SR-PVX: A Source Routing based Network Virtualization Hypervisor to Enable POF-FIS Programmability in vSDNs

Shengru Li, Kai Han, Huibai Huang, Quanying Sun, Junjie Liu, Sicheng Zhao, Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—We design and implement a novel network virtualization hypervisor (NVH), which leverages source routing (SR) and provides network programmability based on the protocol-oblivious forwarding flow instruction set (POF-FIS), to realize protocol-independent virtual software-defined networks (vSDNs). The NVH, namely SR-PVX, uses a novel SR-based mechanism to ensure that the substrate switches can process packets from vSDNs with very small flow-table consumption. We design the network architecture, packet format and packet processing procedure, and implement them to experimentally demonstrate that SR-PVX can control substrate POF switches to accomplish SR-based vSDN slicing. We also perform an experiment to show that the POF-enabled vSDN created by SR-PVX can easily realize a stateful firewall with its POF-FIS programmability.**

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

I. Introduction

Nowadays, with the fast emergence of new applications, service providers are seeking short time-to-market, elastic and cost-effective solutions eagerly because supporting these applications with special-purpose physical network systems would not be realistic anymore, especially for small service providers. Hence, network virtualization has been proposed to allow service providers to lease logically-isolated virtual networks over a shared substrate network and to offer infrastructure providers a powerful way to slice their networks dynamically and adaptively [1–3]. Meanwhile, software-defined networking (SDN) [4] is surely an emerging paradigm for making the networks more flexible, programmable and application-aware. Although network virtualization and SDN are orthogonal to each other, recent studies have suggested that the symbiosis of them would bring more advantages in network programmability, adaptivity and scalability [5–7].

Therefore, how to realize virtual SDNs (vSDNs), *i.e.*, highly programmable SDN-enabled virtual networks, has attracted intensive research interests recently. Previously, to abstract substrate resources and virtualize network elements for tenants, people have considered to slice vSDNs with OpenFlow-based network virtualization hypervisor (NVH), *e.g.*, OpenVirteX [8]. Specifically, OpenVirteX allows tenants to customize the data plane topologies of their vSDNs and helps to bridge the communications between the tenants' OpenFlow controllers and the substrate network elements (*i.e.*, OpenFlow switches). Even though such OpenFlow-based NVHs have been proven to be effective, we hope to point out that the network virtualization system's programmability and flexibility can be further improved if the restrictions due to the protocol-dependent nature of OpenFlow could be removed.

As OpenFlow defines the matching fields in flow-tables based on existing network protocols (*e.g.*, Ethernet and IP), an OpenFlow switch has to be aware of the protocol headers to parse and match to the required fields. Hence, there would be compatibility issues if we need to create vSDNs to support protocols that have not been standardized in the latest OpenFlow specifications. Although the issues can be resolved by introducing OpenFlow extensions, it would take some time to get the extensions standardized. Moreover, the OpenFlow specifications are becoming more complicated due to the extensions. For instance, the number of supported matching fields increases from 12 to 44 from OpenFlow v1.0 to v1.5. To this end, it is desirable that NVHs could be protocol-independent and future-proof such that they can program vSDNs to support new protocols seamlessly. We can realize this by leveraging the existing efforts on protocol-independent forwarding (PIF) [9]. More specifically, protocol-oblivious forwarding (POF) [10, 11] and P4 [12] are the initial practices of PIF, which allow network operators to customize network forwarding protocols and behaviors in a much more flexible manner than OpenFlow.

As the ternary content addressable memory (TCAM) in SDN switches is expensive and power hungry, the flow-entries that can be stored in an SDN switch are usually very limited, *e.g.*, a commercial OpenFlow switch can generally carry less than 2000 flow-entries [13]. However, the flow-entries could be far from enough, since a top-of-rack (ToR) switch would need to require at least 78000 active flow rules to operate effectively if its rule timeout value is set as 60 seconds [14]. Therefore, previous studies have tried different ways to reduce the volume of flow-entries installed in SDN switches. Hu *et al.* [15, 16] converged the flows that share the same path segment with label-in-label encapsulation and forwarded them with a single flow-table on each related SDN switch. On the other hand, source routing (SR), that encodes the output port of each switch along the path in packets, has been approved to be an effective method to reduce the volume of installed flow-entries [17–19]. However, all these technologies concentrate

S. Li, K. Han, H. Huang, Q. Sun, J. Liu, S. Zhao 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 February 14, 2017.

on reducing the flow-entries for end-to-end path setup, while how to save flow-entries during vSDN formulation has not been investigated yet. Note that, as vSDNs need to use isolated flow-entry resources to forward the packets of different tenants, the shortage on flow entries could become even more significant.

In this paper, we design and implement a novel NVH that follows the idea of PIF. Specifically, we present a source routing based NVH, namely SR-PVX, to enable POF flow instruction set (POF-FIS [10]) based programmability in vSDNs. The advantages of our SR-PVX are summarized as follows.

- **Superior programmability**: To the best of our knowledge, SR-PVX is the first NVH that realizes the concept of PIF by supporting POF-FIS. Therefore, the slicing of vSDNs operates in a future-proof manner to provide tenants more freedom on the network innovations for new services and applications.
- **Complete virtualization of address space**: We design a source routing (SR) based header encapsulation scheme for SR-PVX to identify packets from each vSDN, while the actual packet formats in the vSDNs are made transparent to SR-PVX. Also, we design the header encapsulation scheme in the way that the additional overheads can be well controlled.
- **Small flow-table consumption**: We propose a novel SR-based mechanism for substrate switches to process packets from vSDNs with very small flow-table consumption. Specifically, one packet processing pipeline that only consists of few flow-entries can handle the packets from all the vSDNs on any intermediate switch in a substrate path that carries a virtual link.
- **Effective substrate resource isolation**: SR-PVX abstracts flow-tables as a type of substrate resources for the tenants, and make different tenants use non-overlapping flow-table space for better isolation. Moreover, we develop a POF-based meter function to isolate bandwidth usages of the vSDNs.

The rest of this paper is organized as follows. In Section II, we survey the related works. Section III gives a brief introduction on POF. In Section IV, we present the operation principles of SR-PVX. Section V evaluates SR-PVX with numerical simulations. The design, implementation, and experimental demonstration of SR-PVX are discussed in Section VI. Finally, Section VII summaries the paper.

II. Background and Motivations

A. Virtualization of SDN

The combination of network virtualization and SDN enables the network operator to slice a substrate network into several logically independent vSDNs for the tenants [20], and thus each tenant can operate a vSDN with its own protocol and network operating system and provide services to applications independently [21]. Therefore, new network services can be deployed quickly and the development of novel networking technologies can be expedited.

There are several existing solutions to realize vSDNs. FlowVisor [22] is the first NVH for slicing OpenFlow-enabled vSDNs. It introduces the concept of "*flowspace*", which means that each vSDN has to share the same header space with others and overlapped header space usage has to be avoided. This, however, limits the scalability and flexibility of vSDN slicing. Moreover, FlowVisor does not allow vSDNs to have topologies that are different from the substrate one. OpenVirteX (OVX) [8] provides much more enhanced functionalities on network virtualization, since it allows each vSDN to have a customized topology and complete header space. Nevertheless, as the aforementioned solutions are all based on OpenFlow, they cannot efficiently support protocol-independent vSDN slicing. Note that, one important feature of vSDN slicing is to support network innovations, which apparently cannot be realized if each vSDN can only be based on an existing protocol.

B. Source Routing in SDN

People has already verified that the volume of flow-entries in an SDN can be reduced significantly with SR [17], and SecondNet, which is a data-center network architecture that uses port-switching SR to forward packets among servers, has been proposed in [23]. Note that, SecondNet encodes the SR-related fields into MPLS header fields, which might lead to significant overheads. To avoid the overheads, Path Switching [18] proposed to rewrite the source/destination MAC address fields to encapsulate a forwarding path in the SR manner. More recently, Jin *et al.* designed an SR-based data-center network architecture named Sourcey [19], which tried to simplify the switch operations to an extreme case that only includes popping out an SR label and forwarding the packet out.

These existing SDN-based SR techniques usually have a large transmission overhead. Basically, due to the limited programmability of OpenFlow, they can only leverage the headers of existing protocols to encode the path information, which are not specifically designed for SR and thus might have redundant fields. Thanks to the protocol-independent nature of POF, we can tailor the packet header to meet the requirement of SR exactly and improve the transmission efficiency to the maximum extent [24].

III. Overview of POF

As an SDN technology, POF also uses logically centralized controllers residing in the control plane to manage the forwarding devices in the data plane through a south-bound protocol. The new idea brought by POF is to abstract all the objects to be processed in the data plane as $\{$*offset, length*$\}$ tuples. Specifically, when a POF-based forwarding device uses the "match-and-act" principle to process packets, it uses $\{$*offset, length*$\}$ tuples as the search keys of matching fields, where *offset* indicates the start bit-location of the field in a packet and *length* denotes the length of the field in bits. Based on this, POF defines a flow instruction set, *i.e.*, POF-FIS [10], to assist the packet processing in forwarding devices. As all the instructions in POF-FIS locate the data in a packet with $\{$*offset, length*$\}$ tuple, they can manipulate any bits in the packet without being affected by any protocol-dependent restrictions. Hence, POF greatly enhances the programmability of the data plane and can truly make SDN networks future-proof.

Then, the detailed operating procedure of a POF-enabled SDN network is as follows. First of all, all the used protocols and the fields in the metadata memory should be defined in {*offset, length*} tuples. Here, to assist POF switches to cache flow information temporarily, POF suggests to implement a metadata memory in each switch and defines several metadata-related instructions. Secondly, to realize a packet processing task, a flow-table that specifies one or more matching fields in packets or the metadata memory is installed in switch(es). Finally, flow-entries are inserted into the flow-tables, each of which contains several {*offset, length, value*} tuples and a series of instructions in POF-FIS. Here, the {*offset, length, value*} tuple indicates a matching fields and its value, and the matching field should comply with the format defined in the corresponding flow-table.

In this work, our SR-PVX defines the SR-based header fields in {*offset, length*} tuples, and utilizes several flow-tables and the metadata memory to come up with several pipelines for processing the packets from vSDNs.

IV. Operation Principles

A. Source Routing in Substrate Network

It is known that SR encodes routing path information into the header of each packet, *i.e.*, encapsulating a series of output ports in the header to represent the forwarding action on each hop along the routing path. Then, each switch on the routing path just needs to pop out the outmost header field and forward the remaining part of the packet to the output port encoded in the header field. Hence, multiple flows can share the same flow-table in a core switch, and this helps to reduce the flow-table consumption in SDN networks significantly [17–19]. As the TCAM in SDN switches is usually very limited, SR can greatly improve its utilization efficiency and make SDN networks much more scalable.

The aforementioned advantage of SR becomes even more valuable when it comes to consider the creation and operation of vSDNs. Basically, in addition to the bandwidth resources on substrate links, vSDNs also consume TCAM in substrate switches. If the NVH does not try to aggregate the flow-tables used by vSDNs and installs one or more flow-entries for each vSDN's packet flow in every substrate switch that are used to carry it (*e.g.*, the scheme used in OpenVirteX), the TCAM resources in the substrate network would run out easily. Consequently, the infrastructure provider might have to face the dilemma that no more vSDNs can be provisioned, even though the substrate bandwidth resources are still abundant. This actually motivates us to leverage SR to realize the virtual link mapping in vSDN slicing.

B. Network Architecture

Fig. 1 illustrates the overall network architecture for realizing POF-enabled vSDNs with SR-PVX. SR-PVX communicates with all the substrate POF switches to abstract the substrate resources (*i.e.*, bandwidth on substrate links and TCAM on substrate switches). It also receives vSDN mapping tasks from the network embedder, which takes the vSDN requests from tenants and calculates the virtual network embedding

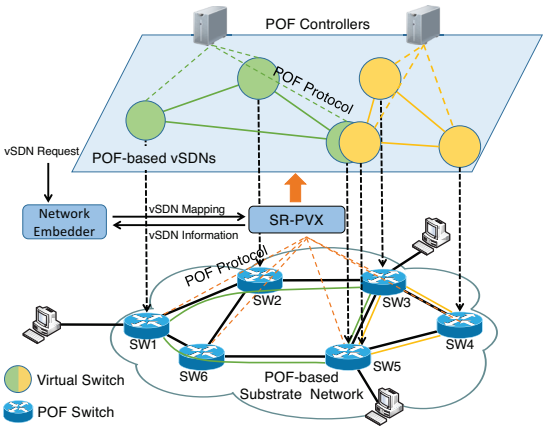

Fig. 1. Realizing POF-enabled vSDNs with SR-PVX.

(VNE) schemes for them. When creating a vSDN according to the mapping scheme from the network embedder, SR-PVX instantiates virtual POF switches over designated substrate switches and establishes virtual links (VLs) through installing SR-based flow-entries. With SR, SR-PVX only needs to install specific flow-entries on the substrate switches that actually carries the virtual switches of the vSDN, while a conventional NVH (*e.g.*, OpenVirteX) needs to install such flow-entries on all the substrate switches involved to carry the vSDN's VLs. Actually, SR-PVX pre-installs a fixed number of default flow-entries on all the substrate switches during substrate network initialization, which can process packets from all the vSDNs on the substrate switches that are involved in VL mapping but not carry virtual switches (*i.e.*, as intermediate substrate switches on the substrate paths to carry VLs). We will explain the default flow-entries in Section 3.4. For instance, in Fig. 2(a), to forward flows from host H1 to host H2 for a vSDN, the conventional NVH needs to install specific flow-entries for the flows on every substrate switch along the path SW1→SW2→SW3→SW4 →SW5→SW6. While our SR-PVX only needs to install such flow-entries with a POF-FIS programmed SR header encapsulating operation on SW1, SW4 and SW6 for all the vSDN's flows from H1 to H2 (as shown in Fig. 2(b)). Finally, when the data plane is established for the vSDN, a POF controller is instantiated for it, and at this moment, the vSDN is created and it is then handed over to the operator of its tenant.

C. Packet Design

For distinguishing the packets from different tenants, previous studies on NVHs tried to either leverage the existing protocol header to encapsulate the vSDN packets or replace the virtual addresses by physical ones. These schemes, however, bear drawbacks such as bandwidth inefficiency or incomplete virtualization of the address space. Fortunately, with the flexibility provided by POF, we can customize the packet format used by the network virtualization system based on SR-PVX to achieve not only high bandwidth efficiency but also complete virtualization of the address space.

Fig. IV-C shows the design of the packet format for network virtualization by SR-PVX. The *NV_SR_Header* field is used

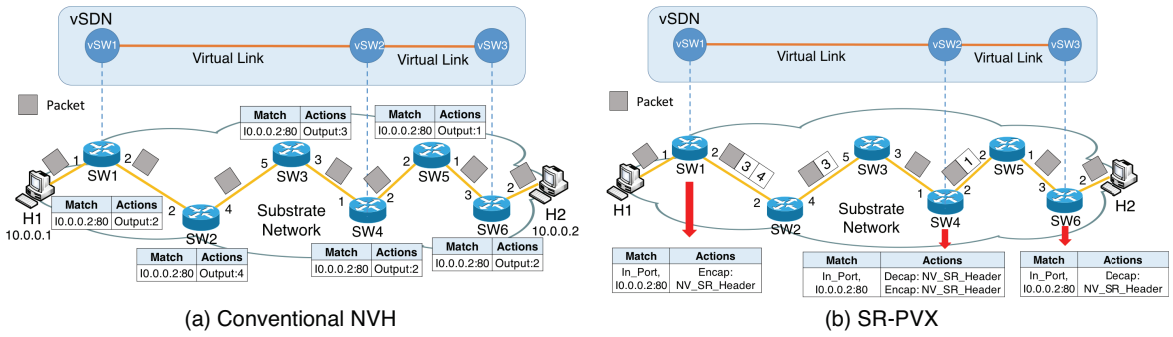

Fig. 2. Network virtualization related flow-entry installation by conventional NVH and SR-PVX.

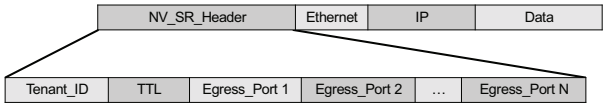

Fig. 3. Packet design for network virtualization by SR-PVX.

to distinguish the packets from different tenants, which locates before the Ethernet header and includes three types of fields. The *Tenant_ID* field (16 bits) stores the tenant's globally-unique ID, which is assigned by SR-PVX during the creation of its vSDN. Substrate switches use this field to identify the packets' ownership and then process them with the flow-tables installed for the corresponding vSDNs. The *TTL* field (8 bits) indicates the number of remaining hops before the packet reaching the next virtual switch. The *Egress_Port* fields (8 bits) encode the designated output ports in sequence for the substrate switches that are used to carry a VL. Hence, each *NV_SR_Header* includes multiple *Egress_Port* fields, when the substrate path to carry a VL consists of more than one hops.

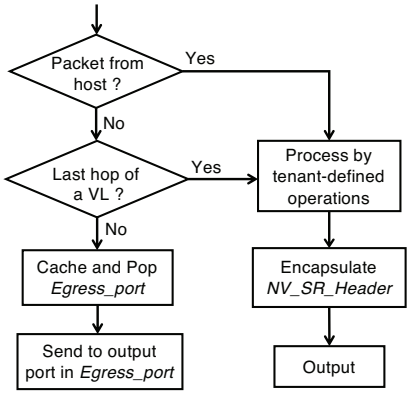

Fig. 4. Flowchart for packet processing in substrate switches.

D. Packet Processing Procedure

Fig. 4 illustrates the packet processing procedure that we designed for the substrate switches controlled by the SR-PVX. First of all, we consider a packet that just enters the substrate POF network from a host, the switch determines the tenant (*i.e.*, the vSDN) to which it belongs with the packet's input port, *i.e.*, *In_Port*. Then, the packet is processed by the tenant-defined flow-tables to execute the instructions defined for an ingress virtual switch of the vSDN. Next, the switch inserts an *NV_SR_Header* into the packet, which stores the designated output ports on subsequent substrate switches along the substrate path that carries the VL starting from the current ingress virtual switch, and forwards the packet out to the next hop. Inside the substrate network, when a substrate switch receives a packet from another switch, it distinguishes the packet's vSDN with the *Tenant_ID* field in its *NV_SR_Header*. Then, the switch determines whether itself is the last hop of a VL in the vSDN by checking the value of the *TTL* field. If yes, the packet is processed by the tenant-defined flow-tables with the similar procedure as for a packet that just enters the substrate network, *i.e.*, a new *NV_SR_Header* is calculated and used to replace the current one. Otherwise, the switch caches the value of the outmost *Egress_Port* field in its metadata memory, pops the field from *NV_SR_Header*, and forwards the packet to the designated output port cached before.

To realize the packet processing procedure, we program flow-table pipelines with POF-FIS in the substrate switches, as explained in Fig. 5. Specifically, for each substrate switch, the incoming packets are classified into three categories, *i.e.*, packets from host, packets at an end-node of a VL, and packets at the intermediate switch of a VL. We program a pipeline for the packets in each category. Here, packets from all the vSDNs share the first three flow-tables in Fig. 5 to be categorized, while the remaining flow-tables are programmed to realize tenant-defined operations for each vSDN.

The first flow-table in Fig. 5 is the "*In_Port* Match Table", which is first checked when a packet enters a substrate switch. Specifically, this flow-table checks the input port of the packet to determine where it comes from. If the input port corresponds to a host of a vSDN, the switch first uses the *Write-metadata* instruction to write the vSDN's *Tenant_ID* in the metadata memory, and then lets the packet be processed by the tenant-defined flow-table(s). Otherwise, the packet should have an *NV_SR_Header* encoded on it and would be processed by the "*TTL* Match Table".

The *TTL* Match Table decides whether the current switch is the last hop on the substrate path for a VL by checking the value of *TTL* field. If not (*i.e.*, $TTL > 0$), the switch applies SR-based forwarding as shown in the second flow-entry of the *TTL* Match Table in Fig. 5. Specifically, the operation of "cache and pop *Egress_Port*" in Fig. 4 is realized by leveraging the *Write-*

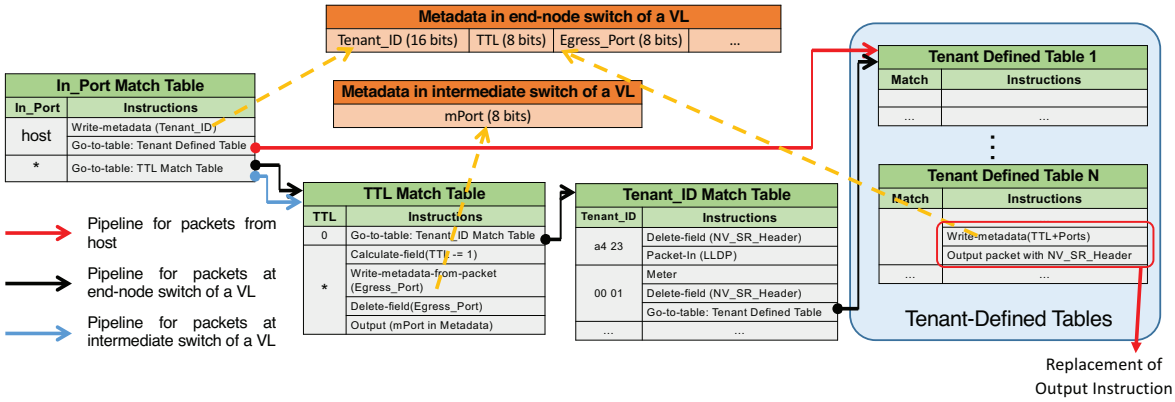

Fig. 5. Packet processing pipelines.

metadata-from-packet and *Delete-field* instructions, while the *Output* instruction sends the packet to its designated output port whose port ID is the value of the popped *Egress_Port* field (*i.e.*, the one cached in *mPort* in the metadata memory). Then, the processing is done for the packet. On the other hand, if we have $TTL = 0$, the packet will be processed by the "*Tenant_ID* Match Table", which matches to the *Tenant_ID* field to determine the packet's vSDN and then lets it to be processed by the corresponding tenant-defined flow-table(s).

The *Tenant_ID* of a vSDN is assigned by SR-PVX but we reserve *Tenant_ID* $= 0xa423$ for the link discovery in SR-PVX. If *Tenant_ID* indicates a valid vSDN, the *Tenant_ID* Match Table first applies the *Meter* instruction to limit the bandwidth usage of the packet's flow according to its service-level agreement (SLA). Then, the switch removes the packet's *NV_SR_Header* and sends it to the corresponding tenant-defined flow-table(s).

For each vSDN, the tenant-defined flow-tables are installed in the virtual switches by the vSDN controller, *i.e.*, "Tenant-Defined Tables" in Fig. 5. When the vSDN is operational, its controller installs flow-tables to its virtual switches, which are translated by SR-PVX. Specifically, for every *Table-Mod* message for installing a flow-table, SR-PVX maps the virtual table ID in the message to a physical one and ensures that each tenant-defined flow-table is used exclusively by one vSDN. SR-PVX also replaces each of the *Output* instructions in the flow-tables with the *NV_SR_Header* encapsulating operation, which encodes *Egress_Port* fields into *NV_SR_Header* according to the virtual output ports instructed by the vSDN controller. Here, for a virtual switch in the vSDN, a virtual output port on it corresponds to a VL, and thus with the link mapping scheme stored in SR-PVX, the substrate path to carry the VL can be determined for encoding the *Egress_Port* fields.

V. Numerical Evaluation

We use numerical simulations to verify SR-PVX's benefit on flow-entry saving. The substrate network uses the Internet2 NDDI topology in Fig. 7, which contains 11 nodes. In the simulations, we change the number of virtual nodes (*i.e.*, virtual SDN switches) in each vSDN and compare the total number of flow-entries that are consumed by SR-PVX and

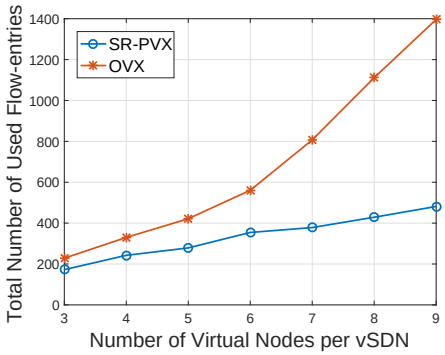

Fig. 6. Simulation results on flow-entries used in the substrate network.

OVX. The VNE algorithm to slice each vSDN is modified from the one developed in [25]. Each vSDN request reports its virtual topology, the bandwidth requirement of each VL, and the flow-entry demand on each virtual switch. The virtual topology of each vSDN is generated by the GT-ITM tool [26] with a connectivity ratio of 0.5, and the number of flow-entry demand on each virtual switch is set randomly within $[10, 100]$. In order to ensure sufficient statistical accuracy, we average the results from 10 independent simulations to obtain each data point.

Fig. 6 shows the simulation results. We observe that compared with OVX, SR-PVX reduces the number of used flow-entries significantly when the size of each vSDN is fixed. This is because with SR-PVX, a vSDN only consumes flow-entries on the substrate nodes that have its virtual switches embedded on, while OVX needs to reserve flow-entries for each vSDN on all the substrate nodes that carry its VLs. Moreover, the advantage of SR-PVX becomes more significant when the size of vSDNs increases. This is due to the fact that to satisfy the bandwidth constraint, the VLs in a larger virtual topology are more likely to be mapped onto relatively long substrate paths.

VI. System Implementation and Experimental Demonstration

We implement SR-PVX based on OpenVirteX (OVX) [8]. Specifically, we realize the POF protocol stack in OVX to develop the south- and north-bound interfaces. We also modify

the link discovery module in OVX and make it comply to the data plane operation defined by SR-PVX, *i.e.*, an LLDP packet is encoded with a special *NV_SR_Header* using *Tenant_ID* $= 0xa423$ and *TTL* $= 0$. Flow-table translation is an important feature of NVH, and our SR-PVX maintains a table to map the virtual table IDs in virtual switches to the physical ones in substrate switches. For each vSDN, we realize its POF controller by extending the ONOS platform [27] and the substrate switches are our home-made ones [11] with Intel data plane development kit (DPDK). In each substrate switch, we isolate the flow-tables used by different tenants since their virtual switches might use different matching fields to forward packets. We also implement the meter function in each switch to isolate bandwidth usages of vSDNs. Specifically, the meter module takes the vSDNs' bandwidth requirements as inputs and configures the *Meter* instruction in the *Tenant_ID* Match Table to limit their bandwidth usages.

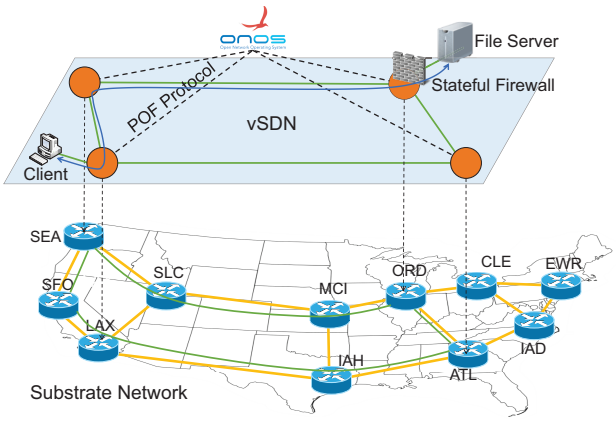

Fig. 7. Experimental setup.

To verify the functionality of our proposed SR-PVX, we perform an experiment to slice a vSDN with it from a POF-enabled substrate network and demonstrate a use case of simple stateful firewall application on the vSDN. Fig. 7 shows the configuration of the substrate network, which takes the 11-node Internet2 NDDI topology. The eleven substrate switches are our home-made POF switches, each of which is implemented on a stand-alone high-performance Linux server. The SR-PVX is also realized on a Linux server, which is directly connected to the substrate switches. The vSDN consists of 4 virtual switches and 4 VLs, and its node and link mapping scheme are also shown in Fig. 7.

Fig. 8 shows the message interchanging that we capture by Wireshark on SR-PVX, for revealing the procedure of vSDN creation. Firstly, the network embedder sends the vSDN mapping task to SR-PVX through several HTTP packets. Then, SR-PVX creates the vSDN accordingly and connects each virtual switch to the POF controller realized with ONOS. As Fig. 8 indicates, SR-PVX uses 4 different TCP ports to connect to the POF controller, where each port is for a virtual switch. It is known that stateful firewall helps to realize intelligent access control by tracking the states of TCP connections. The ability to identify the flags in a TCP header is the basic requirement on the switch that operates as a stateful

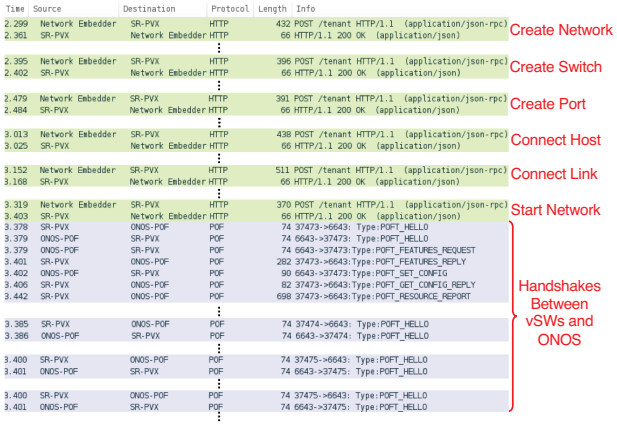

Fig. 8. Messages captured for vSDN creation.

firewall. In a POF-enabled network, this can be easily realized by letting the controller to program POF switches with POF-FIS. Hence, in the experimental demonstration, we consider the scenario in which the POF-enabled vSDN utilizes a stateful firewall to limit the number of simultaneous TCP connections from a client to a file server, for ensuring bandwidth fairness among all the clients. As shown in Fig. 7, the controller instructs the virtual switch that directly connects to the server to realize the stateful firewall.

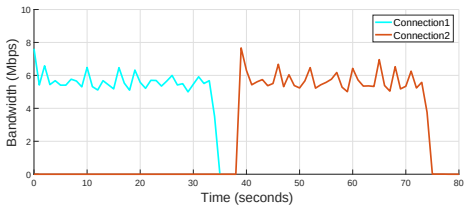

Fig. 9. Results on downloading bandwidth.

The firewall in the switch identifies a TCP connection based on the classic TCP "five-tuple" (*i.e.*, source/destination IP addresses, source/destination ports, and protocol). The controller maintains a counter table to record the connections from a same IP address and uses it to limit simultaneous connections from a client. Specifically, at any time, only one TCP connection is allowed from the client to the server. To realize this, the virtual switch checks the TCP FIN flag in packets and sends a *PacketIn* message to the controller to indicate that an active TCP connection is terminated. Then, the controller will allow to establish the next TCP connection from the client. In the experiment, we use Axel [28], which is a multi-thread tool, to download a file whose size is 40 MB from the server, and each VL in the vSDN can use 6 Mbps bandwidth at most. Fig. 9 shows the experimental results on downloading bandwidth. Even though we configure Axel to download the file with two simultaneous connections (*i.e.*, Axel partitions the file into two parts for downloading), only one connection is allowed at a time. We also observe that the *Meter*-based bandwidth restriction has been applied correctly to the vSDN.

We also trace a packet sent from the server and show the

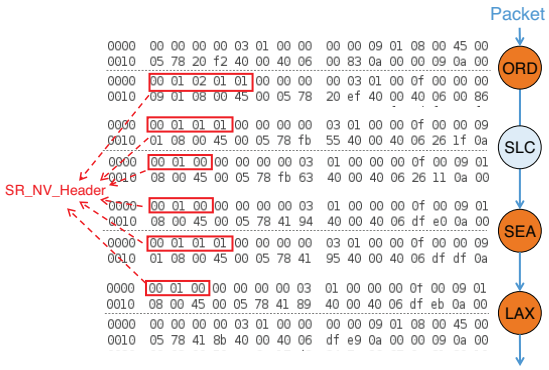

Fig. 10. Packet tracing from server to client.

packet's format at each type of substrate switch in Fig. 10. The data in the rectangles is for the *SR_NV_Header* fields. The *Tenant_ID* assigned to the vSDN is $0x0001$. The packet arrives at ingress virtual switch, where the substrate switch at ORD (as shown in Fig. 7) determines that its next virtual switch in the vSDN is located on the substrate switch at SEA. Hence, the packet is encapsulated in a *NV_SR_Header* with two *Egress_Port* fields. Here, the substrate switch at SLC is an intermediate switch for the VL to deliver the packet, and thus it simply forwards the packet with SR. As shown in Fig. 10, after passing though the substrate switch at SLC, the value of *TTL* field in the packet header has been decreased to 0 and there is no *Egress_Port* field in its *NV_SR_Header*. When the packet reaches the substrate switch at SEA, it will enter another VL in the vSDN and hence it is processed by the tenant-defined flow-tables and encapsulated in a new *SR_NV_Header*. Then, after being forwarded by the switch at SFO, the packet arrives at the switch at LAX, which is its last hop in the substrate network. This switch restores the packet to a common TCP format and forwards it to the client. Finally, we can see that the packet is forwarded correctly according to our design.

VII. Conclusions

In this paper, we proposed SR-PVX, which is an SR-based NVH to realize the slicing of POF-enabled vSDNs. Specifically, we designed the network architecture, packet format and packet processing procedure, evaluated the performance with a numerical simulation and implemented them to experimentally demonstrate that SR-PVX could control substrate POF switches to accomplish the SR-based vSDN slicing with proposed benefits. Moreover, we showed that the POF-enabled vSDN created by SR-PVX could easily realize a stateful firewall with its POF-FIS Programmability.

Acknowledgments

This work was supported in part by NSFC Project 61371117, Natural Science Research Project for Universities in Anhui (KJ2014ZD38), and Strategic Priority Research Program of the CAS (XDA06011202).

References

[1] T. Anderson *et al.*, "Overcoming the Internet impasse through virtualization," *IEEE Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

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

[3] 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] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[5] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, Nov. 2013.

[6] 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.

[7] 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.

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

[9] Protocol Independent Forwarding. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/OF-PI__A_Protocol_Independent_Layer_for_OpenFlow_v1-1.pdf

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

[11] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw., in Press*, 2016.

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

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

[14] A. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[15] D. Hu *et al.*, "Design and demonstration of SDN-based flexible flow converging with protocol-oblivious forwarding (POF)," in *Proc. of GLOBECOM 2015*, pp. 1–6, Dec. 2015.

[16] ——, "Flexible flow converging: A systematic case study on forwarding plane programmability of protocol-oblivious forwarding (POF)," *IEEE Access*, vol. 4, pp. 4707–4719, 2016.

[17] S. Jyothi, M. Dong *et al.*, "Towards a flexible data center fabric with source routing," in *Proc. of ACM SOSR 2015*, pp. 10:1–10:8, Jun. 2015.

[18] A. Hari *et al.*, "Path Switching : Reduced-state flow handling in SDN using path information," in *Proc. of CoNEXT 2015*, pp. 1–7, Dec. 2015.

[19] X. Jin *et al.*, "Your data center switch is trying too hard," in *Proc. of ACM SOSR 2016*, pp. 12:1–12:6, Mar. 2016.

[20] G. Long, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. of INFOCOM 2014*, pp. 1–9, Apr. 2014.

[21] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 655–685, Firstquarter 2016.

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

[23] C. Guo *et al.*, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. of CoNEXT 2010*, pp. 15:1–15:12, Nov. 2010.

[24] S. Li, D. Hu, W. Fang, and Z. Zhu, "Source routing with protocol-oblivious forwarding (POF) to enable efficient e-health data transfers," in *Proc. of ICC 2016*, pp. 1–6, Jun. 2016.

[25] X. Cheng *et al.*, "Virtual network embedding through topology-aware node ranking," *Comput. Commun. Rev.*, vol. 41, no. 2, pp. 38–47, Apr. 2011.

[26] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. of INFOCOM 1996*, vol. 2, pp. 594–602, Mar. 1996.

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

[28] Axel. [Online]. Available: http://axel.alioth.debian.org