Make Big Data Applications More Reliable: Hitless vSDN Migration to Avoid TCAM Depletion

Sicheng Zhao, Yao Jin, Kai Han, Jie Yin, Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
†Email: {zqzhu}@ieee.org

***Abstract*—With network virtualization, an infrastructure provider can create virtual software-defined networks (vSDNs) over a shared substrate network and lease them to service providers (SPs). This enables the SPs to run their Big Data applications in a short time-to-market, flexible and cost-effective way. However, in a dynamic network, both the instances of vSDNs and the traffic in each vSDN can change over time, which would degrade the optimality of the embedding schemes of vSDNs and even cause serious reliability issues. Therefore, in this work, we extend our protocol-oblivious forwarding (POF) based network virtualization hypervisor (NVH) system (*i.e.*, PVX) to realize hitless vSDN migration to avoid ternary content addressable memory (TCAM) depletion. Specifically, we design the PVX system to realize the vSDN migration that is transparent to the controllers of vSDNs and would cause zero or very few packet losses. The proposed PVX is then implemented in a real network testbed, and we conduct experiments to verify its effectiveness.**

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

I. Introduction

Nowadays, to adapt to the fast development of Big Data applications, geographically-distributed datacenter (DC) systems have been built globally to deliver low-latency, high-quality and non-disruptive services to end users [1]. Meanwhile, the emerging of network virtualization technologies [2–4] enables service providers (SPs) to work around the high expenses of building and operating physical DC systems, and thus their Big Data applications can be supported in a short time-to-market, flexible and cost-effective way [5]. Specifically, with network virtualization, the infrastructure provider (InP) that owns a substrate network (*e.g.*, a physical geographically-distributed DC system) can solicit requests from SPs, create logically-isolated virtual networks (VNTs) over the substrate network accordingly, and lease the VNTs to the SPs on demand [6]. Note that, to better support the SPs' Big Data applications, we expect the VNTs to be software-defined networks (SDNs) such that enhanced network programmability can be guaranteed for application-aware network control and management (NC&M). Therefore, how to design and implement the network virtualization system that can slice virtual SDNs (vSDNs) effectively becomes an interesting and urgent problem to solve [7, 8].

To effectively slice vSDNs over a shared substrate network, one generally needs both a virtual network embedding (VNE) algorithm [9] and a network virtualization hypervisor (NVH) [10]. The VNE algorithm is responsible for dealing with the allocation of substrate resources (*i.e.*, switching capacity and flow-table space on substrate switches (S-SWs), and bandwidth on substrate links) to the virtual switches (vSWs) and links in vSDNs, and it has already been studied intensively in literature [9, 11]. The VNE results are realized by the NVH for vSDN slicing, which divides the flow-table space of each S-SW into logic regions accordingly, and serves as a proxy between the S-SWs and the virtual controllers (vCs) of the vSDNs for control message translation [10]. Previously, a few NVH systems have been designed and demonstrated, such as FlowVisor [12], OpenVirtex (OVX) [13], and PVX [7].

Note that, in a dynamic network environment, both the instances of vSDNs and the traffic flowing in each vSDN can change over time. This, however, might make the optimal VNE results implemented by the NVH not optimal anymore, and even lead to serious reliability issues [14]. For instance, the sudden traffic increase in a vSDN can congest the S-SWs on which it is embedded, and as the S-SWs are actually shared by multiple vSDNs, the traffic congestion will interrupt the services of all these vSDNs. Moreover, the ternary content addressable memory (TCAM) on each S-SW also gets shared by vSDNs to store the flow-tables on their vSWs, and it can be used up while the vSWs still have new flow-tables to be installed. This will cause the flooding of *PacketIn* messages to the related vCs and seriously impact the vSDNs' services.

The aforementioned reliability issues can be resolved by invoking vSDN migration to re-balance the resource utilization in the substrate network dynamically [14]. Fig. 1 provides an intuitive example on vSDN migration. Here, the original VNE schemes of *vSDNs* 1 and 2 make *v-SWs* b and a' share *S-SW* 3, and then, during network operation, we find that a TCAM depletion would happen on *S-SW* 3. To resolve the issue, we can migrate *v-SW* a' in *vSDN* 2 from *S-SW* 3 to *S-SW* 5 to re-balance the TCAM utilization in the S-SWs. Then, the virtual links (VLs) in *vSDN* 2 also need to be reconfigured to connect *v-SW* a' to *v-SWs* b' and c'. Although the procedure of vSDN migration is straightforward, to design an NVH that can support it effectively is still challenging due to two reasons. First of all, the vSDN migration should be made transparent to the vSDNs' vCs because the vCs interact with S-SWs through the NVH. Hence, if the vSDN migration can be solely handled by the NVH, simplified vSDN management and fast service recovery can be achieved. Secondly, we would expect the vSDN migration to be "hitless" to the traffic in the vSDN, *i.e.*, very few or even zero packet losses should be caused. However, to the best of our knowledge, the NVH system that

can successfully address the two challenges mentioned above has not been designed or demonstrated before.

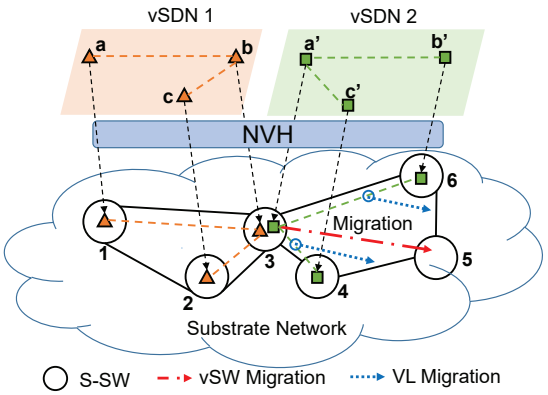


Fig. 1. Example on vSDN migration to avoid TCAM depletion.

In this work, we extend the protocol-oblivious forwarding (POF) based NVH system (*i.e.*, PVX) that we developed in [7] to realize hitless vSDN migration to avoid TCAM depletion. Specifically, we add a TCAM monitor in PVX to check the flow-table usage on each S-SW proactively. When the monitor determines that a TCAM depletion is about to happen, it will trigger a vSDN migration, which is transparent to the vSDNs' vCs and would cause zero or very few packet losses (*i.e.*, less than 0.5% according to the experimental results). We implement the proposed NVH system in a real network testbed and conduct several experiments to verify its effectiveness.

The rest of this paper is organized as follows. Section II provides a brief survey on the related work. The operation principle of the proposed NVH system is explained in Section III, and in Section IV, we discuss the experimental valuations. Finally, Section V summarizes our work.

II. Related Work

NVH is a key component to realize vSDN slicing, and it has already been considered in a few previous studies [10]. FlowVisor [12] is the first NVH that can realize the slicing of Openflow-based vSDNs. However, FlowVisor does not support the creation of vSDNs whose topologies are different from that of the substrate network, and this greatly restricts the flexibility of vSDN slicing. OVX [13] inherits the basic architecture of FlowVisor, but resolves the restriction on vSDNs' topologies. Later on, to realize protocol-independent vSDN slices, we extended OVX to realize PVX [7, 8] and make it support POF, which is a new SDN protocol that can realize the protocol-independent data plane [15]. Specifically, the key idea of POF is to process packet fields as $\{$*offset*, *length*$\}$ tuples, where *offset* is the start bit-location of a field in the packet and *length* tells the field's length in bits. Therefore, POF-based switches can manipulate any bits in a packet without being restricted by network protocols, and greatly enhance the programmability of data plane. Note that, vSDN migration has not been considered in FlowVisor, OVX and PVX.

Previously, vSDN migration has been investigated in [14, 16]. Nevertheless, both of the studies realized vSDN migration through the vSDNs' vCs, which is not very reasonable since the vCs should not know the resource usage in the substrate network and thus cannot determine the migration scheme (*i.e.*, when and where to migrate). More importantly, the experimental results in [14, 16] indicated that the vSDN migration mechanisms were not hitless.

III. Operation Principle

A. System Architecture

Fig. 2 shows the proposed system architecture to support hitless vSDN migration in PVX. Here, the PVX connects to all the S-SWs in the POF-based substrate network, and it works as the SDN controller from the perspective of the S-SWs. When the virtual network manager (VNMgr) receives a vSDN request from tenants, it will calculate the VNE scheme for the vSDN and pass the scheme to the PVX through the Restful API for implementation. After the vSDN has been sliced, its vC talks with the vSWs through the PVX for installing, updating and removing flow-tables. Then, during network operation, the TCAM monitor in PVX checks the TCAM usage on each S-SW proactively. If the TCAM monitor finds that a TCAM depletion is about to happen, it will inform the VNMgr to calculate the vSDN migration scheme. In our design, the vSDN migration is handled solely by the PVX without any participation of the vCs, *i.e.*, the whole process is made transparent to the vCs.

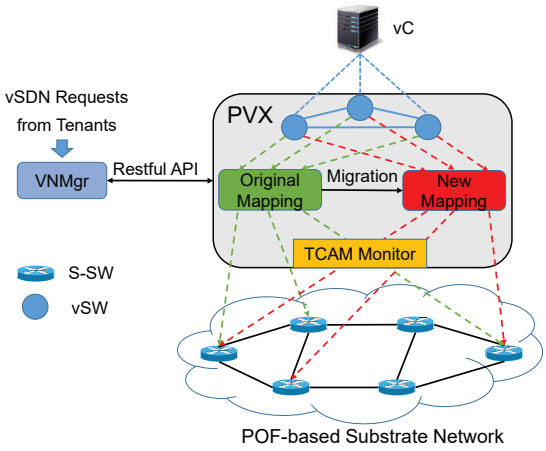


Fig. 2. Architecture of our PVX to support vSDN migration.

The packet format that we design to realize vSDN slicing with PVX is illustrated in Fig. 3. Specifically, to distinguish the packets in different vSDNs, PVX inserts a *Virtual Network Header* field (6 bytes) before the Ethernet header of each packet flowing in the vSDNs. The *Virtual Network Header* includes two subfields, *i.e.*, the *Tenant ID* (4 bytes) and *Link ID* (2 bytes) subfields. The *Tenant ID* indicates the vSDN to which the packet belongs, while the *Link ID* identifies the VL on which the packet is being transmitted.

B. Design of Functional Modules

Fig. 4 illustrates the functional modules designed in the PVX for realizing hitless vSDN migration.

Flow-Table Database (FT-DB): This module stores the flow-tables that have been installed in the vSDNs. Note that, the PVX translates the virtual flow-tables that are from the vCs and for the vSWs into the flow-tables that can be deployed on the S-SWs. Here, to realize hitless vSDN migration, FT-DB stores both types of flow-tables and the flow-entries in them in the forms of *TableMod* and *FlowMod* messages, respectively. Meanwhile, the mapping relations between the virtual and substrate flow-tables/flow-entries are also preserved in FT-DB.

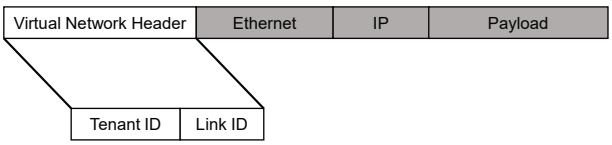


Fig. 3. Packet format to realize vSDN slicing with PVX.

VNE Database (VNE-DB): This module stores the mapping relations between the vSDNs and the substrate network, *i.e.*, the node and link mapping results.

TCAM Monitor: This module is responsible for monitoring the TCAM usages in the S-SWs proactively. Specifically, the TCAM Monitor allocates a counter for each S-SW to record the number of installed flow-entries, *i.e.*, the counter gets incremented by 1 every time when a vC installs a new flow-entry to the S-SW. Then, when the TCAM Monitor finds that the value of the counter exceeds a preset threshold, it will inform the VNMgr to calculate the vSDN migration scheme for avoiding TCAM depletion.

Migration API: This module implements the vSDN migration scheme sent from the VNMgr through the Restful API. Specifically, the Migration API can parse the vSDN migration scheme stored in the JSON file from the VNMgr, configure the related S-SWs to implement the scheme, and update the new network status in FT-DB and VNE-DB when the vSDN migration has been done.

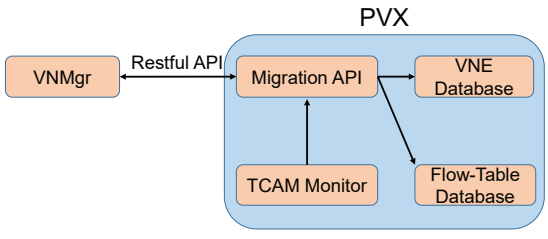


Fig. 4. Design of migration-related functional modules in PVX.

C. Design of vSDN Migration Procedure

In this subsection, we use the example in Fig. 5 to explain our design of the vSDN migration procedure. Here, the numbers aside the S-SWs and vSWs are the corresponding port numbers. It can be seen that the vSDN migration aims to migrate *vSW b* from *S-SW* 2 (*i.e.*, the original mapping) to *S-SW* 5 (*i.e.*, the new mapping), and in the meantime, the related VLs should be re-mapped too. The detailed procedure to achieve this vSDN migration is explained as follows, and a graphical illustration can be found in Fig. 6.

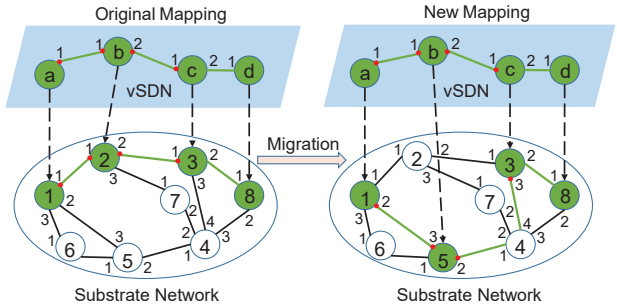


Fig. 5. An example on vSDN migration.

Step-1 (Changing Mapping Relations): We first change the mapping relation (*i.e.*, the node and link mapping results) between the vSDN and the substrate network in VNE-DB. Note that, link mapping also involves port mapping, *i.e.*, how to map the ports on vSWs to those on S-SWs. Hence, in this step, PVX changes the node, link and port mappings. In Fig. 5, the port mappings that need to be changed are marked as red, while the link mappings are marked as green.

Step-2 (Copying Flow-Tables/Flow-Entries): In this step, we extract all the flow-tables/flow-entries (*i.e.*, both the virtual and substrate ones) that are for the vSW(s) and S-SW(s) involved in the vSDN migration from FT-DB, store them in the temporary memory in PVX, and delete them from FT-DB. For instance, for the example in Fig. 5, all the virtual flow-tables/flow-entries for *vSW b* and all the substrate ones for *S-SW* 2 are coped to the temporary memory in PVX. Note that, we remove the original flow-tables/flow-entries in FT-DB after copying them to avoid dual-entries, since we will insert the new ones in FT-DB in subsequent steps.

Step-3 (Installing Flow-Tables/Flow-Entries on New S-SWs): In this step, PVX realizes the migration of vSW(s). Note that, the virtual flow-tables/flow-entries on the vSW(s) (*e.g.*, *vSW b* in Fig. 5) to be migrated has already been extracted and stored in the temporary memory, and they will not change during the vSDN migration. Hence, the PVX installs the substrate flow-tables/flow-entries on the new S-SW(s) by passing the corresponding virtual ones through the De-virtualization Module in PVX. The De-virtualization Module was developed in our previous work [7], and it can translate virtual flow-tables/flow-entries to substrate ones according to the mapping relations recorded in VNE-DB. Then, the obtained substrate flow-tables/flow-entries are installed on the new S-SW(s) (*e.g.*, *S-SWs* 4 and 5 in Fig. 5) by PVX.

Step-4 (Updating Flow-Entries on Related S-SWs): In the previous steps, PVX only tries to realize the new mapping results in the substrate network, but does not remove the original mapping results in it. Hence, we incorporate the "make-before-break" scenario [17] to realize hitless vSDN

migration. In other words, if there is traffic flowing in the vSDN, its routing path(s) in the substrate network would not be changed until this step. For example, if we assume that there is a flow running from *vSW* a to *vSW* d in the vSDN in Fig. 5, its routing path in the substrate network will stay as 1→2→3→8 until this step. Then, we take this flow as an example to explain how to update the flow-entries on S-SWs that carry the neighboring vSWs of vSW b for vSDN migration. PVX first takes the virtual flow-entries which contain *OUTPUT* action with the output port of 1 of *vSW* a out from FT-DB, changes their *FlowEntryCmd* (*i.e.*, the type of operation in a *FlowMod* message) from *OFPFC-ADD* to *OFPFC-MODIFY*. Then, the virtual flow-entries are sent to *S-SW* 1 through the De-virtualization Module, which will update the substrate flow-entries on *S-SW* 1 to change the output port of the aforementioned flow from 1 to 2. Next, the same procedure is applied to the substrate flow-entries on *S-SW* 3 too, and then the flow will take the routing path of 1→5→4→3→8 in the substrate network.

Step-5 (Removing Flow-Tables/Flow-Entries on Original S-SWs): In this step, PVX first gets the substrate flow-tables/flow-entries for the original S-SWs (*e.g.*, *S-SW* 2 in Fig. 5) from its temporary memory, changes the *TableModCmd* of the corresponding *TableMod* messages from *OFPTC-ADD* to *OFPTC-DELETE*, changes the *FlowEntryCmd* of the corresponding *FlowMod* messages from *OFPFC-ADD* to *OFPFC-DELETE*, and then send all the *FlowMod* and *TableMod* messages to the original S-SWs directly to remove the related flow-tables/flow-entries there.

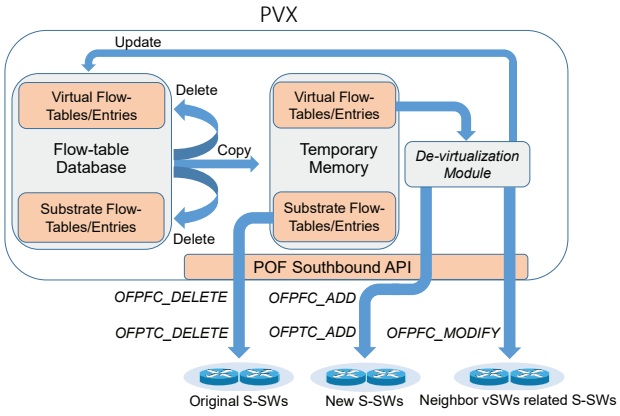


Fig. 6. Operations in PVX for vSDN migration.

IV. Experimental Valuations

This section discusses the experimental valuations of our proposed PVX system and analyzes the results. The PVX system is implemented in a real network testbed, with which we perform vSDN migration experiments with live traffic on. In the testbed, we realize the vCs by extending the POF controller developed in our previous work [15] and run them on commodity Linux servers, and each S-SW is implemented by running our software-based POF switch [15] on a high-performance Linux server.

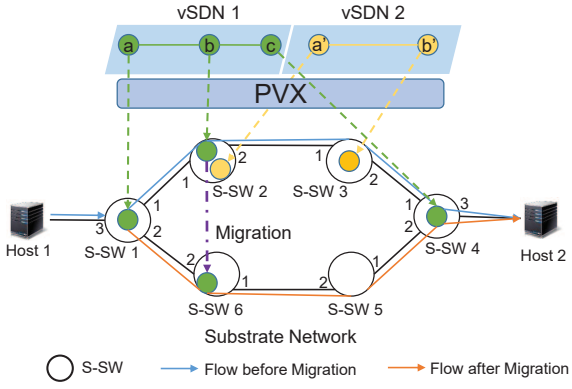


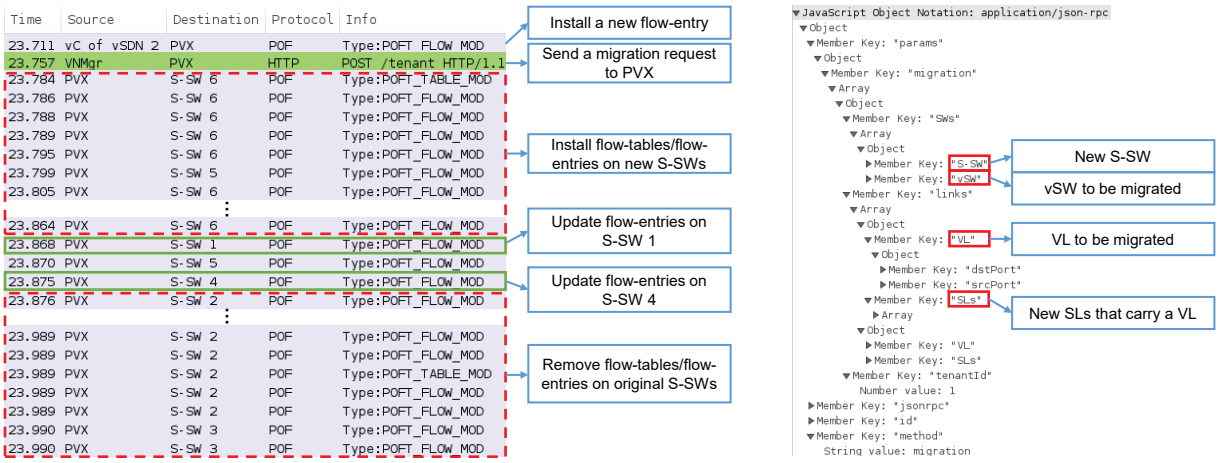
Fig. 7. Experimental setup for functionality verification.

A. Verification of Functionalities

We first conduct an experiment to verify that the proposed PVX system can realize hitless vSDN migration. Fig. 7 shows the experimental setup and scenario that we use for functionality verification. Here, the PVX system slices two vSDNs, *i.e.*, *vSDNs* 1 and 2, over the substrate network, and the VNE schemes are illustrated in Fig. 7. *vSW* b in *vSDN* 1 and *vSW* a' in *vSDN* 2 are both mapped onto *S-SW* 2. Note that, since the S-SWs are realized by software-based POF switches and there is no TCAM in them, we emulate the behavior of limited TCAM by limiting the maximum number of flow-entries that can installed in each S-SW below 1000. In the experiment, we first let the vCs of *vSDN* 1 and 2 install certain numbers of flow-entries on *vSWs* b and a', respectively, to use all the emulated TCAM on *S-SW* 2. Meanwhile, the service of a video streaming is running between *Hosts* 1 and 2 in *vSDN* 1. Then, we try to set up a new flow in *vSDN* 2 such that its vC would install a new flow-entry in *vSW* a'. Upon receiving the related *FlowMod* message from the vC, PVX determines that the TCAM on *S-SW* 2 would be insufficient and hence, it will trigger a vSDN migration to readjust the mapping of *vSDN* 1. Specifically, *vSW* b will be migrated from *S-SW* 2 to *S-SW* 6, as shown in Fig. 7.

The procedure of the vSDN migration is illustrated in the wireshark capture in Fig. 8(a). It can be seen that upon receiving the *FlowMod* message from the vC of *vSDN* 2, PVX determines that a vSDN migration would be necessary since otherwise a TCAM depletion would happen. Hence, PVX communicates VNMgr to determine the vSDN migration scheme, and the detailed message VNMgr to PVX to inform the vSDN migration scheme is shown and explained in Fig. 8(b). Next, PVX starts the vSDN migration according to the received scheme. Specifically, PVX follows the procedure discussed in Section III-C to accomplish the vSDN migration. The time-stamps shown in Fig. 8(a) suggest that the whole vSDN migration process only takes 133 msec. Also, the messages in Fig. 8(a) indicate that the vSDN migration only involves the PVX, VNMgr and S-SWs, which means that the migration is made transparent to the vCs of *vSDN* 1 and 2.

To verify that the vSDN migration is implemented suc-



(a) Messages captured for virtual network migration.

(b) Message from VNMgr to PVX to inform the vSDN migration scheme.

Fig. 8. Messages captured on experiment of functional verification.

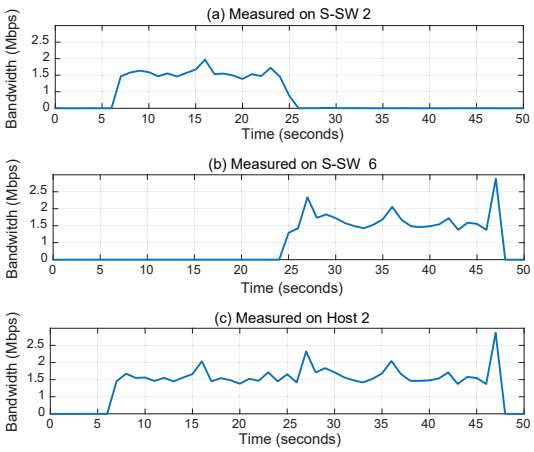


Fig. 9. Bandwidth of video streaming on *S-SWs* 2 and 6 and *Host* 2.

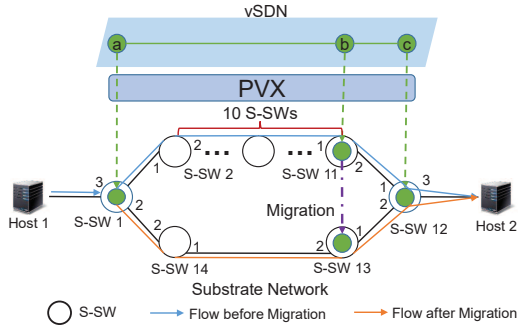


Fig. 10. Experimental setup for packet loss analysis.

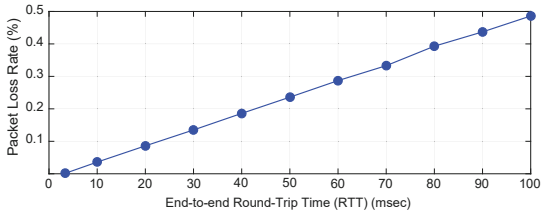


Fig. 11. Results on packet loss rate during vSDN migration.

cessfully in the network testbed and the operation of vSDN 1 would not be impacted by it, we measure the receiving bandwidth of the video streaming on *S-SWs* 2 and 6 and *Host* 2 and show the results in Fig. 9. The results on the receiving bandwidth on *S-SWs* 2 and 6 suggest that the vSDN migration is happened at $t = 25$ seconds, while the receiving bandwidth on *Host* 2 indicates that the service of the video streaming runs smoothly end-to-end before, during and after the vSDN migration. Therefore, the results in Fig. 9 verify that our PVX has implemented the vSDN migration successfully in the testbed and the overall process is hitless to the application traffic flowing in the migrated vSDN.

B. Performance on Packet Losses during Migration

Note that, even though our design of PVX can ensure hitless vSDN migration, a small amount of packet losses can still occur in **Steps-4** and **5** discussed in Section III-C. This is because the packets transmitted on the original substrate links can be dropped during the path switching conducted for the vSDN migration. More specifically, the probability of packet losses increases with the average transmission time of the packets. Hence, we design an experiment to measure the end-to-end packet loss rate during the vSDN migration when changing the average round-trip time (RTT) from *Host* 1 to *Host* 2. The experimental setup and scenario are shown in Fig. 10. This time, we include 12 S-SWs in the original substrate path, and then change the RTT from *Host* 1 to *Host* 2 artificially with Netem (*i.e.*, a tool for network traffic control). Next, in each experiment, we use iPerf to generate a UDP flow from *Host* 1 to *Host* 2 for 10 seconds, let the PVX to invoke a vSDN migration as shown in Fig. 10 at $t = 5$ seconds, *i.e.*, the vSW on *S-SW* 11 gets re-mapped to *S-SW* 13, and measure

the packet loss rates for different RTT in iPerf.

Fig. 11 shows the experiment results on packet loss rate *versus* RTT. It can be seen that in the worst case (*i.e.*, the RTT is 100 msec), the packet loss rate is still below 0.5%. In our future work, we will try to further optimize the design and implementation of our PVX to reduce the packet loss rate.

C. Performance on Migration Latency

Finally, we evaluate the PVX's performance on migration latency, *i.e.*, the time used to accomplish the vSDN migration. Specifically, we define the migration latency as the time period from when PVX receives the vSDN migration scheme from the VNMgr to when the scheme has been successfully implemented in the substrate network. The experimental setup and scenario are showed in Fig. 12. Here, we further stress our PVX system to let it 1) migrate multiple vSDNs simultaneously, and 2) move hundreds of flow-entries on each vSDN during the vSDN migration. Three experiments are conducted to migrate 1, 2, and 3 vSDNs simultaneously, and to realize fair comparisons, we make sure that the total number of flow-entries that are moved in each experiment is the same.

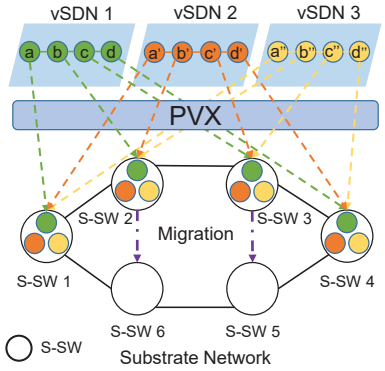


Fig. 12. Experimental setup for migration latency analysis.

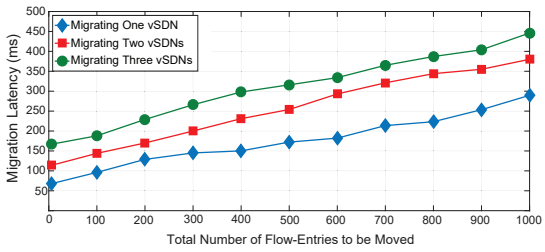


Fig. 13. Results on migration latency.

Fig. 13 shows the results on migration latency for different experimental scenarios. We observe that no matter how many vSDNs that we migrate simultaneously, the migration latency generally increases linearly with the number of flow-entries to be moved. As expected, the migration latency also increases with the number of vSDNs. In the worst case scenario, when there are three vSDNs to be migrated simultaneously and the total number of flow-entries to be moved is 1000, our PVX system takes ~ 450 msec to accomplish the vSDN migration.

V. Conclusion

In this work, we realized hitless vSDN migration in PVX to avoid TCAM depletion. Specifically, we added a TCAM monitor in PVX to check the flow-table usage on each S-SW proactively. When the monitor determined that a TCAM depletion is about to happen, it would trigger a vSDN migration. The whole process was made transparent to the vSDNs' vCs and would cause zero or very few packet losses. We implemented the proposed NVH system in a real network testbed and conducted several experiments to verify its effectiveness.

Acknowledgments

This work was supported in part by the NSFC Project 61371117, the Key Project of the CAS (QYZDY-SSW-JSC003), and the NGBWMCN Key Project under Grant No. 2017ZX03001019-004.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

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

[17] 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.