Source Routing with Protocol-oblivious Forwarding (POF) to Enable Efficient e-Health Data Transfers

Shengru Li, Daoyun Hu, Wenjian Fang, Zuqing Zhu†
School of Information Science and Technology,
University of Science and Technology of China, Hefei, China
†Email: {zqzhu}@ieee.org

***Abstract*—It has already been confirmed that software-defined networking (SDN) can make the networks more programmable, adaptive and application aware. However, due to the large-scale and geographically-distributed nature of wide-area networks (WAN), the scalability could become a critical issue if we incorporate SDN for WANs (*i.e.*, realizing SD-WANs). In this paper, we design and implement a novel network system that can leverage source routing with the protocol-oblivious forwarding (POF) to facilitate efficient e-Health data transfers with low setup latency. We develop the POF-based source routing protocol to realize a pipeline based packet processing procedure, which can replace the table-lookup based approach in traditional SDN networks and make the forwarding plane more efficient. The proposed scheme is demonstrated experimentally, and the results verify that with it, the flow-tables installed in each core switches in a POF-controlled SD-WAN can be minimized and the path setup latency of traffic flows can be reduced significantly as well.**

***Index Terms*—Software-defined network (SDN), Wide area network (WAN), Protocol-oblivious forwarding (POF)**

I. Introduction

Nowadays, the fast development of data-intensive applications, such as e-Health, e-Science, e-commerce, *etc*, has brought us into the Big Data era [1]. It is known that certain Big Data applications may generate huge volumes of data, which needs to be transferred over wide-area networks (WANs) for timely processing. For instance, in a telemedicine network, the health-monitoring devices worn by a large population of subscribers can contribute a fair amount of traffic for real-time processing [2, 3]. As the subscribers usually locate in a geographically dispersed manner, it would be challenging to transfer the data to the processing module(s) with low latency. Hence, both flexible architecture and efficient protocols are required to achieve flexible traffic engineering in WANs, especially when cloud-based data gathering and processing are needed [4–9].

Today's WAN architecture uses distributed traffic engineering mechanisms to reduce bandwidth congestion, but it has been proven to be inefficient due to the close coupling of the control and forwarding planes of the network [10]. In order to support Big Data applications more efficiently, WANs also need a more programmable and adaptive architecture with effective network control and management (NC&M), which is similar to the innovation trends in other types of networks [11–14]. Recently, software-defined networking (SDN) has been proposed as a break-through technology that is promising for the next-generation Internet due to the fact that it decouples control and forwarding planes of a network and leverages centralized NC&M to make it more programmable, adaptive and application-aware [10]. Thanks to the advances on SDN, the Big Data related traffic can be managed more efficiently in WANs. However, due to the large-scale and geographically-distributed nature of WANs, the scalability could become a critical issue when incorporating SDN for WANs. Specifically, when the traffic volume increases, more and more flow-tables will be installed in the switches, which can use up their memory, make the table look-up and traffic scheduling increasingly complex, and increase the communication overhead significantly.

Unfortunately, the aforementioned issue cannot be addressed properly with the initial implementation of SDN, *i.e.*, OpenFlow [15]. This is because OpenFlow defines protocol-dependent flow-matching rules, which can lead to repeated flow-table installations and look-ups in the forwarding plane. In an OpenFlow-controlled WAN, the interactions between the switches and OpenFlow controller need to bear a relatively long communication latency, which is due to the physical distance between them and intrinsic. Hence, when setting up a flow by configuring the flow-table on each switch along the path, the hop-by-hop operation can cause a long setup delay. Note that, this is especially unwanted for the delay-sensitive e-Health data transfers that usually operate as mice flows [16].

On the other hand, the increasing volume of flow-tables could be another killing factor for the software-defined WANs (SD-WANs). Specifically, due to its user population, an OpenFlow-controlled SD-WAN usually needs to deploy a huge number of flows [17], each of which may require to install a flow-table on all the switches along the routing path. Consequently, the switches' memory space for flow-tables can vanish quickly [18]. Further more, as pointed out by [10], most of the commercially-available OpenFlow-enabled switches cannot achieve a processing throughput of more than 500 *Flow_Mod*s per second.

In order to address the issues of OpenFlow mentioned above, recent researches have considered to enhance the programmability and flexibility of SDN forwarding plane, with the protocol-oblivious forwarding (POF) technology [19] or the programming protocol-independent packet processors (P4) [20]. The basic ideas behind POF and P4 are similar, *i.e.*, trying to decouple network protocols from the forwarding

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

procedure in SDN-enabled switches and make the forwarding plane reconfigurable, programmable and future-proof. More specifically, POF develops a protocol-independent instruction set that allows to express much more flexible packet processing than the current OpenFlow specifications [21, 22], while P4 mainly focuses on designing a high-level network programming language for protocol innovations. Note that both approaches have attracted noticeable interests from the open networking foundation (ONF), and are considered in its project on protocol-independent forwarding (PIF) [23].

In this paper, we design and implement a novel network system that can leverage source routing with POF to facilitate efficient e-Health data transfers with low setup latency. We develop the POF-based source routing protocol, and experimentally demonstrate that with the proposed scheme, the flow-tables installed in each core switches in a POF-controlled SD-WAN can be minimized. Our experimental results also indicate that the setup latency of the traffic flows can be reduced significantly. The rest of the paper is organized as follows. Section II provides a brief survey on the related work. The operation principle of POF is introduced in Section III. Then, we describe our design of POF-based source routing in Section IV, and the experimental demonstrations are discussed in Section V. Finally, Section VI summarizes the paper.

II. Related Work

In order to address the performance issues of SD-WANs, people have proposed a few approaches. The authors of [24] proposed the scheme of HyperFlow, which utilizes a logically centralized but physically distributed control plane to enhance the performance of OpenFlow-enabled SDN networks. In the same direction, Dixit *et al.* [25] designed and implemented an elastic architecture to coordinate distributed SDN controllers for reducing the setup latency in SD-WANs. However, it is known that in order to keep the network status consistent among the distributed controllers, complicated synchronization scenarios have to be implemented [26], while the approaches mentioned above did not address the increased operational complexity due to the status synchronization. Therefore, the efficiency of NC&M would be impacted.

On the other hand, researchers have also considered to reduce the interactions between the control and forwarding planes. In [27], an architecture named as KeyFlow was proposed, which leverages a residue numeral system (RNS) to make a forwarding device flow-stateless. Nevertheless, the approach requires each switch to equip the special reminder operation in its hardware, and moreover, the network architecture can hardly adapt to dynamic topology changes. The authors of [28] considered the OpenFlow-based source routing approach. Specifically, in each packet, the scheme encapsulates a series of output ports in one or more header fields supported by OpenFlow (*e.g.*, VLAN header, MPLS label, *etc*), to represent the routing path and the forwarding action on each hop along it. Then, on the routing path, each OpenFlow switch will extract its forwarding action in sequence (*i.e.*, matching to the corresponding field that contains its output port) to direct the packet to its destination. Hence, multiple flows can share the same flow-table in a core switch, if their forwarding actions use the same output port. Nevertheless, the proposed source routing approach is still protocol-dependent, which means that the definition of the output port related matching fields needs to comply with the existing protocols and cannot be adjusted adaptively for each network. Moreover, with this approach, the volume of flow-table entries in each core switch still depends on the maximum number of output ports on a switch. We believe that this is still sub-optimal, and as we will show later in this paper, the volume of flow-table entries can be further reduced with a POF-based approach.

III. POF Primer

This section briefly reviews the operation principle of POF and its flow instruction set (POF-FIS) to provide a context for the rest of the article. Basically, POF inherits the network architecture of OpenFlow, *i.e.*, a centralized controller resides in the control plane to manage the forwarding behaviors of the switches in the forwarding plane with flow-tables. However, the innovations provided by POF are the protocol-oblivious description for flow-matching fields and a set of generic flow instructions, with which protocol-independent packet forwarding can be realized easily in the switches.

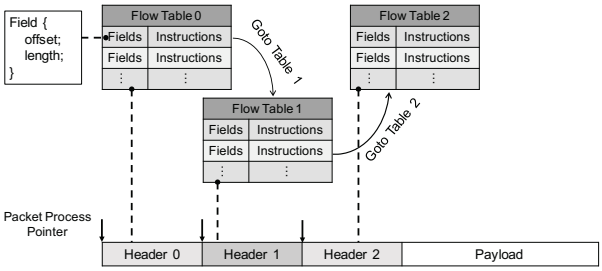


Fig. 1. Packet forwarding procedure in POF switches.

Fig. 1 shows the packet forwarding procedure used in POF switches. Specifically, the switches use a sequence of generic key assembly and table lookup instructions to accomplish packet parsing and flow matching. The key concepts regarding POF are explained as follows.

- **Matching Fields**: POF simply defines the search key of a matching field as a tuple <*offset, length*>. Here, *offset* indicates the start-location of the field in a packet (*i.e.*, how many bits the packet process pointer should skip from the beginning of the packet to locate the field), while *length* tells the field's length in bits [19].
- **POF-FIS**: All the instructions in POF-FIS utilize the tuple <*offset, length*> to locate the data that they need to operate on [21]. This provides us the freedom to manipulate any bit(s) in packets at will, which is far more flexible than the scheme in OpenFlow. For instance, in the latest OpenFlow switch specifications (*i.e.*, version 1.5 [15]), the *push* action is still protocol-specific and multiple actions have to be defined for each legacy protocol (*e.g.*, *push*-MPLS, *push*-PBB, and *push*-VLAN).

However, all these *push* actions can be realized with one generic instruction in POF, which is *add-field*. Specifically, by using *add-field*, we can insert any field at any position in a packet. POF-FIS even includes a *calculate-field* instruction to provide the support on arithmetic and logical operations.

- **Flow Tables**: The flow tables stored in a POF-enabled switch can be classified into four types, *i.e.*, 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 various 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-enabled switch can be abstracted as a data-path pipeline, and hence the network programmability and flexibility can be improved significantly.
- **Metadata Memory**: When a switch needs to handle multiple tables in packet forwarding, it uses metadata memory to store the flow information that the current table processing generated for the next. POF-FIS defines three metadata-related instructions (*i.e.*, *write-metadata*, *write-metadata-from-packet*, and *set-field-from-metadata*).

IV. POF-based Source Routing in SD-WANs

In this section, we explain the proposed POF-based source routing for SD-WANs, and describe both the network architecture and the forwarding procedure used by the POF-enabled switches. Fig. 2 shows the network architecture of a POF-based SD-WAN, which consists of a centralized POF controller, and core and edge switches. Note that, the edge switches work as the gateways to peer networks, while the core switches focus on packet forwarding inside the SD-WAN.

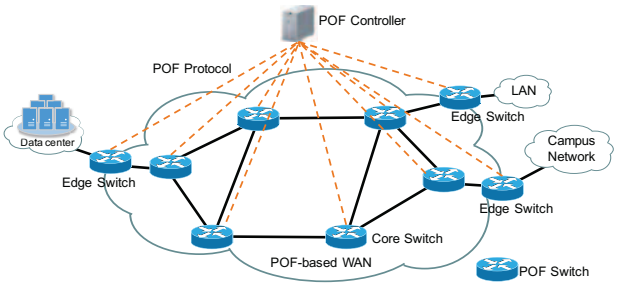


Fig. 2. Network architecture.

A. Packet Design for Source Routing

Thanks to the protocol-independent nature of POF, there is no need to reuse the header fields in legacy protocols (*e.g.*, VLAN and MPLS). Basically, the packet fields can be tailored specially to enable efficient source routing. Here, the fields are still designed to store the path information, and Fig. 3 describes the proposed packet format for POF-based source routing. Specifically, the source routing related header fields are inserted in between Ethernet header and IP header. Moreover, after inserting the new fields, we modify the *type* field in Ethernet header to "0x0908" to indicate that the Ethernet frame contains a POF-based source routing packet. The detailed descriptions on the new fields are as follows.

- *Time-to-Live (TTL)*: This field occupies 8 bits and indicates the remaining hops for the packet to travel to its destination in the POF-based SD-WAN. Therefore, the value of this field will be set at the ingress edge switch, and in each subsequent switch, its value is decreased by 1. When the packet is about to leave the POF-based SD-WAN, the egress edge switch remove it by applying the *del-field* instruction.
- *Port*: This field occupies 32 bits[1], and its value identifies an output port on a POF-enabled switch. Note that, a source routing packet can contain multiple *Port* fields to represent the forwarding path, and all the fields form a *Port* stack. Each switch pops the first *Port* field (*i.e.*, with *<offset=120 bits, length=32 bits>*) from the *Port* stack to find the output port for the packet. This is done by writing the value of the *Port* field to the metadata memory and removing the field from the packet, *i.e.*, with the *write-data-from-packet* and *del-field* instructions, respectively.

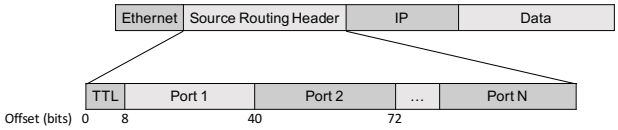


Fig. 3. Packet header format designed for POF-based source routing.

B. Procedure for Source Routing based Packet Processing

Figs. 4 and 5 show the principle and detailed procedure of POF-based source routing, respectively. When the first packet of a flow arrives at an ingress edge switch, it triggers a *Packet-In* message to be sent to the POF controller, since there is no flow entries to match against. Upon receiving the *Packet-In* message, the controller calculates a routing path for the flow, and sends a *Flow-Mod* message to the ingress edge switch, which encodes the designated output port to use on each switch along the path. The ingress edge switch stores the output ports in its metadata memory, and will insert them into every packet of the flow by using POF-FIS. The subsequent core switches use a pre-installed pipeline-like matching rule that consists of multiple flow tables to process the packets, which will be explained in detail in Fig. 6. Note that, since the forwarding path is encoded in each packet, the core switches do not need to have any interactions with the controller, and hence the setup latency is reduced significantly.

POF-FIS enable a lot of functions for the POF-enabled switches, and thus by leveraging it, we can reduce the processing burden on the controller and use source routing to

[1]The length of the field is determined according to the *Port-ID* defined for POF-enabled switches [29], as we encode the *Port-ID* of an output port in this field.

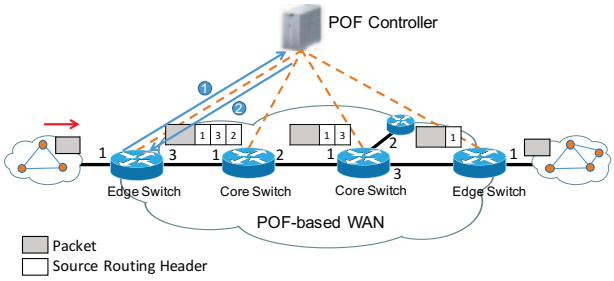


Fig. 4. Operational principle of POF-based source routing.

make the data-path much more intelligent. Before explaining the detailed packet forwarding procedure in the switches, we introduce several notations to assist the description.

- <*offset, length*>: the field starting from *offset* with *length* bits.
- {*offset, length*}: the value of the field determined with <*offset, length*> in a packet.
- [*offset, length*]: the value of the field determined with <*offset, length*> in the metadata memory.

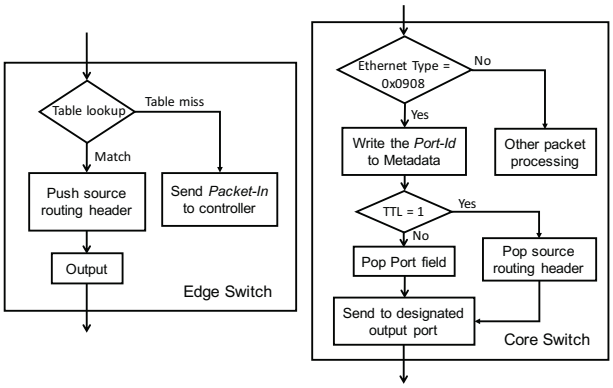


Fig. 5. Procedures used by edge and core switches for source routing.

Fig. 6 shows the proposed procedure used to process the flow tables in a core switch for source routing. We use three flow tables, including two MM tables and one DT table, to realize the overall source routing functionality as follows.

- The source routing packet arrives at the switch first goes to Table 0, which includes an entry to check the *type* field in its Ethernet header, *i.e.*, with <*offset, length*> equals <96 bits, 16 bits>. If its value equals 0x0908, we determines that it is a source routing packet and should be sent to Table 1.
- Table 1 is a DT, and the entries in it only include instructions, as shown in Fig. 6. Here, the switch executes the *write-metadata-from-packet* instruction to copy the value at <120 bits, 32 bits> (*i.e.*, the *Port* field to encode the output port for this hop) to its metadata memory.
- Table 2 includes two entries that are used to determine whether the switch is the packet's last hop. Specifically, it checks the *TTL* field (*i.e.*, <112 bits, 8 bits>). If the value of the *TTL* field equals 1, the switch is the last hop of this packet and it invokes the *del-field* instruction to delete the whole source routing related fields in the packet and restore the *type* field in the Ethernet header to its original value (*e.g.*, 0x0800 for an IPv4 packet). Otherwise, the switch only removes the *Port* field for this hop. The *output* instruction is then used to forward the packet to its designated output port (*i.e.*, using the *Port-ID* stored in the metadata memory).

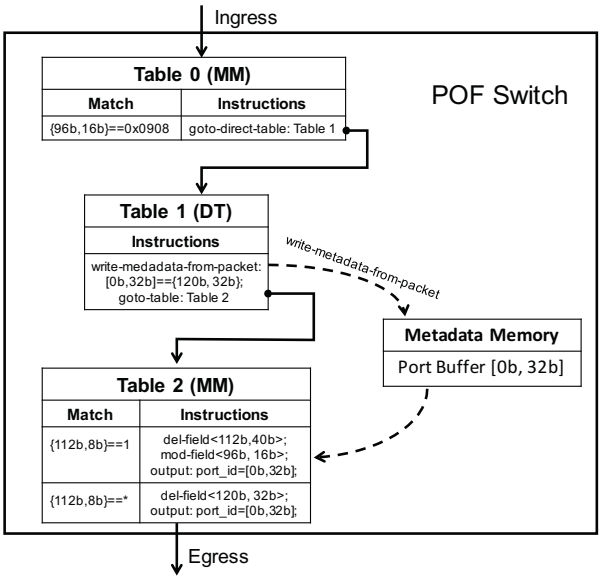


Fig. 6. Procedure used to process the flow tables in core switches for source routing.

Finally, we can see that the overall processing in each switch behaves like a software program, which verifies the programmability of POF. Specifically, the processing on the flow tables can be considered as functions, whose inputs and outputs are the fields in the packet and the processed packet, respectively. The metadata memory provides the support on saving information in temporary variables. The proposed procedure makes the switches work as a stand-alone entity for packet processing and minimizes the interactions between the control and forwarding planes. Furthermore, the number of flow tables installed on each core switch is fixed as 3, which is a small constant and does not depends on the maximum number of output ports on the switches any more. Therefore, compared with the OpenFlow-based source routing scheme in [28], our proposed POF-based scheme consumes less memory on the switches and requires less communication overhead between the controller and switches.

V. Experimental Demonstration

In this section, we describe our proof-of-concept demonstration to verify the functionality of the proposed POF-based source routing scheme, and evaluates its performance in terms of path setup latency.

A. Experiments for Functionality Verification

We first build a POF-based network testbed to verify the functionality of the proposed POF-based source routing scheme. The testbed consists of several software-based POF-enabled switches, which are realized by modifying the open-source POFSwitch [29] and run its instances on stand-alone high-performance Linux servers. We also extend the POX platform [30] to develop a POF controller and also runs it on a Linux server. Fig. 7 shows the configuration of the testbed for functionality verification, which possesses a line topology including 4 POF-enabled switch, *i.e.*, two core switches and two edge switches. Each edge switch connects to an IPv4 host. We send *ICMP_Request* packets from *Host* 1 to *Host* 2, and capture the packets in the POF-based network with Wireshark.

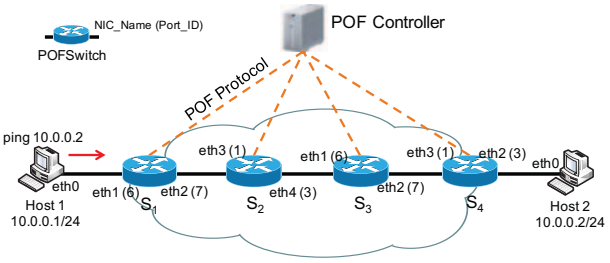
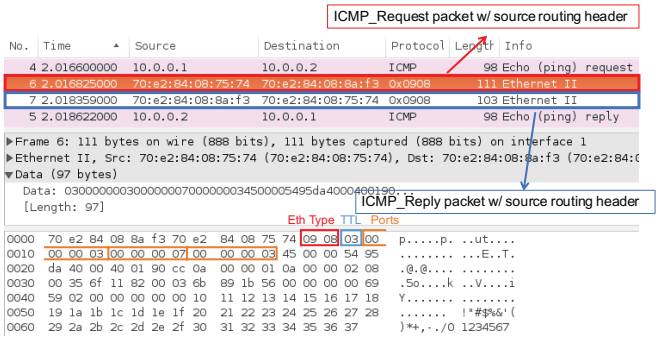
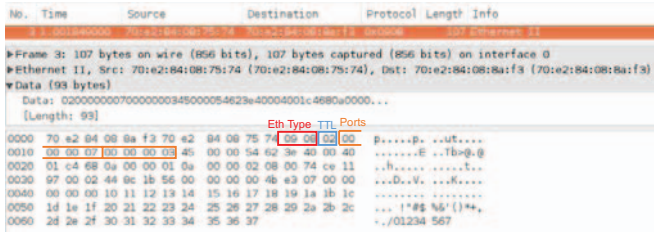


Fig. 7. Experimental testbed.

An *ICMP_Request* packet first arrives at edge switch S_1, where it finds that there is no match rules configured for it. Then, S_1 sends a *Packet-In* message to the POF controller to ask for the forwarding policy. The POF controller handles the *Packet-In* message, calculates a path together with the output ports on each hop along it, and instructs edge switch S_1 to convert the packets to source routing ones by encoding the output port information in them.



(a) ICMP packets captured on edge POF switches.



(b) ICMP packets captured on core POF switches.

Fig. 8. Wireshark captures to verify POF-based source routing.

Fig. 8(a) shows the ICMP packets captured on edge switch S_1. The *ICMP_Request* packet enters the switch from Ethernet interface *eth1*, and its designated output port on S_1 is *eth2*. We observe that the packet is converted to a source routing one, in which the *type* field in its Ethernet header is changed to "0x0908". Meanwhile, we notice that the packet's length increases from 88 Bytes to 111 Bytes, which also confirms that one *TTL* field (1 Bytes) and three *Port* fields (12 Bytes) are inserted into the packet. The *ICMP_Reply* packet from *Host* 2 to *Host* 1 travels in the opposite direction of the *ICMP_Request* packet. By looking at the *type* field in its Ethernet header, we can also see that the packet is converted to a source routing one at the edge switch. Meanwhile, since *Host* 2 sends *ICMP_Reply* to *Host* 1 to respond to *ICMP_Request*, we can verify that the egress edge switch of the *ICMP_Request* packet (*i.e.*, S_4) does restore it to a common IPv4 packet.

Fig. 8(b) shows the ICMP packet captured at the Ethernet interface *eth4* of core switch S_2. We observe that after passing one hop, the first *Port* field in the source routing header has been removed and the value of the *TTL* field has been decreased by 1. The results in Fig. 8(b) indicate that by using the packet processing procedure in Fig. 6, the core switch can forward the source routing packets to the designated output ports successfully.

B. Path Setup Latency

To evaluate the proposed POF-based source routing scheme further, *i.e.*, without being restricted by the network elements that we have, we modify Mininet [31] to support POFSwitch. Then, we emulate the topology in Fig. 7 with Mininet and increase the number of core switches in the setup to evaluate the system's performance on path setup latency. Specifically, in each experiment in Mininet, we emulate different numbers of core switches and measure the path setup latency from *Host 1* to *Host 2*. Note that, to emulate the situation in a practical SD-WAN, we also generate background traffic in the network. We use OpenFlow as the benchmark for performance comparison.

Fig. 9 shows the results on path setup latency. We can see that our proposed POF-based source routing scheme achieves much shorter path setup latency than the traditional scheme with OpenFlow. Moreover, it can be observed that the path setup latency from our proposed scheme does not increase with the number of switches on forwarding path, while for OpenFlow, the latency does increase significantly. This observation is crucial to verify that our proposed scheme can fit into the background of large-scale WANs better.

The reason why the POF-based source routing can achieve these advantages is two-fold. Firstly, since the controller does not have an unlimited processing capacity, it could be saturated if there are a lot of *Packet-In* messages for new flows and it has to respond by sending *Flow-mod* messages to all the switch on the forwarding paths. Apparently, this would not be an issue in our proposed scheme, since the controller only needs to configure the edge switches for path setup and does not

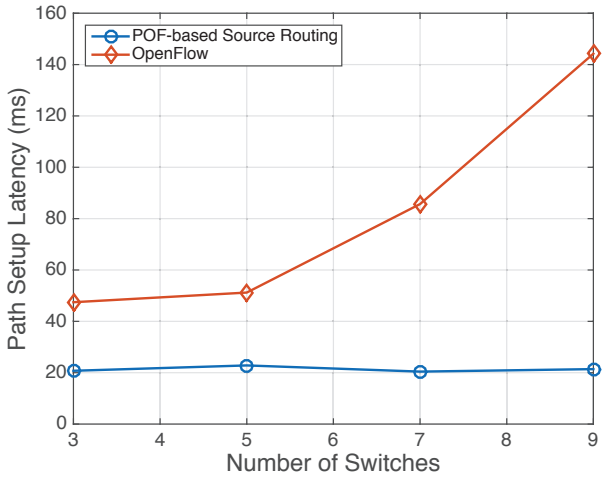


Fig. 9. Results on path setup latency.

have any interaction with the core switches. Specifically, the three flow tables shown in Fig. 6 can be pre-installed on each core switch to process any source routing packets. Secondly, in OpenFlow, the forwarding path can only be established after the controller has configured all the switches on it, while our POF-based source routing scheme only needs to configure the ingress and egress edge switches. Hence, the message propagation time can be saved as well.

VI. Conclusion

In this paper, we designed and implemented a novel network system that could leverage source routing with POF to facilitate efficient e-Health data transfers with low setup latency. We developed the POF-based source routing protocol to realize a pipeline based packet processing procedure, which could replace the table-lookup based approach in traditional SDN networks and make the forwarding plane more efficient. The proposed scheme was demonstrated experimentally, and the results verified that with it, the flow-tables installed in each core switches in a POF-controlled SD-WAN could be minimized and the path setup latency of traffic flows could be reduced significantly as well.

Acknowledgment

This work was supported in part by the NSFC Project 61371117, the Fundamental Research Funds for Central Universities (WK2100060010), Natural Science Research Project for Universities in Anhui (KJ2014ZD38), and the Strategic Priority Research Program of the CAS (XDA06011202).

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] O. Diallo, J. Rodrigues, M. Sene, and J. Niu, "Real-time query processing optimization for cloud-based wireless body area networks," *Inform. Sci.*, vol. 284, pp. 84–94, Nov. 2014.
[3] J. Rodrigues, S. Misra, H. Wang, and Z. Zhu, "Ambient assisted living communications," *IEEE Commun. Mag.*, vol. 53, pp. 24–25, Jan. 2015.
[4] S. Li, Z. Zhu, H. Li, and W. Li, "Efficient and scalable cloud-assisted SVC video streaming through mesh networks," in *Proc. of ICNC 2012*, pp. 944–948, Jan. 2012.
[5] Z. Bai *et al.*, "Experimental demonstration of SVC video streaming using QoS-aware multi-path routing over integrated services routers," in *Proc. of ICC 2013*, pp. 2276–2280, Jun. 2013.
[6] Z. Zhu, S. Li, and X. Chen, "Design QoS-aware multi-path provisioning strategies for efficient cloud-assisted SVC video streaming to heterogeneous clients," *IEEE Trans. Multimedia*, vol. 15, pp. 758–768, Jun. 2013.
[7] C. Tsai and J. Rodrigues, "Metaheuristic scheduling for cloud: A survey," *IEEE Syst. J.*, vol. 8, pp. 279–291, Mar. 2014.
[8] P. Lu, Q. Sun, K. Wu, and Z. Zhu, "Distributed online hybrid cloud management for profit-driven multimedia cloud computing," *IEEE Trans. Multimedia*, vol. 17, pp. 1297–1308, Aug. 2015.
[9] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.
[10] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. of the IEEE*, vol. 103, pp. 14–76, Jan. 2015.
[11] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.
[12] S. Li *et al.*, "Flexible traffic engineering (F-TE): When OpenFlow meets multi-protocol IP-forwarding," *IEEE Commun. Lett.*, vol. 18, pp. 1699–1702, Oct. 2014.
[13] W. Lu *et al.*, "Implementation and demonstration of revenue-driven provisioning for advance reservation requests in OpenFlow-controlled SD-EONs," *IEEE Commun. Lett.*, vol. 18, pp. 1727–1730, Oct. 2014.
[14] S. Ma *et al.*, "QoS-aware flexible traffic engineering with OpenFlow-assisted agile IP-forwarding interchanging," in *Proc. of ICC 2015*, pp. 8490–8495, Jun. 2013.
[15] OpenFlow Switch Specifications. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[16] S. Shirali-Shahreza and Y. Ganjali, "ReWiFlow: Restricted wildcard OpenFlow rules," *Comput. Commun. Rev.*, no. 45, pp. 29–35, Sep. 2015.
[17] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, pp. 136–141, Feb. 2013.
[18] M. Rifai, D. Lopez-Pacheco, and G. Urvoy-Keller, "Coarse-grained scheduling with software-defined networking switches," in *Proc. of SIGCOMM 2015*, pp. 95–96, Aug. 2015.
[19] 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.
[20] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[21] J. Yu *et al.*, "Forwarding programming in protocol-oblivious instruction set," in *Proc. of ICNP 2014*, pp. 577–582, Oct. 2014.
[22] 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.
[23] OF-PI: A Protocol Independent Layer. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/OF-PI__A_Protocol_Independent_Layer_for_OpenFlow_v1-1.pdf
[24] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. of INM/WREN 2010*, pp. 3–3, Apr. 2010.
[25] A. Dixit *et al.*, "Towards an elastic distributed SDN controller," in *Proc. of ACM HotSDN 2013*, pp. 7–12, Aug. 2013.
[26] X. Chen *et al.*, "Leveraging master-slave openflow controller arrangement to improve control plane resiliency in SD-EONs," *Opt. Express*, vol. 23, pp. 7550–7558, Mar. 2015.
[27] M. Martinello, M. Ribeiro, R. de Oliveira, and R. de Angelis Vitoi, "KeyFlow: a prototype for evolving SDN toward core network fabrics," *IEEE Netw.*, vol. 28, pp. 12–19, Mar. 2014.
[28] S. Jyothi, M. Dong, and P. Godfrey, "Towards a flexible data center fabric with source routing," in *Proc. of ACM SOSR 2015*, pp. 10:1–10:8, Jun. 2015.
[29] POFSwitch Introduction. [Online]. Available: http://www.poforwarding.org/document/POFSwitch_Introduction.pdf
[30] POX. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-InstallingPOX
[31] Mininet. [Online]. Available: http://mininet.org