On Table Resource Virtualization and Network Slicing in Programmable Data Plane

Yuhan Xue, Jianquan Peng, Kai Han, and Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—Recently, the advances on programmable data plane (PDP) promote the studies on the network virtualization in a PDP-based substrate network (SNT). In this paper, we address the table resource virtualization and network slicing in PDP-based SNTs. We first leverage the idea of "Big-Switch" to design an effective table resource virtualization scheme in which the flow tables of a virtual switch (V-SW) can be installed in multiple adjacent substrate switches (S-SWs) according to their table sizes, while the feasible table size(s) on each S-SW are determined based on the global information of the SNT. By doing so, we can regulate the flow tables in the S-SWs in a more organized way to minimize memory fragmentation. Next, we address the network slicing based on the virtualization scheme, and come up with a three-layer VNE problem. To the best of our knowledge, such a VNE problem has not been studied before and the existing algorithms designed for traditional two-layer VNE problems can hardly solve it. We formulate an integer linear programming (ILP) model to solve the VNE problem exactly, and also design a time-efficient heuristic that can provide near-optimal solutions. Finally, we implement the heuristic in TPVX, which is a network hypervisor based on protocol-oblivious forwarding (POF), and also improve its performance by introducing source routing. The new TPVX is experimentally demonstrated in a real network testbed, and the results verify that our proposal maintains the additional latency caused by the three-layer VNE well and would not degrade the network services in virtual networks (VNTs).**

***Index Terms*—Network virtualization, Virtual network embedding (VNE), Software-defined networking (SDN), Protocol-oblivious forwarding (POF), Programmable data plane (PDP).**

I. Introduction

OVER the past decade, we have witnessed astonishing advances on network technologies to overcome the ossification of the current Internet infrastructure [1–4]. Among them, software-defined networking (SDN) [1] and network virtualization [5] are two most referenced ones, since they provide network operators and service providers the capability to deliver short time-to-market, flexible and cost-effective solutions. Specifically, SDN decouples the control and data planes of a network for enhanced programmability and application-awareness [6–8], while network virtualization allows an infrastructure provider (InP) to slice its substrate network (SNT) into logically-isolated virtual networks (VNTs) and lease them to service providers (SPs) dynamically and adaptively [9–11].

The symbiosis of these two orthogonal technologies would amplify their individual advantages dramatically and bring us significantly more programmability, agility and scalability [12–14]. However, these benefits cannot be fully explored without the programmable data plane (PDP) [15, 16]. This is because an SDN-based network is not full-stack programmable without PDP, and PDP removes unnecessary restrictions on VNT slicing to make it future-proof [17]. PDP is a general term for packet processing and forwarding elements that have programmable features and depend less on existing protocols. People have come up with PDP proposals such as P4 [16] and protocol-oblivious forwarding (POF) [18], and the development and deployment in this field have gained momentum from innovations in semiconductor industry [19–21].

Some issues remain, however, in PDP environments, *e.g.*, the table resource management in switches. Actually, the issue with table resource management is not newly introduced by PDP but has been a tricky problem throughout the development of SDN [22]. This is because both the ternary content addressable memory (TCAM) and static random-access memory (SRAM) are very limited hardware resources in switches, which usually makes them insufficient to accommodate all the active flow tables [23]. The problem becomes more intimidating when network virtualization tries to create various VNTs over an SDN-based environment. Furthermore, when it comes to considering the network virtualization in an SNT built with PDP, table resource management can be even more challenging for the following two reasons. Firstly, in PDP environments, the enhanced programmability also exacerbates the pressure on memory usage, due to the tendency of offloading network functions directly into the data plane [16]. Secondly, PDP makes the sizes of flow tables change irregularly when supporting VNTs running different protocols. Therefore, without effective table resource virtualization, such variable-sized flow tables can cause severe memory fragmentation in substrate switches and eat up their table resources quickly [24–26].

In order to realize high-performance network virtualization in an SNT built with PDP, we have to address two problems properly, *i.e.*, 1) table resource virtualization and 2) network slicing based on the virtualization scheme. Specifically, an effective table resource virtualization scheme should be able to partition and organize the table resources in the SNT such that they can be allocated to VNTs to store their variable-sized flow tables with minimum memory fragmentation. Then, based on the virtualization scheme, a specific version of virtual network embedding (VNE) should be formulated and solved to facilitate cost-effective network slicing. However, to the best of our knowledge, these two problems have not been considered jointly to seek for a systematic solution before.

In this paper, we study the two problems for PDP. We first leverage the idea of "Big-Switch" to design an effective table

Y. Xue, J. Peng, 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 January 27, 2019.

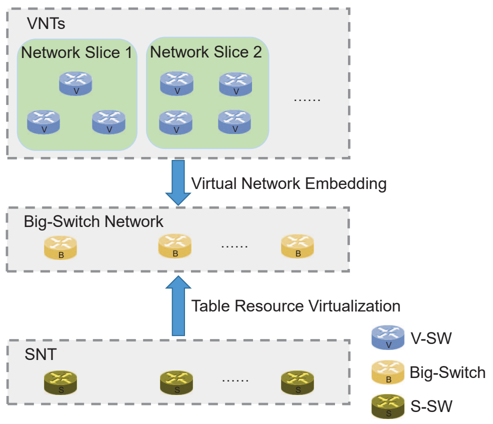


Fig. 1. Three-layer VNE considered in this work.

resource virtualization scheme in which the flow tables of a virtual switch (V-SW) can be installed in multiple adjacent substrate switches (S-SWs) according to their table sizes, while the feasible table size(s) on each S-SW are determined based on the global information of the SNT. In other words, our table resource virtualization forms a Big-Switch over multiple adjacent S-SWs, each of which has feasible table size(s) that are predetermined, and then embeds a V-SW onto the Big-Switch by distributing the variable-sized flow tables in the V-SW to a proper S-SW in it. By doing so, we can regulate the flow tables in the S-SWs in a more organized way to minimize memory fragmentation.

Next, we address the network slicing based on the virtualization scheme by formulating a three-layer VNE problem (as shown in Fig. 1), to describe the one-to-one mapping between V-SWs and Big-Switches and the many-to-many mapping between V-SWs and S-SWs. To the best of our knowledge, such a VNE problem has not been investigated before, and the existing algorithms designed for two-layer VNE problems can hardly solve it. We formulate an integer linear programming (ILP) model to solve it exactly, and also design a time-efficient heuristic that can provide near-optimal solutions.

Finally, we extend TPVX, which is the POF-enabled network hypervisor that we developed in [25], implement our VNE algorithm in it, and improve its performance by leveraging POF-based source routing [27]. The new TPVX is then experimentally demonstrated in a real network testbed that includes 11 stand-alone S-SWs. Our experimental results suggest that the additional latency caused by the many-to-many mapping between V-SWs and S-SWs is maintained well and thus would not cause noticeable performance degradation.

The rest of paper is organized as follows. Section II introduces the background regarding PDP and provides a brief survey on the related work. We explain the network model and problem description in Section III. Then, the ILP for the three-layer VNE problem is formulated in Section IV, while the time-efficient heuristic is designed in Section V. Section VI evaluates the algorithms with numerical simulations. Next, we describe the system implementations for extending TPVX and the experimental results to demonstrate its effectiveness in Section VII. Finally, Section VIII summarizes the paper.

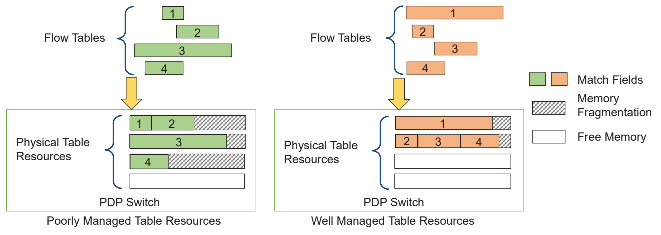


Fig. 2. Organizations of flow tables in PDP switches.

II. Background and Related Work

In this section, we first explain the operation principle of PDP, and then conduct a literature survey on the related work.

A. Background

Different from the OpenFlow switches that operate on the match fields defined based on existing network protocols, a switch with PDP has the freedom of defining arbitrary match fields and packet processing procedure. To achieve this, PDP schemes such as P4 and POF have been proposed [16, 18]. Supported by several commercially-available solutions [19, 20], the P4-based scheme has been widely considered. Specifically, P4 defines a high-level programming language to program the processor(s) in a PDP switch for customizing the formats and processing procedure of packets. On the other hand, a POF switch matches to an arbitrary packet field with the tuple $<$*offset*, *length*$>$, where *offset* represents the start location (in bits) of the field in a packet and *length* denotes its length (also in bits) [15], and processes the field with the protocol-oblivious forwarding instruction set (POF-FIS) [18].

As both P4 and POF allow us to define arbitrary packet fields with various lengths, the sizes and organization of flow tables in a PDP switch are much different from those in an OpenFlow switch. The sizes of OpenFlow-based flow tables are the same, since each of them needs to include all the match fields supported by the OpenFlow specifications. For the fields that will not be matched during packet processing, the OpenFlow switch just ignores them with wildcards. However, with enhanced programmability, a PDP switch uses variable-sized flow tables and irrelevant match fields can be excluded from them. Hence, as shown in Fig. 2, if an effective table resource management scheme is absent, memory fragmentation can happen and eventually lead to severe waste of the table resources in the PDP switch [19]. This actually motivates us to study the problem that in the case of network virtualization, how to efficiently accommodate variable-sized flow tables from V-SWs in S-SWs to reduce memory fragmentation.

B. Related Work

The authors of [19] have considered how to optimize the hardware design to relieve the memory fragmentation in SDN switches, but their proposals are hardware-specific, which means that they might not be enabled on all the PDP switches. Both the studies in [28, 29] proposed to use slow but abundant software-based table resources as the supplement to overcome the shortage of hardware-based table resources. However, the long lookup time of software-based table resources could be an issue. Meanwhile, we notice that hardware resource disaggregation [30], *i.e.*, spreading the usage of one or multiple types of hardware resources across multiple switches, might be leveraged to potentially solve the memory fragmentation in PDP switches. This is because existing techniques such as the remote direct memory access (RDMA) [23] can be utilized to access remote memory with relatively low latency.

Shirali-Shahreza *et al.* [31] considered how to expedite flow rule evictions and delay flow rule installations to adapt to the limited table resources in SDN switches. The authors of [32] tried to address the shortage of table resources in large-scale SDN-based networks with deploying aggregated default paths specified by wildcard forwarding rules. They formulated the problem as an ILP model and designed a time-efficient approximate algorithm. In [33], the proposal is to divide the TCAM in an SDN switch into small blocks with a fixed size and leverage fine-grained memory operations to reduce the memory fragmentation in TCAM. Nevertheless, the scheme might need to segment and reassemble flow tables frequently, which would lead to extra delay and overheads.

A few network hypervisor systems have been developed to realize network virtualization in PDP [17, 24, 25, 34]. HyPer4 [34] was implemented to achieve network virtualization in P4-based PDP. However, since VNT requests cannot be known in advance and runtime reconfiguration of physical tables would be prohibitively difficult, HyPer4 has to pre-allocate a relatively large number of physical tables if it needs to reconfigure virtual tables at runtime. This might waste a lot of memory in typical HyPer4 deployments. Previously, we have developed PVFlow [24] and SR-PVX [17] to facilitate network virtualization in POF-based PDP. Nevertheless, the systems developed in [14, 17, 24, 34] did not consider how to address the memory fragmentation in S-SWs. In [25], we showed the preliminary results on TPVX, which is a POF-enabled network hypervisor that can leverage the idea of Big-Switch to realize effective table resource virtualization for reducing the memory fragmentation in S-SWs. However, the system performance has not been optimized, and more importantly, the three-layer VNE algorithm for network slicing has not been designed yet.

Finally, we hope to point out that the three-layer VNE considered in this work is fundamentally different from the traditional multilayer VNE [35, 36]. More specifically, the traditional multilayer VNE tries to embed VNTs onto an SNT that includes multiple physically-separated layers (*e.g.*, an IP-over-optical network), while in our three-layer VNE, the layers of Big-Switches and S-SWs are actually based on the same network elements and only logically-separated. In other words, the one-to-one mapping between V-SWs and Big-Switches and the many-to-many mapping between V-SWs and S-SWs in our problem are correlated and thus cannot be optimized separately. Therefore, our VNE problem is more complex.

III. Problem Description

In this section, we first describe the table resource virtualization scheme that leverages Big-Switch to realize network virtualization in PDP, then explain the network model based on the scheme, and finally define the three-layer VNE problem. As abbreviations are frequently used in this paper, we list all the major but not well-known ones in Table I.

TABLE I
Major Abbreviations

Abbrev.	Full Name	Abbrev.	Full Name
POF	Protocol-oblivious forwarding	SNT	Substrate network
TCAM	Ternary content-addressable memory	S-SW	Substrate switch
VNE	Virtual network embedding	SL	Substrate link
PDP	Programmable data plane	VNT	Virtual network
SP	Service provider	VL	Virtual link
InP	Infrastructure provider	V-SW	Virtual switch

A. Table Resource Virtualization in PDP

In network virtualization, the memory fragmentation in Fig. 2 cannot be removed with the one-to-one mapping between V-SWs and S-SWs in traditional VNE schemes [37, 38]. This is because the flow tables in each V-SW have irregular sizes.

***Definition** 1:* Sub-tables are obtained by partitioning a physical table, and mapping a flow table in a V-SW to a sub-table means to put all the flow table's entries in the sub-table.

One approach to reduce such memory fragmentation is to map flow tables to partitioned table resources (*i.e.*, sub-tables of different sizes) [39]. However, since the table resources on each PDP switch are very limited, we have the dilemma that each physical table can only carry very few flow tables if sub-tables with many sizes are partitioned on each S-SW; otherwise, memory fragmentation still exists since we need to round up flow table sizes frequently. Meanwhile, considering the fact that each S-SW can be shared by several V-SWs, we cannot repartition the sub-tables in it at runtime. Hence, our table resource virtualization scheme allows the flow tables of a V-SW being mapped to the sub-tables in multiple adjacent S-SWs (*i.e.*, they form a Big-Switch), while the feasible size(s) of the sub-tables on each S-SW are predetermined.

Fig. 3 explains the operation principle of the table resource virtualization scheme. The three-layer VNE scenario is shown in Fig. 3(a), where we form two Big-Switches over the three S-SWs and embed the V-SWs onto the Big-Switches. To minimize memory fragmentation in the S-SWs, we partition the table resources in them into sub-tables with different sizes and install flow tables of the V-SWs in the sub-tables in a best-fit way (as illustrated in Fig. 3(b)). Specifically, if the size of a flow table does not equal to any of the feasible ones, we pad it with wildcards to round up its size to the next available one. Then, the S-SWs in a Big-Switch cooperate with each other to realize the packet processing in each embedded V-SW.

B. Network Model of Three-layer VNE

We model the SNT as an undirected graph $G_s(V_s, E_s)$, where V_s and E_s are the sets of S-SWs and substrate links (SLs), respectively. For simplicity, we assume that each S-SW $v_s \in V_s$ only contains sub-tables with one size[1]. Hence, the table resources in each S-SW have a table size w_{v_s} and

[1]Note that, this assumption would not limit the generality of the algorithms designed in this work, since they can handle S-SWs with multiple table sizes after minor modifications.

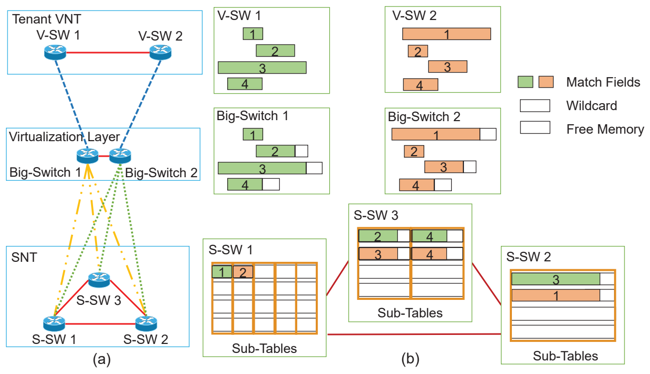


Fig. 3. Table resource virtualization in PDP leveraging Big-Switch.

an entry depth d_{v_s}, which denote the size of each physical sub-table and the number of entries that the S-SW can store, respectively. Note that, we allow one S-SW can be used by multiple Big-Switches in the three-layer VNE, *i.e.*, the mapping between S-SWs and Big-Switches is many-to-many. The bandwidth capacity of an SL $e_s \in E_s$ is denoted as b_{e_s}.

The Big-Switches generated over the S-SWs form a network, which can also be modeled as an undirected graph $G_b(V_b, E_b)$. Here, V_b and E_b are the sets of Big-Switches and the links to interconnect them, respectively. Since each Big-Switch $v_b \in V_b$ includes several adjacent S-SWs that are fully connected, v_b has a set of table sizes and a set of corresponding entry depths. We use $w^i_{v_b}$ to denote the table size of the i-th S-SW in Big-Switch $v_b \in V_b$ and the corresponding entry depth is $d^i_{v_b}$. A VNT is also an undirected graph $G_r(V_r, E_r)$, where V_r and E_r are the sets of V-SWs and virtual links (VLs), respectively. Each V-SW requires certain table resources, which can be represented by pairs of flow table sizes and entry depths. Similarly, we use $w^i_{v_r}$ to denote the table size of the i-th flow table array in V-SW $v_r \in V_r$ and the corresponding entry depth is $d^i_{v_r}$. The bandwidth requirement of a VL $e_r \in E_r$ can be denoted as b_{e_r}. The mapping between V-SWs and Big-Switches is one-to-one.

C. Problem Description of Three-Layer VNE

To facilitate the three-layer VNE, we first determine all the feasible Big-Switch configurations in the SNT. Specifically, this can be done by search $G_s(V_s, E_s)$ to find all the complete subgraphs in it, and to limit the complexity of the search, we can restrict the node-count of each complete graph below certain threshold (*e.g.*, 3). Then, we obtain $G_b(V_b, E_b)$ and can solve the three-layer VNE problem with either an ILP model or a time-efficient heuristic. To explain the three-layer VNE, we show an illustrative example in Fig. 4. Here, Fig. 4(b) shows the SNT, where the number on each SL e_s is its bandwidth capacity b_{e_s} while the tuple aside each S-SW v_s is its table size (in bits) and entry depth, *i.e.*, (w_{v_s}, d_{v_s}). For example, the $(64, 100)$ aside *S-SW* 1 means that the S-SW has a table size of 64 bits and the corresponding entry depth is 100 entries. The two VNTs are plotted in Fig. 4(a), where the number on each VL is its bandwidth requirement, and the tuples aside each VN are its table resource requirement in the pairs of table size (in bits) and entry depth. For instance, the $(32, 60)$ and $(20, 80)$ aside *V-SW* a means that the V-SW requires two types of flow tables: 1) 60 flow tables whose sizes are 32 bits and 2) 80 entries of 20-bit ones.

Fig. 4(c) explains how to map the VNTs onto the SNT by leveraging the Big-Switches, where the dotted cycles indicate the configurations of Big-Switches (*i.e.*, the many-to-many mapping between Big-Switches and S-SWs). It can be seen clearly that our three-layer VNE problem is different from and more complex than the traditional two-layer and multilayer VNE problems. Specifically, our problem not only has more complex node mapping but also needs to address more sophisticated link mapping. For instance, each link between two Big-Switches actually relates to a set of SLs and/or substrate paths, since each Big-Switch includes multiple S-SWs.

IV. ILP Formulation

In this section, we formulate an ILP model to solve the aforementioned three-layer VNE problem for embedding a VNT in the SNT. We first preprocess the SNT and VNT as follows. The connectivity of $G_s(V_s, E_s)$, which is the probability that any two S-SWs in V_s are directly connected in G_s, is represented as ε. We calculate K shortest paths between each S-SW pair in $G_s(V_s, E_s)$[2] and store them in the path set P, where a subset P_{u_s,v_s} includes all the calculated paths between u_s and v_s $(u_s, v_s \in V_s)$. If we have $K = 1$, P_{u_s,v_s} degenerates as the shortest substrate path p_{u_s,v_s}. Then, we define $hop(\cdot)$ to return the hop-count of a path, and use $\bar{h}$ and σ_h to denote the average value and standard deviation of the hop-counts of all the paths in P, respectively. The average value and standard deviation of the required table sizes of the VNT are denoted as $\bar{w}$ and σ_w, respectively.

Notations:

- $G_s(V_s, E_s)$: the topology of the SNT.
- ε: the connectivity of the SNT.
- P: the set of substrate paths, where $p_{u_s,v_s} \in P$ is the path to connect u_s and v_s $(u_s, v_s \in V_s)$.
- $\bar{h}$ and σ_h: the statistics regarding the hop-counts of all the paths in P.
- $b_{(u_s,v_s)}$: the available bandwidth on substrate path p_{u_s,v_s}.
- w_{v_s}: the table size of the sub-tables on S-SW v_s.
- d_{v_s}: the entry depth of S-SW v_s.
- $G_b(V_b, E_b)$: the topology of the Big-Switch network.
- $\xi^{v_b}_{v_s}$: the indicator that equals 1 if S-SW v_s is included in Big-Switch v_b, and 0 otherwise.
- $G_r(V_r, E_r)$: the topology of the VNT.
- $w^i_{v_r}$: the table size of the i-th flow table array in V-SW $v_r \in V_r$.
- $d^i_{v_r}$: the entry depth of the i-th flow table array in V-SW $v_r \in V_r$.
- b_{e_r}: the bandwidth requirement of VL $e_r \in E_r$.
- $\bar{w}$ and σ_w: the statistics regarding the required table sizes of the VNT.

[2]In an SNT where the bandwidth resources are much more abundant than the table resources (*i.e.*, in most of the practical cases), we can only calculate the shortest path between each S-SW pair (*i.e.*, $K = 1$) to limit the complexity of the model and avoid causing unnecessary latency in the VNTs.

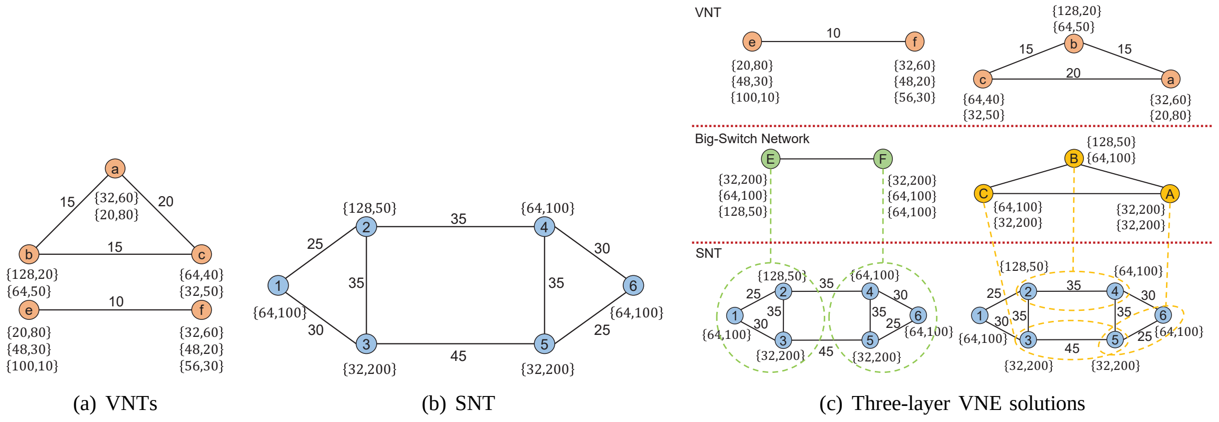


Fig. 4. Example on three-layer VNE for network virtualization in a PDP-based SNT.

Variables:

- $\delta_{v_b}^{v_r}$: the boolean variable that equals 1 if V-SW v_r is embedded onto Big-Switch v_b, and 0 otherwise.
- $\rho_{v_s}^{i,v_r}$: the boolean variable that equals 1 if the i-th flow table array in V-SW v_r is embedded onto S-SW v_s, and 0 otherwise.
- $\omega_{(u_s,v_s)}^{e_r}$: the boolean variable that equals 1 if VL e_r is embedded onto substrate path p_{u_s,v_s}, and 0 otherwise.

Objective:

In our three-layer VNE, each VNT actually consumes two types of resources in the SNT, *i.e.*, the table resources on S-SWs and the bandwidth resources on SLs. For the table resources, since a flow table array in a V-SW can be embedded onto an S-SW whose table size is larger than its, we should try to minimize such table resource wastes (*i.e.*, the memory fragmentation). Hence, we leverage the coefficient of variation to normalize the memory fragmentation in the SNT as

$$\psi_t = \frac{1}{\sigma_w}\left[\frac{1}{\sum\limits_{v_r,i} d_{v_r}^i}\left(\sum_{v_s,v_r}\sum_{i}\rho_{v_s}^{i,v_r}\cdot w_{v_s}\cdot d_{v_r}^i\right)-\bar{w}\right]. \quad (1)$$

Similarly, the bandwidth utilization can be normalized as

$$\psi_b = \frac{1}{\sigma_h}\left\{\frac{1}{|E_r|}\left[\sum_{p_{u_s,v_s}\in P}\sum_{e_r}\omega_{(u_s,v_s)}^{e_r}\cdot hop(p_{u_s,v_s})\right]-\bar{h}\right\}. \quad (2)$$

We try to minimize the total normalized resource utilization. Meanwhile, we hope to point out that the connectivity of SNT actually affects the formulation of Big-Switches in it, *i.e.*, a better connected SNT leads to more feasible Big-Switches. For instance, the well-known 14-node NSFNET topology [40] only provides 23 Big-Switches while we can form 455 Big-Switches in a 14-node complete graph, if we limit the number of S-SWs in each Big-Switch within $[2,3]$. Therefore, we design the overall optimization objective of the ILP as

$$\text{Minimize} \quad \psi = \alpha\cdot\psi_t + \varepsilon\cdot(1-\alpha)\cdot\psi_b, \quad (3)$$

where α is the weight coefficient to balance the importance of the two terms. In general, we have $\alpha > 0.5$ to ensure that minimizing the first term is the major objective, while the actual setting of α will be discussed in Section VI when we present the numerical simulation results.

Constraints:

1) *Node Mapping Constraints:*

$$\sum_{v_b\in V_b}\delta_{v_b}^{v_r}=1, \quad \forall v_r\in V_r. \quad (4)$$

Eq. (4) ensures that a V-SW can be embedded onto one and only one Big-Switch.

$$\rho_{v_s}^{i,v_r}\cdot\left(d_{v_s}-d_{v_r}^i\right)\geq 0, \quad \forall v_r\in V_r,\ v_s\in V_s. \quad (5)$$

Eq. (5) ensures that all the flow table arrays in each V-SW are embedded onto the S-SWs whose entry depths are large enough to accommodate them.

$$\rho_{v_s}^{i,v_r}\cdot\left(w_{v_s}-w_{v_r}^i\right)\geq 0, \quad \forall v_r\in V_r,\ v_s\in V_s. \quad (6)$$

Eq. (6) ensures that if a V-SW is embedded onto a Big-Switch, each of its flow table array can be embedded onto an S-SW in the Big-Switch and the S-SW's table size is large enough.

$$\sum_{v_s\in V_s}\rho_{v_s}^{i,v_r}\cdot\xi_{v_s}^{v_b}=\delta_{v_b}^{v_r}, \quad \forall v_r\in V_r,\ v_b\in V_b,\ i. \quad (7)$$

Eq. (7) ensures that the node mapping relations defined by two variables $\rho_{v_s}^{i,v_r}$ and $\delta_{v_b}^{v_r}$ are consistent.

2) *Link Mapping Constraints:*

$$\sum_{e_r\in E_r}\omega_{(u_s,v_s)}^{e_r}\cdot b_{e_r}\leq b_{(u_s,v_s)}, \quad \forall p_{u_s,v_s}\in P. \quad (8)$$

Eq. (8) ensures that if a VL is embedded onto a substrate path, its bandwidth demand would not exceed the path's capacity.

$$\sum_{p_{u_s,v_s}\in P}\omega_{(u_s,v_s)}^{e_r}=1, \quad \forall e_r\in E_r. \quad (9)$$

Eq. (9) ensures that each VL is embedded onto one and only one substrate path.

$$\sum_{v_s\in V_s}\sum_{v_r\in V_r}\omega_{(u_s,v_s)}^{(u_r,v_r)}\cdot\xi_{u_s}^{u_b}=\delta_{u_b}^{u_r}, \quad \forall u_b\in V_b,\ u_s\in V_s,\ \{u_r:(u_r,v_r)\in E_r\}. \quad (10)$$

$$\sum_{u_s\in V_s}\sum_{u_r\in V_r}\omega_{(u_s,v_s)}^{(u_r,v_r)}\cdot\xi_{v_s}^{v_b}=\delta_{v_b}^{v_r}, \quad \forall v_b\in V_b,\ v_s\in V_s,\ \{v_r:(u_r,v_r)\in E_r\}. \quad (11)$$

Eqs. (10) and (11) ensure that if a VL gets embedded onto a substrate path, the node mappings of its two end V-SWs are correctly represented by variable $\delta^{v_r}_{v_b}$, and *vice versa*.

$$\omega^{(u_r,v_r)}_{(u_s,v_s)} = \omega^{(v_r,u_r)}_{(v_s,u_s)}, \quad \forall (u_r,v_r) \in E_r,\ p_{u_s,v_s} \in P. \qquad (12)$$

Eq. (12) ensures that all the link mappings are bidirectional.

V. Heuristic Algorithm

To solve large-scale three-layer VNE problems time-efficiently, we design a heuristic in this section. *Algorithm* 1 shows its overall procedure, which includes four major steps: 1) preprocessing to build the Big-Switch network, 2) finding node mapping candidates, 3) finding link mapping candidates, and 4) determining the three-layer VNE scheme based on dispersed mapping. The following subsections explain the details of the four steps (*i.e.*, *Algorithms* 2-5).

Algorithm 1: Overall procedure of dispersed three-layer VNE (D3L-VNE)

```
Input: Latest SNT $G_s(V_s, E_s)$, VNT $G_r(V_r, E_r)$.
1 preprocess SNT $G_s$ with Algorithm 2 to build a
  Big-Switch network $G_b(V_b, E_b)$ for carrying $G_r$;
2 find node mapping candidates with Algorithm 3;
3 find link mapping candidates with Algorithm 4;
4 try to get the VNE scheme for $G_r$ with Algorithm 5;
5 if a feasible VNE scheme cannot be found then
6     mark VNT $G_r$ as blocked;
7 end
```

A. Preprocessing of SNT

Algorithm 2 explains how we preprocess the SNT $G_s(V_s, E_s)$ to build a Big-Switch network $G_b(V_b, E_b)$ on which the VNT $G_r(V_r, E_r)$ can potentially be embedded. Note that, to limit the problem size when the SNT is relatively large, we set upper-bounds on the number of Big-Switches that will be included in the Big-Switch network and the number of S-SWs in each Big-Switch, which are $\mathcal{V}$ and $\mathcal{V}_m$, respectively. Here, we usually set $\mathcal{V}_m = 3$ since complete subgraphs with a larger size are difficult to be found in a normal SNT topology, while the setting of $\mathcal{V}$ should be determined based on the size of the VNT. *Lines* 1 and 2 are for the initialization, where we ignore the S-SWs whose table resources are not compatible with at least one required flow table array in the VNT. Then, the while loop that covers *Lines* 3-15 tries to build Big-Switches one by one based on S-SWs that form complete subgraphs. The time complexity of *Algorithm* 2 is $O(|V_s|^2 \cdot \mathcal{V}_m)$.

B. Finding Node Mapping Candidates

With the Big-Switch network $G_b(V_b, E_b)$ obtained by *Algorithm* 2, *Algorithm* 3 checks the Big-Switches in it based on their table resources to find the candidate Big-Switches for node mapping. *Line* 1 is for the initialization. Then, the for-loop covering *Lines* 2-14 checks whether each V-SW in $G_r(V_r, E_r)$ can be embedded onto at least one Big-Switch in $G_b(V_b, E_b)$. Here, we use $flag$ to indicate whether a V-SW v_r can be accommodated by at least one Big-Switch in $G_b(V_b, E_b)$ (*Line* 3). In the for-loop that covers *Lines* 4-10, each Big-Switch is checked to see whether the table resources in its S-SWs can support all the required flow table arrays in the V-SW v_r. Note that, to avoid severe memory fragmentation in the substrate table resources, we set an upper-bound on the largest wasted table fragmentation, *i.e.*, $\mathcal{W}_f$. Suppose we plan to embed the i-th required flow table array in v_r onto S-SW v_s, then we must have $w^i_{v_r} \le w_{v_s}$ and the wasted table fragmentation can be calculated as

$$\mathcal{W} = \frac{w_{v_s} - w^i_{v_r}}{w^i_{v_r}}, \qquad (13)$$

which should be less than $\mathcal{W}_f$. Meanwhile, since each required flow table array applies two-dimensional requirements, we also have to ensure the entry depth in v_s is sufficient, *i.e.*, $d_{v_s} \ge d^i_{v_r}$. If the two-dimensional requirements of all the required flow table arrays in v_r can be satisfied by v_b, we insert the mapping as a tuple (v_r, v_b) in the node mapping candidate set $\mathcal{M}_v$ and update $flag$ (*Lines* 6-9). Otherwise, if any of the required flow table arrays in v_r cannot find at least one feasible S-SW in all of the Big-Switches, we will return an empty $\mathcal{M}_v$ to indicate that the VNT G_r should be blocked (*Lines* 11-13). Finally, in *Line* 15, we return the obtained node mapping candidate set $\mathcal{M}_v$. The time complexity of *Algorithm* 3 is $O(|V_b| \cdot |V_r| \cdot \mathcal{V}_m \cdot \mathcal{V}^r_m)$, where $\mathcal{V}^r_m$ is the maximum number of flow table arrays required by V-SWs.

Algorithm 2: Preprocessing of SNT to build big-switch network

```
Input: Latest SNT $G_s(V_s, E_s)$, number of
       Big-Switches in Big-Switch network $\mathcal{V}$,
       maximum S-SWs in a Big-Switch $\mathcal{V}_m$.
Output: Big-Switch network $G_b(V_b, E_b)$.
1 remove S-SWs in $V_s$ whose table size and entry
  depth cannot satisfy at least one required flow table
  array in $V_r$;
2 $i = 0$, $V^t_s = V_s$;
3 while $i < \mathcal{V}$ or $V^t_s \neq \emptyset$ do
4     select an S-SW in $v_s \in V^t_s$;
5     $V^t_s = V^t_s \setminus v_s$, $j = 1$, $i = i + 1$;
6     create a new Big-Switch $v_b$, insert $v_s$ in $v_b$, and
      update $\xi^{v_b}_{v_s}$;
7     while $j < \mathcal{V}_m$ do
8         try to find an S-SW $u_s \in V_s$ that is adjacent
          to all the S-SWs in $v_b$;
9         if such an S-SW cannot be found then
10            break;
11        end
12        insert $u_s$ in Big-Switch $v_b$ and update $\xi^{v_b}_{u_s}$;
13        $j = j + 1$;
14    end
15 end
16 return($G_b(V_b, E_b)$);
```

Algorithm 3: Finding node mapping candidates

```
Input: Big-Switch network $G_b(V_b, E_b)$, VNT $G_r(V_r, E_r)$, largest wasted table fragmentation that is permitted $\mathcal{W}_f$.
Output: Set of node mapping candidates $\mathcal{M}_v$.
1  $\mathcal{M}_v = \emptyset$;
2  for each V-SW $v_r \in V_r$ do
3      $flag = 0$;
4      for each Big-Switch $v_b \in V_b$ do
5          try to embed all the required flow table arrays in $v_r$ onto the S-SWs in $v_b$ while making sure that the constraint on $\mathcal{W}_f$ is not violated;
6          if all the flow table arrays can be embedded then
7              insert tuple $(v_r, v_b)$ in $\mathcal{M}_v$;
8              $flag = 1$;
9          end
10     end
11     if $flag = 0$ then
12         return($\mathcal{M}_v = \emptyset$);
13     end
14 end
15 return($\mathcal{M}_v$);
```

C. Finding Link Mapping Candidates

Algorithm 4 finds all the feasible link mapping candidates based on the set of node mapping candidates $\mathcal{M}_v$ from *Algorithm* 3. *Line* 1 is for the initialization. We use the for-loop covering *Lines* 2-23 to find all the feasible link mapping candidates for each VL $(u_r, v_r) \in E_r$ in the VNT. Based on $\mathcal{M}_v$, *Lines* 3 and 4 find all the feasible Big-Switches that can carry the end V-SWs of (u_r, v_r) and store them in $V^t_{b,1}$ and $V^t_{b,2}$, respectively. We still use $flag$ to indicate whether the VL (u_r, v_r) can be accommodated by at least one substrate path (*Line* 5). Then, the multi-level for-loops from *Line* 6 to 19 try to determine the link mapping candidates. Note that, the mapping between Big-Switches and S-SWs are many-to-many. Hence, for each feasible Big-Switch u_b in $V^t_{b,1}$, we need to check all of its S-SWs for the link mapping, and the same thing is applied to each feasible Big-Switch v_b in $V^t_{b,2}$.

Here, we would like to point out that the three-layer VNE and the introduction of Big-Switches make the link mapping in this work significantly different from those in traditional VNE problems. Specifically, as long as an S-SW is in the Big-Switch on which a V-SW gets embedded on, the V-SW can use it as an end S-SW in the related link mapping even though none of the required flow table arrays actually gets embedded on the S-SW. This additional freedom in link mapping is brought by the operation principle of Big-Switches, *i.e.*, the S-SWs in a Big-Switch cooperate with each other to process packets as in a logic switch. In *Lines* 11-15, we insert the tuple $((u_r, v_r), p_{u_s,v_s})$ as a link mapping candidate in $\mathcal{M}_e$, if the substrate path p_{u_s,v_s} has sufficient bandwidth capacity to carry (u_r, v_r) and its hop-count does not exceed the preset threshold h_m. Here, the preset threshold on hop-count is used to avoid considering the substrate paths that are too long in link mapping, for saving bandwidth resources. Next, after having checked all the feasible S-SW pairs, we will return empty $\mathcal{M}_v$ and $\tilde{\mathcal{M}}_v$ to indicate that the VNT G_r should be blocked (*Lines* 20-22) if zero feasible link mapping candidate has been found. Here, the updated node mapping candidate set $\tilde{\mathcal{M}}_v$ is used to store each node mapping candidate that will lead to feasible link mapping candidate(s), and thus we have $\tilde{\mathcal{M}}_v \subseteq \mathcal{M}_v$. Finally, *Line* 24 returns the obtained $\mathcal{M}_v$ and $\tilde{\mathcal{M}}_v$. The time complexity of *Algorithm* 4 is $O(|E_r|^2 \cdot |V_b|^2 \cdot \mathcal{V}_m{}^2)$.

Algorithm 4: Finding link mapping candidates

```
Input: SNT $G_s(V_s, E_s)$, Big-Switch network $G_b(V_b, E_b)$, VNT $G_r(V_r, E_r)$, node mapping candidate set $\mathcal{M}_v$, maximum hop-count $h_m$.
Output: updated node mapping candidate set $\tilde{\mathcal{M}}_v$, link mapping candidate set $\mathcal{M}_e$.
1  $\mathcal{M}_e = \emptyset$, $\tilde{\mathcal{M}}_v = \emptyset$;
2  for each VL $(u_r, v_r) \in E_r$ do
3      check $\mathcal{M}_v$ to find all the feasible Big-Switches that can carry V-SW $u_r$ and store them in $V^t_{b,1}$;
4      check $\mathcal{M}_v$ to find all the feasible Big-Switches that can carry V-SW $v_r$ and store them in $V^t_{b,2}$;
5      $flag = 0$;
6      for each Big-Switch $u_b$ in $V^t_{b,1}$ do
7          for each S-SW $u_s$ in Big-Switch $u_b$ do
8              for each Big-Switch $v_b$ in $V^t_{b,2}$ do
9                  for each S-SW $v_s$ in Big-Switch $v_b$ do
10                     get substrate path $p_{u_s,v_s}$ from $P$;
11                     if $hop(p_{u_s,v_s}) \leq h_m$ and the bandwidth requirement $b_{((u_r,v_r))}$ can be satisfied by $p_{u_s,v_s}$ then
12                         insert tuple $((u_r, v_r), p_{u_s,v_s})$ in $\mathcal{M}_e$;
13                         insert tuples $(u_r, u_b)$ and $(v_r, v_b)$ in $\tilde{\mathcal{M}}_v$;
14                         $flag = 1$;
15                     end
16                 end
17             end
18         end
19     end
20     if $flag = 0$ then
21         return($\mathcal{M}_e = \emptyset$, $\tilde{\mathcal{M}}_v = \emptyset$);
22     end
23 end
24 return($\mathcal{M}_e$, $\tilde{\mathcal{M}}_v$);
```

D. Determining Three-Layer VNE Scheme

Finally, we design *Algorithm* 5 to calculate the three-layer VNE scheme for the VNT $G_r(V_r, E_r)$ based on $\mathcal{M}_v$ and $\tilde{\mathcal{M}}_v$. Here, we leverage dispersed mapping to avoid the "big island" problem [37], *i.e.*, when multi-dimensional resources are considered, the unbalanced utilization of certain type(s) of resources could make the remaining types of available resources unusable [37, 41, 42]. Therefore, in the initialization in *Lines* 1-10, we assign a counter to each Big-Switch to check how many times its S-SWs could be used in a potential node mapping of certain V-SW. Then, in the node mapping of each S-SW, we should try the feasible Big-Switches in ascending order of their counters, as shown in *Lines* 11-19. Note that, the "total entry depth" of a V-SW v_r (*Line* 11) means the summation of the entry depths of all its required flow table arrays, *i.e.*, $\sum_i d^i_{v_r}$. *Lines* 20-22 take care of the exception that one or more V-SWs cannot be embedded in the SNT. The link mapping for each VL is handled in *Lines* 23-33. Here, when there are multiple feasible substrate paths, *Line* 26 randomly selects one and tries to use it for the link mapping. This is also for realizing the dispersed mapping. The time complexity of *Algorithm* 5 is $O(|V_r| \cdot |V_b| \cdot \mathcal{V}_m + |E_r| \cdot |P|)$.

Algorithm 5: Determining VNE scheme with dispersed mapping

```
1  assign a counter $c_{v_b} = 0$ to each Big-Switch $v_b \in V_b$;
2  for each tuple $(v_r, v_b) \in \tilde{\mathcal{M}}_v$ do
3      for each required flow table array $i$ in $v_r$ do
4          for each S-SW $v_s$ in Big-Switch $v_b$ do
5              if embedding of $v_r$ on $v_b$ can take table resources in $v_s$ then
6                  $c_{v_b} = c_{v_b} + 1$;
7              end
8          end
9      end
10 end
11 for each V-SW $v_r \in V_r$ in descending order of total entry depth do
12     $flag = 0$, get all the feasible Big-Switches for $v_r$ from $\tilde{\mathcal{M}}_v$ and store them in $V_b^t$;
13     for each $v_b \in V_b^t$ in ascending order of $c_{v_b}$ do
14         try to embed $v_r$ on the S-SWs in $v_b$;
15         if the embedding is successful then
16             $flag = 1$, update $G_s$ and $G_b$, break;
17         end
18     end
19 end
20 if $flag = 0$ then
21     return(FAILURE);
22 end
23 for each VL $e_r \in E_r$ do
24     $flag = 0$, get all the feasible link mappings for $e_r$ from $\mathcal{M}_e$ and store them in $P^t$;
25     while $P^t \neq \emptyset$ do
26         select a substrate path $p \in P^t$ randomly;
27         $P^t = P^t \setminus p$;
28         if $p$ has enough bandwidth to carry $e_r$ then
29             embed $e_r$ onto $p$ and update $G_s$ and $G_b$;
30             $flag = 1$, break;
31         end
32     end
33 end
34 if $flag = 0$ then
35     return(FAILURE);
36 else
37     return(SUCCESS);
38 end
```

VI. Performance Evaluation

In this section, we first evaluate the performance of our proposed algorithms with extensive numerical simulations. Specifically, we consider two scenarios: 1) the small-scale one time operation and 2) the large-scale dynamic operations. Due to the complexity of the ILP, the small-scale scenario is used to analyze the ILP's sensitivity to parameter in the optimization objective (*i.e.*, α in Eq. (3)) and to compare its performance with that of the heuristic (D3L-VNE). In the large-scale scenario, we consider the dynamic VNT requests that come and leave on-the-fly in relatively large SNTs to further investigate the performance of D3L-VNE with respect to two benchmark algorithms. In the simulations, the topologies of the SNT and VNTs are all randomly generated with the GT-ITM tool [43]. Each data point is obtained by averaging the results from 20 independent simulations.

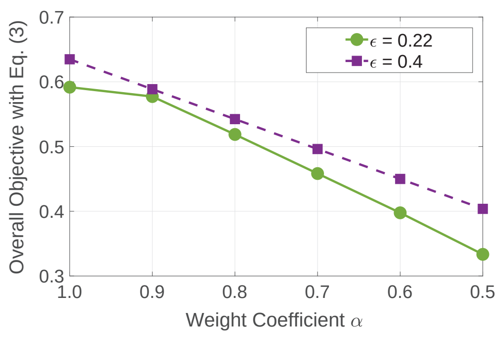


Fig. 5. Sensitivity analysis of the ILP.

A. *Small-Scale One Time Operation*

In this scenario, we use an SNT that includes 14 S-SWs (*i.e.*, $|V_s| = 14$) whose connectivity ε is within $\{0.22, 0.3, 0.4\}$. For the sub-tables on each S-SW, their table sizes and entry depths are randomly selected from $\{32, 48, 64, 128\}$ bits and within $[50, 100]$, respectively. The available bandwidth on each SL is uniformly distributed within $[50, 100]$ units. Meanwhile, the size of a VNT is fixed as $|V_r| = 6$ and the connectivity of its V-SWs is 1. We assume that each V-SW requires $[2, 3]$ flow table arrays, while the table size and entry depth of each array are randomly selected within $\{16, 20, 32, 48, 64, 96, 128\}$ bits and $[5, 20]$, respectively. The bandwidth requirement of each VL is uniformly distributed within $[5, 10]$ units.

1) Sensitivity Analysis of ILP: We first conduct simulations with different settings of the weight coefficient α to analyze the sensitivity of the ILP. Here, we fix $|V_s| = 14$ and $|V_r| = 6$ and consider two connectivity settings for the SNT, *i.e.*, $\varepsilon = \{0.22, 0.4\}$. Fig. 5 shows the simulation results on the overall optimization objective calculated with Eq. (3). It can be seen that in both scenarios, the overall objective decreases with α. This suggests that the value of μ_t (*i.e.*, the normalized memory fragmentation in the SNT) in the first term of Eq. (3) is actually larger and thus contributes more to the overall objective. Therefore, the results in Fig. 5 confirm that the optimization objective of the ILP is properly designed, and we will set $\alpha = 0.9$ in the subsequent simulations.

2) Comparison between ILP and D3L-VNE: Then, we compare the performance of the ILP and D3L-VNE. Here, we consider the different connectivity of substrate network, and Table II summarizes the results. The overall objective from D3L-VNE is close to that from the ILP, while the running time of D3L-VNE is much shorter than that of the ILP. The results verify the effectiveness of our proposed algorithm.

B. Large-Scale Dynamic Operations

In this scenario, we consider two large-scale SNTs with 50 S-SWs whose connectivity ε is 0.2 and 0.3 (*i.e.*, there are 238 and 361 SLs, respectively). For the sub-tables on each S-SW, their table sizes and entry depths are randomly selected from $\{32, 48, 64, 128\}$ bits and within $[50, 100]$, respectively. The available bandwidth on each SL is uniformly distributed within $[50, 100]$ units. On the other hand, the size of each VNT is $|V_r| \in [2, 6]$ and we set the connectivity as 0.5. Each V-SW requires $[2, 3]$ flow table arrays whose entry depths are randomly selected within $[10, 20]$. To mimic the table size distribution in a real network, we follow the procedure in [19, 39] to analyze a program for L2/L3 switches, which is derived from the open-source P4 program switch.p4 [44], and determine that the table sizes of the flow table arrays should be selected from $\{32, 42, 60, 108, 128\}$ bits with probabilities of $\{0.21, 0.04, 0.66, 0.01, 0.08\}$, respectively. The bandwidth requirement of each VL is within $[10, 20]$ units. The dynamic VNT requests follow the Poisson process that has an average arrival rate of λ VNTs per time-unit and the average life-time of each VNT as $\frac{1}{\mu}$ time-units, *i.e.*, their load is $\frac{\lambda}{\mu}$ in Erlangs.

In addition to D3L-VNE, the simulations consider two benchmarks as follows.

- *Normal two-layer VNE (N2L-VNE)*: this algorithm does not consider Big-Switches, and it applies the normal two-layer VNE to embed the V-SWs in each VNT onto the S-SWs with one-to-one mapping.
- *First-fit based three-layer VNE (FF3L-VNE)*: this algorithm considers Big-Switches in the same manner as D3L-VNE, but it utilizes the first-fit scenario to determine the VNE schemes. In other words, FF3L-VNE leverages *Algorithms* 1-4 in D3L-VNE, but replaces *Algorithm* 5 with a first-fit based mapping selection scenario.

The simulations compare the algorithms in terms of the following performance metrics.

- *Acceptance ratio*: the ratio of accepted VNTs to total arrived ones in each simulation.
- *Average bandwidth usage*: the average bandwidth usage on each SL over the time of each simulation.
- *Average table usage*: the average usage of the sub-tables in the S-SWs over the time of each simulation[3].
- *Average table waste*: the average table waste due to memory fragmentation on the sub-tables in the S-SWs over the time of each simulation.

[3]Here, an entry in a sub-table is either fully used or not used at all, and thus this average table usage includes the memory fragmentation.

Fig. 6 shows the simulation results with the SNT that has $|V_s| = 50$ and $\varepsilon = 0.2$. In Fig. 6(a), we can see that among the algorithms, D3L-VNE provides the highest acceptance ratio, which confirms its effectiveness. As expected, the acceptance ratio from N2L-VNE is the lowest and much worse than those from D3L-VNE and FF3L-VNE since it does not consider Big-Switches. The fact of D3L-VNE accepting more VNTs than FF3L-VNE confirms that the dispersed mapping in *Algorithm* 5 helps to accommodate more VNTs. The average bandwidth and table usages in Figs. 6(b) and 6(c), respectively, further explain the superiority of D3L-VNE, *i.e.*, it can organize the VNTs in the SNT in the best way such that the most bandwidth and table resources can be utilized in the service provisioning. Again, since N2L-VNE does not consider Big-Switches or memory fragmentation, it causes the highest table waste as shown in Fig. 6(d). Meanwhile, it is interesting to notice that D3L-VNE and FF3L-VNE perform similarly in terms of the average table waste. This is because both of them try to minimize the memory fragmentation during node mapping. The results in Fig. 7, which are from the simulations using the SNT with $|V_s| = 50$ and $\varepsilon = 0.3$, exhibit the similar trends as those in Fig. 6.

VII. System Implementation and Experimental Demonstrations

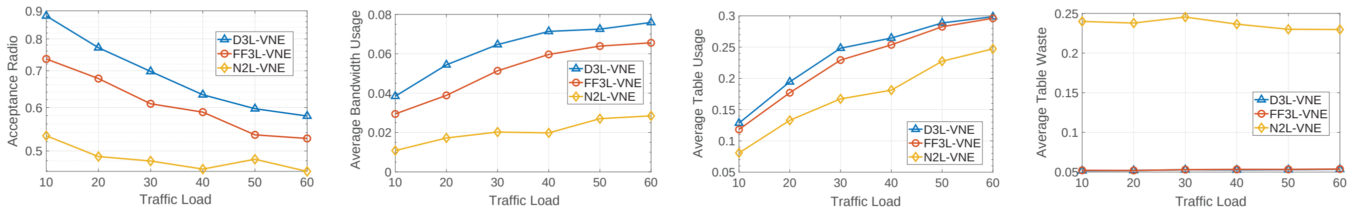
A. System Implementation

Previously, in [25], we have designed and implemented a network hypervisor, namely, TPVX, which can realize the table resource virtualization and network slicing for PDP. Specifically, TPVX can create and manage VNTs over an SNT that is built with POF-based S-SWs, and when mapping the flow tables in V-SWs to S-SWs, TPVX considers their table sizes and the pre-formatted sub-tables in the S-SWs to improve table resource utilization and avoid memory fragmentation. However, TPVX should still be improved from two perspectives, which are 1) an effective three-layer VNE algorithm (*e.g.*, D3L-VNE) should be implemented in it to ensure that cost-effective VNE schemes can be calculated for various VNT requests, and 2) the overheads on table resource utilization and packet processing due to the network virtualization should be minimized. The first perspective can be easily understood while the explanations on the second one are as follows.

Note that, to realize network virtualization, a network hypervisor needs to tell the S-SWs how to distinguish packets belonging to different VNTs, and this is done by the hypervisor installing control flow tables in the S-SWs in addition to the flow tables from the VNTs (*i.e.*, the tenant flow tables). For example, control flow tables need to be installed in intermediate S-SWs to route the packets in different VNTs correctly. Here, the intermediate S-SWs refer to the S-SWs that are intermediate nodes on the substrate paths carrying VLs. As shown in Fig. 8(a), *VL a-b* in *VNT* 1 is embedded on substrate paths 1-2-3. Hence, *S-SW* 2 is an intermediate S-SW, and control flow tables need to be installed on it for processing

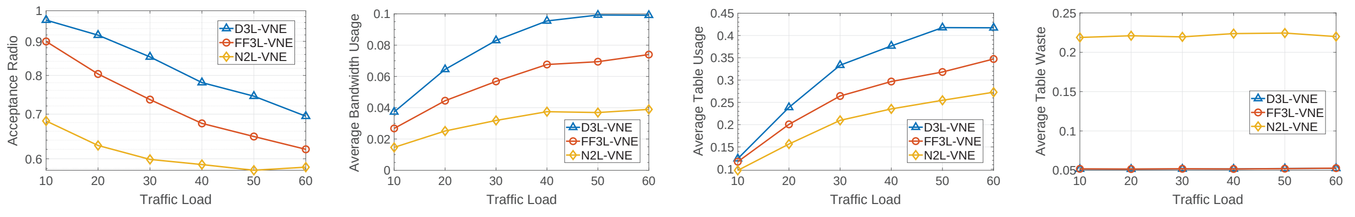
TABLE II
PERFORMANCE COMPARISONS BETWEEN ILP AND D3L-VNE

SNT		VNT		ILP				D3L-VNE			
$\lvert V_s \rvert$	ε	$\lvert V_r \rvert$	ε	First Term	Second Term	Overall Objective	Running Time (s)	First Term	Second Term	Overall Objective	Running Time (s)
14	0.22	6	1	0.56	0.01	0.57	35.5	0.67	0.02	0.69	0.06
14	0.3	6	1	0.51	0.07	0.58	47.5	0.54	0.05	0.59	0.09
14	0.4	6	1	0.49	0.11	0.60	289.3	0.57	0.07	0.64	0.13



(a) Acceptance ratio (b) Average bandwidth usage (c) Average table usage (d) Average table waste

Fig. 6. Large-scale dynamic simulation results (SNT: $|V_s| = 50$, $\varepsilon = 0.2$).



(a) Acceptance ratio (b) Average bandwidth usage (c) Average table usage (d) Average table waste

Fig. 7. Large-scale dynamic simulation results (SNT: $|V_s| = 50$, $\varepsilon = 0.3$).

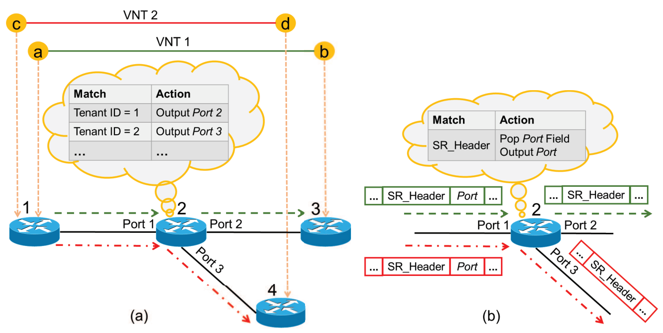


Fig. 8. Examples on control flow tables in an intermediate S-SW for (a) original TPVX, and (b) extended TPVX that uses SR.

packets going through *VL* a-b, even though *S-SW* 2 does not carry any V-SW. On an intermediate S-SW, we normally need to install one control flow table for each VL that uses it, to match to the tenant ID encoded in packets going through the VL and route the packets correctly. For instance, in Fig. 8(a), we also have VL c-d in *VNT* 2 embedded on substrate path 1-2-4, and thus two control flow tables needs to be installed in *S-SW* 2, each of which matches to the tenant ID of a VL and route the corresponding packets correctly. This is also how the previous version of TPVX in [25] was implemented.

Nevertheless, the control flow tables lead to additional overheads, since each intermediate S-SW needs to spend table resources on them and matching packets to them would bring in additional operation complexity and thus latency. This explains why TPVX should be improved from the second perspective. Therefore, we leverage POF-based source routing (SR) [27, 45] to extend TPVX for using much fewer control flow tables. Specifically, as described in [27], POF-based SR inserts an *SR_Header* in each packet to explain its forwarding path to each intermediate switch (*i.e.*, which output port the packet should take on a particular switch). Hence, each intermediate S-SW only needs a fixed number of shared control flow tables to process all the packets going through the VLs that use it. For each packet, the shared control flow tables pop the first *Port* field in its *SR_Header*, parse the designated output port, and then forward the packet accordingly [27]. As shown in Fig. 8(b), the TPVX with this extension can minimize the overheads due to control flow tables in intermediate S-SWs.

The overall system architecture of the improved TPVX is illustrated in Fig. 9, where we divide the system into the service, control, orchestration, virtualization and infrastructure layers according to their functionalities. The service layer provides the API to the service providers (SPs), through which they can submit their VNT requests to the infrastructure provider (InP). After a VNT request has been provisioned successfully, the InP creates a tenant controller in the control layer for the VNT and passes it to the VNT's SP, and then the SP can use it as an SDN controller to manage the tenant flow tables in the VNT. The orchestrator in the orchestration layer works as the "brain" of TPVX to calculate the VNE schemes for the

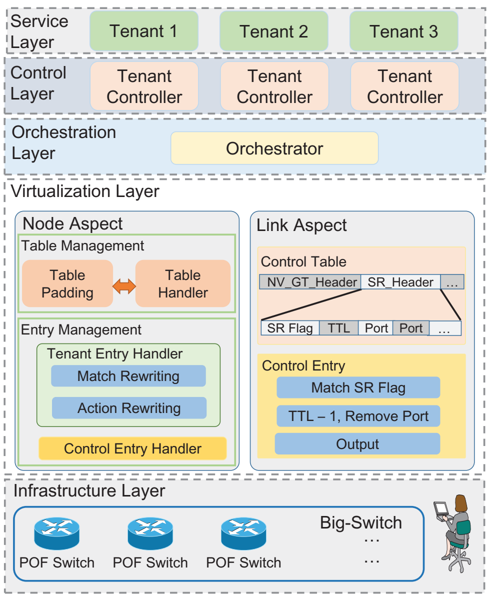


Fig. 9. Overall system architecture of improved TPVX.

VNT requests based on the latest SNT state, and we implement our D3L-VNE algorithm in it. The virtualization layer bridges the control communications between the tenant controllers and the S-SWs in the infrastructure layer, and according to the VNE schemes, it installs both control flow tables and translated tenant flow tables in the SNT. Specifically, as plotted in Fig. 9, this layer handles the network slicing from both the node and link aspects, where the POF-based SR is considered in the link aspect. The infrastructure layer contains the SNT, where POF-based switches can form Big-Switches. Since the design and implementation of TPVX have already been elaborated in [25], we save the discussion on them here.

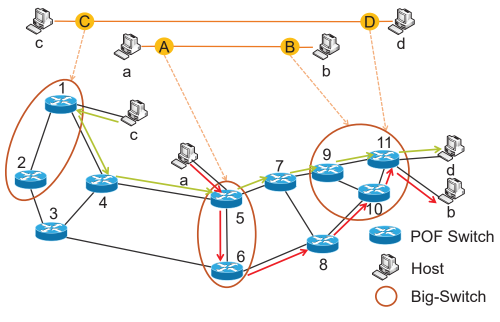


Fig. 10. Experimental setup.

B. Experimental Setup

The SNT in our experimental setup is shown in Fig. 10, which consists of 11 stand-alone POF-based S-SWs and 4 hosts. Each S-SW is based on our homemade software POF switch [46] running on a high-performance Linux server. All the network connections in the SNT are based on 1GbE. The TPVX system in Fig. 9 is implemented on a Linux server too. Note that, even though the simulations in Section VI have already evaluated D3L-VNE from a few aspects, the additional latency induced by the Big-Switches cannot be evaluated with simulations. Therefore, our experiments will first measure the latency in various situations and then send high-definition (HD) video streams through the VNTs built over Big-Switches to further verify the practicalness of our proposal.

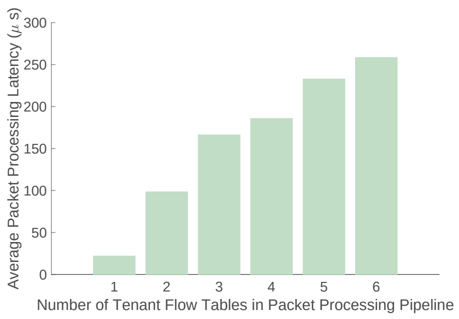


Fig. 11. The worst-case latency in a Big-Switch that includes three V-SWs.

C. Latency Measurements

We first consider a Big-Switch including three S-SWs (*i.e.*, *S-SWs* 9-11 in Fig. 10) and embed a V-SW on it. Then, we change the packet processing pipeline in the V-SW from containing 1 to 6 tenant flow tables, and measure the worst-case latency for the packet processing in the Big-Switch. The results in Fig. 11 show that as expected, the average latency increases with the number of tenant flow tables in the pipeline, but the latencies are in the order of μs and thus very short. Next, we consider the multi-hop scenario and map two VNTs in the SNT as shown in Fig. 10. Specifically, each VNT only consists of one VL, and the VL: a-A-B-b is embedded as a-(5, 6)-8-(10, 9, 11)-b (*Path* 1) while the VL: c-C-D-d is embedded as c-(1, 2)-4-(5, 6)-7-(9,10,11)-b (*Path* 2), where the S-SWs in "()" form a Big-Switch. We still fix the total number of tenant flow tables over each VL as 6, consider different distributions of the flow tables over each path, measure the total packet processing latency with the standard scheme in [47], and plot the results in Fig. 12. It can be seen that the longest latencies in the worst-case scenarios are still in the order of μs. Since our proposal actually sacrifices the memory access latency in exchange of the efficiency of memory utilization, we expect it to be useful in the environment where the network latency can be controlled well (*e.g.*, in a datacenter network).

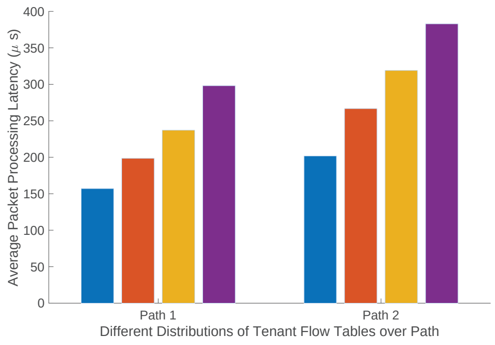
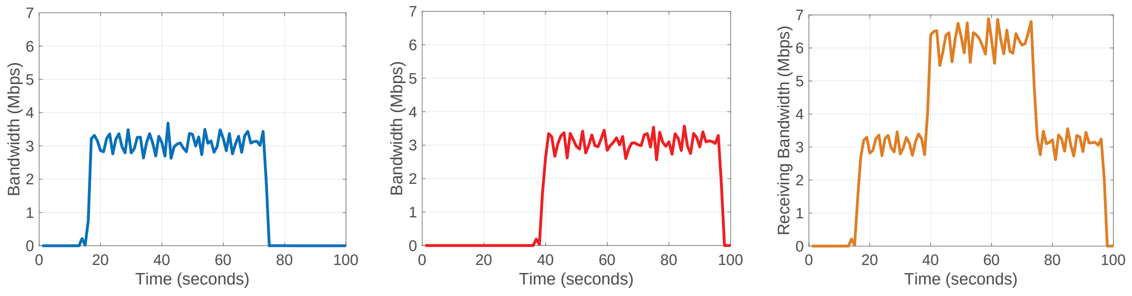


Fig. 12. Packet processing latency in multi-hop scenarios.

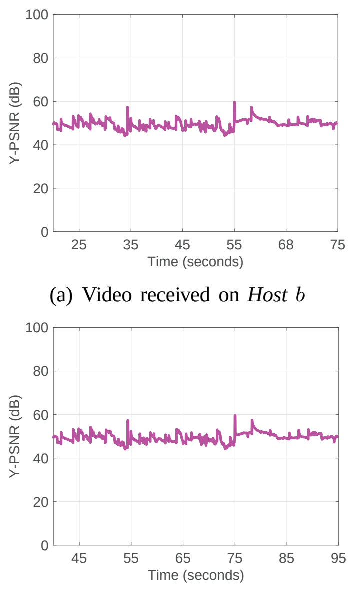


(a) on *Host b* (b) on *Host d* (c) on *S-SW* 9

Fig. 13. Receiving bandwidth for HD video streaming.

D. HD Video Streaming in VNTs Embedded on Big-Switches

To further verify the practicalness of our proposal, we keep the VNTs embedded in the previous subsection and stream HD videos over them. The experiment measures the receiving bandwidths and quality of video playbacks to confirm that our three-layer VNE would not cause noticeable service degradation on upper-layer applications, *i.e.*, our proposal maintains good isolation between VNTs that share the same S-SWs. We still use the embedding schemes in Fig. 10, and thus the two VLs share the Big-Switch (9,10,11). Then, two HD video streams are sent from a to b and c to d at different time in the experiment. Fig. 13 shows the experimental results on the receiving bandwidth measured at different locations. The results in Figs. 13(a) and 13(b) verify that with the three-layer VNE, the video streams on different VLs get delivered correctly in the SNT. Meanwhile, we can see that the forwarding of the packets over different VLs gets isolated well. This is because the insertion of the video stream to *Host* d at around $t = 40$ seconds does not cause any noticeable bandwidth reduction on that to *Host* b, even though the results in Fig. 13(c) indicate that the two streams share *S-SW* 9. Finally, we measure the luminance component's peak signal-to-noise ratio (Y-PSNR) of the video playbacks on *Hosts* b and d, to confirm the quality of the video streaming. As shown in Fig. 14, the Y-PSNR of the video playbacks stay at relatively high values throughout the streaming of the two videos.



(b) Video received on *Host d*

Fig. 14. Y-PSNR of video playbacks.

VIII. Conclusion

This paper studied the table resource virtualization and network slicing in PDP-based SNTs. We first leveraged the idea of "Big-Switch" to design an effective table resource virtualization scheme to regulate the flow tables in the S-SWs in a more organized way to minimize memory fragmentation. Then, we addressed the network slicing based on the virtualization scheme by formulating a three-layer VNE problem. An ILP model and a time-efficient heuristic, namely, D3L-VNE, were designed to solve the problem. Simulation results confirmed that D3L-VNE can provide near-optimal solutions to small-scale problems, while for dynamic operations in large-scale SNTs, it outperforms two benchmark algorithms. Finally, we implemented D3L-VNE in TPVX, which is a POF-enabled network hypervisor, and improved the performance of TPVX by introducing source routing. Experimental demonstrations of the new TPVX showed that the additional latency caused by the three-layer VNE can be maintained well and thus would not degrade the network services in VNTs.

Acknowledgments

This work was supported by the NSFC projects 61871357 and 61701472, CAS key project (QYZDY-SSW-JSC003), and NGBWMCN key project (2017ZX03001019-004).

References

[1] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[2] 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.

[3] 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.

[4] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[5] 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.

[6] Z. Zhu *et al.*, “OpenFlow-assisted online defragmentation in single-/multi-domain software-defined elastic optical networks,” *J. Opt. Commun. Netw.*, vol. 7, pp. A7–A15, Jan. 2015.

[7] 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.

[8] 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.

[9] L. Gong and Z. Zhu, “Virtual optical network embedding (VONE) over elastic optical networks,” *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[10] 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.

[11] K. Han *et al.*, “Application-driven end-to-end slicing: When wireless network virtualization orchestrates with NFV-based mobile edge computing,” *IEEE Access*, vol. 6, pp. 26 567–26 577, 2018.

[12] R. Jain and S. Paul, “Network virtualization and software defined networking for cloud computing: a survey,” *IEEE Commun. Mag.*, vol. 51, pp. 24–31, Nov. 2013.

[13] 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.

[14] 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.

[15] S. Li *et al.*, “Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability,” *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[16] P. Bosshart *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[17] 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.

[18] H. Song, “Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane,” in *Proc. of ACM HotSDN 2013*, pp. 127–132, Aug. 2013.

[19] P. Bosshart *et al.*, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 99–110, Oct. 2013.

[20] The world’s fastest & most programmable networks. [Online]. Available: https://www.barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/

[21] S. Chole *et al.*, “dMRT: Disaggregated programmable switching,” in *Proc. of SIGCOMM 2017*, pp. 1–14, Aug. 2017.

[22] D. Kreutz *et al.*, “Software-defined networking: A comprehensive survey,” *Proc. IEEE*, vol. 103, pp. 14–76, Jan. 2015.

[23] D. Kim *et al.*, “Generic external memory for switch data planes,” in *Proc. of ACM HotNets 2018*, pp. 1–7, Nov. 2018.

[24] 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.

[25] Y. Xue *et al.*, “Virtualization of table resources in programmable data plane with global consideration,” in *Proc. of GLOBECOM 2018*, pp. 1–6, Dec. 2018.

[26] S. Zhao, D. Li, K. Han, and Z. Zhu, “Proactive and hitless vSDN reconfiguration to balance substrate TCAM utilization: From algorithm design to system prototype,” *IEEE Trans. Netw. Serv. Manag.*, vol. 16, pp. 647–660, Jun. 2019.

[27] 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.

[28] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, “Cacheflow: Dependency-aware rule-caching for software-defined networks,” in *Proc. of SOSR 2016*, pp. 1–12, Mar. 2016.

[29] A. Mimidis-Kentis *et al.*, “A novel algorithm for flow-rule placement in SDN switches,” in *Proc. of NetSoft 2018*, pp. 1–9, Jun. 2018.

[30] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, “LegoOS: A disseminated, distributed OS for hardware resource disaggregation,” in *Proc. of OSDI 2018*, pp. 69–87, Oct. 2018.

[31] S. Shirali-Shahreza and Y. Ganjali, “Delayed installation and expedited eviction: An alternative approach to reduce flow table occupancy in SDN switches,” *IEEE/ACM Trans. Netw.*, vol. 26, pp. 1547–1561, Aug. 2018.

[32] G. Zhao *et al.*, “Joint optimization of flow table and group table for default paths in SDNs,” *IEEE/ACM Trans. Netw.*, vol. 26, pp. 1837–1850, Aug. 2018.

[33] W. Li, X. Li, and H. Li, “MEET-IP: Memory and energy efficient TCAM-based IP lookup,” in *Proc. of ICCCN 2017*, pp. 1–8, Jul. 2017.

[34] D. Hancock and J. Merwe, “HyPer4: Using P4 to virtualize the programmable data plane,” in *Proc. of CoNEXT 2016*, pp. 35–49, May 2016.

[35] J. Zhang *et al.*, “Dynamic virtual network embedding over multilayer optical networks,” *J. Opt. Commun. Netw.*, vol. 7, pp. 918–927, Sept. 2015.

[36] S. Chowdhury *et al.*, “MULE: Multi-layer virtual network embedding,” in *Proc. of CNSM 2017*, pp. 1–9, Nov. 2017.

[37] G. Long, 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] Y. Wang, P. Lu, W. Lu, and Z. Zhu, “Cost-efficient virtual network function graph (vNFG) provisioning in multidomain elastic optical networks,” *J. Lightw. Technol.*, vol. 35, pp. 2712–2723, Jul. 2017.

[43] E. Zegura, K. Calvert, and S. Bhattacharjee, “How to model an internetwork,” in *Proc. INFOCOM 1996*, pp. 594–602, Mar. 1996.

[44] switch.p4. [Online]. Available: https://github.com/p4lang/switch/tree/master/p4src

[45] 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.

[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.

[47] S. Bradner and J. McQuaid, “Benchmarking methodology for network interconnect devices,” *RFC 2544*, Mar. 1999. [Online]. Available: https://tools.ietf.org/html/rfc2544