{\{v_s : (v_s, u_s) \in E_s\}} \rho_{(u_s, v_s)}^{(u_r, v_r)} - \omega_{u_s}^{(u_r, v_r)} = \delta_{u_s}^{u_r}, \ \forall u_s \in V_s, \\ &\forall (u_r, v_r) \in E_r, \ \forall r \in R. \end{aligned} \tag{11}$$

Eq. (11) ensures that the intermediate S-SWs on the substrate path that carries VL (u_r, v_r) get marked correctly.

- *Resource Constraints:*

$$\begin{aligned} &t_{v_s} + \sum_{r \in R} \sum_{v_r \in V_r} t_{v_r} \cdot (\delta_{v_s}^{v_r} - \widetilde{\delta}_{v_s}^{v_r}) \\ &+ \sum_{r \in R} \sum_{(u_r, v_r) \in E_r} \left[\omega_{v_s}^{(u_r, v_r)} - \widetilde{\omega}_{v_s}^{(u_r, v_r)} \right] \leq T_{v_s}, \ \forall v_s \in V_s. \end{aligned} \tag{12}$$

Eq. (12) ensures that the TCAM utilization on each S-SW does not exceed its TCAM capacity.

$$\begin{aligned} &b_{(u_s, v_s)} + \sum_{r \in R} \sum_{(u_r, v_r) \in E_r} \left[\rho_{(u_s, v_s)}^{(u_r, v_r)} - \widetilde{\rho}_{(u_s, v_s)}^{(u_r, v_r)} \right] \cdot b_{(u_r, v_r)} \leq B_{v_s}, \\ &\forall v_s \in V_s. \end{aligned} \tag{13}$$

Eq. (13) ensures that the bandwidth utilization on each SL does not exceed its bandwidth capacity.

$$c_{v_s} = \sum_{r \in R} \sum_{v_r \in V_r} t_{v_r} \cdot \delta_{v_s}^{v_r}, \ \forall v_s \in V_s. \tag{14}$$

Eq. (14) calculates the TCAM utilization of vSWs on each S-SW in the SNT after reconfiguration.

- *Other Constraints:*

$$\begin{aligned} c_{\max} &\geq c_{v_s}, \ \forall v_s \in V_s, \\ c_{\min} &\leq c_{v_s}, \ \forall v_s \in V_s. \end{aligned} \tag{15}$$

Eq. (15) ensures that $c_{\max}$ and $c_{\min}$ are got correctly.

2) Node-First Remapping Algorithm (NF-R): As the MILP model is not scalable and can become intractable for a relatively large SNT, we design two heuristics to improve the time efficiency. Here, we first consider a remapping algorithm that determines the remapping schemes of vSWs and VLs in two steps, namely, node-first remapping algorithm (NF-R). *Algorithm* 3 shows the detailed procedure of NF-R.

Here, *Lines* 1-5 are for the initialization. In *Line* 1, we introduce two temporary variables P and $\tilde{V}_s$, where P will be used to store certain vSWs for reconfiguration, and $\tilde{V}_s$ is initialized as $\tilde{V}_s = V_s$ to store the S-SWs in the SNT. *Line* 2 determines the vSDNs that need to be reconfigured based on V_R^s and stores them in R. The for-loop that covers *Lines* 3-5 calculates the TCAM utilization $\tilde{t}_{v_s}$ on each S-SW v_s when the vSWs in V_R^s have been migrated away from it. Then, the while-loop covering *Lines* 6-22 determines the remapping schemes of vSWs in V_R^s one by one. In *Line* 7, we select an S-SW v_s from $\tilde{V}_s$ such that its TCAM utilization $\tilde{t}_{v_s}$ is the smallest. Then, for each vSDN that needs to be reconfigured, the for-loop that covers *Lines* 8-13 selects a vSW that provides the minimum value for $|\tilde{t}_{v_s} + t_{v_r} - \bar{t}|$ and inserts it in set P. Note that, as shown in *Line* 9, we will skip to process a vSDN if all of its vSWs for reconfiguration have been processed or one of its vSWs has already been mapped on *S-SW* v_s.

Next, in *Line* 14, we test whether P is empty. If no, we select the vSW in P, which provides the minimum value for $|\tilde{t}_{v_s} + t_{v_r} - \bar{t}|$, mark its new S-SW as v_s (*Lines* 15-16), update the related variables (*Lines* 17-18). Otherwise, if P is empty, we remove v_s from $\tilde{V}_s$ since it cannot carry any vSW in V_R^s. Finally, we determine the new link mapping schemes for all the VLs that need to be reconfigured based on shortest path routing (*Lines* 23-24) The time complexity of *Algorithm* 3 is

$O(|V_R^s|\cdot(|V_s|+|V_R^s|^2)+|E_R|\cdot(|E_s|+|V_s|\cdot\log(|V_s|)))$, where E_R is the set of VLs in all the vSDNs.

Algorithm 3: Node-First Remapping Algorithm

1 $P=\emptyset$, $\tilde{V}_s=V_s$;
2 determine the vSDNs that need to be reconfigured based on V_R^s and store them in R;
3 **for** *each* $v_s\in\tilde{V}_s$ **do**
4 calculate TCAM utilization $\tilde{t}_{v_s}$ on S-SW v_s when the vSWs in V_R^s have been migrated away;
5 **end**
6 **while** $V_R^s\neq\emptyset$ **do**
7 $v_s=\underset{v_s\in\tilde{V}_s}{\operatorname{argmin}}(\tilde{t}_{v_s})$;
8 **for** *each vSDN* $r\in R$ **do**
9 **if** *($V_R^s\cap V_r\neq\emptyset$) and (vSDN r currently does not have a vSW mapped onto S-SW v_s)* **then**
10 $v_r=\underset{v_r\in(V_R^s\cap V_r)}{\operatorname{argmin}}(|\tilde{t}_{v_s}+t_{v_r}-\bar{t}|)$;
11 insert v_r in set P;
12 **end**
13 **end**
14 **if** $P\neq\emptyset$ **then**
15 $v_r=\underset{v_r\in P}{\operatorname{argmin}}(|\tilde{t}_{v_s}+t_{v_r}-\bar{t}|)$;
16 decide to re-map vSW v_r onto S-SW v_s;
17 $\tilde{t}_{v_s}=\tilde{t}_{v_s}+t_{v_r}$, $P=\emptyset$;
18 remove v_r from V_R^s;
19 **else**
20 remove v_s from $\tilde{V}_s$;
21 **end**
22 **end**
23 **for** *each VL that needs to be reconfigured* **do**
24 calculate its link mapping scheme as the shortest substrate path with sufficient bandwidth/TCAM resources;
25 **end**

3) Link-First Remapping Algorithm (LF-R): Note that, the link mapping schemes can also affect the performance of vSDN reconfiguration, since each additional intermediate S-SW will cause more TCAM utilization in the SNT. Therefore, we design another remapping algorithm that gives priority to the link mapping, *i.e.*, link-first remapping algorithm (LF-R).

To describe the procedure of LF-R in *Algorithm* 4, we use the example in Fig. 3 to explain its principle and several concepts related to it. *Lines* 1-7 are for the initialization. The original state of the vSDN before reconfiguration is shown in Fig. 3(a), where the red nodes (*i.e.*, b, c and d) are for the vSWs that need to be reconfigured. Hence, we define a reconfiguration flag m_{v_r} for each vSW $v_r\in V_r$. Specifically, we have $m_{v_r}=0$ if the reconfiguration scheme of v_r has been determined, and 1 otherwise. Meanwhile, if a vSW v_r does not need to be reconfigured, its m_{v_r} is set as 0 too. Therefore, in Fig. 3(a), we set the reconfiguration flags of *vSWs* b, c and d as 1, and the remaining vSWs have their reconfiguration flags as 0 (*Lines* 5-7). Then, LF-R tries to re-map the selected vSWs in each vSDN sequentially with the for-loop covering *Lines* 8-35. The while-loop that covers *Lines* 9-34 handles the vSW remapping for a vSDN, while for the example in Fig. 3, it determines the remapping schemes of $\{b,c,d\}$ in sequence.

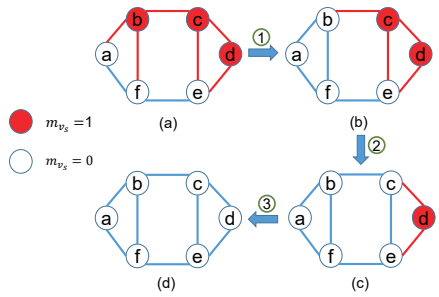

Fig. 3. Procedure of calculating remapping schemes in LF-R.

Here, for each vSDN, the order of vSW remapping is determined by how many fixed adjacent vSWs that a vSW has (*Line* 11). Note that, to remap a vSW onto a new S-SW, we also need to remap all the VLs that connect to it accordingly. Therefore, if a vSW has the maximum number of fixed adjacent vSWs (*i.e.*, with reconfiguration flags as 0), determining its remapping scheme first would help to finalize the maximum number of VL remapping schemes. Hence, in Fig. 3(b), the remapping scheme of *vSW* b is determined first since it has two fixed adjacent vSWs, *i.e.*, a and f. After selecting a vSW v_r from $V_R^s\cap V_r$ in *Line* 11, we calculate h_{v_r} as the hop-count of the determined substrate paths connecting to v_r in *Line* 12. Here, the determined substrate paths connecting to v_r refer to those that are originally used by the VLs from v_r to its fixed adjacent vSWs before the remapping. For example, in Fig. 3(a), the determined substrate paths connecting to *vSW* b are those that carry *VLs* a-b and b-f before the remapping. We store the summation of the hop-counts of these substrate paths in h_{v_r}, and will use it as a metric to avoid using over-length substrate paths in the subsequent link remapping.

Next, the for-loop covering *Lines* 13-22 tries to use an available S-SW to carry vSW v_r. For an S-SW v_s, *Line* 14 gets h_{v_s} as the hop-count of determined substrate paths connecting to it. Here, the determined substrate paths connecting to v_s refer to the shortest available substrate paths from v_s to where the fixed adjacent vSWs of v_r are mapped onto. For instance, in Fig. 3(b), they are the shortest available substrate paths from v_s to where *vSWs* a and f are mapped onto. In *Line* 15, we check whether remapping v_r onto v_s can balance the TCAM utilization in the SNT. If yes, *Lines* 16-20 classify v_s based on the relation between h_{v_r} and h_{v_s} and insert it in Q_1 or Q_2, respectively. Next, in *Lines* 23-30, we select a proper S-SW v_s based on Q_1, Q_2 and the TCAM utilization of S-SWs, and decide to re-map vSW v_r onto it. Finally, we update both the link mapping of VLs that connect to v_r and the values of related variables in *Lines* 31-33. The time complexity of *Algorithm* 4 is $O(|V_s|)\cdot(|V_R^s|+|E_R|\cdot(|E_s|+|V_s|\cdot\log(|V_s|)))$.

V. Numerical Simulations

In this section, we evaluate the performance of the proposed vSDN reconfiguration algorithms with numerical simulations.

Algorithm 4: Link-First Remapping Algorithm

determine the vSDNs that need to be reconfigured based on V_R^s and store them in R;
for *each* $v_s \in \tilde{V}_s$ **do**
  calculate TCAM utilization $\tilde{t}_{v_s}$ on S-SW v_s when the vSWs in V_R^s have been migrated away;
end
for *each* $v_r \in V_R^s$ **do**
  set its reconfiguration flag m_{v_r} as 1;
end
for *each vSDN* $r \in R$ **do**
  while $V_R^s \cap V_r \neq \emptyset$ **do**
    $Q_1 = \emptyset$, $Q_2 = \emptyset$;
    find the vSW $v_r \in V_R^s \cap V_r$ such that it has the maximum number of adjacent vSWs whose reconfiguration flags are 0;
    get h_{v_r} as the hop-count of determined substrate paths connecting to v_r;
    for *each S-SW* v_s *that can carry* v_r **do**
      get h_{v_s} as the hop-count of determined substrate paths connecting to v_s;
      if $|\tilde{t}_{v_s} + t_{v_r} - \bar{t}| \leq |\tilde{t}_{v_s} - \bar{t}|$ **then**
        if $h_{v_s} \leq h_{v_r}$ **then**
          insert v_s in set Q_1;
        else
          insert v_s in set Q_2;
        end
      end
    end
    if $Q_1 \neq \emptyset$ **then**
      $v_s = \underset{v_s \in Q_1}{\operatorname{argmin}}(\tilde{t}_{v_s})$;
    else if $Q_2 \neq \emptyset$ **then**
      $v_s = \underset{v_s \in Q_2}{\operatorname{argmin}}(h_{v_s})$;
    else
      select the S-SW v_s that can carry v_r and has the smallest $\tilde{t}_{v_s}$;
    end
    decide to re-map vSW v_r onto S-SW v_s;
    update link mapping of VLs connecting to v_r;
    $m_{v_r} = 0$, $\tilde{t}_{v_s} = \tilde{t}_{v_s} + t_{v_r}$;
    remove v_r from V_R^s;
  end
end

Specifically, we combine the selection and remapping algorithms according to the overall procedure in *Algorithm* 1 and obtain three algorithms. Since the algorithms use the same overall procedure and selection algorithm, we refer to them according to the names of their remapping algorithms, *i.e.*, MILP, NF-R, and LF-R. In the simulations, we implement the MILP model with GLPK 4.64 and program the heuristics in MATLAB R2016. All the simulations run on a Windows server with 3.3 GHz Intel CPU and 8 GB RAM.

We consider two topologies for the SNT, *i.e.*, the 8-node and NSFNET topologies [44] shown in Fig. 4. Considering the time complexity of the MILP, we only simulate it in the 8-node topology, while the time-efficient heuristics are evaluated in both topologies. In each simulation, we first use a greedy algorithm to embed vSDNs, and then serve traffic flows in them to generate a network scenario with unbalanced TCAM utilization. Then, the network scenario is treated as the input of the vSDN reconfiguration algorithms. We consider two performance metrics. The first one is the balanced degree of TCAM utilization, *i.e.*, $\tilde{c}$ defined in Eq. (2). The second one is the difference in total TCAM utilization in the SNT before and after vSDN reconfiguration, *i.e.*, $\tilde{t}$ defined in Eq. (3). To ensure sufficient statistical accuracy, we average the results from 10 independent simulations to get each data point.

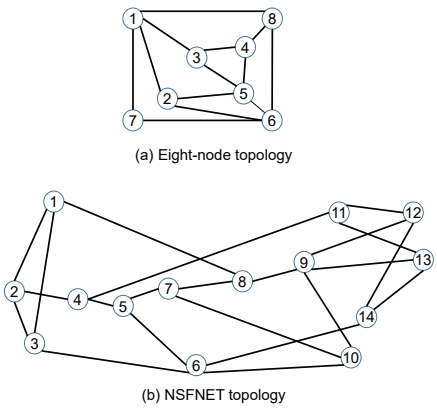

Fig. 4. SNT topologies used in simulations.

A. Results with 8-Node Topology

We first run simulations in the small-scale 8-node topology. Here, we assume that the TCAM capacity of each S-SW is 2000 flow-entries and the bandwidth capacity of each SL is 1000 units. Then, we generate 20 vSDNs with random topologies, each of which has its number of vSWs uniformly distributed within $[2, 6]$. During network operation, the TCAM utilization of each vSW ranges within $[10, 110]$ flow-entries[2], while the bandwidth usage of each VL is also randomly selected, within $[1, 10]$ units. We pause the SNT when it has unbalanced TCAM utilization, where there are at least three S-SWs that are heavy loaded and $\tilde{c}$ is more than 1100 flow-entries. Then, we use the three proposed algorithms to balance the TCAM utilization in the SNT, and adjust the selection ratio γ from 0.2 to 1 to monitor the tradeoff between the complexity and gain of vSDN reconfiguration. Note that, the parameters α and β in the MILP's objective are set as $\alpha \gg \beta$ to ensure that the first term in Eq. (4) is the primary objective. And because as long as these two parameters satisfy this condition, their actual values do not affect the optimization in the MILP, we will not discuss their impacts in the following simulations.

[2]In this work, both the simulations and experiments select the TCAM capacity of each S-SW within the practical range reported for commercial SDN switches [45], while the range of the TCAM utilization on each vSW depends on the actual network services running in its vSDN and we also determine the used values according to the results in [45].

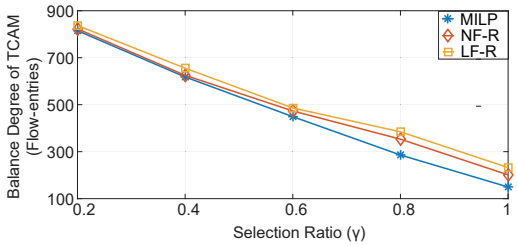

Fig. 5. Results on balance degree of TCAM usage.

Fig. 5 shows the results on the balance degree of TCAM utilization. It can be seen that the MILP model achieves the best performance on balancing the TCAM usage in the SNT, which is actually expected since minimizing $\tilde{c}$ is its primary objective. NF-R can balance the TCAM utilization as well as the MILP model when γ is less than 0.6, while the performance gap between them becomes slightly larger after $\gamma = 0.6$. LF-R performs slightly worse than NF-R in terms of $\tilde{c}$. For all the three algorithms, $\tilde{c}$ decreases with γ, which means that the TCAM utilization becomes more balanced for a larger γ. This is because with a larger γ, the algorithms can reconfigure more vSWs to balance the TCAM utilization, *i.e.*, the optimization space is larger. In practical cases, how to set γ depends on the actual needs of the InP that owns the SNT.

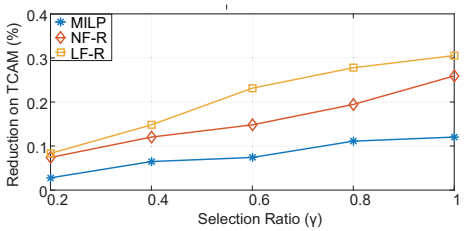

Fig. 6. Results on reduction on total TCAM usage.

The algorithms' results on reduction on total TCAM usage are plotted in Fig. 6. It is promising to observe that all the algorithms can reduce the total TCAM utilization in the SNT with vSDN reconfiguration even when γ is as small as 0.2. Therefore, for an SNT with unbalanced TCAM utilization, our proposed algorithm can not only balance the TCAM utilization but also reduce the absolute TCAM usage. Moreover, we can see that LF-R achieves the largest reduction on TCAM usage, and this phenomenon can be explained as follows. Since LF-R always tries to use the shortest available substrate paths to remap VLs, it saves the most TCAM utilization on intermediate S-SWs. NF-R reduces less TCAM usage than LF-R, while the reduction on total TCAM usage from the MILP model is the smallest among the three. This is because the MILP model gives most of its optimization power to its primary objective, *i.e.*, minimizing $\tilde{c}$. Note that, even though the MILP model gets the worst results on the reduction on TCAM usage, the absolute difference between its results and those from LF-R is actually small, which is only 0.13% when $\gamma = 1$.

The average running time of the algorithms is listed in Table II. All the algorithms generally take more running time when γ increases, due to the fact that a larger γ means that more vSWs are selected for reconfiguration. However, the running time of the MILP increases much faster than that of NF-R and LF-R, which makes it inappropriate for solving vSDN reconfiguration problems whose scales are relatively large. Both NF-R and LF-R are much more time-efficient than the MILP, and NF-R runs slightly faster than LF-R.

TABLE II
RUNNING TIME OF ALGORITHMS (SECONDS)

Selection Ratio (γ)	MILP	NF-R	LF-R
0.2	0.1	0.0037	0.0043
0.4	0.6	0.0045	0.0049
0.6	2.5	0.0050	0.0060
0.8	5.8	0.0058	0.0070
1.0	45.8	0.0061	0.0081

B. Simulation Results with the NSFNET Topology

We then evaluate the performance of NF-R and LF-R in the NSFNET topology. The TCAM capacity of each S-SW is assumed to be 5000 flow-entries, while the bandwidth capacity of each SL is set as 2000 units. We generate 80 vSDNs this time with random topologies whose numbers of vSWs are randomly selected within $[2, 8]$. The TCAM utilization of each vSW is uniformly distributed within $[10, 110]$ flow-entries, while the bandwidth usage of each VL is still randomly selected from $[1, 10]$ units. This time, we pause the SNT to invoke vSDN reconfiguration, when there are at least five heavy loaded S-SWs and $\tilde{c}$ is more than 2900 flow-entries.

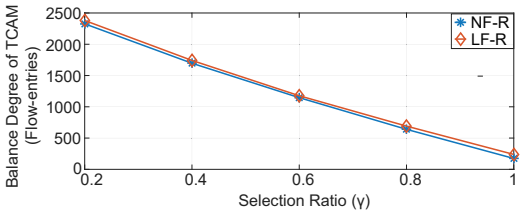

Fig. 7. Results on balance degree of TCAM usage.

Fig. 7 shows the results on the balance degree of TCAM utilization, which indicate that NF-R still performs slightly better than LF-R. This is because NF-R always tries to migrate a vSW to the one with minimum TCAM usage during vSDN reconfiguration. Nevertheless, the performance difference in terms of $\tilde{c}$ is very small. The results on reduction on total TCAM usage are illustrated in Fig. 8, and we can see that when the SNT is larger, the advantage of LF-R on TCAM saving over NF-R actually becomes larger. The results on the reduction on the average hop-count of substrate paths used by the reconfigured VLs are illustrated in Fig. 9, which indicates that the vSDN reconfiguration actually helps to reduce the hop-counts of the reconfigured VLs' substrate paths. This is because the vSDN reconfiguration organizes the vSDNs better and thus make shorter substrate paths available for link remapping. As their performance on $\tilde{c}$ is almost identical, these results suggest that LF-R is a better algorithm to use

for network scenarios with relatively larger SNT topologies, for vSDN reconfiguration.

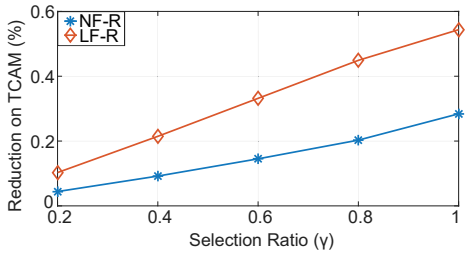

Fig. 8. Results on reduction on total TCAM usage.

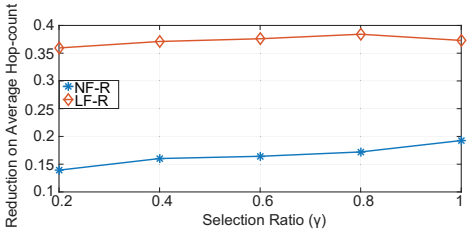

Fig. 9. Results on reduction on average hop-count of substrate paths.

VI. Experiment Demonstrations

A. System Implementation and Experimental Setup

The proposed LF-R is implemented in the network virtualization system that we developed in [27], and we conduct experiments with it to operate on the six-node SNT shown in Fig. 10. Note that, the scale of the network testbed is still relatively small, and thus the setup can only be considered as a preliminary prototype of our proposal. In our future work, we will consider large-scale verifications with substrate topologies taken from typical wide-area and datacenter networks. Here, PVX is the protocol-oblivious forwarding (POF) based NVH system that gets connected to all the S-SWs in the SNT and bridges the communications between vSWs (*i.e.*, embedded on the S-SWs) and their vCs.

The relevant modules in the system work as follows.

- **Flow-Table Database (FT-DB)**: It stores the flow-tables for vSDNs, which include both the virtual and substrate flow-tables and the mapping between them.
- **VNE Database (VNE-DB)**: It stores the VNE schemes of vSDNs, which help PVX translate the virtual flow-entries from vCs to substrate ones.
- **TCAM Monitor**: It checks the TCAM utilizations in S-SWs proactively, and when finding that the TCAM utilization in the SNT is severely unbalanced, it will generate a request to invoke a vSDN reconfiguration.
- **Input/Output API (I/O API)**: It coordinates the communications within PVX and enables the communication between VNMgr and PVX. Before reconfiguring a vSDN, it parses the reconfiguration scheme sent from the virtual network manager (VNMgr). Then, it updates the network status in FT-DB and VNE-DB and configures the related S-SWs to implement the reconfiguration.
- **Virtual Network Manager (VNMgr)**: It is a homemade module that is programmed with Python, for calculating vSDN reconfiguration schemes based on the requests from PVX. It stores the reconfiguration schemes in JSON files and sends them to I/O API through a Restful API.

Note that, with the reconfiguration scheme from VNMgr, PVX schedules the vSDN reconfiguration with I/O API. Firstly, I/O API updates the mapping relation in VNE-DB according to the reconfiguration scheme. Secondly, I/O API gets the related virtual and substrate flow-tables from FT-DB and updates them. Then, it lets PVX implement the vSDN reconfiguration by installing the updated flow-entries in the related S-SWs. Finally, it deletes the outdated flow-entries in the SNT. More details related to the implementation of our system can be found in [27].

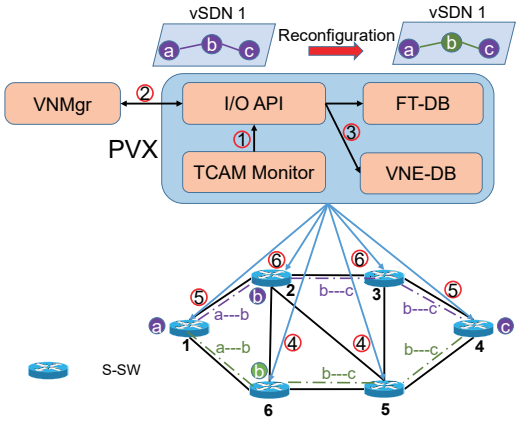

Fig. 10. System design and operation procedure.

Fig. 10 illustrates an example on how to accomplish a vSDN reconfiguration operation in the system. After having been generated by the TCAM monitor (①), the request for vSDN reconfiguration is forwarded to VNMgr by I/O API. VNMgr then uses LF-R and the network status stored in VNE-DB and FT-DB to calculate the vSDN reconfiguration scheme (②). For instance, it decides to re-map *vSW* b in *vSDN* 1 from *S-SW* 2 to *S-SW* 6, and the VLs that connect to *vSW* b (*i.e.*, *VLs* a-b and b-c) should also be re-mapped accordingly. Upon receiving the remapping scheme (③), the I/O API parses it and updates VNE-DB accordingly. Then, PVX checks FT-DB to obtain the flow-entries that get installed in *vSW* b and installs them in *S-SW* 6, and the flow-entries related to *VL* b-c also gets installed on the new intermediate node *S-SW* 5 (④).

By now, the new node and link mapping schemes have been implemented in the SNT, but for the traffic flowing in *vSDN* 1, it still takes the original substrate paths, *i.e.*, we incorporate the "make-before-break" scenario [37] to minimize traffic disruption. Next, PVX updates the flow-entries on *S-SWs* 1 and 4 to reroute the traffic on new substrate paths (⑤). Then, the traffic flowing in *vSDN* 1 gets switched to the new substrate paths instantaneously. Note that, our previous results in [27] showed that packet loss could occur during the path switching. To minimize such packet loss, we modify PVX to let it wait for a duration that is slightly longer than the maximum transmission delay of the original substrate paths

before proceeding to ⑥. Hence, we can avoid the situation in which packets transmitted on the original substrate paths get dropped during the path switching. Finally, the flow-entries on the original substrate paths are removed from *S-SWs* 2 and 3, and the vSDN reconfiguration has been accomplished (⑥).

Therefore, we can see that vSDN reconfiguration is handled solely by the VNMgr and PVX and the vCs of the vSDNs would not need to be involved. Our experimental demonstrations are based on a real network system prototype, in which PVX, VNMgr and the vCs are realized on commodity Linux servers, and each S-SW is based on the software-based POF switch [46] running on an independent high-performance Linux server with multiple 1GbE linecards. The SNT consists of six S-SWs whose topology is as that in Fig. 10.

B. Proactive and Hitless vSDN Reconfiguration

The experimental scenario for verifying the performance of our system prototype is shown in Fig. 11. Here, the SNT carries three vSDNs, *i.e.*, *vSDNs* 1-3. Except for those used for pumping background traffic, the experiment concerns four hosts connecting to the vSDNs. Since the S-SWs are realized with software-based POF switches and they operate on random-access memory (RAM) but not TCAM, we emulate TCAM limitation in them by applying an upper-limit on the number of flow-entries that can be installed in each of them.

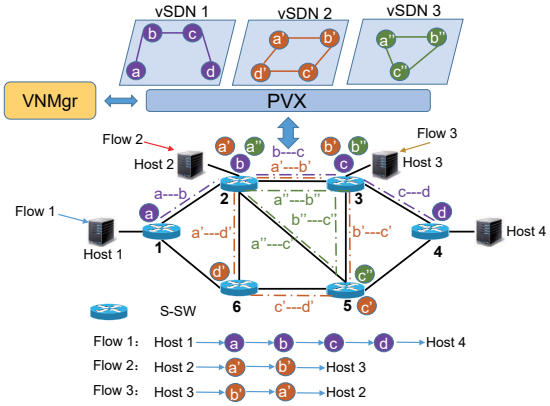

Fig. 11. Experimental scenario of hitless vSDN Reconfiguration.

The experiment then runs as follows. After the three vSDNs have been built over the SNT (at $t = 0$), two flows (*Flows* 1 and 2) start to run in *vSDNs* 1 and 2, respectively. Then, the TCAM utilization in the SNT becomes unbalanced as time goes on, and more specifically, *S-SWs* 2 and 3 are heavy loaded. At $t = 20$ seconds, the TCAM monitor in PVX determines that the TCAM on *S-SW* 2 is about to be depleted, and thus send a vSDN reconfiguration request to VNMgr. VNMgr uses LF-R with $\gamma = 0.5$ to get the vSDN reconfiguration scheme as remapping *vSWs* b and c in *vSDN* 1 onto *S-SWs* 6 and 5, respectively. Later on, at $t = 30$ seconds, *Flow* 3 tries to join in *vSDN* 2 and gets routed on *VL* b'-a'. Following the principle of SDN, since the first packet of *Flow* 3 does not match to any of the installed flow-entries in the S-SWs that carry *VL* b'-a', the vC of *vSDN* 2 will install flow-entries in *S-SWs* 2 and 3, for processing its packets. This, however, will consume the TCAM resources on *S-SWs* 2 and 3. *Flows* 2 and 3 are for the video streaming with around 3 Mbps throughput, while *Flow* 1 is for the data transfer whose throughput is fixed at 2 Mbps.

We first measure the bandwidth of *Flow* 1 on *S-SWs* 2 and 6 and *Host* 4 over time, and show the results in Fig. 12. The curves in Fig. 12(a) indicate that the vSDN reconfiguration to re-map *vSW* b in *vSDN* 1 from *S-SW* 2 to *S-SW* 6 gets implemented successfully at $t = 20$ seconds, since all the traffic in *Flow* 1 gets rerouted through *S-SW* 6 after $t = 20$ seconds. More importantly, Fig. 12(b) suggests that the end-to-end data transfer of *Flow* 1 has not been impacted by the vSDN reconfiguration, since its bandwidth measurement on *Host* 4 does not show any dip. Therefore, the results in Fig. 12 confirm that our system realizes hitless vSDN reconfiguration.

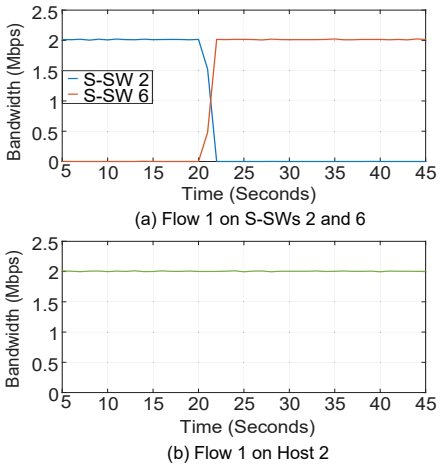

Fig. 12. Bandwidth of *Flow* 1 with vSDN reconfiguration.

Then, we measure the receiving bandwidth of *Flows* 2 and 3 on *Hosts* 3 and 2, respectively. To show the advantage of proactive vSDN reconfiguration, we also conduct an experiment without the vSDN reconfiguration at $t = 20$ seconds and use its results as the benchmark. As shown in Fig. 13(a), with the proactive vSDN reconfiguration at $t = 20$ seconds, the joining of *Flow* 3 at $t = 30$ seconds would not induce any performance degradation on both flows, since the TCAM utilization has been balanced and the new flow would not cause TCAM depletion. On the other hand, in Fig. 13(b), when the vSDN reconfiguration is absent, the receiving bandwidth of both flows cannot reach around 3 Mbps after $t = 30$ seconds. This is because since *S-SWs* 2 and 3 stay heavy loaded without the vSDN reconfiguration, the joining of *Flow* 3 at $t = 30$ seconds will use up the TCAM on them. Consequently, *vSDNs* 1 and 2 will have to compete for the TCAM space on *S-SWs* 2 and 3 after $t = 30$ seconds, which makes their vSWs send *PackectIn* messages to their vCs repeatedly and severely degrades their packet processing performance. Finally, to further confirm the advantage of the proactive vSDN reconfiguration, we measure the luminance components peak signal-to-noise-ratio (Y-PSNR) of the video received on *Host* 3 for *Flow* 2 and plot the results in Fig. 14. The results also indicate that with the vSDN reconfiguration, the playback quality of the video would not degrade, while

without the reconfiguration, there is significant degradation on the playback quality after $t = 30$ seconds.

Note that, our approach is still not completely hitless and packet loss could occur in the following two rare cases. Firstly, since PVX only waits for a duration that is slightly longer than the maximum transmission delay of the original substrate paths before deleting the old flow-entries, packets that are stuck in the original substrate paths longer than the duration would be lost. Secondly, not deleting the old flow-entries right after the path switching might lead to incorrect routing loop(s) when the original and new substrate paths of a VL share one or more S-SWs, and this would cause packet loss too. We will try to address these two cases in our future work.

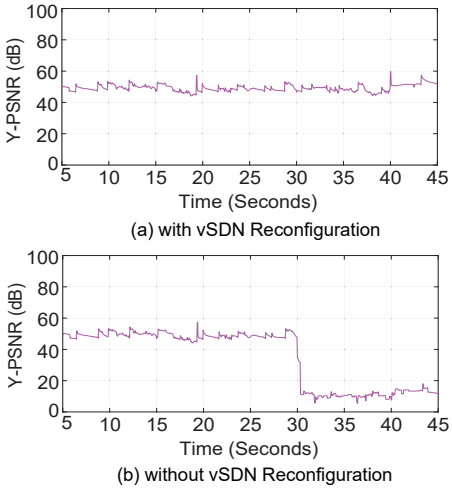

Fig. 14. Y-PSNR of video received on *Host* 3 for *Flow* 2.

C. Reconfiguration Latency and System Scalability

Finally, we conduct an experiment to evaluate the reconfiguration latency as well as system scalability of the system prototype with LF-R as the vSDN reconfiguration algorithm. Here, we define the reconfiguration latency as the time period from when a vSDN reconfiguration request reaches VNMgr to when all the selected vSWs together with all the VLs connecting to them have been re-mapped successfully. The experimental setup is shown in Fig 15, where we originally embed 10 vSDNs, each of which has four vSWs, in the SNT and make *S-SWs* 2 and 3 heavy loaded. Then, our system will select vSWs to reconfigure with $\gamma \in [0.2, 1]$ and we measure the reconfiguration latency for each case.

Table III explains the actual workload in each vSDN reconfiguration operation with different γ. It can be seen that in the worst case when $\gamma = 1$, we need to reconfigure 10 vSWs in 7 vSDNs and migrate 1164 flow-entries in total. The experimental results on reconfiguration latency are illustrated in Fig. 16, which indicates that it only takes our system prototype $\sim$550 milliseconds to accomplish the vSDN reconfiguration operation with $\gamma = 1$. Note that, it would be reasonable to assume that in a practical network environment, the interval between two adjacent vSDN reconfigurations would be at least in tens of minutes. Hence, the reconfiguration latency

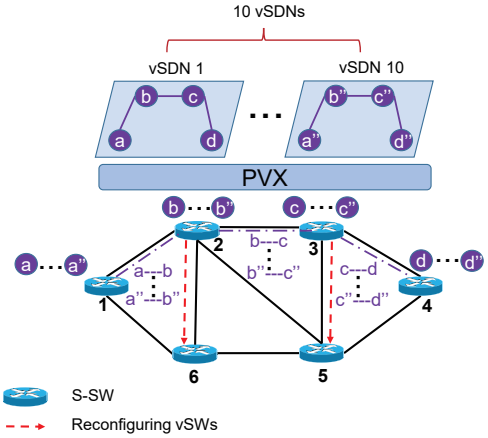

Fig. 15. Experimental setup for reconfiguration latency evaluation.

is much shorter than the interval and would not cause significant performance degradation. Meanwhile, we observe that the reconfiguration latency actually increases with γ almost linearly. This is because our current implementation tries to reconfigure vSWs in sequence. Note that, this scheme can actually be improved if we consider reconfiguring vSWs in the parallel way, and thus the reconfiguration latency could be further reduced. Therefore, we will study how to reconfigure the vSDNs in parallel in our future work.

TABLE III
WORKLOAD OF VSDN RECONFIGURATION OPERATIONS

Selection Ratio (γ)	0.2	0.4	0.6	0.8	1.0
# of Selected vSDNs	2	4	6	7	7
# of Selected vSWs	2	4	6	8	10
Flow-entries to be Migrated	277	539	773	1005	1164

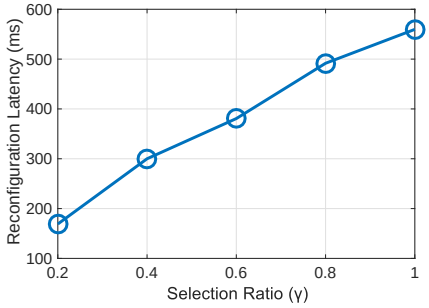

Fig. 16. Results on reconfiguration latency.

VII. CONCLUSION

In this paper, we performed a relatively comprehensive study on how to realize proactive and hitless vSDN reconfiguration in dynamic network environment. We first solved the problem of "what to reconfigure" by designing an algorithm to proactively select the vSWs that should be migrated for balancing the TCAM utilization in an SNT. Then, the problem of "how to reconfigure", *i.e.*, where to re-map the selected vSWs and the VLs connecting to them, was also studied. Specifically, we formulated an MILP model to solve the

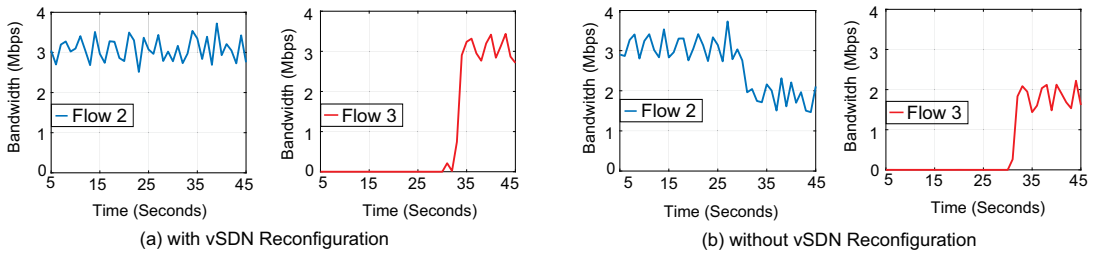

Fig. 13. Receiving bandwidth of *Flows* 2 and 3 on *Hosts* 3 and 2, respectively.

problem exactly, and designed two heuristics to improve time efficiency. Next, the proposed algorithms were implemented in our POF-enabled NVH system, and we conducted experiments to demonstrate proactive and hitless vSDN reconfiguration. The experimental results indicated that our system did make vSDN reconfiguration transparent to the vSDNs' vCs and proactive, and when reconfiguring a vSDN with live traffic, it achieved hitless operations without traffic disruption. Moreover, the scalability of the proposed system was also verified, since the results showed that it only took $\sim$550 milliseconds to reconfigure 7 vSDNs and migrate 1164 flow-entries.

Acknowledgments

This work was supported in part by the NSFC Project 61701472, CAS Key Project (QYZDY-SSW-JSC003), NGB-WMCN Key Project (2017ZX03001019-004), China Postdoctoral Science Foundation (2016M602031), and Fundamental Research Funds for the Central Universities (WK2100060021).

References

[1] T. Anderson *et al.*, "Overcoming the Internet impasse through virtualization," *IEEE Computer*, vol. 38, pp. 34–41, Apr. 2005.
[2] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.
[3] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding (A-SVNE) in optical datacenter networks," *J. Opt. Commun. Netw.*, vol. 7, pp. 1160–1171, Dec. 2015.
[4] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.
[5] J. Yin *et al.*, "Experimental demonstration of building and operating QoS-aware survivable vSD-EONs with transparent resiliency," *Opt. Express*, vol. 25, pp. 15 468–15 480, 2017.
[6] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.
[7] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.
[8] H. Huang *et al.*, "Realizing highly-available, scalable and protocol-independent vSDN slicing with a distributed network hypervisor system," *IEEE Access*, vol. 6, pp. 13 513–13 522, 2018.
[9] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, pp. 14–76, Jan. 2015.
[10] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.
[11] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.
[12] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.
[13] D. Hu *et al.*, "Flexible flow converging: A systematic case study on forwarding plane programmability of protocol-oblivious forwarding (POF)," *IEEE Access*, vol. 4, pp. 4707–4719, 2016.
[14] OpenFlow Switch Specifications. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[15] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[16] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 275–288, Mar. 2018.
[17] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, pp. 655–685, First Quarter 2016.
[18] S. Li *et al.*, "SR-PVX: A source routing based network virtualization hypervisor to enable POF-FIS programmability in vSDNs," *IEEE Access*, vol. 5, pp. 7659–7666, 2017.
[19] H. Huang *et al.*, "Embedding virtual software-defined networks over distributed hypervisors for vDC formulation," in *Proc. of ICC 2017*, pp. 1–6, May 2017.
[20] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. of INFOCOM 2014*, pp. 1–9, Apr. 2014.
[21] S. Li, K. Han, H. Huang, and Z. Zhu, "PVFlow: flow-table virtualization in POF-based vSDN hypervisor (PVX)," in *Proc. of ICNC 2018*, pp. 1–5, Mar. 2018.
[22] H. Huang *et al.*, "Cost minimization for rule caching in software defined networking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, pp. 1007–1016, Apr. 2016.
[23] M. Demirci and M. Ammar, "Design and analysis of techniques for mapping virtual networks to software-defined network substrates," *Comput. Commun.*, vol. 45, pp. 1–10, Mar. 2014.
[24] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, pp. 1888–1906, Fourth Quarter 2013.
[25] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proc. of HotSDN 2014*, pp. 175–180, Aug. 2014.
[26] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in *Proc. of ICNP 2003*, pp. 120–131, Nov. 2003.
[27] S. Zhao *et al.*, "Make big data applications more reliable: Hitless vSDN migration to avoid TCAM depletion," in *Proc. of ICC 2017*, pp. 1–6, May 2018.
[28] M. Chowdhury and M. Rahman, "ViNEYard : Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping," *IEEE/ACM Trans. Netw.*, vol. 20, pp. 206–219, Jan. 2012.
[29] J. Yin *et al.*, "On-demand and reliable vSD-EON provisioning with correlated data and control plane embedding," in *Proc. of GLOBECOM 2016*, pp. 1–6, Dec. 2016.
[30] R. Sherwood *et al.*, "FlowVisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1–13, 2009.
[31] A. Al-Shabibi *et al.*, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. of ACM HotSDN 2014*, pp. 25–30, Aug. 2014.
[32] P. Berde *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. of ACM HotSDN 2014*, pp. 1–6, Aug. 2014.
[33] R. Munoz *et al.*, "Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks," *J. Opt. Commun. Netw.*, vol. 7, pp. B62–B70, Nov. 2015.
[34] P. Lu *et al.*, "Highly-efficient data migration and backup for big data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[35] Z. Zhu *et al.*, "Build to tenants' requirements: On-demand application-driven vSD-EON slicing," *J. Opt. Commun. Netw.*, vol. 10, pp. A206–A215, Feb. 2018.
[36] B. Kong *et al.*, "Demonstration of application-driven network slicing and orchestration in optical/packet domains: On-demand vDC expansion for Hadoop MapReduce optimization," *Opt. Express*, vol. 26, pp. 14 066–14 085, 2018.
[37] M. Zhang, C. You, H. Jiang, and Z. Zhu, "Dynamic and adaptive bandwidth defragmentation in spectrum-sliced elastic optical networks with time-varying traffic," *J. Lightw. Technol.*, vol. 32, pp. 1014–1023, Mar. 2014.
[38] W. Fang *et al.*, "Joint defragmentation of optical spectrum and IT resources in elastic optical datacenter interconnections," *J. Opt. Commun. Netw.*, vol. 7, pp. 314–324, Mar. 2015.
[39] M. Zangiabady, C. Aguilar-Fuster, and J. Rubio-Loyola, "A virtual network migration approach and analysis for enhanced online virtual network embedding," in *Proc. of CNSM 2016*, pp. 324–329, Oct. 2016.
[40] S. Ghorbani *et al.*, "Transparent, live migration of a software-defined network," in *Proc. of SOCC 2014*, pp. 1–14, Nov. 2014.
[41] S. Lo, M. Ammar, E. Zegura, and M. Fayed, "Virtual network migration on real infrastructure: A PlanetLab case study," in *Proc. of IFIP 2014*, pp. 1–9, Jun. 2014.
[42] Y. Zhao *et al.*, "Virtual network migration on the GENI wide-area SDN-enabled infrastructure," in *Proc. of INFOCOM WKSHPS 2017*, pp. 265–270, Apr. 2017.
[43] P. Pisa *et al.*, "OpenFlow and Xen-based virtual network migration," in *Proc. of IFIP 2010*, pp. 170–181, Jun. 2010.
[44] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.
[45] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. of NSDI 2015*, pp. 103–115, May 2015.
[46] Q. Sun, Y. Xue, S. Li, and Z. Zhu, "Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks," *IEEE Access*, vol. 5, pp. 24 004–24 011, 2017.