On Parallel and Hitless vSDN Reconfiguration

Sicheng Zhao, Xing Wu, and Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—The symbiosis of network virtualization and software-defined networking (SDN) enables an infrastructure provider (InP) to build various virtual software defined networks (vSDNs) over a shared substrate network (SNT). To handle a dynamic network environment, the InP may need to reconfigure the mapping schemes of vSDNs for a variety of reasons. Although previous studies have addressed how to calculate the new virtual network embedding (VNE) schemes for vSDN reconfiguration under different objectives, the transition to migrate vSDNs from their original VNE schemes to new ones is still under-explored.**

Hence, this paper studies how to realize parallel and hitless vSDN reconfiguration, by leveraging the "make-before-break" scenario. We come up with a generic solution to optimize the transition to remap vSDNs to new VNE schemes, such that the remappings can be done in the parallel, hitless and resource-efficient manner, as long as the new VNE schemes are feasible. More specifically, our proposal is the multi-stage parallel vSDN reconfiguration based on maximal connected reconfigurable subgraph (MCRSG). To ensure the efficiency of our proposal, we formulate the optimization for selecting MCRSGs to reconfigure in each stage, and prove the $\mathcal{NP}$-hardness of the problem. Then, we design an approximation algorithm based on Lagrangian relaxation to solve it time-efficiently. Extensive simulations verify that the proposed algorithm can obtain near-optimal solutions quickly. In addition to the algorithmic study, we also realize our multi-stage parallel vSDN reconfiguration in a practical NVH system, and demonstrate its performance in a real network testbed. Our experimental study identifies in what condition losing of packets during remapping would be inevitable, studies the tradeoff between reconfiguration latency and packet loss rate, and reveal an empirical method to adjust key parameters of our NVH system, for adapting to various network environments.

***Index Terms*—Network virtualization, Software-defined networking (SDN), Parallel and hitless vSDN reconfiguration, Lagrangian relaxation, Approximation algorithm.**

I. Introduction

AFTER many years of development, the Internet is now facing numerous challenges, most of which cannot be addressed with traditional network control and management (NC&M) mechanisms [1, 2]. Hence, to prevent the ossification of Internet infrastructure, people turned to develop new network technologies. Among them, the two most-referred-to ones are network virtualization [3] and software-defined networking (SDN) [4]. Network virtualization transforms the conventional ISPs into infrastructure providers (InPs) and service providers (SPs). An InP owns one substrate network (SNT), and it can slice the SNT into logically-isolated virtual networks (VNTs) with virtual network embedding (VNE) [5, 6], and lease them to the SPs to satisfy their demands [7–9]. SDN takes the features of NC&M out of each network element to build a logically-centralized control plane, and makes network elements operate according to the instructions from the control plane, for ensuring enhanced programmability and application-awareness in various networks [10–12].

Network virtualization and SDN are mutually beneficial [13–15]. This makes an InP provision software-defined VNTs, *i.e.*, virtual software-defined networks (vSDNs), to the SPs that subscribe to its service. To build vSDNs over its SNT, the InP needs a VNE algorithm [16] and a network virtualization hypervisor (NVH) [17]. The VNE algorithm determines how to allocate the memory resources on substrate switches (S-SWs) to compose virtual switches (vSWs) (*i.e.*, node mapping), and how to route virtual links (VLs) over substrate links (SLs) and assign bandwidth to them (*i.e.*, link mapping). The memory resources concerned in the node mapping usually refer to the ternary content-addressable memory (TCAM) and static random-access memory (SRAM), which store flow-entries/tables in S-SWs. Note that, due to the power consumption and cost of TCAM and SRAM, their amounts are usually very limited in an S-SW [18, 19]. For instance, the memory resource in a start-of-art commercial SDN-based S-SW can usually only accommodate a few thousands of flow-entries at most [18]. The NVH translates the control communications between S-SWs and virtual controllers (vCs) of vSDNs [17] to implement the node and link mappings.

The algorithmic studies on VNE and the system designs of NVH have already been covered comprehensively in the literature [16, 17]. However, since an SNT usually possesses dynamic network environment, its InP might need to reconfigure the VNE schemes of vSDNs, for a variety of reasons. The first example is for fault tolerance and failure recovery [20]. Since various failures can happen in a production SNT due to random faults, human errors or even malicious attacks, the InP needs to migrate the affected vSDNs to minimize service disruptions. The second example is for service upgrading/downgrading [21] and mobility support [22]. Specifically, an SP may request to add/remove vSWs/VLs in its vSDN, and it may also hope to relocate some of its vSWs/VLs to adapt to the movement of its clients. The last example is for balancing the usages of substrate resources [23]. As TCAM and SRAM are very limited hardware resources in S-SWs [18], the memory provided by them for storing flow-entries/tables could become the bottleneck resources in vSDN slicing. Therefore, without dynamic load-balancing, the memory capacity of an S-SW could be insufficient to accommodate all the active flow-entries/tables from its vSWs during rush hours.

To the best of our knowledge, the existing algorithmic studies on vSDN reconfiguration only addressed how to calculate the new VNE schemes for vSDNs under various objectives, but have not considered the transition to migrate vSDNs from their original VNE schemes to the new ones. We argue that to realize high-performance vSDN reconfiguration in practical

S. Zhao, X. Wu, 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 November 10, 2019.

systems, it is very important for the InP to carefully handle the remapping transition of each vSDN. More specifically, it is desired that the InP handles the transition with parallel and hitless vSDN reconfiguration. Here, "parallel reconfiguration" means to reconfigure multiple vSWs and VLs simultaneously, while "hitless reconfiguration" refers to the one whose resulting packet losses are very few or even zero. The demand for parallel reconfiguration comes from the fact that the InP might need to remap a few vSWs and VLs in a vSDN or even multiple vSDNs within a short period of time (*e.g.*, for failure recovery or load-balancing). Hence, parallelizing the remapping operations would bring in benefits such as short reconfiguration latency, low operational complexity, and less disruption to the services of vSDNs. The rationale behind the demand for hitless reconfiguration is also obvious, *i.e.*, vSDNs can carry live traffic and thus any service disruptions on their vSWs and VLs would affect their quality-of-service (QoS).

However, since there are complex dependencies among vSWs, VLs, S-SWs and SLs, it is never easy to achieve parallel vSDN reconfiguration. For instance, remapping a vSW will change the link mapping schemes of all the VLs that use it as an end-node, which will in turn affect the resource usages on many SLs and S-SWs[1]. Moreover, if we want the parallel reconfiguration to be hitless, we will need to leverage the "make-before-break" scenario [26]. Specifically, the InP first builds new vSWs and VLs in new locations, then redirects the traffic passing through original vSWs and VLs to use the new ones, and finally removes the original ones from the SNT [27]. Hence, vSWs and VLs can be remapped in the hitless way. Nevertheless, make-before-break further complicates the dependencies among vSWs, VLs, S-SWs and SLs, and makes parallel reconfiguration even more difficult. Last but not least, when parallel vSDN reconfiguration is considered, the make-before-break scenario might consume too many redundant resources because the original vSWs and VLs cannot be torn down before the new ones are up and operational.

In this work, we aim to realize parallel and hitless vSDN reconfiguration. Specifically, we optimize the transition to remap vSDNs, regardless of how or for what purpose the new VNE schemes were computed. In other words, our proposal is so generic that for a given set of vSDNs, it can remap them to arbitrary VNE schemes in the parallel, hitless and resource-efficient manner, as long as the schemes are feasible. We consider both algorithm design and system implementation.

As for the algorithm design, we first analyze the problem of parallel and hitless vSDN reconfiguration, and point out the issues of one-stage parallel reconfiguration. Then, to address the issues, we propose a multi-stage parallel vSDN reconfiguration scheme based on the maximal connected reconfigurable subgraph (MCRSG). Here, an MCRSG refers to a maximal structure in a vSDN, which is built with connected vSWs and VLs concerned in a reconfiguration. Next, we consider how to select MCRSGs to reconfigure in each stage, and prove its $\mathcal{NP}$-hardness. Hence, we develop an approximation algorithm based on Lagrangian relaxation, for improved time-efficiency. Simulation results confirm that for problems with relatively large sizes, our approximation algorithm can obtain near-optimal solutions within reasonable numbers of iterations. We also conduct extensive simulations to evaluate the performance of our proposal in dynamic network environments.

As for the system implementation, we expand our NVH system designed in [27] to facilitate parallel vSDN reconfiguration, and experimentally demonstrate it in a network testbed that consists of six S-SWs. Experimental results indicate that compared with sequential reconfiguration, our parallel vSDN reconfiguration effectively reduces the reconfiguration latency. We also conduct experiments to identify in what condition losing of packets during remapping would be inevitable, study the tradeoff between reconfiguration latency and packet loss rate, and reveal an empirical method to adjust key parameters of our NVH system, for adapting to various network environments.

The major contributions of our work can be summarized as:

- To the best of our knowledge, our proposal is the first work to realize parallel and hitless vSDN reconfiguration in the resource-efficient manner.
- We propose a novel multi-stage parallel vSDN reconfiguration scheme based on MCRSG to address the issues of one-stage parallel vSDN reconfiguration.
- To ensure the efficiency of our multi-stage parallel vSDN reconfiguration, we design the optimization to select MCRSGs in each stage, and prove its $\mathcal{NP}$-hardness.
- We design an approximation algorithm based on Lagrangian relaxation to solve the problem of MCRSG selection time-efficiently, and simulation results verify that the algorithm can obtain near-optimal solutions quickly.
- We realize the multi-stage parallel vSDN reconfiguration in an NVH system for experimental demonstrations.

The rest of the paper is organized as follows. We survey the related work briefly in Section II. Section III describes the problem of parallel and hitless vSDN reconfiguration. Our proposed multi-stage parallel vSDN reconfiguration scheme is introduced in Section IV. Since the selection of indeterministic MCRSGs to reconfigure, which is a subproblem in our proposal, is $\mathcal{NP}$-hard, we leverage Lagrangian relaxation to solve it time-efficiently in Section V. Section VI discusses the numerical simulations for performance evaluation, while the experimental demonstrations and studies are presented in Section VII. Finally, we summarize the paper in Section VIII.

II. Related Work

For network virtualization, VNE is the fundamental problem and it has been studied intensively with various networks as the SNT [3, 5–8]. Compared with the VNE in packet networks, the VNE in optical networks (*e.g.*, the fixed-/flexible-grid wavelength-division multiplexing networks [28–30]) could be more complex since the link mapping involves routing and spectrum assignment, which itself is an $\mathcal{NP}$-hard problem. The studies in [31–33] proposed algorithms based on decompositions and Lagrangian relaxation to solve the VNE problem. Although the mathematical methods are similar, our problem is not about VNE and thus it is fundamentally different from

[1]Each VL consumes bandwidth resources on each of the SLs along the substrate path that it is embedded on, while in addition to the S-SWs that carry its two end vSWs, the VL also uses memory resources in all the intermediate S-SWs along its substrate path [24, 25].

those studied in [31–33]. For a comprehensive survey on the existing VNE algorithms, one is suggested to refer to [16].

The symbiosis of network virtualization and SDN leads to the creation of vSDNs [13], which brings new challenges to VNE algorithm design. In [24], the authors pointed out that in addition to the S-SWs carrying its end vSWs, each VL in a vSDN also consumes memory resources in all the intermediate S-SWs along the substrate path that it is embedded on. Considering the dependency between the control and data planes of vSDNs, the studies in [25, 34] tackled the problems of correlated data and control plane embedding. More recently, to address the memory fragmentation in S-SWs with programmable data plane (PDP), Xue *et al.* [35] leveraged "Big-Switches" to design a three-layer VNE scheme, which could realize resource-efficient vSDN creation. Nevertheless, all these studies are orthogonal to this work, because they did not investigate the transition to realize vSDN reconfiguration.

With VNE algorithms, an InP can use an NVH system to create vSDNs over its SNT. Due to the benefits of this approach, numerous studies have been dedicated to developing effective and powerful NVH systems [17]. Initially, the development efforts were focused on designing NVH systems to support the creation of OpenFlow-based vSDNs, *e.g.*, the FlowVisor [36] and OpenVirteX [37]. Later on, the designs tried to make NVH compatible with PDP. Here, PDP refers to the packet processing and forwarding elements that have powerful programmable features to remove the dependence on existing protocols, such as P4 [38] and protocol-independent forwarding (POF) [39]. HyPer4 [40] was designed to slice vSDNs over the S-SWs that are based on P4-based PDP. Meanwhile, by leveraging famous open-source projects such as OpenVirtex and ONOS [41], people have also come up with various NVH systems that support POF [14, 42–44]. However, none of the aforementioned NVHs have been demonstrated to support parallel and hitless vSDN reconfiguration. This is because without a sophisticated algorithm to sort out the dependencies among vSWs, VLs, S-SWs and SLs, NVH can hardly make the remapping of vSWs and VLs satisfy the two conflicting demands (*i.e.*, parallel and hitless) simultaneously.

Previously, the studies in [45–47] considered how to migrate virtual machines (VMs) in SDNs, but these approaches did not address the vSDN reconfiguration that remaps VLs and vSWs simultaneously. Jiao *et al.* [48] tackled the online resource allocation in multi-tier distributed cloud networks, to jointly optimize the reconfiguration cost and the operational cost of the target configuration. However, the background was still not vSDN reconfiguration. The authors of [49] studied how to calculate the new VNE schemes for realizing VNT reconfiguration in a dynamic network environment. The studies in [50, 51] tried to achieve vSDN reconfiguration with the help of vSDNs' vCs. However, since a vC cannot have a global view of the SNT, it would have difficulty determining when and how to reconfigure its vSDN. The system developed in [52] could realize SDN migration, but it did not put network virtualization into consideration. Lo *et al.* [53] considered vSDN reconfiguration based on a network virtualization scenario that is different from ours, *i.e.*, instead of being created by slicing S-SWs, the vSWs were all software-based ones that needed to be instantiated in VMs. In our previous work [23, 27], we studied how to calculate new VNE schemes for vSDNs such that the memory resource usages in an SNT can be re-balanced by leveraging sequential vSDN reconfiguration operations based on make-before-break. Nevertheless, the solution was neither generic nor parallel. Therefore, to the best of our knowledge, the problem of parallel and hitless vSDN reconfiguration has not been explored yet, especially when the algorithm design needs to be considered together with system implementation.

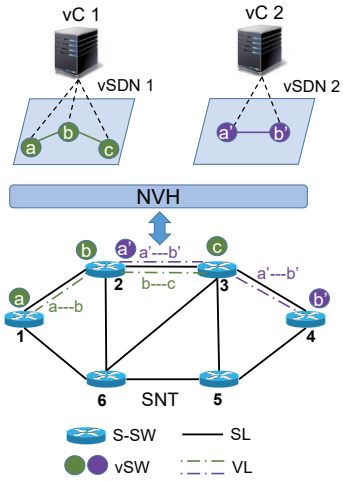

Fig. 1. Network system for vSDN creation and reconfiguration.

III. Problem Description

In this section, we first introduce the network model of vSDN reconfiguration, then explain how to remap vSWs and VLs with the make-before-break scenario, and finally define the problem of parallel and hitless vSDN reconfiguration. Since several abbreviations are frequently used in this paper, we list them here in Table I for the readers' convenience.

TABLE I
Major Abbreviations

Abbrev.	Full Name	Abbrev.	Full Name
SDN	Software-defined networking	InP	Infrastructure provider
VNE	Virtual network embedding	SNT	Substrate network
vSW	Virtual switch	SP	Service provider
QoS	Quality-of-service	VNT	Virtual network
S-SW	Substrate switch	VL	Virtual link
MtV	Move-to-vacancy	VM	Virtual machine
SRAM	Static random-access memory	SL	Substrate link
MbB	Make-before-break	vC	Virtual controller
ILP	Integer linear programming	DG	Dependency graph
LR	Lagrangian relaxation	RTT	Round-trip-time
vSDN	Virtual software-defined network		
NC&M	Network control and management		
TCAM	Ternary content-addressable memory		
MCRSG	Maximal connected reconfigurable subgraph		
MKP	Multi-dimensional 0-1 knapsack problem		
FPTAS	Fully polynomial-time approximation scheme		

A. Network Model

Fig. 1 shows the architecture of the network system concerned in this work for vSDN creation and reconfiguration. The SNT can be modeled as an undirected graph $G_s(V_s, E_s)$, where V_s and E_s denote the sets of its S-SWs and SLs,

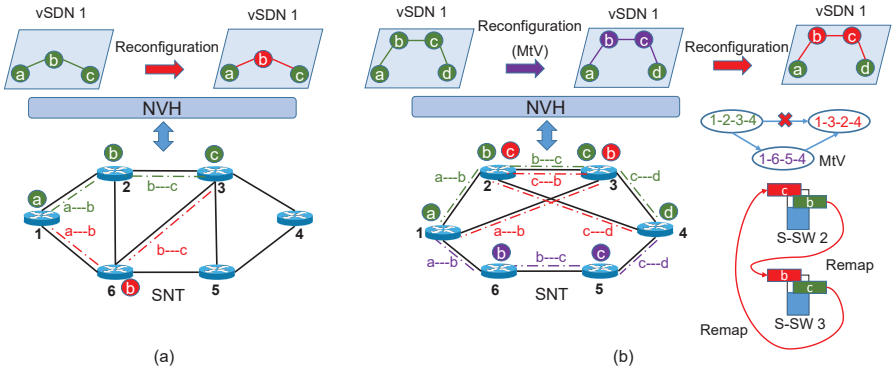

Fig. 2. Examples on hitless vSDN reconfiguration with MbB, a) MbB is feasible, and b) MtV has to be used.

respectively. In this work, we consider two types of substrate resources, *i.e.*, the memory resources on S-SWs to store flow-entries and the bandwidth resources on SLs to carry traffic. The NVH manages the SNT for vSDN creation and reconfiguration. The data plane of a vSDN can also be modeled as an undirected graph $G_i^r(V_i^r, E_i^r)$, where i is its unique index, and V_i^r and E_i^r represent the sets of its vSWs and VLs, respectively. To create a vSDN, the NVH first utilizes a VNE algorithm to calculate the node and link mapping schemes of $G_i^r(V_i^r, E_i^r)$, subject to the resource constraints[2], as follows.

$$\mathcal{M}_i = \begin{cases} \mathcal{M}_N: & V_i^r \rightarrow V_s, \\ \mathcal{M}_L: & E_i^r \rightarrow P_s, \end{cases} \tag{1}$$

where $\mathcal{M}_N$ and $\mathcal{M}_L$ refer to the node and link mappings, respectively, and P_s is the set of pre-calculated substrate paths in the SNT. Then, the NVH implements the VNE scheme $\mathcal{M}_i$, and hands over the vSDN's NC&M to its vC. At this moment, the vSDN is up and operational, and its vC can install, update, and remove flow-entries in the vSWs to forward application traffic. According to $\mathcal{M}_i$, the NVH performs two-way translation of the control messages between the vC and the S-SWs that carry the vSDN's vSWs. As shown in Fig. 1, the S-SWs and SLs in an SNT can be shared by multiple vSDNs, while the NVH enforces proper isolation such that the vSDNs' operations would not interfere with each other.

To invoke a vSDN reconfiguration, the NVH first obtains the new VNE scheme $\mathcal{M}_i'$ of $G_i^r(V_i^r, E_i^r)$, and then remaps the related vSWs and VLs to implement $\mathcal{M}_i'$. Since the generic vSDN reconfiguration is considered in this work, we do not care how or for what purpose $\mathcal{M}_i'$ is calculated, but focus on the transition to migrate $G_i^r(V_i^r, E_i^r)$ from $\mathcal{M}_i$ to $\mathcal{M}_i'$. Moreover, our proposal supports the vSDN reconfiguration that needs to migrate multiple vSDNs simultaneously. In our design, both the vSDN creation and reconfiguration are handled solely by the NVH, *i.e.*, they are completely transparent to the vCs.

B. Hitless Reconfiguration with Make-before-Break

Since the vSDN reconfiguration is performed during when vSDN(s) are in operation, we need to make the transition "hitless", *i.e.*, the resulting packet losses should be very few or even zero. To achieve this, a common practice is to leverage the "make-before-break" scenario (MbB) [27]. Specifically, this means that to reconfigure $G_i^r(V_i^r, E_i^r)$, the InP needs to

1) set up new vSWs and VLs according to $\mathcal{M}_i'$,
2) switch traffic from $\mathcal{M}_i$ to $\mathcal{M}_i'$,
3) remove the original vSWs and VLs in $\mathcal{M}_i$.

For instance, in Fig. 2(a), we reconfigure *vSW* b in *vSDN* 1 from *S-SW* 2 to *S-SW* 6. Due to the dependency between *vSW* b and *VLs* a-b and b-c, the migration of *vSW* b involves one node remapping and two link remappings. Specifically, the NVH first copies all the flow-entries of *vSW* b to *S-SW* 6, then steers the traffic passing through *vSW* b in *vSDN* 1 to use *S-SW* 6 instead of *S-SW* 2, and finally removes the flow-entries of *vSW* b on *S-SW* 2. Hence, the transition will not cause noticeable packet losses in *vSDN* 1.

One might think that the aforementioned procedure is feasible, as long as $\mathcal{M}_i'$ is a feasible VNE scheme for $G_i^r(V_i^r, E_i^r)$ based on the network status. Unfortunately, this is not true, and MbB is infeasible in certain cases where $\mathcal{M}_i$ and $\mathcal{M}_i'$ share some resource-insufficient S-SWs or/and SLs. Fig. 2(b) illustrates such an example. This time, we assume that the bandwidth usages on *SLs* 1-2 and 3-4 are relatively high, while *SLs* 1-3 and 2-4 have abundant bandwidth resources. Hence, for the purpose of re-balancing the bandwidth usages, the vSDN reconfiguration wants to remap *vSWs* b and c and the related VLs from the original VNE scheme (marked in green) to the new one (marked in red). However, as shown in Fig. 2(b), the available memory resources on *S-SWs* 2 and 3 are insufficient to carry the flow-entries/tables of the original and new VNE schemes simultaneously, which makes MbB infeasible. Therefore, we consider the move-to-vacancy (MtV) scenario [54], *i.e.*, first remapping *vSWs* b and c to a third VNE scheme (onto *S-SWs* 6 and 5, respectively), then tearing down the original VNE scheme, and finally remapping the vSWs and related VLs from the third VNE scheme to the new one. Here, the third VNE scheme is marked in purple in Fig. 2(b).

Finally, with the procedure shown in Fig. 2(b), we accomplish the vSDN reconfiguration with MbB and MtV, and minimize the service disruptions during the vSDN reconfiguration. Nevertheless, MtV could be infeasible too, and thus it can only be used in the best-effort way, *i.e.*, service disruption would be inevitable if both MbB and MtV are not possible.

[2]Note that, in a dynamic network environment, each vSDN can change both the memory usages of its vSWs and the bandwidth usages on its VLs on-the-fly. Hence, its initial VNE scheme can be calculated based on the estimated statistics of the resource usages (*e.g.*, their mean or maximum values).

C. Parallel vSDN Reconfiguration

Ideally, we would like to parallelly remap all the concerned vSWs and VLs, which could be from multiple vSDNs, in each vSDN reconfiguration. Hence, the reconfiguration latency can be reduced significantly, and it would not increase with the total number of concerned vSWs and VLs anymore. However, considering the complex dependencies among vSWs, VLs, S-SWs and SLs, we can hardly achieve this in one shot, especially when the vSDN reconfiguration has to leverage MbB and MtV for hitless operation. Here, the major difficulty of realizing one-stage parallel and hitless vSDN reconfiguration comes from the resource constraints.

Fig. 3 shows an example on the conflicts in memory usages caused by parallel vSDN reconfiguration. Here, we assume that there are four S-SWs and the vSDN reconfiguration is for re-balancing the memory usages on the S-SWs. Therefore, to address the unbalanced usages in Fig. 3(a), the InP needs to migrate *vSWs* a, b and c from *S-SWs* 1, 2 and 3 to *S-SWs* 3, 3 and 4, respectively (as in Fig. 3(c)). However, due to the insufficient memory capacity on *S-SW* 3 to accept all the flow-entries from *vSWs* a, b and c, one-stage parallel reconfiguration with MbB will have the conflict in Fig. 3(b). We can see that the conflicts in memory usages happen in the first step of MbB, *i.e.*, setting up new vSWs and VLs according to the new VNE.

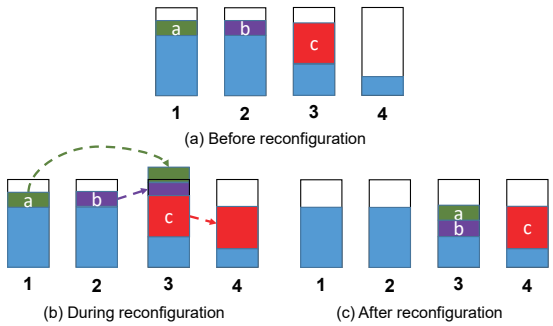

Fig. 3. Conflict in memory usages in parallel reconfiguration.

Meanwhile, the conflicts in bandwidth usage can also prevent one-stage parallel vSDN reconfiguration. Fig. 4 provides such an example. This time, the vSDN reconfiguration needs to remap two *VLs* a-b and a'-b' from 1-2-3 and 4-5-3 to 1-5-3 and 4-3, respectively (*i.e.*, from the mapping in Fig. 4(a) to that in Fig. 4(c)), while both VLs carry active traffic flows. Hence, during the one-stage parallel reconfiguration with MbB, the new substrate path of *VL* a-b can share *SL* 5-3 with the original one of *VL* a'-b', as shown in Fig. 4(b). In this case, if the total throughput of the two flows is larger than the capacity of *SL* 5-3, there will be a conflict in bandwidth usage. The conflicts in bandwidth usages happen in the second step of MbB, *i.e.*, switching active traffic to new vSWs and VLs. Note that, the conflicts in bandwidth usages are much less devastating than those in memory usages. This is because the traffic on original and new substrate paths might not encounter each other if path switching is done quickly enough, but according to MbB, coexistence of flow-entries in original and new vSWs will certainly happen. Therefore, we should focus more on minimizing the conflicts in memory usages.

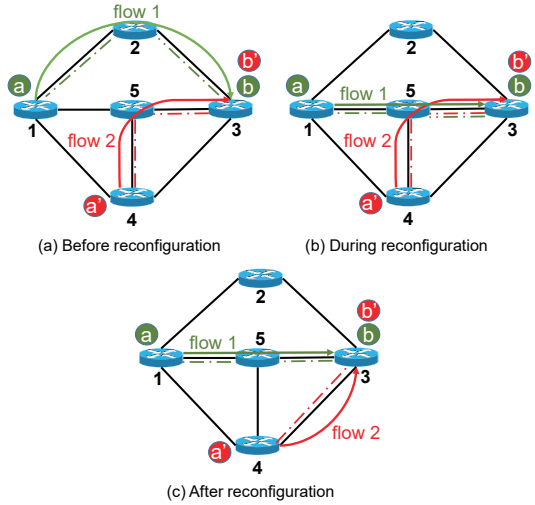

Fig. 4. Conflict in bandwidth usages in parallel reconfiguration.

IV. Multi-stage Parallel vSDN Reconfiguration

One-stage parallel vSDN reconfiguration can hardly avoid the conflicts in resource usages, and thus it can cause severe service disruptions. In this section, we design a multi-stage parallel vSDN reconfiguration scheme. Specifically, we optimize the reconfiguration procedure such that both the stages of operations and the service disruptions can be minimized.

A. Maximal Connected Reconfigurable Subgraph (MCRSG)

Intuitively, we would consider vSWs and VLs as the basic elements in vSDN reconfiguration. Nevertheless, this would not benefit our algorithm design due to the complex dependencies among vSWs and VLs. Fig. 5 shows an illustrative example on why the basic elements should not simply be vSWs and VLs. For the reconfiguration scheme in Fig. 5(a), we have the conflict in memory usages in Fig. 5(b), *i.e.*, *vSW* b cannot be remapped with MbB before the successful remapping of *vSW* a'. Hence, *VL* a-b and *vSW* a' cannot be reconfigured in parallel. But if we treat *vSW* a as a basic element and remap it together with *vSW* a', the intermediate state will be the one in Fig. 5(c), where a temporary substrate path 3-4-2 is set up to support the interim *VL* a-b. However, because the remapping of *vSW* b has to be put on hold until *vSW* a' has been reconfigured (as in Fig. 5(d)), the remapping of *vSW* a in Fig. 5(c) is apparently redundant. In other words, it would be more beneficial in terms of saving operation complexity and ensuring reconfiguration success, if we treat *VL* a-b as a basic element and remap it after *vSW* a'.

The dilemma in Fig. 5 inspires us to define the basic element in our vSDN reconfiguration as a maximal connected reconfigurable subgraph (MCRSG).

Definition 1. *The* ***maximal connected reconfigurable subgraph (MCRSG)*** *refers to a connected structure in a vSDN, which only consists of the vSWs and VLs that need to be reconfigured and cannot be enlarged anymore by adding in more such vSWs and VLs. Note that, an MCRSG might not always be a graph because the VLs in it can be dangling ones (i.e., the MCRSG might not include both end vSWs of a VL).*

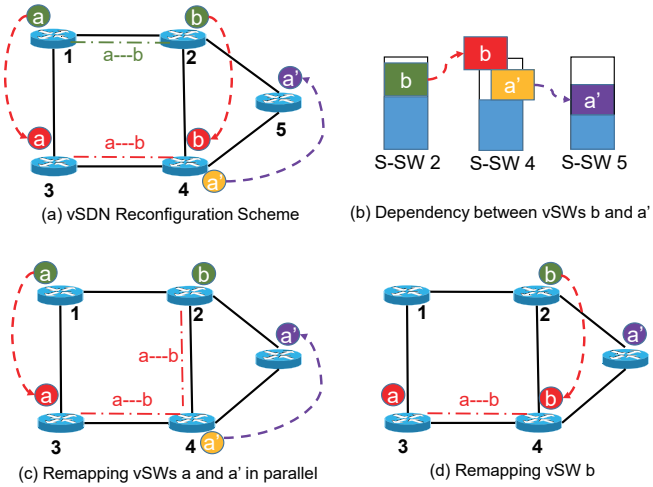

Fig. 5. Example on setting vSWs and VLs as basic reconfiguration element.

Therefore, a vSDN can have multiple MCRSGs (as shown in Fig. 6). The smallest MCRSG is a VL, while the largest one is a whole vSDN when all of its vSWs need to be remapped. Note that, a single vSW cannot be an MCRSG, because remapping the vSW will change the link mappings of all the VLs that use it as an end-node. For a vSDN $G_i^r(V_i^r, E_i^r)$, its MCRSGs can be obtained by performing a breadth-first search on it to compare $\mathcal{M}_i$ and $\mathcal{M}_i'$. Hence, the time complexity of finding all of its MCRSGs is $O(|V_i^r| + |E_i^r|)$.

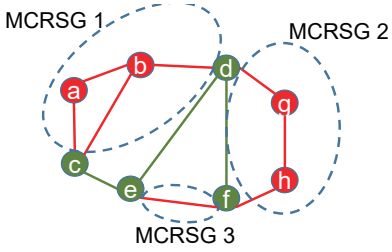

Fig. 6. Example on finding MCRSGs in a vSDN.

B. Overall Algorithm Design

Algorithm 1 shows the overall procedure of our multi-stage parallel vSDN reconfiguration based on MCRSGs. Specifically, the algorithm operates in the greedy manner to let the InP select the most weighted MCRSGs to reconfigure in parallel in each stage. Here, the weight of each MCRSG will be defined in the next subsection. *Lines* 1-7 are for the initialization. Specifically, for a reconfiguration operation, we store the indices of concerned vSDNs in set I (*Line* 1), check each concerned vSDN to find its MCRSGs and store them in set $\mathbb{SG}$ (*Line* 5), and obtain the new S-SWs and SLs that each MCRSG will use after remapping (*Line* 6). Then, the while-loop of *Lines* 8-36 realizes the multi-stage parallel reconfiguration. Each iteration is a stage of reconfiguration, which divides the pending MCRSGs in $\mathbb{SG}$ into three categories, *i.e.*, the moveable, unmoveable, and indeterministic MCRSGs.

Firstly, we hypothetically remap all the MCRSGs in $\mathbb{SG}$ but keep their original resource usages (*i.e.*, conducting the "make" operation in MbB), find all the MCRSGs that are surely moveable, and move the moveable MCRSGs from $\mathbb{SG}_t$ to $\mathbb{SG}_1$ (*Lines* 10-16). Here, $\mathbb{SG}_t$ gets initialized as $\mathbb{SG}$ in *Line* 9. Secondly, we check whether each remaining MCRSG in $\mathbb{SG}_t$ can be remapped while keeping its original resource usage (*Line* 18). If this cannot be done, the MCRSG is certainly an unmoveable one and we move it from $\mathbb{SG}_t$ to $\mathbb{SG}_2$ (*Lines* 19-21). At this moment, we store the moveable and unmoveable MCRSGs in $\mathbb{SG}_1$ and $\mathbb{SG}_2$, respectively, while the remaining ones in $\mathbb{SG}_t$ are indeterministic. Thirdly, *Lines* 23-25 leverage *Algorithm* 5 to transform certain indeterministic MCRSGs into moveable ones, if $\mathbb{SG}_t$ is not empty. The detailed procedure of *Algorithm* 5 will be discussed in the next subsection.

Finally, we remap the MCRSGs according to their categories (*Lines* 26-35). If $\mathbb{SG}_1$ is not empty, all the MCRSGs in it are the moveable ones that can be remapped in parallel with MbB. Hence, the stage just reconfigures them in one shot (*Lines* 27-29), and the while-loop will proceed to the next iteration. Note that, when remapping an MCRSG, we change its VNE scheme to the new one, *i.e.*, the resource utilization of its original VNE scheme in the SNT is released. Otherwise, if $\mathbb{SG}_1$ is empty but $\mathbb{SG}_2$ is not empty, we are only left with unmoveable MCRSGs whose remappings have cyclic dependencies such that resource conflicts will happen if they are remapped with MbB. *Lines* 31-34 remap the MCRSGs in $\mathbb{SG}_2$ with MbB and MtV in the best-effort way, by leveraging *Algorithm* 2 below. Then, the multi-stage parallel reconfiguration is accomplished.

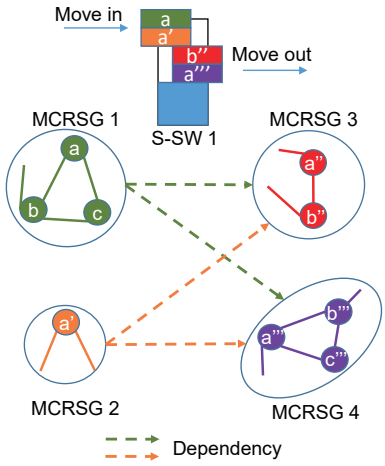

Fig. 7. Example on dependencies among MCRSGs.

Fig. 7 gives an example on the dependencies among MCRSGs. Here, *vSW* a in *MCRSG* 1 and *vSW* a' in *MCRSG* 2 need to be remapped to *S-SW* 1, while *vSW* b'' in *MCRSG* 3 and *vSW* a''' in *MCRSG* 4 need to be reconfigured from *S-SW* 1. As the memory resources in *S-SW* 1 are limited, if we remap the four MCRSGs in parallel with MbB, there will be resource conflicts in *S-SW* 1. Hence, *MCRSGs* 3 and 4 should be remapped before *MCRSGs* 1 and 2, and the dependencies among them are illustrated in Fig. 7. In the worst cases, the dependencies can be cyclic as shown in Fig. 8(a), which makes all the related MCRSGs unmoveable and can only be resolved with MtV in the best-effort manner (as explained in Fig. 2(b)).

Algorithm 1: Multi-stage Parallel Reconfiguration

store the indices of vSDNs to reconfigure in set I;
$\mathbb{SG} = \emptyset$;
for *each index* $i \in I$ **do**
    perform breadth-first search on $G_i^r(V_i^r, E_i^r)$ to find all the MCRSGs based on $\mathcal{M}_i$ and $\mathcal{M}_i'$;
    store the MCRSGs in $\mathbb{SG}$ as $\{SG_j\}$, where j is the global index of an MCRSG;
    store the new S-SWs and SLs of SG_j in set $\mathbb{R}_j$;
end
while $\mathbb{SG} \neq \emptyset$ **do**
    $\mathbb{R} = \emptyset$, $\mathbb{SG}_t = \mathbb{SG}$, $\mathbb{SG}_1 = \emptyset$, $\mathbb{SG}_2 = \emptyset$;
    find all S-SWs and SLs with resource conflicts, if we hypothetically remap all the MCRSGs in $\mathbb{SG}_t$ but keep their original resource usages;
    store the S-SWs and SLs in $\mathbb{R}$;
    for *each* $SG_j \in \mathbb{SG}_t$ **do**
        if $\mathbb{R} \cap \mathbb{R}_j = \emptyset$ **then**
            move SG_j from $\mathbb{SG}_t$ to $\mathbb{SG}_1$;
        end
    end
    for *each* $SG_j \in \mathbb{SG}_t$ **do**
        remap SG_j hypothetically but keep its original resource usages;
        if *there are resource conflicts* **then**
            move SG_j from $\mathbb{SG}_t$ to $\mathbb{SG}_2$;
        end
    end
    if $\mathbb{SG}_t \neq \emptyset$ **then**
        select certain MCRSGs in $\mathbb{SG}_t$ with *Algorithm* 5 and move them from $\mathbb{SG}_t$ to $\mathbb{SG}_1$;
    end
    move all the remaining MCRSGs in $\mathbb{SG}_t$ to $\mathbb{SG}_2$;
    if $\mathbb{SG}_1 \neq \emptyset$ **then**
        remap MCRSGs in $\mathbb{SG}_1$ with MbB in parallel;
        $\mathbb{SG} = \mathbb{SG} \setminus \mathbb{SG}_1$;
    else
        if $\mathbb{SG}_2 \neq \emptyset$ **then**
            remap MCRSGs in $\mathbb{SG}_2$ with MbB and MtV in the best-effort way (*Algorithm* 2);
            $\mathbb{SG} = \mathbb{SG} \setminus \mathbb{SG}_2$;
        end
    end
end

Algorithm 2 explains how to break cyclic dependencies with MtV. *Lines* 1-2 are for the initialization, where we get the dependencies among the unmoveable MCRSGs in $\mathbb{SG}_2$ and build a dependency graph (DG) $G_d(V_d, E_d)$ to represent it. Specifically, each MCRSG in $\mathbb{SG}_2$ is represented as a node in V_d while the dependencies among the MCRSGs are denoted as directed edges in E_d. If an MCRSG $v_{d,1}$ cannot be remapped before the MCRSG $v_{d,2}$ has been reconfigured, there will be a directed edge as $(v_{d,1}, v_{d,2})$ in E_d. Note that, a direct edge from an MCRSG can point to itself, if the remapping schemes

Algorithm 2: Break Cyclic Dependencies with MtV

find dependencies among MCRSGs in $\mathbb{SG}_2$ by hypothetically remapping all of them but keeping their original resource usages;
construct a DG $G_d(V_d, E_d)$ to denote dependencies;
while $\mathbb{SG}_2 \neq \emptyset$ **do**
    if G_d *has node(s) whose out-degrees are* 0 **then**
        remap the corresponding MCRSGs with MbB;
        remove the MCRSGs from $\mathbb{SG}_2$ and update $G_d(V_d, E_d)$ to remove their nodes;
    else
        perform MtV (in best-effort) to remap the MCRSG whose in-degree is the maximum;
        remove the MCRSG from $\mathbb{SG}_2$ and update $G_d(V_d, E_d)$ to remove its node;
    end
end

of its vSWs and VLs have resource conflicts.

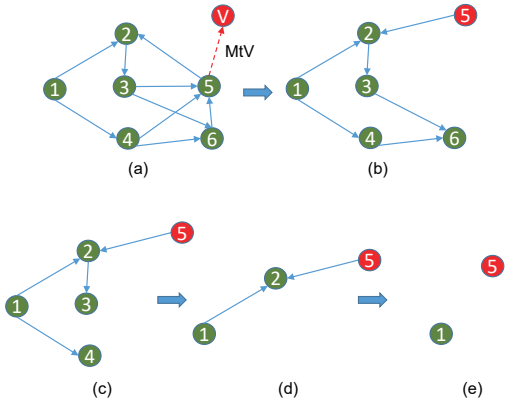

Fig. 8. Example on breaking cyclic dependencies with MtV.

Fig. 8(a) shows an example on the DG, where *MCRSGs* 2, 3 and 5 have cyclic dependencies. In each iteration of the while-loop, we first check whether the DG contains node(s) whose out-degrees are 0. If yes, the MCRSGs represented by these nodes do not depend on other ones, and thus can be remapped with MbB (*Lines* 4-6). Otherwise, we find the MCRSG whose in-degree is the maximum, and try to remap it with MtV in the best-effort way, *i.e.*, for breaking cyclic dependencies (*Lines* 8-9). Here, we always make sure that there will not be any new dependencies when calculating the vacancy for each MCRSG. Meanwhile, MtV can also become infeasible, when the SNT is heavy loaded. In that case, the MCRSGs are remapped with the scheme that will cause service disruption, *i.e.*, first tearing down their original VNE schemes and then setting up the new ones. For the DG in Fig. 8(a), *Algorithm* 2 will first select *MCRSG* 5 to apply the remapping with MtV. If this can be done, *MCRSG* 5 will be remapped in the hitless manner, and there will be service disruptions, otherwise. Next, the remapping sequence will be $6 \rightarrow \{3, 4\} \rightarrow 2 \rightarrow \{1, 5\}$, as explained in Figs. 8(b)-8(e). The time complexity of *Algorithm* 2 is $O(|\mathbb{SG}_2|^2 \cdot (|V_s| + |E_s|))$.

C. Selection of Indeterministic MCRSGs to Reconfigure

As explained in *Algorithm* 1, we divide the pending MCRSGs into three categories in each stage. Specifically, in addition to those that are surely moveable and unmoveable, we also have indeterministic MCRSGs that can be transformed into moveable ones, if we have a proper selection algorithm to sort out the potential resource conflicts. Here, the potential resource conflicts can be obtained by hypothetically remapping all the indeterministic MCRSGs (*i.e.*, those in $\mathbb{SG}_t$ when *Algorithm* 1 reaches *Line* 24), while keeping their original resource usages. Then, the S-SWs and SLs with resource conflicts are stored in set $\mathbb{R}_{in}$, with which we formulate the following integer linear programming (ILP) model to optimize the MCRSG selection.

Input Parameters:

- $\mathbb{SG}_t$: the set of indeterministic MCRSGs.
- $\mathbb{R}_{in}$: the set of substrate elements (*i.e.*, S-SWs and SLs) with resource conflicts.
- c_{se}: the available resources on an element $se \in \mathbb{R}_{in}$.
- $w_{j,se}$: the resource usage on an element $se \in \mathbb{R}_{in}$, if an MCRSG $SG_j \in \mathbb{SG}_t$ is reconfigured.
- p_j: the preset weight of remapping SG_j successfully.

Variables:

- x_j: the boolean variable that equals 1 if SG_j is selected to be transformed into a moveable one, and 0 otherwise.

Objective:

In this work, we introduce a preset weight p_j for each MCRSG SG_j to generalize our problem formulation and to give the InP the freedom to prioritize each MCRSG in the parallel reconfiguration. For instance, if the InP simply wants to reconfigure as many MCRSGs in each batch, it should set $\{p_j = 1,\ \forall SG_j \in \mathbb{SG}_t\}$. As we do not restrict how to define the preset weight, our MCRSG selection algorithm can optimize the following objective, *i.e.*, to maximize the total weight of the selected MCRSGs, regardless of the definition.

$$\text{Maximize} \quad \sum_{SG_j \in \mathbb{SG}_t} p_j \cdot x_j. \tag{2}$$

Constraints:

$$\sum_{SG_j \in \mathbb{SG}_t} w_{j,se} \cdot x_j \le c_{se}, \quad \forall se \in \mathbb{R}_{in}. \tag{3}$$

Eq. (3) ensures that the remappings of the selected indeterministic MCRSGs with MbB will not cause any resource conflicts, *i.e.*, the selection transforms them into moveable ones.

Theorem 2. *The MCRSG selection problem is $\mathcal{NP}$-hard.*

Proof: We prove the $\mathcal{NP}$-hardness of the MCRSG selection problem by transforming it into a general case of a well-known $\mathcal{NP}$-hard problem. By observing the MCRSG selection problem defined by Eqs. (2) and (3), we find that each MCRSG in $\mathbb{SG}_t$ can consume the resources in multiple substrate elements. Hence, we can treat each substrate element $se \in \mathbb{R}_{in}$ as a dimension (with a capacity of c_{se}) of a multi-dimensional knapsack ($\mathbb{R}_{in}$), and consider each MCRSG SG_j as an item, which has a multi-dimensional size ($\{w_{j,se},\ \forall se \in \mathbb{R}_{in}\}$) and a value ($p_j$). Then, the problem becomes how to select items to put into the multi-dimensional knapsack such that their total value is maximized. This is essentially the general case of a multi-dimensional 0-1 knapsack problem (MKP), which is known to be $\mathcal{NP}$-hard [55]. Hence, we prove the $\mathcal{NP}$-hardness of the MCRSG selection problem. ■

Since the MCRSG selection problem is $\mathcal{NP}$-hard, we first would like to resort to a fully polynomial-time approximation scheme (FPTAS) to design a time-efficient algorithm, which can get near-optimal solutions with reasonable running time. Nevertheless, the study in [56] has already proven that it is not feasible to find an FPTAS for MKP. Hence, in the next section, we will leverage Lagrangian relaxation (LR) to design a time-efficient algorithm to solve the MCRSG selection problem.

V. Approximation Algorithm to Select MCRSGs

In this section, we explain how to leverage LR to optimize the selection of MCRSGs. The MCRSG selection problem is a maximization problem. Therefore, we first dualize the hard-side constraints of the original problem, and construct a dual problem whose solution gives an upper-bound on the optimal solution of the original one. Then, we obtain a feasible solution of the original problem based on the solution of the dual problem. Since we have a maximization problem, the feasible solution provides a lower-bound. Next, we optimize the solution of the original problem by obtaining upper-/lower-bounds iteratively. Meanwhile, the gap between the upper- and lower-bounds indicates the distance between the current feasible solution and the optimal one.

A. Lagrangian Dual Problem

The number of constraints in Eq. (3) is $|\mathbb{R}_{in}|$. By dualizing $|\mathbb{R}_{in}|-1$ constraints, we get the following dual problem, where the $|\mathbb{R}_{in}|-1$ constraints compose set $\mathbb{R}'_{in} \subsetneqq \mathbb{R}_{in}$.

$$\begin{aligned} \text{Minimize} \quad Z_{dual}(\Lambda) = \max_{\{x_j\}} \Bigg[& \left(\sum_{SG_j \in \mathbb{SG}_t} p_j \cdot x_j \right) + \\ & \sum_{se \in \mathbb{R}'_{in}} \lambda_{se} \cdot \left(c_{se} - \sum_{SG_j \in \mathbb{SG}_t} w_{j,se} \cdot x_j \right) \Bigg], \\ \text{s.t.} \quad & \sum_{SG_j \in \mathbb{SG}_t} w_{j,se} \cdot x_j \le c_{se},\ se = \mathbb{R}_{in} \setminus \mathbb{R}'_{in}, \end{aligned} \tag{4}$$

where $\Lambda = \{\lambda_{se}\}$ is the vector of Lagrangian multipliers. We have $\lambda_{se} \ge 0,\ \forall se \in \mathbb{R}'_{in}$ to ensure that $Z_{dual}(\Lambda)$ will be an upper-bound on the optimal solution in each iteration, where $Z_{dual}(\Lambda)$ is maximized for a specific Λ. Then, we have

$$Z_{dual}(\Lambda) = \max_{\{x_j\}} \left(\sum_{SG_j \in \mathbb{SG}_t} \bar{p_j} \cdot x_j + \sum_{se \in \mathbb{R}'_{in}} \lambda_{se} \cdot c_{se} \right), \tag{5}$$

where $\bar{p_j}$ is the Lagrangian-modified weight of MCRSG SG_j

$$\bar{p_j} = p_j - \sum_{se \in \mathbb{R}'_{in}} \lambda_{se} \cdot w_{j,se}. \tag{6}$$

The second term in Eq. (5) is independent of $\{x_i\}$, and thus we only need to solve the following optimization to get $Z_{dual}(\Lambda)$.

$$\begin{aligned} \text{Maximize} \quad & \sum_{SG_j \in \mathbb{SG}_t} \bar{p_j} \cdot x_j, \\ \text{s.t.} \quad & \sum_{SG_j \in \mathbb{SG}_t} w_{j,se} \cdot x_j \le c_{se},\ se = \mathbb{R}_{in} \setminus \mathbb{R}'_{in}. \end{aligned} \tag{7}$$

The optimization in Eq. (7) can be solved with the dynamic programming in *Algorithm* 3 [57]. The time complexity of *Algorithm* 3 is $O(|\mathbb{SG}_t| \cdot c_{se})$, where $se = \mathbb{R}_{in} \setminus \mathbb{R}'_{in}$.

Algorithm 3: Dynamic Programming to Get $Z_{dual}(\Lambda)$

Input: $\{\bar{p_j}, c_{se}, w_{j,se} : \ se = \mathbb{R}_{in} \backslash \mathbb{R}'_{in}, \forall SG_j \in \mathbb{SG}_t\}$.
Output: $\{x_j\}$.
temporarily reassign indices of MCRSGs in $\mathbb{SG}_t$ as $i \in [1, |\mathbb{SG}_t|]$;
initialize $\mathbf{M} = \{m_{i,j}, \ i \in [1, |\mathbb{SG}_t|], \ j \in [1, c_{se}]\}$ as an all-zero matrix;
for *each* $i \in [1, |\mathbb{SG}_t|]$ **do**
  for *each* $j \in [1, c_{se}]$ **do**
    if $j < w_{i,se}$ **then**
      $m_{i,j} = m_{i-1,j}$;
    else if $j = w_{i,se}$ **then**
      $m_{i,j} = \max(m_{i-1,j}, \ \bar{p_i})$;
    else
      $m_{i,j} = \max\left(m_{i-1,j}, \ m_{i-1,j-w_{i,se}} + \bar{p_i}\right)$;
    end
  end
end
for *each* $k \in [0, |\mathbb{SG}_t| - 2]$ **do**
  $i = |\mathbb{SG}_t| - k$;
  if $m_{i,c_{se}} > m_{i-1,c_{se}}$ **then**
    $x_i = 1$;
    $c_{se} = c_{se} - w_{i,se}$;
    if $c_{se} = 0$ **then**
      break;
    end
  end
end
restore indices of MCRSGs in $\mathbb{SG}_t$ and get $\{x_j\}$ based on $\{x_i, \ i \in [1, |\mathbb{SG}_t|]\}$ accordingly;

B. Construction of Feasible Solution

After solving the optimization defined in Eq. (7) with *Algorithm* 3, we get the optimal solution $\{x_j\}$ regarding a specific Λ in an iteration. Then, we need to construct a feasible solution of the original problem based on the solution $\{x_j\}$, which can be achieved by leveraging *Algorithm* 4. More specifically, *Algorithm* 4 removes certain MCRSGs from the solution provided by *Algorithm* 3 until the resulting solution becomes feasible to the original problem. The time complexity of *Algorithm* 4 is $O(|\mathbb{SG}_t| \cdot |\mathbb{R}'_{in}|)$.

C. Solving Lagrangian Dual Problem

The optimization objective of the Lagrangian dual problem defined in Eq. (4) is to minimize $Z_{dual}(\Lambda)$. Since $Z_{dual}(\Lambda)$ is a piecewise linear programming, we leverage the sub-gradient method in [58] to optimize Λ iteratively until $Z_{dual}(\Lambda)$ converges to the minimum. Specifically, we first set Λ to an initial value, and then update Λ in each iteration as follows.

$$\Lambda_{k+1} = \Lambda_k - \mu_k \cdot f(\Lambda_k), \tag{8}$$

Algorithm 4: Construction of Feasible Solution

Input: $\{x_j\}$, $Z^* = 0$.
Output: $\{x_j\}$, Z^*.
for *each* $se \in \mathbb{R}'_{in}$ **do**
  $a = 0$;
  for *each* $SG_j \in \mathbb{SG}_t$ **do**
    if $x_j = 1$ **then**
      $a = a + w_{j,se}$;
    end
  end
  while $a > c_{se}$ **do**
    for $SG_j \in \mathbb{SG}_t$ **do**
      if $x_j = 1$ *AND* $w_{j,se} > 0$ **then**
        $x_j = 0$, $a = a - w_{j,se}$;
        $Z_{dual}(\Lambda) = Z_{dual}(\Lambda) - p_j$;
      end
      if $a \leq c_{se}$ **then**
        break;
      end
    end
  end
end
for *each* $SG_j \in \mathbb{SG}_t$ **do**
  if $x_j = 1$ **then**
    $Z^* = Z^* + p_j$;
  end
end

where Λ_k and μ_k are the Lagrangian multiplier and step-size, respectively, and $f(\Lambda_k)$ is the sub-gradient vector of $Z_{dual}(\Lambda)$ regarding Λ, in the k-th iteration. Specifically, we have

$$f(\Lambda) = \frac{\partial Z_{dual}}{\partial \Lambda}, \tag{9}$$

where Λ is a vector, and each of its elements is

$$\lambda_{se} = c_{se} - \sum_{SG_j \in \mathbb{SG}_t} w_{j,se} \cdot x_j. \tag{10}$$

As μ_k affects the convergence performance of $Z_{dual}(\Lambda)$, we get its value with the following formulation, based on [59].

$$\mu_k = \frac{\nu \cdot (Z_{dual}(\Lambda) - Z^*)}{||f(\Lambda_k)||^2}. \tag{11}$$

Here, $Z_{dual}(\Lambda)$ is obtained by solving the optimization in Eq. (7) with a specific Lagrangian multiplier Λ_k, Z^* is the maximum feasible solution until the k-th iteration, and ν is a scaler variable whose initial value is set as 2. Specifically, if the value of $Z_{dual}(\Lambda)$ cannot be improved after a fixed number of iterations, we will divide ν by 2. Note that, we need to have $\Lambda \succeq 0$ to guarantee that $Z_{dual}(\Lambda)$ is an upper-bound on the optimal solution, and this is achieved by applying

$$(\Lambda_{k+1})_{se} = \max\{0, \ [\Lambda_k - \mu_k \cdot f(\Lambda_k)]_{se}\}. \tag{12}$$

D. Overall Procedure

Algorithm 5 shows the overall procedure of the LR-based algorithm to select indeterministic MCRSGs to reconfigure.

Line 1 is for the initialization, where ub and lb are introduced to store the upper- and lower-bounds obtained in iterations, respectively, and n is the counter to monitor the convergence performance of $Z_{dual}(\Lambda)$. The while-loop covering *Lines* 2-19 tries to improve the solution's quality until the relative dual gap is smaller than a preset threshold γ. In the k-th iteration, we first use *Algorithm* 3 to get the optimal solution $\{x_j\}$ of the dual problem in Eq. (7) regarding Λ_k (*Lines* 3-4). Then, we use $Z_{dual}(\Lambda_k)$ to update the upper-bound ub, and if the value of ub has not been updated after T_h iterations, we divide v by 2 (*Lines* 5-11). Next, we construct a feasible solution Z^* of the original problem with *Algorithm* 4, and update the lower-bound lb according to Z^* (*Lines* 12-15). Finally, *Lines* 16-18 prepare for the next iteration.

Algorithm 5: Overall Algorithm Procedure

```
1  k = 1, Λ_k = 0, ν = 2, ub = +∞, lb = 0, n = 0;
2  while (ub − lb)/ub ≥ γ do
3      calculate {p̄_j} with Eq. (6) and Λ_k;
4      solve the optimization in Eq. (7) with Algorithm
       3 to get Z_dual(Λ_k) and {x_j};
5      if Z_dual(Λ_k) < ub then
6          ub = Z_dual(Λ_k), n = 0;
7      else if n > T_h then
8          ν = ν/2, n = 0;
9      else
10         n = n + 1;
11     end
12     get a feasible solution Z* with Algorithm 4;
13     if Z* > lb then
14         lb = Z*;
15     end
16     calculate μ_k with Eq. (11);
17     calculate Λ_{k+1} with Eqs. (8)-(10) and (12);
18     k = k + 1;
19 end
```

The optimal solution of the MCRSG selection problem can be obtained by directly solving the ILP defined by Eqs. (2)-(3). Hence, we can denote the optimal solution as Z_{ILP}. Since the optimization is a maximization problem, the approximation ratio of *Algorithm* 5 can be defined as

$$\varepsilon = \frac{Z^*}{Z_{\text{ILP}}}, \tag{13}$$

where Z^* is the feasible solution obtained with *Algorithm* 4. Meanwhile, the principle of LR ensures that by solving the Lagrangian dual problem with *Algorithm* 3, we can get $Z_{dual}(\Lambda_k)$ as an upper-bound on Z_{ILP}. The while-loop in *Algorithm* 5 guarantees that the algorithm's output satisfies

$$1 - \frac{lb}{ub} < \gamma, \tag{14}$$

where we have $ub = Z_{dual}(\Lambda_k)$ and $lb = Z^*$ according to *Lines* 6 and 14, respectively. Therefore, we can get

$$\varepsilon = \frac{Z^*}{Z_{\text{ILP}}} \geq \frac{Z^*}{Z_{dual}(\Lambda_k)} = \frac{lb}{ub} > 1 - \gamma. \tag{15}$$

This verifies that *Algorithm* 5 is an approximation algorithm whose approximation ratio ε is at least $1 - \gamma$ for the maximization in Eqs. (2)-(3). Moreover, its approximation becomes better if we can get a feasible solution with a smaller γ.

VI. Numerical Simulations

In this section, we perform simulations to evaluate the performance of the multi-stage parallel vSDN reconfiguration.

A. Simulation Setup

Section III-A has already explained that for a vSDN $G_i^r(V_i^r, E_i^r)$, our vSDN reconfiguration scheme optimizes the transition (*i.e.*, from its original VNE $\mathcal{M}_i$ to the new one $\mathcal{M}_i'$) to make the transition parallel and hitless, but we do not care how or for what purpose $\mathcal{M}_i'$ is calculated. In other words, our proposal is independent of $\mathcal{M}_i$ to the new one $\mathcal{M}_i'$ for each $G_i^r(V_i^r, E_i^r)$, and thus is generic. Nevertheless, in numerical simulations, we still need a scenario that can obtain new VNE schemes as the inputs to our vSDN reconfiguration scheme. Without loss of generality, we choose the load-balancing scenario considered in [27]. Specifically, in a dynamic SNT, the memory usages in S-SWs can become unbalanced due to various reasons [60], and thus the InP needs to invoke vSDN reconfiguration from time to time to re-balance them.

In the simulations, we consider three SNT topologies with different sizes, *i.e.*, the 14-node NSFNET topology [28], and two large-scale random topologies that have 50 S-SWs and 122 SLs (RT-50) and 100 S-SWs and 496 SLs (RT-100), respectively. Each S-SW has a random memory capacity to accommodate $[2500, 5000]$ flow-entries, while the bandwidth capacity of each SL is also randomly selected within $[1000, 2000]$ units. For each vSDN, its number of vSWs uniformly distributes within $[5, 10]$, the vSWs are randomly connected with a probability of 0.5, the memory requirement of each vSW is randomly selected within $[50, 200]$ flow-entries, and the bandwidth demand of each VL is within $[1, 80]$ units. The parameters above are set according to the realistic cases discussed in [14, 17, 61]. Our simulations consider dynamic network environment, where vSDNs are generated according to the Poisson traffic model, *i.e.*, the average arrival rate is λ vSDNs per time-unit and each vSDN has an average life-time of $\frac{1}{\mu}$ time-units. Hence, the load of vSDNs can be quantified as $\frac{\lambda}{\mu}$ in Erlangs. In order to ensure sufficient statistical accuracy, we average the results from 5 independent runs to get each data point.

The simulations run as follows. When a vSDN first comes in, we use an existing VNE algorithm, which is the GRC-VNE in [3], to provision it in the SNT. Although GRC-VNE tries to embed each vSDN such that the substrate resource usages can be balanced in the greedy manner, the dynamic arrivals and departures of vSDNs will still induce unbalanced memory usages in S-SWs. Then, when the unbalanced memory usages get accumulated to certain extent, the InP will leverage the LF-R algorithm in [27] to select certain vSDNs to reconfigure and calculate the new VNE schemes for them. Hence, for each selected vSDN $G_i^r(V_i^r, E_i^r)$, $\mathcal{M}_i$ is calculated by GRC-VNE at when it first comes in, while $\mathcal{M}_i'$ is obtained by LF-R at

when the InP detects severe unbalanced memory usages in its SNT. Next, our multi-stage parallel vSDN reconfiguration scheme kicks in to handle the transition from $\mathcal{M}_i$ to $\mathcal{M}'_i$.

B. Single Operations

We first evaluate an operation of vSDN reconfiguration in detail. For the definition of p_j (*i.e.*, the weight of remapping an MCRSG SG_j successfully), we consider two types: 1) $\{p_j = 1,\ \forall SG_j \in \mathbb{SG}_t\}$, and 2) p_j is set as the number of vSWs in SG_j. Hence, the first definition motivates the optimization in Section IV-C to maximize the number of selected MCRSG (MCRSG-prioritized), while the second one pushes it to select as many vSWs as possible (vSW-prioritized). For the LR-based approach in *Algorithm* 5, we set $\gamma \in \{0.2, 0.5, 0.8\}$ as the preset threshold on the relative dual gap (*i.e.*, the condition to stop iterations), while T_h is empirically set as 15.

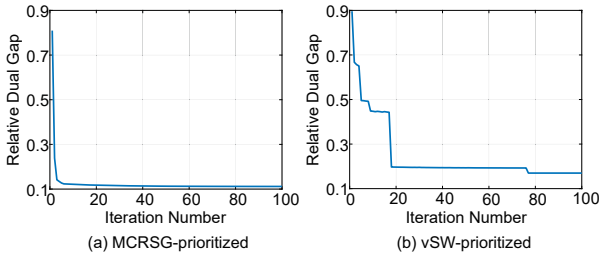

Fig. 9. Convergence performance of LR-based *Algorithm* 5 on NSFNET.

1) Convergence Performance: We first evaluate the convergence performance of *Algorithm* 5. With the simulation scenario explained above, we randomly select vSDN reconfiguration cases in SNTs based on the NSFNET, RT-50, and RT-100 at when the load of dynamic vSDNs is 40, 120, and 240 Erlangs, respectively. Note that, we also test the algorithm with other loads, and confirm that its convergence performance would not be affected significantly. Figs. 9-11 show the results on convergence performance from the simulations with NSFNET, RT-50 and RT-100, respectively. For each SNT, we consider both the MCRSG-prioritized and vSW-prioritized definitions of preset weight p_j. We observe that for all the simulation scenarios, the relative dual gap becomes less than 0.2 within 20 iterations, and this verifies that *Algorithm* 5 converges fast to provide good time-efficiency.

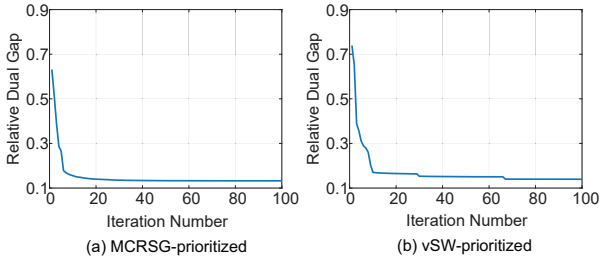

Fig. 10. Convergence performance of LR-based *Algorithm* 5 on RT-50.

2) Parallel Reconfiguration Stages: Then, we change the load of vSDNs in each SNT, apply our multi-stage parallel vSDN reconfiguration scheme, and get the average number of parallel stages in each reconfiguration operation at different loads. Meanwhile, to check the effect of γ on algorithm performance, we select its value from $\{0.2, 0.5, 0.8\}$ and compare the results from the multi-stage parallel reconfiguration using *Algorithm* 5, and those from the one using the ILP model in Section IV-C. Note that, if we use the ILP model to solve the problem of indeterministic MCRSG selection, the solution would be optimal. Figs. 12-14 show the simulation results. In Figs. 12 and 13, the average number of stages in each operation from the multi-stage parallel reconfiguration using *Algorithm* 5 with $\gamma = 0.2$ is very close to the optimal result from that using the ILP model. For RT-100, the scheme using the ILP model is too time-consuming, and thus we cannot obtain optimal solutions with it in that case, but the general trend that the multi-stage parallel reconfiguration becomes more efficient for a smaller γ can still be seen in Fig. 14.

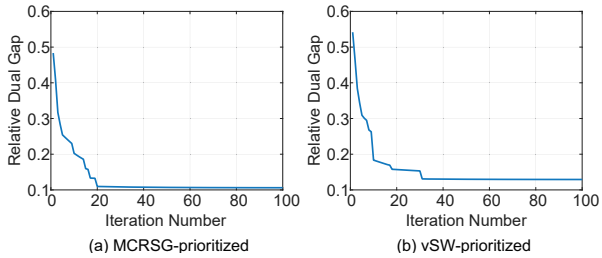

Fig. 11. Convergence performance of LR-based *Algorithm* 5 on RT-100.

Note that, the loads of vSDNs in Figs. 12-14 are selected to cover the light, medium and heavy loaded network environments in each SNT, as indicated by the results in Table II. Here, "Mem. Usage" and "BW Usage" mean the average utilizations of memory and bandwidth resources in the SNT, respectively, "Dep. per MCRSG" means the average number of dependencies from each MCRSG to substrate elements (S-SWs and SLs) before each vSDN reconfiguration, and "Non-MbB Ratio" means the average ratio of MCRSGs that cannot be remapped with MbB in each vSDN reconfiguration. Meanwhile, we can see that the average number of dependencies increases with the load of vSDNs, *i.e.*, the dependencies among the MCRSGs become more complicated. However, our proposal can always realize effective parallel vSDN reconfigurations to achieve large degree of conflict avoidance (as shown in Figs. 12-14). The results in Table II also suggest that for each of the SNTs, the non-MbB ratios are very low (*i.e.*, below 1.5%) in all the simulation scenarios, and we only have MCRSGs that cannot be remapped with MbB when the average usages of substrate resources are relatively high. This further verifies the performance of our proposal.

3) Time Complexity: To compare the algorithms' performance on time complexity, we list the average running time for different simulation scenarios in Table III. Here, each running time is the average result from multiple scenarios in Figs. 12-14. For instance, the running time listed in (ILP, NSFNET) in Table III is the average result of 10 ILP-related scenarios (*i.e.*, 10 combinations of p_j definitions and vSDN loads) in Fig. 12. The results indicate that our LR-based approximation algorithm is much more time-efficient than solving the ILP,

TABLE II
RESULTS ON SUBSTRATE RESOURCE USAGES, DEPENDENCIES PER MCRSG, AND RATIO OF MCRSGS NOT REMAPPED WITH MBB

	NSFNET				
Load of vSDNs	20	30	40	50	60
Mem. Usage (%)	42.3	54.3	65.7	74.3	85.6
BW Usage (%)	36.7	47.6	59.4	63.4	73.4
Dep. per MCRSG	3.4	4.3	5.1	6.5	7.8
Non-MbB Ratio (%)	0	0	0.12	0.59	1.45
	RT-50				
Load of vSDNs	60	90	120	150	180
Mem. Usage (%)	34.6	47.8	65.2	79.5	87.0
BW Usage (%)	29.2	40.1	58.2	68.1	74.0
Dep. per MCRSG	2.4	3.9	4.5	5.9	6.6
Non-MbB Ratio (%)	0	0	0	0.24	0.75
	RT-100				
Load of vSDNs	120	180	240	300	360
Mem. Usage (%)	32.3	47.7	64.9	78.2	88.1
BW Usage (%)	26.7	39.8	56.8	67.3	75.2
Dep. per MCRSG	2.4	3.7	4.0	4.6	5.6
Non-MbB Ratio (%)	0	0	0	0.14	0.35

i.e., the running time could be reduced by more than two magnitudes. Meanwhile, since a smaller γ means that the LR-based algorithm needs to run more iterations to reduce the relative dual gap, the running time decreases with γ.

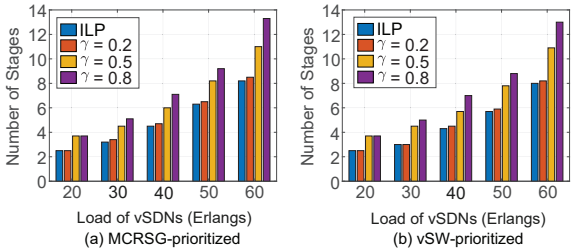

Fig. 12. Average number of parallel stages in each operation in NSFNET.

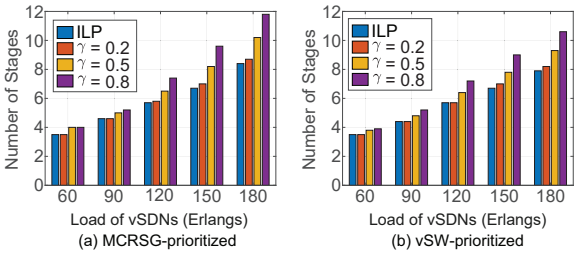

Fig. 13. Average number of parallel stages in each operation in RT-50.

C. Dynamic Operations

Next, we conduct dynamic simulations to confirm that multi-stage parallel vSDN reconfiguration is beneficial to the operation of SNT. This time, we use RT-50 as the SNT's topology, run the network for $5,000$ time-units to embed, reconfigure and remove dynamic vSDNs, and collect the

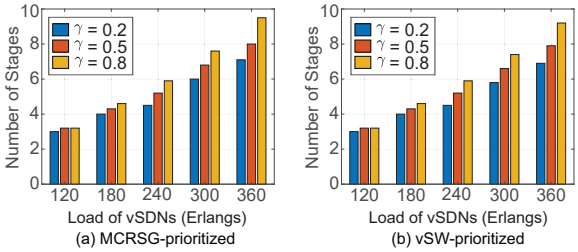

Fig. 14. Average number of parallel stages in each operation in RT-100.

TABLE III
AVERAGE RUNNING TIME (SECONDS)

	NSFNET	RT-50	RT-100
ILP	1.012	94.943	–
$\gamma = 0.2$	0.096	0.287	1.230
$\gamma = 0.5$	0.042	0.098	0.263
$\gamma = 0.8$	0.032	0.062	0.175

blocking probability of vSDNs. To ensure that the blocking probabilities are in the range of $[10^{-4}, 10^{-1}]$, we reduce the bandwidth demand of each VL to be within $[1, 10]$ units. We compare the performance of the SNT with and without multi-stage parallel vSDN reconfiguration. When vSDN reconfiguration is enabled, the operations are triggered every $\{50, 100, 200\}$ time-units. Hence, we denote the corresponding algorithms as LF-R-Parallel-50, LF-R-Parallel-100, and LF-R-Parallel-200, respectively. Figs. 15 shows the results on blocking probability, which confirms that multi-stage parallel vSDN reconfiguration can reduce the blocking probability.

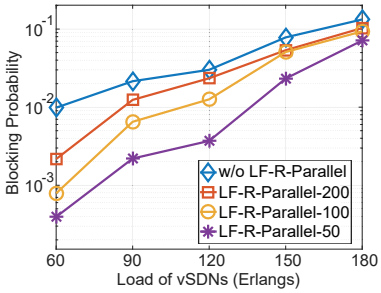

Fig. 15. Results on blocking probability.

VII. EXPERIMENTAL DEMONSTRATIONS

Although numerical simulations have already verified the effectiveness of our multi-stage parallel vSDN reconfiguration, its practicalness cannot be confirmed without experimental investigations. Moreover, experiments can reveal the scheme's performance on the overall reconfiguration latency and packet loss rate during reconfiguration, which cannot be simulated. Therefore, we expand the NVH system developed in [27] to enable multi-thread based parallel operations, and implement our multi-stage parallel vSDN reconfiguration scheme in it.

A. Experimental Setup

Fig. 16 shows the experimental setup, where the NVH can leverage the multi-thread scheme to remap multiple vSWs

and VLs (*i.e.*, either in the same vSDN or multiple vSDNs) simultaneously. The SNT consists of six S-SWs, each of which is based on a software-based SDN switch running on an independent high-performance Linux server. As indicated in Fig. 16, the experiments set up three vSDNs with chain topologies, and let each vSDN carry an active traffic flow with a throughput of 1 Mbps. Before vSDN reconfiguration, all of the flows go through *S-SWs* 2 and 3, while the vSDN reconfiguration remaps all the vSWs mapped on *S-SWs* 2 and 3 to *S-SWs* 6 and 5, respectively. In addition to the active flows, we also install a fixed number of flow-entries in each vSW, to emulate the background traffic.

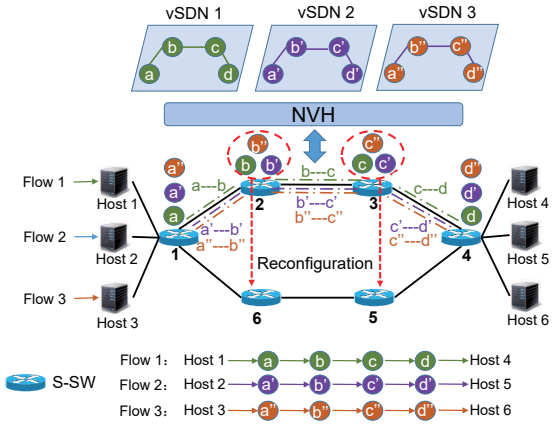

Fig. 16. Experimental setup.

B. Benefits of Parallel Reconfiguration

We first conduct an experiment to verify that the NVH can realize parallel vSDN reconfiguration. In this experiment, we first install 500 flow-entries on each vSW to reconfigure, and then invoke the parallel reconfiguration. During the transition, we record the total bandwidth of the traffic going through *S-SW* 2. Fig. 17 compares the experimental results of sequential and parallel vSDN reconfigurations. In Fig. 17(a), the sequential reconfiguration takes ∼600 ms, and the three bandwidth drops clearly indicate the remappings of the three vSDNs. On the other hand, the parallel reconfiguration in Fig. 17(b) only consumes ∼100 ms, which is much more time-efficient.

Next, we make the vSDN reconfiguration to move a total number of $[1000, 6000]$ flow-entries, and measure the total reconfiguration latency, which the total time needed to remap all the three vSDNs with MbB. Fig. 18 shows the results, which still clearly indicate the better time-efficiency of parallel reconfiguration. Specifically, the latency of parallel reconfiguration not only is much shorter than that of sequential reconfiguration, but also increases much slower with the total number of flow-entries to move. Note that, the reconfiguration latency in Fig. 18 for moving $3,000$ flow-entries in total is longer than that obtained by observing Fig. 17, for both the sequential and parallel schemes. This is because the latency shown in Fig. 18 is measured by considering the whole procedure of MbB, while the third step of MbB, which is to remove the original vSWs and VLs, might not have significant impacts on the bandwidth of active traffic flows.

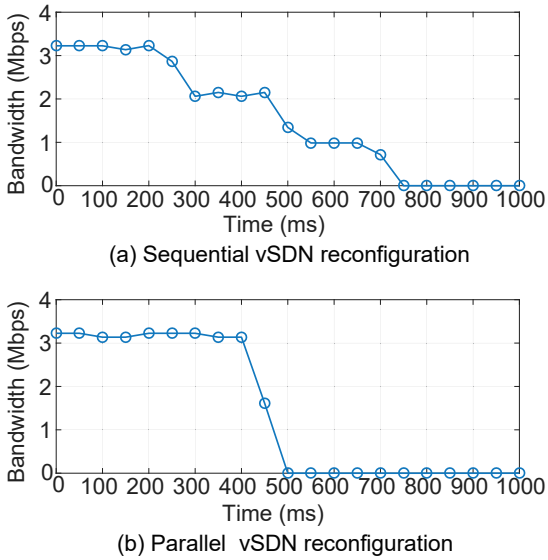

Fig. 17. Bandwidth of traffic going through *S-SW* 2 during vSDN remapping.

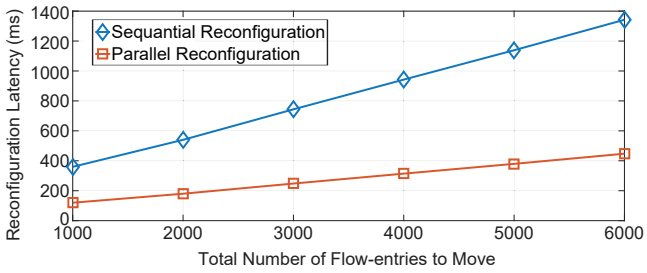

Fig. 18. Results on reconfiguration latency.

C. Tradeoff between Latency and Packet Losses

Finally, even though we leverage MbB and MtV to minimize packet losses, packet losses might not be completely avoided due to the inconsistency of packet handling during the parallel vSDN reconfiguration. More specifically, if the traffic flows affected by a vSDN reconfiguration have longer end-to-end round-trip-time (RTT), the probability that their packets get handled inconsistently because of the vSDN reconfiguration will become larger. In other words, the packet loss rate during a vSDN reconfiguration will increase with the longest end-to-end round-trip-time (RTT) of the affected traffic flows [27]. Meanwhile, the inconsistency of packet handling will disappear after each stage of the parallel vSDN reconfiguration (*i.e.*, the related flow-entries/tables have been updated in the corresponding S-SWs), and thus it would not cause severe packet losses because each stage can be accomplished very quickly (within a few hundred milliseconds in Fig. 18).

To check how hitless our parallel vSDN reconfiguration is, we conduct more experiments. Specifically, we intentionally increase the end-to-end RTT of the traffic flows, perform the parallel vSDN reconfiguration again, and measure the packet loss rate during the transition[3]. Fig. 19 shows the experimental

[3]Here, packet losses are transient as they only happen during the remapping, and because the reconfiguration latency is always less than one second, we average the packet loss rate within a second.

results, which show that the packet loss rate increases with the end-to-end RTT. This is because if the end-to-end RTT is relatively long, some packets might still be transmitting on the original path when the third step of MbB is invoked to remove the original vSWs and VLs. Hence, to reduce the packet loss rate, we should insert certain waiting time in between the second and third steps of MbB. However, this would increase the total reconfiguration latency. Hence, we can see that for vSDN reconfiguration, if the longest end-to-end RTT is fixed, there is a tradeoff between the total reconfiguration latency and the packet loss rate during remapping.

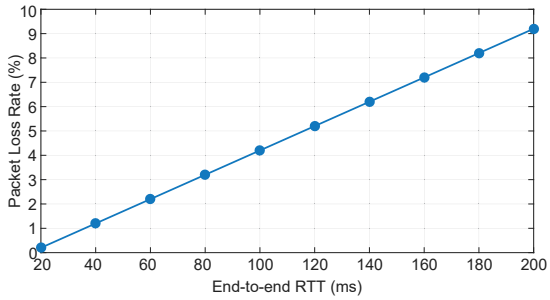

Fig. 19. Packet loss rate during parallel vSDN reconfiguration.

Hence, we use experiments to find out the shortest waiting time that can make the vSDN reconfiguration in Fig. 16 completely hitless (*i.e.*, the packet loss rate during the transition is zero), for both the sequential and parallel schemes. The total waiting time is plotted in Fig. 20, which indicates that the parallel vSDN reconfiguration balances the tradeoff between the total reconfiguration latency and the packet loss rate during remapping better. This is still because parallel reconfiguration can remap the vSDNs with much fewer MbB operations.

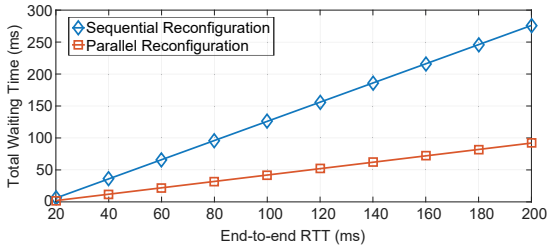

Fig. 20. Extra waiting time to ensure zero packet loss.

VIII. Conclusion and Future Work

In this paper, we optimized the transition to migrate vSDNs from their original VNE schemes to the new ones, and proposed a scheme that can realize parallel and hitless vSDN reconfiguration in a resource-efficient manner, by leveraging the "make-before-break" scenario. To resolve the issues of one-stage parallel reconfiguration, we proposed a multi-stage parallel vSDN reconfiguration scheme based on MCRSG, which operates in the greedy manner to let the InP select the most weighted MCRSGs to reconfigure in parallel in each stage. The optimization to select MCRSGs to reconfigure in each stage was formulated as an ILP model, and we proved its $\mathcal{NP}$-hardness. Then, we designed an approximation algorithm based on Lagrangian relaxation to solve the problem time-efficiently. Extensive simulations verified that the proposed algorithm can obtain near-optimal solutions quickly, *i.e.*, for a relatively large SNT with 100 S-SWs, it can reduce the relative dual gap below 0.2 within only 20 iterations. Moreover, we also considered the system implementation of our proposed algorithms, realized the multi-stage parallel vSDN reconfiguration in a practical NVH system, and demonstrated its performance in a real network testbed. Our experimental study analyzed the tradeoff between reconfiguration latency and packet loss rate, and revealed an empirical method to adjust key parameters of our NVH system, for adapting to various network environments.

Meanwhile, we hope to point out that this work still has some unresolved issues, which will be addressed in our future work. Firstly, the overall procedure of the multi-stage parallel vSDN reconfiguration in *Algorithm* 1 operates in the greedy manner. Therefore, it will be interesting to optimize the overall procedure towards a specific objective (*e.g.*, minimizing the number of parallel reconfiguration stages) and design an exact or approximation algorithm for the optimization. Secondly, the algorithms designed in this work do not care how the new VNE schemes for vSDN reconfiguration were calculated. Although this brings some advantages (*e.g.*, simplifying the algorithm design, and supporting modular design of the NVH system), it would still be relevant to study the optimization that jointly considers the calculations of new VNE schemes and reconfiguration procedure. Hence, our future work will address the joint optimization and try to balance the tradeoff between the reconfiguration and operational costs of vSDNs.

Acknowledgments

This work was supported in part by the NSFC projects 61871357, 61771445 and 61701472, ZTE Research Fund PA-HQ-20190925001J-1, Zhejiang Lab Research Fund 2019LE0AB01, CAS Key Project (QYZDY-SSW-JSC003), and SPR Program of CAS (XDC02070300).

References

[1] 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.

[2] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[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] 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.

[16] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, pp. 1888–1906, Fourth Quarter 2013.

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

[19] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in *Proc. of ICNP 2003*, pp. 120–131, Nov. 2003.

[20] 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.

[21] 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.

[22] 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.

[23] S. Zhao *et al.*, "Make Big Data applications more reliable: Hitless vSDN migration to avoid TCAM depletion," in *Proc. of ICC 2018*, pp. 1–6, May 2018.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] X. Gao *et al.*, "A new algorithm with coordinated node and link mapping for virtual network embedding based on LP relaxation," in *Proc. of ACP 2010*, pp. 152–153, Dec. 2010.

[32] G. Chochlidakis and V. Friderikos, "Low latency virtual network embedding for mobile networks," in *Proc. of ICC 2016*, pp. 1–6, May 2016.

[33] R. Lin, C. Du, S. Wang, and S. Luo, "Virtual network embedding in flexi-grid optical networks," in *Proc. of ICCT 2017*, pp. 777–782, Oct. 2017.

[34] 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.

[35] Y. Xue, J. Peng, K. Han, and Z. Zhu, "On table resource virtualization and network slicing in programmable data plane," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 319–331, Jan. 2020.

[36] R. Sherwood *et al.*, "FlowVisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1–13, 2009.

[37] A. Al-Shabibi *et al.*, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. of ACM HotSDN 2014*, pp. 25–30, Aug. 2014.

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

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

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

[41] P. Berde *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. of ACM HotSDN 2014*, pp. 1–6, Aug. 2014.

[42] 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.

[43] 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.

[44] 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.

[45] B. Boughzala *et al.*, "OpenFlow supporting inter-domain virtual machine migration," in *Proc. of WOCN 2011*, pp. 1–7, May. 2011.

[46] S. Zhang *et al.*, "Fast network flow resumption for live virtual machine migration on SDN," in *Proc. of ICNP 2015*, pp. 446–452, Nov. 2015.

[47] C. Benet, K. Noghani, and A. Kassler, "Minimizing live VM migration downtime using OpenFlow based resiliency mechanisms," in *Proc. of CloundNet 2015*, pp. 27–32, Oct. 2016.

[48] L. Jiao *et al.*, "Smoothed online resource allocation in multi-tier distributed cloud networks," *IEEE/ACM Trans. Netw.*, vol. 25, pp. 2556–2570, Aug. 2017.

[49] 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.

[50] P. Pisa *et al.*, "OpenFlow and Xen-based virtual network migration," in *Proc. of IFIP 2010*, pp. 170–181, Jun. 2010.

[51] R. Mijumbi *et al.*, "Dynamic resource management in SDN-based virtualized networks," in *Proc. of CNSM 2014*, pp. 412–417, Nov. 2014.

[52] S. Ghorbani *et al.*, "Transparent, live migration of a software-defined network," in *Proc. of SOCC 2014*, pp. 1–14, Nov. 2014.

[53] 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.

[54] M. Zhang, C. You, and Z. Zhu, "On the parallelization of spectrum defragmentation reconfigurations in elastic optical networks," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 2819–2833, Oct. 2016.

[55] A. Freville, "The multidimensional 0–1 knapsack problem: An overview," *Eur. J. Oper. Res.*, vol. 155, pp. 1–21, May 2004.

[56] B. Korte and R. Schrader, "On the existence of fast approximation schemes," in *Nonlinear Programming 4*. Elsevier, 1981, pp. 415–437.

[57] H. Kellerer, U. Pferschy, and D. Pisinger, "Multidimensional knapsack problems," in *Knapsack Problems*. Springer, 2004, pp. 235–283.

[58] M. Held, P. Wolfe, and H. Crowder, "Validation of subgradient optimization," *Math. Program.*, vol. 6, pp. 62–88, Feb. 1974.

[59] D. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[60] 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.

[61] 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.