Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability

Shengru Li, Daoyun Hu, Wenjian Fang, Shoujiang Ma, Cen Chen, Huibai Huang, and Zuqing Zhu

Abstract

Software-defined networking separates the control and forwarding planes of a network to make it more programmable and application-aware. As one of the initial implementations of SDN, OpenFlow abstracts a forwarding device as a flow table and realizes flow processing by applying the "match-and-act" principle. However, the protocol-dependent nature of OpenFlow still limits the programmability of the forwarding plane. Hence, in this article, we discuss how to leverage protocol-oblivious forwarding (POF) to further enhance the network programmability such that the forwarding plane becomes protocol-independent and can be dynamically reprogrammed to support new protocol stacks seamlessly. We first review the development of OpenFlow and explain the motivations for introducing POF. Then we explain the working principle of POF, discuss our efforts on realizing the POF development ecosystem, and show our implementation of POF-based source routing as a novel use case. Finally, we elaborate on the first WAN-based POF network testbed that includes POF switches located in two cities in China.

Introduction

Over the past decade, the Internet's rapid development has pushed the scales of end users, networking applications, and network elements to grow exponentially. Meanwhile, due to the competitive tussle among different stakeholders, the network architecture has become more and more complicated, brought to "ossification." Consequently, the introduction of new protocols and services can be slowed down and even impeded. In response to these issues, software-defined networking (SDN) has emerged as a new paradigm to change how the Internet operates [1]. Specifically, SDN separates the control and forwarding planes of a network, and leverages centralized network control and management (NC&M) to make it more programmable, flexible and application-aware. Therefore, new networking protocols and services can easily be developed with the forwarding plane abstraction provided by the control plane.

As one of the initial implementations of SDN, OpenFlow [2] provides a powerful tool set for network operators to program and manage their networks adaptively [3]. However, its protocol-dependent nature still limits the programmability of the forwarding plane. Specifically, OpenFlow defines the matching fields in flow tables according to existing network protocols (e.g., Ethernet and IP). Therefore, OpenFlow switches need to understand the protocol headers to parse packets and perform flow matching, which may cause serious compatibility issues when new protocols try to add or remove header fields. Hence, it is desirable that the network programmability can be further enhanced such that the forwarding plane is protocol-independent and can be dynamically reprogrammed to support new protocol stacks seamlessly. Following this idea, recent studies have proposed a few new SDN technologies, such as protocol-oblivious forwarding (POF) [4] and programming protocol-independent packet processors (P4) [5]. The basic idea behind POF and P4 are similar, as they both try to decouple network protocols from packet forwarding and make the forwarding plane reconfigurable, programmable, and future-proof.

More specifically, POF introduces a protocol-independent instruction set, which allows a network operator to define the protocol stack and packet processing procedure in a much more flexible manner than that in the current OpenFlow specifications, while P4 designs a high-level language to program an SDN switch more flexibly for protocol innovations. As POF defines the southbound interface that can be treated as a promising enhanced version of OpenFlow, we consider it in this work and try to demonstrate that network innovations can easily be realized with it.

This article discusses how to enhance the programmability of forwarding plane in SDN with POF. We first review the development of OpenFlow and explain the motivations for introducing POF. Then we discuss our efforts on realizing the POF development ecosystem, and show our implementation of POF-based source routing as a novel use case. Our work to build the first WAN-based POF network testbed, which includes POF switches located in two cities in China, is also presented. Finally, we summarize the article.

Review of OpenFlow

Since its inception, OpenFlow has used a protocol-dependent forwarding plane. Specifically, the evolution of OpenFlow specification takes a reactive approach instead of a proactive one, that is, all the matching fields and actions are defined based on existing protocols. This means that we have to update the matching fields and corresponding actions constantly as it becomes more

The authors are with the University of Science and Technology of China.

Digital Object Identifier: 10.1109/MNET.2017.1600030NM

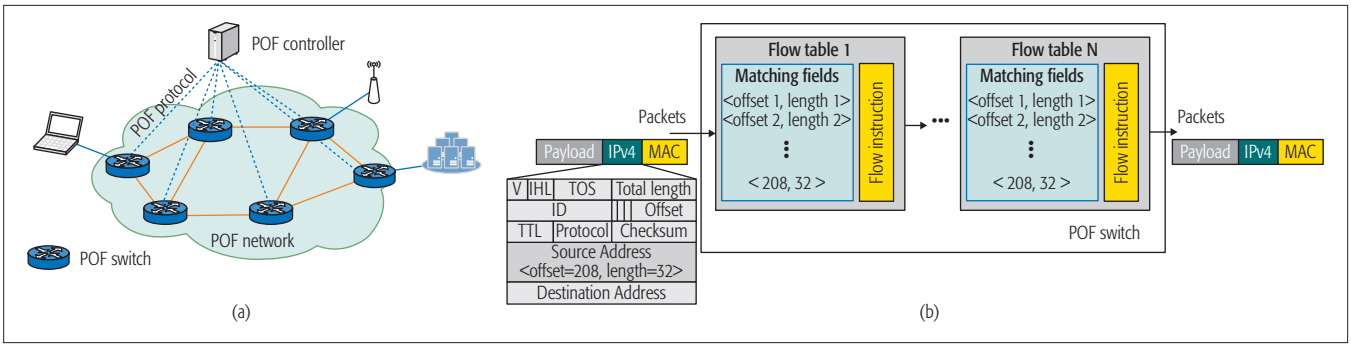

FIGURE 1. Overview of POF: a) architecture of a POF-based network; b) packet forwarding procedure in POF.

widely deployed with the need to support more protocols. Hence, the numbers of actions and matching fields keep increasing with the evolution of OpenFlow specification; for example, the latest OpenFlow v1.5 has to support 44 matching fields and 19 actions, while the numbers in OpenFlow v1.0 were 12 and 10, respectively. Therefore, when we want to support a new network protocol with OpenFlow, we have to check whether it can leverage the existing matching fields and actions in the latest OpenFlow specification. If not, we have to extend the specification and wait for the new version to be standardized. Apparently, this would inevitably make the OpenFlow specification more and more complicated, which would eventually restrict the innovation capability of OpenFlow. Hence, to enhance OpenFlow, POF [4] and P4 [5] have been put forward to create a protocol-independent and vendor-agnostic SDN environment. As both approaches have attracted intensive interest from the Open Networking Foundation (ONF), they are considered in ONF's project on protocol-independent forwarding (PIF) [6].

Protocol-Oblivious Forwarding

Overview of POF

Figure 1 shows the overview of POF. The network architecture in Fig. 1a indicates that POF also uses a centralized controller to manage the packet forwarding in the switches. However, to make the forwarding plane protocol-independent, POF uses the forwarding procedure in Fig. 1b. Specifically, the packet parsing and flow matching in POF are based on a sequence of generic key assembly and table lookup instructions [4]. Hence, POF switches do not need to know the packet formats in advance. Actually, the search key of a matching field is defined as a tuple <*offset*, *length*>, where *offset* indicates the matching field's start bit location in a packet, and *length* tells the field's length in bits. For instance, in Fig. 1b, the IPv4 Source Address field in an IPv4-over-Ethernet frame is denoted as <*offset* = 208 *bits*, *length* = 32 bits>.

POF also includes a generic flow instruction set (POF-FIS) to facilitate POF switches to parse, edit, and forward packets arbitrarily [7]. It is known that in OpenFlow, the instructions and actions are also protocol-dependent; for example, actions like *push*, *pop*, and *set* are all subject to a specific packet field. While with POF-FIS, all the instructions and actions become protocol-independent. Hence, for packet forwarding, all a POF switch needs to do is to extract the matching fields from packets based on certain tuples <*offset*, *length*>, perform flow table lookups, and then execute the associated instructions defined in POF-FIS.

On top of these innovations, POF defines four types of flow tables to enhance the programmability of the forwarding plane, which are the masked-match (MM) table, the longest-prefix-match (LPM) table, the extract-match (EM) table, and the direct table (DT). These types of tables occupy different memory sizes and can be searched with specific table lookup algorithms. Note that a flow entry in all the tables consists of both matching field(s) and related instruction(s), except for DT, whose flow entries only include instructions. By leveraging these tables, the forwarding procedure in a POF switch can be abstracted as a data path pipeline. To handle the situation in which the switch needs to store the flow information temporarily, POF defines a metadata memory for each switch and introduces several metadata-related instructions.

Ecosystem of POF Development

We realize a simple ecosystem to facilitate the network innovations with POF:

Control Plane: We can realize a POF controller by leveraging the existing open source platforms for OpenFlow controllers. Hence, we develop a POF controller by extending the POX platform. In addition, when the SDN networks are managed and operated by different operators, they need to deal with the inter-domain traffic flows. Even though POF specification has already defined the protocol for the interactions between the control and forwarding planes, the mechanism and implementation for POF-based inter-domain operations still have not been addressed. Therefore, to fill in the missing puzzle piece, we design an inter-domain module (IDM) and include it in our POF controller. The IDM is realized with JavaScript Object Notation (JSON) and helps the domain managers (i.e., POF controllers) to exchange inter-domain information.

Figure 2a shows the inter-domain operation scheme of POF networks. Here, we design the IDM to exchange the information on network reachability, intra-domain network protocol, and so on among the controllers. Each controller abstracts its own domain as a "big switch" and generates a domain forwarding table to describe the connectivity among the domain and its neighbors. The domain forwarding table will be broadcast to the controllers in neighbor domains when there is a change in the inter-domain topology. After collecting the domain forwarding tables

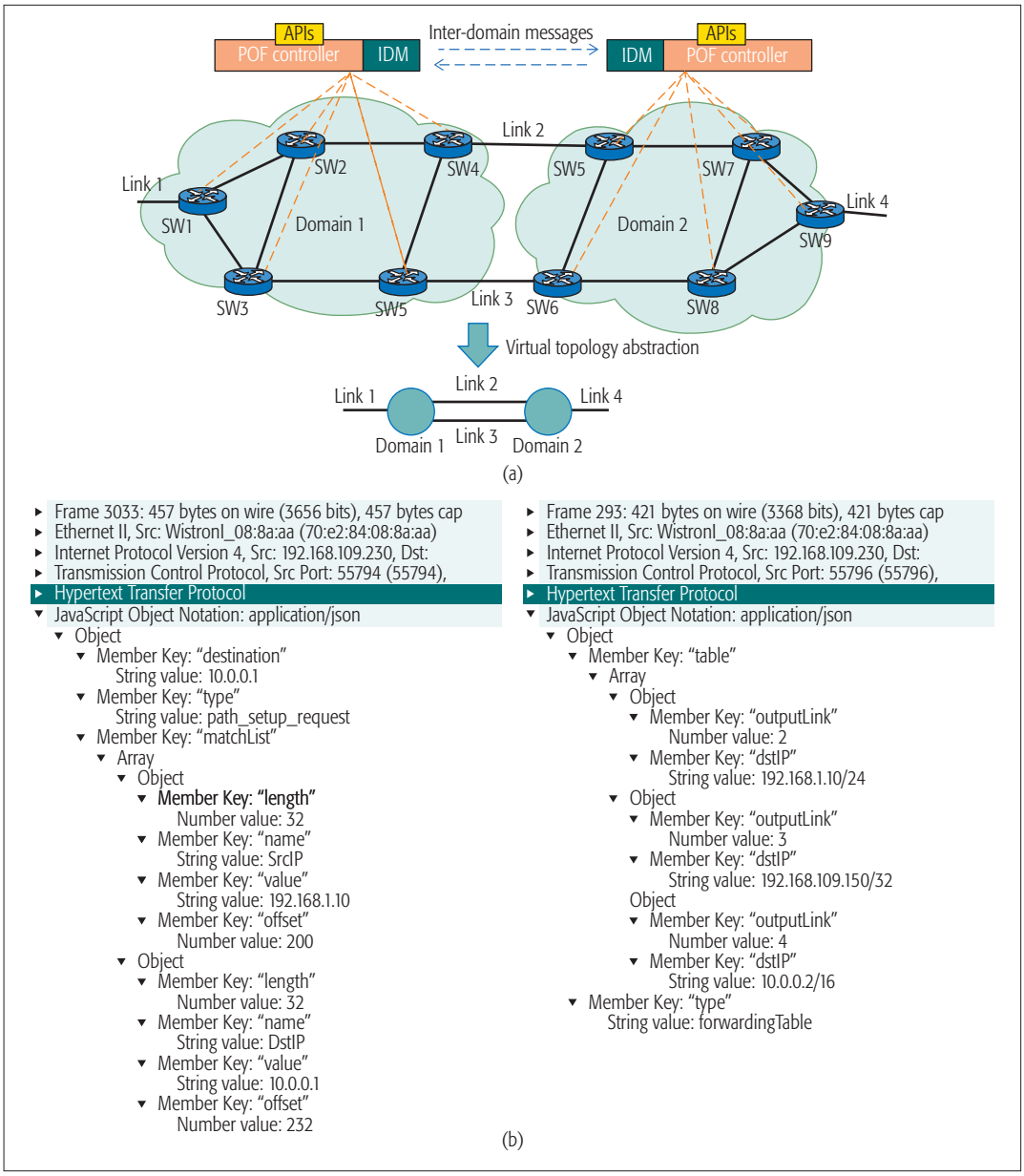

FIGURE 2. POF control plane for multi-domain scenarios: a) POF inter-domain operation; b) Wireshark captures of control packets to/from IDMs in POF controllers.

Our software-based switch has much higher packet processing/forwarding capacity and hence can make sure that all the video packets are handled in a timely and lossless manner even though there is 500 Mb/s background traffic.

from all of its neighbors, each controller constructs a global virtual topology. Then, when an inter-domain traffic flow arrives, it can calculate and find the right next domain to forward the flow to. For the flows that require specified quality of service (QoS), the controller can also set up an end-to-end routing path across multiple domains by collaborating with other controllers. Specifically, to realize this feature, we define an inter-domain message to encode the flows' requests and also design the protocol for controller collaboration. Figure 2b shows the packets captured between the IDMs for exchanging a domain forwarding table and sending the inter-domain message for end-to-end flow path setup, respectively.

Forwarding Plane: Similar to the case of OpenFlow, we need two types of switches (i.e., commercial and software-based ones) to realize the packet forwarding functionalities described earlier. Fortunately, network equipment vendors are in a joint force to design and implement POF. For instance, Huawei has partially supported POF with their router platforms (e.g., Huawei NE40E and NE5000E) and also published an open source prototype of a software-based POF switch online [8]. However, the commercial routers are prohibitively expensive for academic research, while the open source software-based switch has performance issues, that is, the packet forwarding throughput is very limited (i.e., below 200 Mb/s per port).

The major issue with the software-based POF switch prototype published in [8] is that it was

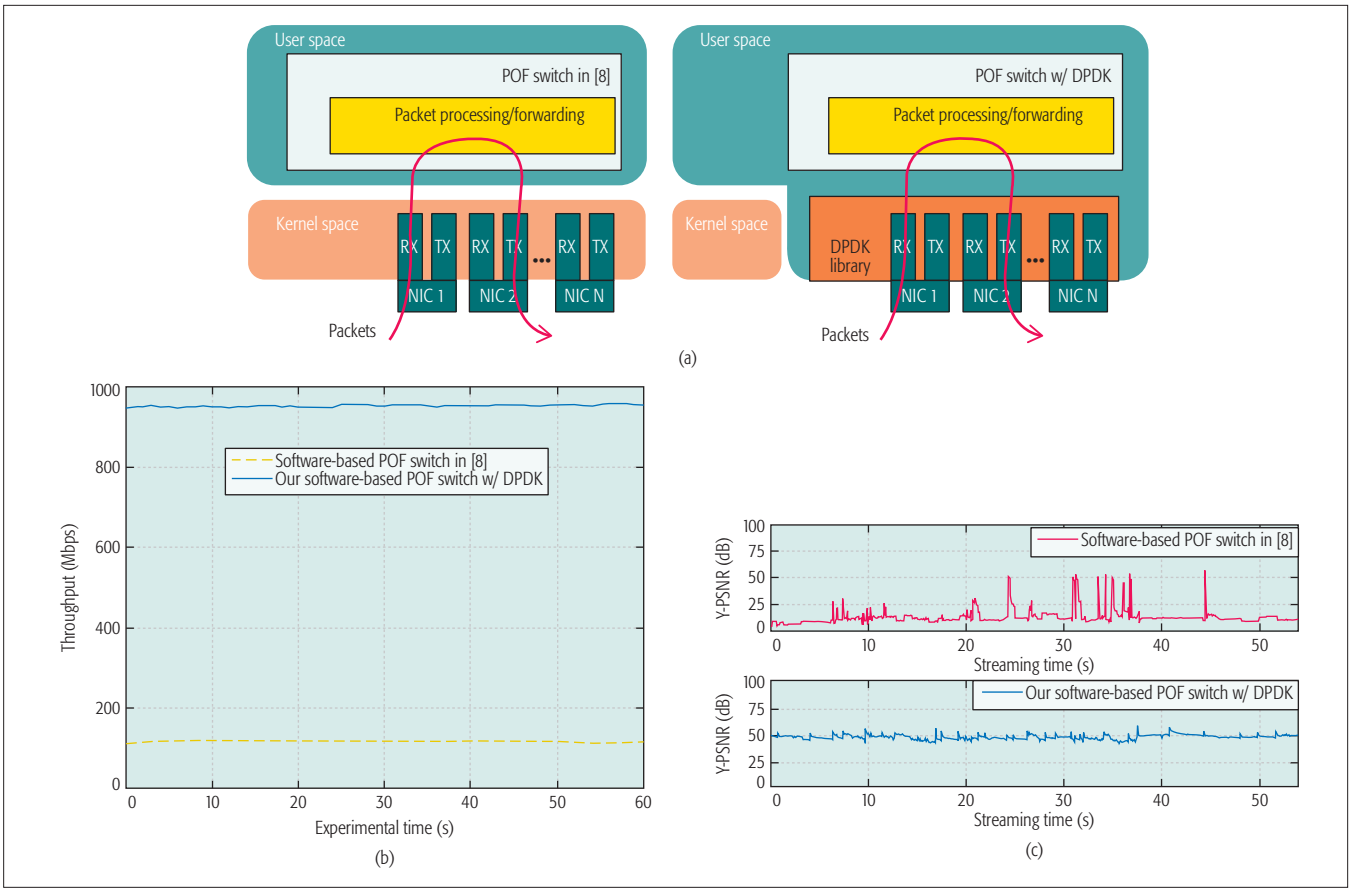

FIGURE 3. Design and performance of our software-based POF switch: a) architecture of our software-based POF switch with Intel DPDK; b) throughput comparison of software-based POF switches; and c) comparison on the Y-PSNR of received videos from different software-based POF switches.

designed to move packets between the kernel space and user space of Linux too frequently, which makes the packets being handled in the switch go through a few time-consuming and unnecessary processes, including CPU interrupt handling, user-space-to-kernel-space context switching, and so on. Therefore, the long processing time limits the throughput of the software-based switch. We try to overcome this drawback by ensuring that the packet processing and forwarding in a software-based POF switch are all conducted in the user space of Linux [9]. Basically, we design and implement our own software-based POF switch by leveraging the Intel data plane development kit (DPDK) [10] to forward the packets received by a network interface card (NIC) directly to the packet processing/forwarding module in the user space. Hence, they would experience much less latency and can adapt to the sophisticated processing in the forwarding plane of POF. Figure 3a shows the architecture of our software-based POF switch, and it can be seen that the DPDK driver helps exchange packets between the Ethernet ports in a NIC and the POF packet processing/forwarding module in the user space.

To illustrate the performance improvement achieved by our design, we measure the packet forwarding throughput of our software-based POF switch and compare it to that of the one published in [8] (i.e., without DPDK-based implementation). Figure 3b shows the experimental results. Here, we use an IXIA packet generator to pump 1 Gb/s packets to the two types of software-based switches running on independent high-performance servers that are equipped with Gigabit Ethernet (GbE) NICs, and measure the received throughput at the outputs of the switches. It can be seen that our software-based POF switch achieves much higher throughput and almost reaches the line rate of the GbE NICs. We then use the software-based switches to forward the packets for high-resolution video streaming and perform experiments to further verify the performance of our switch. Specifically, we emulate a network environment in which a 3.5 Mb/s video streaming packet flow needs to be forwarded together with 500 Mb/s background traffic (e.g., UDP packets). Figure 3c plots the luminance component's peak signal-to-noise ratios (Y-PSNR) of the received videos, which indicate that the quality of the video forwarded by our software-based switch is much higher and more stable than that of the benchmark. This is because our software-based switch has much higher packet processing/forwarding capacity and hence can make sure that all the video packets are handled in a timely and lossless manner even though there is 500 Mb/s background traffic.

Use Case: POF-Based Source Routing

With the ecosystem discussed above, we can realize new network applications/operations more easily and efficiently. In this section, we show

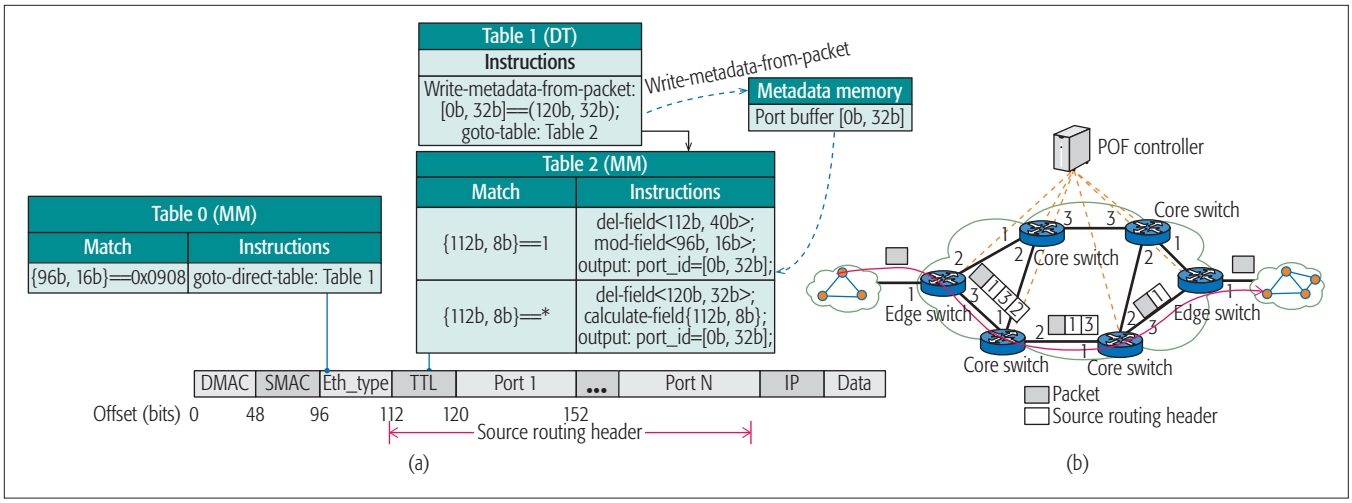

FIGURE 4. POF-based source routing: a) the procedure used to process the flow table in core switches and packet header designed for source routing; b) the operation principle of POF-based source routing.

POF-based source routing as a use case.

Principle of Source Routing

It is known that since the inception of SDN, the scalability of flow table space has always been an important issue [1]. The issue is especially tricky when the SDN network needs to carry a huge volume of traffic from many flows (e.g., in a data center network). Basically, when the flow entries installed in the switches increase rapidly, they can exhaust the switches' ternary content addressable memory (TCAM) easily and degrade forwarding performance.

SDN-based source routing has been considered as a promising way to solve the scalability issue mentioned above [11]. Specifically, in each packet, source routing encapsulates a series of output ports in the header fields to represent the forwarding action on each hop along the routing path. Then, when the packet is forwarded along the routing path, each switch extracts and performs its designated forwarding action in sequence (i.e., popping out the header field that contains the designated output port for the current hop and directing the packet accordingly). With this scheme, the authors of [11] discussed OpenFlow-based source routing, which encodes the output port sequence in one or more header fields supported by OpenFlow (e.g., VLAN header and multiprotocol label switching [MPLS] header). Then they showed that multiple flows can share the same flow table in a core switch if their designated forwarding actions at that switch use the same output port.

Note that this OpenFlow-based source routing scheme is still protocol-dependent, that is, the encoding of output ports has to reuse the header fields of legacy protocols and thus cannot be adjusted adaptively. For instance, the lengths of the encoded fields are either fixed or at least rigid and can hardly adapt to the hop count of routing paths in an arbitrary network. In addition, because of the flow matching principle of OpenFlow, the volume of installed flow entries in each core switch is still proportional to the number of output ports on the switch. In the discussions below, we show the design of POF-based source routing, and verify that it is more flexible and time-efficient, and the volume of installed flow entries can be reduced substantially.

Packet Design for POF-Based Source Routing

By using the protocol-independent feature of POF, we can realize source routing without reusing the header fields in legacy protocols [12]. Basically, we can tailor the packet fields to store the path information efficiently and enable effective source routing. Figure 4a describes the POF-based source routing packet format that we design in this work. For backward compatibility, we insert the source routing header between the Ethernet header and IP header. In order to identify source routing packets, we set the *type* field in the Ethernet header to 0x0908 to indicate that the Ethernet frame contains a POF-based source routing packet. Actually, this field can use another feasible value as long as it does not conflict with those defined for existing protocols [13].

The detailed descriptions on the fields included in the source routing header are as follows:

1. The *time-to-live* (*TTL*) field occupies 8 bits and represents the number of remaining hops for the packet to travel to its destination in a POF network. Thus, the value of this field will be set at the ingress edge switch to the POF network and decreased by 1 on each hop in it. When the packet is about to leave the POF network, the egress switch will remove the source routing header by using the *del-field* instruction in POF-FIS.
2. The *Port* field contains 32 bits, and its value identifies an output port of a POF switch.

Note that we design this field according to the *Port-ID* field used for POF-enabled switches [14], which is 32 bits. However, considering the fact that a switch would normally have fewer than 256 output ports, we can shorten the length of this field. This can easily be realized with either of the two following approaches.

- We keep the POF switch as it is, but shorten the *Port* field in source routing packets. Then we use padding bits to make the field 32 bits when it is being written into the metadata memory and matched to a switch port. Note that this can easily be realized with POF-FIS, and thus we do not need to modify any

codes in the software-based POF switch.
- We redesign the POF switch to use a shorter variable to define the *Port-ID* field, and then implement necessary changes in the POF controller.

Packet Forwarding with POF-Based Source Routing

Figure 4b describes the packet forwarding procedure with POF-based source routing. For a traffic flow, when its first packet arrives at an ingress switch to the POF network, the switch detects a flow table mismatch and sends a *Packet-In* message to the controller. Then the controller calculates a routing path for the flow to traverse the POF network, determines the designated output port in the switch on each hop along the path, encodes the information in a *Flow-Mod* message, and sends it to the ingress switch. After receiving the *Flow-Mod* message, the ingress switch sets up a flow entry for the flow and stores the output ports in its metadata memory. Then, for every packet of the flow, the ingress switch will convert it to the POF-based source routing format in Fig. 4a and insert the output ports by using POF-FIS.

The intermediate core switches use our proposed pipeline-like rule to process the POF-based source routing packets. Note that since we design the rule in such a way that it can be shared by all the POF-based source routing packets, we only need to install a small fixed number of flow entries (i.e., four) in each core switch in the POF network. Therefore, the volume of installed flow entries in the core switches can be reduced substantially. Moreover, since we encode the routing path in each packet, the core switches do not need to interact with the controller during the path setup, and thus the path setup latency can be reduced significantly too. Figure 4a explains the pipeline-like rule that we propose to process the POF-based source routing packets.

Before explaining the rule, we need to introduce the following three types of notations to assist the description.
- <*offset*, *length*> represents a field that starts from the bit location *offset* in a packet or the metadata memory of a POF switch and contains *length* bits.
- {*offset*, *length*} is the value of the field that is described by <*offset*, *length*> in a packet.
- [*offset*, *length*] is the value of the field that is described by <*offset*, *length*> in the metadata memory of a switch.

As shown in Fig. 4a, we leverage the forwarding plane programmability provided by POF to realize the pipeline-like source routing packet processing in core switches with three tables: two MM tables and one DT. Upon receiving a source routing packet, a core switch first passes it to Table 0 and uses it to check the *type* field in the Ethernet header, which is described by <96 bits, 16 bits>. If the field's value equals 0x0908, the packet is a source routing one and should be sent to Table 1, which is a DT whose entry only includes an instruction. With Table 1, the core switch copies the value of field <120 bits, 32 bits> (i.e., the *Port* field that encodes the designated output port of this hop) to its metadata memory by executing the *write-metadata-from-packet* instruction. Finally, the packet reaches Table 2, which includes two entries to determine whether the switch is the packet's last hop in the POF network. Specifically, it checks the value of field <112 bits, 8 bits> (i.e., *TTL*). If the switch is the last hop (i.e., the packet's egress switch in the POF network), the value of its *TTL* field will be 1. The field will match the first entry in Table 2, and then the *del-field* instruction is invoked to remove the whole source routing header from the packet and restore the *type* field in the Ethernet header to its original value. Otherwise, the field will match the second entry in Table 2, and the switch only removes the leftmost *Port* field in the source routing header (i.e., the one represented by <*offset* = 120 bits, *length* =32 bits>) with the *del-field* instruction, decreases the *TTL* field by 1 with the *calculate-field* instruction, and forwards the packet to the designated output port that has been stored in the metadata memory with the *output* instruction.

> We can see that the packet forwarding procedure in each POF switch works just like a software program, which verifies the forwarding plane programmability provided by POF.

To this end, we can see that the packet forwarding procedure in each POF switch works just like a software program, which verifies the forwarding plane programmability provided by POF. Specifically, the processing with the three tables can be considered as the functions whose inputs and outputs are the fields in the source routing packets, while the metadata memory behaves like temporary variables to save necessary information. The number of flow entries installed on each core switch for source routing is fixed at four, which is relatively small and does not increase with the number of output ports on switches anymore. What is more promising is that as the pipeline-like packet processing in the core switches is applicable to all the source routing packets, the interactions between the control and forwarding planes are minimized. Hence, compared to the OpenFlow-based source routing in [11], our POF-based scheme not only uses fewer flow entries but also reduces the communication overhead between the controller and switches.

Performance Evaluation

In order to further verify the proposed functionality and related benefits, we build an experimental testbed to evaluate POF-based source routing. Specifically, we use chain topologies, and change the number of switches that are included in the chain to observe the changes on the path setup latency. Here, each switch is realized by running our software-based POF switch on a standalone Linux server. Here, the path setup latency is measured by leveraging the Ping program, and to emulate a real network situation, we also inject background traffic in the testbed. Figure 5a shows the results on path setup latency from the schemes with and without the proposed source routing scheme. It can be seen clearly that with the proposed source routing scheme, the path setup latency can be reduced significantly and does not increase with the number of switches along the path. These advantages can be explained as follows. Basically, without source routing, the path of a flow can only be set up after the controller has configured all the switches

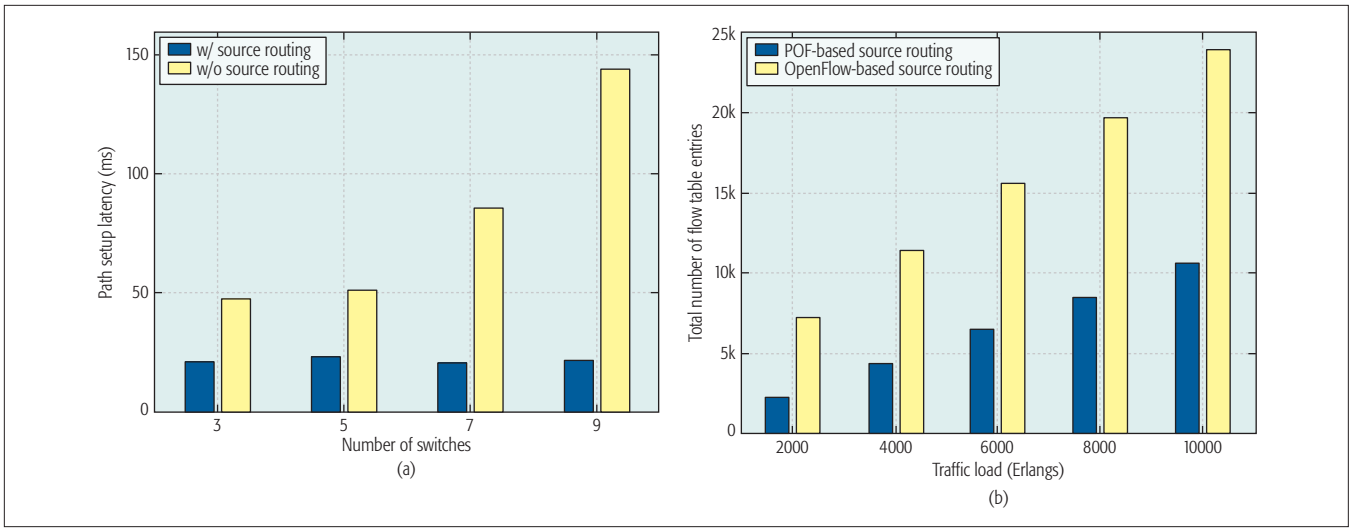

FIGURE 5. Experimental results: a) Results on path setup latency; b) results on total number of flow entries.

on it. Hence, the time spent on the interactions between the controller and switches becomes longer when there are more switches to configure. However, our proposed scheme would not have this issue, since no matter how many hops the path contains, the controller only needs to configure the ingress switch to set it up and does not need to interact with other switches.

Another benefit of our proposed POF-based source routing is that compared to OpenFlow-based routing, it uses fewer total flow table entries in the network. Figure 5b compares the total flow table entry usage with our POF-based source routing scheme and the OpenFlow-based benchmark. Here, the results are obtained by running simulations with a FatTree topology that contains 16 end nodes. The dynamic traffic flows are generated between the end nodes with the Poisson traffic model and have an average duration of 10 s. We observe that compared to the OpenFlow-based benchmark, our POF-based source routing scheme reduces the total flow table entry usage by 55–68 percent. More promisingly, with the increase of the traffic load, the advantage of our proposed scheme becomes more and more significant. This is because our POF-based source routing scheme only needs to install additional flow table entries in the ingress switches to accommodate a new flow, and thus the numbers of used flow table entries on the core and aggregation switches do not increase or have relation with the number of output ports.

WAN-Based POF Network Testbed in China

In addition to our lab-based network environment, we also realize the first wide-area network (WAN)-based POF network testbed that connects both commercial and software-based POF switches located in two major cities in China (i.e., Beijing and Hefei). Basically, to make the POF networks practical, the ecosystem we developed should also be tested in real deployed networks, to verify their performance, functionality, and effectiveness. As shown in Fig. 6a, the WAN-based POF network testbed includes four sites, which are located in the Computer Network Information Center (CNIC) of the Chinese Academy of Science (CAS), the Institute of Acoustics (IOA) of CAS, the Huawei Institute of Research and Development (Huawei R&D), and the University of Science and Technology of China (USTC), respectively. The first three sites are in Beijing, and the last one is located in Hefei.

Each site consists of a commercial POF hardware switch (Huawei's NE40E-X3) and several software-based POF switches, while they are all managed by the POF controller located at USTC. Among these sites, we have network connections set up across the WAN with the virtual extensible LAN (VXLAN) tunnels, and the network bandwidth among USTC, CNIC, and IOA achieves up to 70 Mb/s through the China Education and Research Network (CERNET). With this scheme, the end users at different sites are transparent to the WAN and can communicate as in a LAN.

By leveraging the VXLAN tunnels, the whole WAN-based testbed can be considered as a pure POF network, and thus network innovations such as POF-based source routing can easily be realized in this testbed. Moreover, the testbed can use VXLAN to support dynamic virtual machine (VM) migration seamlessly, and we describe our experimental demonstration on WAN-based VM migration in this section to further verify the flexibility of POF.

It is known that VXLAN is widely used in inter-data-center networks and can make geographically distributed VMs communicate as if in the same LAN. Unfortunately, VXLAN has not been supported by the latest OpenFlow specification [5], which means that we cannot realize it with OpenFlow without non-standardized extensions. Meanwhile, most of the existing VXLAN implementations in legacy networks use static configuration, which limits network flexibility and can hardly cooperate with a dynamic network environment. On the other hand, with the enhanced forwarding plane programmability provided by POF, we can support VXLAN easily and realize live VM migration across WAN in the testbed. We program the POF switches at each site, and use them to emulate the ingress/egress switches of data centers to facilitate the function of a virtual tunnel endpoint (VTEP). Specifically, each inter-data-center packet is encapsulated with a VXLAN header in the POF switches by perform-

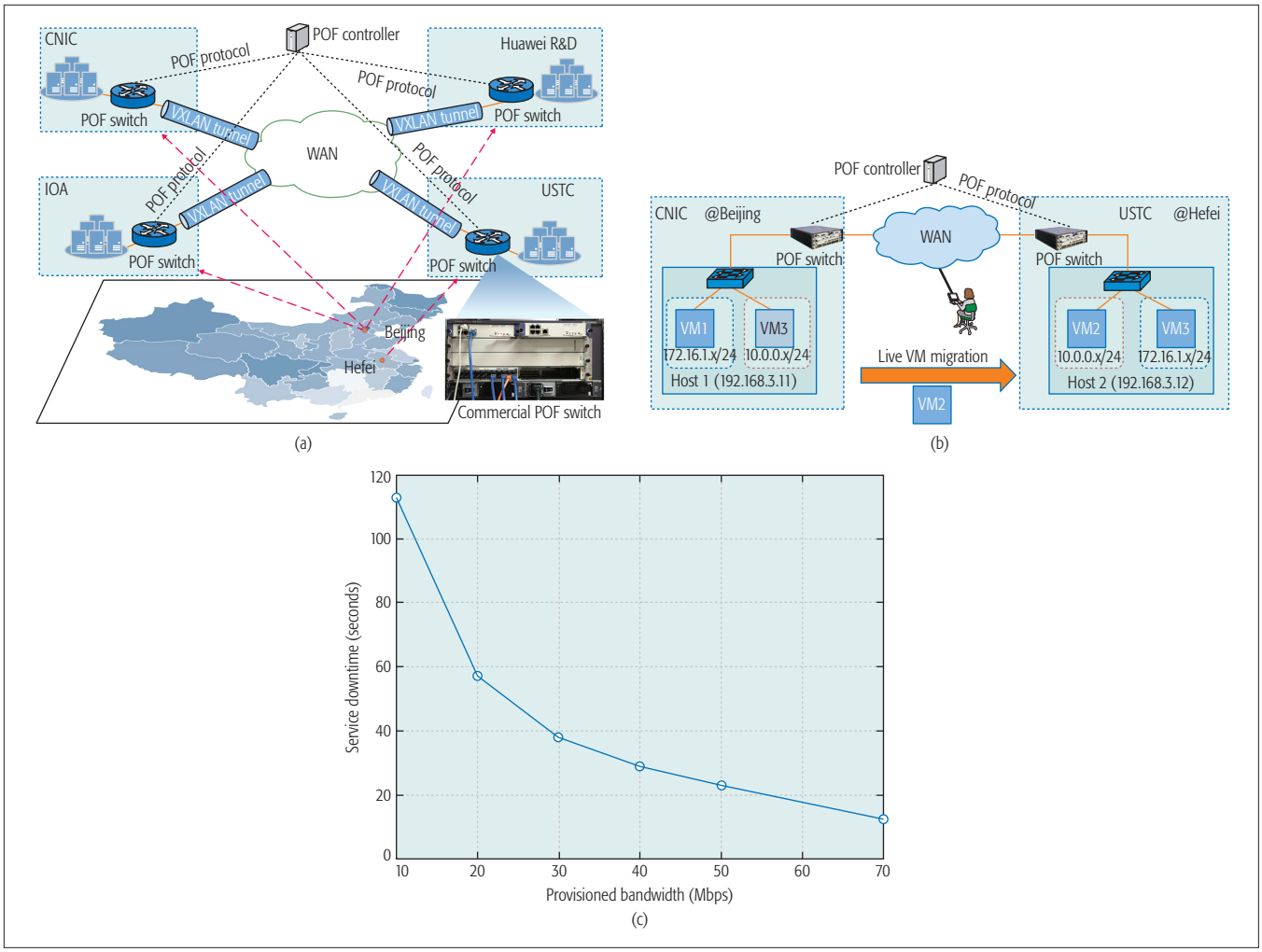

FIGURE 6. WAN-based POF network testbed in China: a) WAN-based POF network testbed in China; b) VM migration across WAN in POF network testbed; c) results on service downtime due to VM migration across WAN.

ing the instructions from the controller, and a virtual network identifier (VNI) is allocated to each tenant. Then, based on the VMs' locations, the POF controller can update the flow entries in the switches to facilitate VM migration.

We experimentally demonstrate a live VM migration between the USTC and CNIC sites. Figure 6b illustrates the experimental scenario. We migrate a 1 GB VM running Ubuntu OS from USTC to CNIC. In order to evaluate the performance of the VM migration, we let the VM send a 2 Mb/s UDP packet flow to an end user located in the USTC domain, and measure the service downtime during which the end user cannot receive the packets due to the VM migration across the WAN. Figure 6c shows the experimental results on service downtime when we allocate different bandwidth in the POF network testbed to facilitate the VM migration.

Conclusion

This article discusses how to enhance the programmability of the forwarding plane in SDN with POF. We first describe the working principle of POF and elaborate on our efforts on enriching the ecosystem of POF development. Then our design and implementation of the POF-based source routing are discussed, and we also elaborate on the first WAN-based POF network testbed, which includes POF switches located in two cities in China.

Acknowledgments

This work was supported in part by the NSFC Project 61371117, the Fundamental Research Funds for the Central Universities (WK2100060010), the Natural Science Research Project for Universities in Anhui (KJ2014ZD38), and the Strategic Priority Research Program of the CAS (XDA06011202). The authors would also like to thank their colleagues from CAS and the Huawei institute of Research and Development for their kind help on the experiments.

References

[1] D. Kreutz *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, Jan. 2015, pp. 14–76.
[2] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *Comp. Commun. Rev.*, vol. 38, Feb. 2008, pp. 69–74.
[3] N. Xue *et al.*, "Demonstration of OpenFlow-Controlled Network Orchestration for Adaptive SVC Video Manycast," *IEEE Trans. Multimedia*, vol. 17, Sept. 2015, pp. 1617–29.
[4] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane," *Proc. ACM HotSDN 2013*, Aug. 2013, pp. 127–32.
[5] P. Bosshart *et al.*, "P4: Programming Protocol-independent Packet Processors," *Comp. Commun. Rev.*, vol. 44, July 2014, pp. 87–95.

[6] Protocol Independent Forwarding, https://www.opennetworking.org/protocol-independent-forwarding.
[7] J. Yu *et al.*, "Forwarding Programming in Protocol-Oblivious Instruction Set," *Proc. ICNP 2014*, Oct. 2014, pp. 577–82.
[8] Protocol Oblivious Forwarding, http://www.poforwarding.org.
[9] B. Pfaff *et al.*, "The Design and Implementation of Open vSwitch," *Proc. NSDI 2015*, May 2015, pp. 117–30.
[10] DPDK: Data Plane Development Kit, http://dpdk.org/.
[11] S. Jyothi, M. Dong, and P. Godfrey, "Towards a Flexible Data Center Fabric with Source Routing," *Proc. ACM SOSR 2015*, June 2015, pp. 10:1–10:8.
[12] S. Li *et al.*, "Source Routing with Protocol-Oblivious Forwarding (POF) to Enable Efficient e-Health Data Transfers," *Proc. ICC 2016*, May 2016, pp. 1–6.
[13] EtherType, https://en.wikipedia.org/wiki/EtherType.
[14] D. Hu *et al.*, "Design and Demonstration of SDN-Based Flexible Flow Converging with Protocol-Oblivious Forwarding (POF)," *Proc. GLOBECOM 2015*, Dec. 2015, pp. 1–6.
[15] OpenFlow Switch Specification, v. 1.5.0, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.pdf.

Biographies

Shengru Li received his B.S. degree from the School of Information Science and Engineering, Shenyang Ligong University (SYLU), China, in 2013. Now he is working toward his Ph.D. degree at the School of Information and Technology, University of Science and Technology of China (USTC). His research interests include software-defined networking and network architecture.

Daoyun Hu received his B.S. degree from the Department of Information Science and Technology, Southwest Jiaotong University (SWJTU), Chengdu, China, in 2014. He is working toward his M.S. degree at the School of Information and Technology, USTC. His research interest is software-defined networking.

Wenjian Fang received his B.S. degree from the School of Information Science and Technology, USTC in 2014. Now he is working toward his M.S. degree at the same school at the same university. His research interests include elastic optical networks, data center networks, and software-defined networking.

Shoujiang Ma received his B.S. degree from the Department of Information Science and Technology, SWJTU, Chengdu, China, in 2013. He is working toward his M.S. degree at the School of Information and Technology, USTC. His research interests include software-defined networking and next generation network architecture.

Cen Chen received his B.S. degree from the School of Electronic Engineering, Xidian University, China, in 2013. Now he is working toward his M.S. degree at the School of Information and Technology, USTC. His research interests include elastic optical networks and software-defined networking.

Zuqing Zhu [M'07, SM'12] (zqzhu@ieee.org) received his Ph.D. degree from the Department of Electrical and Computer Engineering, University of California, Davis, in 2007. From July 2007 to January 2011, he worked in the Service Provider Technology Group of Cisco Systems, San Jose, California, as a senior R&D engineer. In January 2011, he joined USTC, where currently he is a full professor. He has published more than 160 papers in peer-reviewed journals and conferences. He is an Editorial Board member of *IEEE Communications Magazine*, the *Journal of Optical Switching and Networking* (Elsevier), the *Telecommunication Systems Journal* (Springer), *Photonic Network Communications* (Springer), and other publications. He received Best Paper Awards from IEEE ICC 2013, IEEE GLOBECOM 2013, ICNC 2014, and ICC 2015. He is a Senior Member of OSA.