Using Path Label Routing in Wide Area Software-Defined Networks with OpenFlow

Weidong Lin*, Yukun Niu*, Xia Zhang†, Lingbo Wei*, Chi Zhang*
*Key Laboratory of Electromagnetic Space Information, Chinese Academy of Sciences
University of Science and Technology of China, Hefei 230027, China
Email: wdlin@mail.ustc.edu.cn, xiaoniu@mail.ustc.edu.cn, lingbowei@ustc.edu.cn, chizhang@ustc.edu.cn
†School of Computer Science and Technology, Wuhan Univeristy of Technology, Wuhan 430070, China
Email: xia_whut@163.com

***Abstract*—Separating the control and data planes has brought many advantages such as greater control plane programmability, more vendor independence, and lower operational expenses in Software-Defined Networking (SDN). SDN deployments are possible in a variety of contexts including Wide Area Networks (WANs). However, the large propagation delays between the controller and switches raise several concerns for WANs such as performance limitations, longer routing time for flows and sensitivity of the controller placement. To address this issue, we propose a path label routing approach in SDN-based WANs using OpenFlow. Through hybridizing traditional hop-by-hop routing with path label routing, we reduce flow routing time and controller state distributions. In addition, we present an implement method without modifications to OpenFlow. Finally a prototype is implemented as a proof of the method in OPNET simulator. Experimental results demonstrate that the approach can bring significant performance gains such as the reduction of state distribution and the sensitivity of controller placement in SDN-based WANs.**

Keywords—Path Label Routing; WAN; Software-Defined Networking

I. Introduction

The current WANs mainly use MPLS [1] which has a complicated control plane. To improve the MPLS control plane, GMPLS [2] was designed as a superset of the MPLS control plane protocols. But GMPLS has failed in terms of actual deployment in wide area networks as unified control plane (UCP) for a variety of technologies C packet, time-slots, wavelengths, and fibers [3]. In [3], the authors propose SDN-based architecture to replace MPLS/GMPLS as it is simple, extensible, programmable and can be gradually adopted.

Software-Defined Networking (SDN) [4] is a new emerging network architecture that is aimed towards overcoming the limitations of traditional networking, making networks more dynamic, manageable, and innovative. Separation of the control and data planes in SDN significantly simplifies modifications to the network logically centralized control plane, enables the data and control planes to evolve and scale independently. A uniform vendor agnostic interface called OpenFlow [5] between control and data planes has succeeded in attracting commercial vendors [6]. The great advantages of SDN make the architecture a preferred deployment option in variety of use cases including WANs. B4 [7] presented by Google is one of the most famous SDN deployments in WANs. It has utilized SDN approaches to achieve above 90% utilization on many WAN links in their network.

Despite the great advantages of leveraging SDN/OpenFlow in WANs, certain challenges still arise. Since SDN depends on a logically centralized controller, all switches must communicate with the controller to obtain new state information to forward network flows. However, WANs are usually characterized by geographically dispersed network elements that span broad areas. Therefore this naturally leads to relatively long propagation delays between the controller and switches. Reducing the latency in the communication process between the controller and a switch is usually not an option in a WAN because it is constrained by the physical topology. This raises a need for an approach that reduces the volume of communication between switches and the controller.

In particular, from an SDN perspective, the placement of the controller in WAN becomes a major factor in network design as it affects the latencies between the controller and switches. A study in [8] has verified that optimizing the controller placement in a SDN-based WAN is a complicated and an expensive procedure. In addition when this procedure is completed only a small percentage of the possible network placements are near optimal, once again raising the need for an approach that reduces the sensitivity of such a network design parameter.

Due to most of the control communications involve state distributions, a method that reduces or eliminates state distributions would have a positive impact on performance. To address the problems related to limitations imposed by state distribution in SDN, a variation of source routing is proposed By Ashwood-Smith [9]. Inspired by him, Soliman *et al.* discuss using Strict Link Source Routing (SLSR) in OpenFlow detailedly [10]. But Soliman *et al.* just analyze it in theory without a corresponding implementation. SlickFlow [11] gives the specific source routing implementation which provides links failure recovery by carrying an alternative path message based on Slick Packet [12] in SlickFlow header. But the implementation requires too much path information as

This work was supported by the Natural Science Foundation of China under Grants 61202140 and 61328208, by the Program for New Century Excellent Talents in University under Grant NCET-13-0548, and by the Fundamental Research Funds for the Central Universities under Grand WK2101020006.

well as varies the header length greatly leading to being not conducive to hardware implementation. Moreover, OpenFlow protocol was modified in SlickFlow and it is difficult to promote.

To avoid modifying the existing OpenFlow protocol, this paper presents a hybrid routing scheme, mixing traditional hop-by-hop routing and path label routing. Path label routing is similar to source routing, but the data packets carry only a path label instead of full path information. Headers of path labels are fixed length size and much shorter than that of S-lickFlow. In MPLS, labels are local valid and will be modified before forwarding [5]. This causes some unnecessary delay. In comparison with labels in MPLS, ours are globally unique. The packets using path label routing, only will be pushed label headers in ingress switches and popped the headers in egress switches. The intermediate switches will just rely on the packet path label headers for forwarding but do not change the path label headers. In regard to intermediate switches in the paths, we can distribute corresponding flow table entries to them during network initialization. The intermediate switches forward the packets matched a path label directly and need no communications with the controller.

It is impossible to allocate a path label to each path in a large-scale network. So the hybrid approach is adopted instead of the pure path label routing. The controller can allocate path labels according to statistics of network characteristics collected from the OpenFlow switches. Of course, network operators can allocate path labels via the controller based on business needs or traffic patterns. If a flow needs to pass a path without a path label, the controller will perform conventionally. The larger percentage of path label routing in hybrid routing, the better the network performs.

Furthermore, we propose an implement method without modifications to OpenFlow. We demonstrate the implement method in OPNET simulator. The experimental results indicate that through mixing path label routing the network average latency significantly reduces.

The contributions of this work are summarized as follows:

• To our knowledge, this is the first paper that presents mixing path label routing with conventional routing in SDN-based WANs using OpenFlow.

• We proposes a workable implement method of path label routing without modifications to the OpenFlow protocol.

The rest of this paper is organized as follows. First we design our hybrid approach in Section II. In Section III, we analyze our idea from several aspects. In Section IV, we introduce our implementation in OPNET simulator and exhibit results of our simulation scenarios. Finally, we conclude our work in Section V.

II. Architecture Design

Fig. 1 highlights the main idea of the network architecture. Switches are OpenFlow-enable switches in Fig. 1. They can forward a flow using path label routing or conventional routing. The controller is a logically centralized controller [13] in this paper, can be a single controller in physics such as NOX [14], Floodlight [15] and ryu [16], as well as a physically distributed control plane, like Onix [17], HyperFlow [18] e.g.. Whether the control plane is physically centralized or distributed is not our concern. A study in [8] has shown one controller often suffices. Therefore, we use a single controller for concise description in this paper.

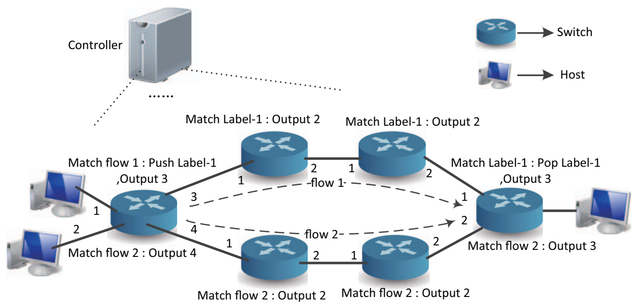

Fig. 1. Overview of the hybrid routing architecture.

We mix conventional routing and path label routing using OpenFlow. In conventional SDN/OpenFlow routing, the controller determines a path for an unknown flow and sends new flow table entries to all switches along the path concurrently except for the ingress switch. Once all acknowledgements are received, the controller sends a new flow table entry to the ingress switch and the forwarding path is established. This paper concentrates on the design of the path label routing, and the following analysis involves the path label routing emphatically.

A. A path with a label Identification

In MPLS, labels are local valid and will be modified before forwarding [1]. This causes some unnecessary delay. Contrastingly, our labels are only pushed in ingress switches and popped in egress switches, do not be modified in each intermediate switch. We can distribute corresponding flow table entries to intermediate switches during network initialization. The intermediate switches directly forward the packets matched an path label and need no state distribution from the controller.

As shown in Fig. 2, a label can identify a path between two switches. It is worth noting that we can identify multiple paths which have the same source and destination switches via different labels.

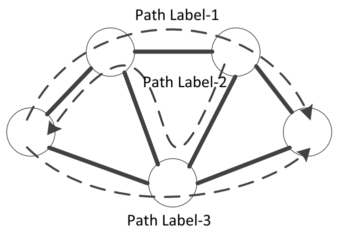

Fig. 2. Schematic diagram of path labels.

B. Store path labels in packets

To avoid modifications of OpenFlow, we store our fixed-length labels in OpenFlow support header fields. According to the last OpenFlow Specification [19] until now, there are three header fields suitable for storing, which are MPLS header field, VLAN header field and PBB header field. This paper uses VLAN header field, the simplest among them, as storage place of labels. In VLAN header, there are 12 bits of VLAN ID and 3 bits of VLAN priority [20] for storing use. It can provide up to 32768 label identifications. In view of a VLAN header probably already exists in users' data packets, we use VLAN header field of QinQ [21]. Switches push the VLAN header must contain the appropriate QinQ header field type indication (0x88a8). Fig. 3 exhibits a VLAN tagged frame and a QinQ tagged frame. The gray fields displayed in Fig. 3 are used for storing labels. It is worth mentioning that SlickFlow header requires several hundred bits [11] while ours is 12 bits. Therefore, the header size of path label routing leads to lower transmission latency than that of SlickFlow.

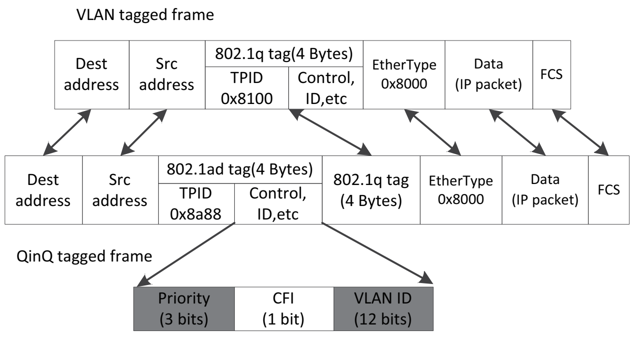

Fig. 3. VLAN frame versus QinQ frame.

C. Failure and modifications of path labels

1) Idle timeout and remove of path labels: The controller can decide idle timeout intervals of path labels. The corresponding flow table entries will time out after the intervals with no received traffic and the switches sends flow removed messages to notify the controller. Certainly, for optimization, management or other reasons, the controller can remove path labels initiatively by removing corresponding flow table entries. But the controller must establish an alternative path for the flows firstly. The timeout and removed labels' identifications will be reused by the controller.

2) Smooth Handover: If two paths with the same source and destination addresses are identified with two labels, we can handover the traffic from one to the other smoothly. To accomplish this task, the controller merely needs to modify the push label entry in the ingress switch replacing the pushed label with the other. This feature is extremely convenient to balance traffic load and failure recovery.

D. Link failure recovery

In [22], the authors use fast-failover type of the group-entry[19] to switch traffic from the faulty path to the fault-free alternative path upon failure. They simulate in high speed testbed using German backbone network topology, the simulation results are satisfied. In an analogous manner, path label routing adopts the same program and stores an alternative path label in group-entry. Table I describes main components of a group-entry in a group table. We put fast-failover type indication in Group Type field and an alternative path label change action in action buckets.

TABLE I
MAIN COMPONENTS OF A GROUP-ENTRY IN A GROUP TABLE.

Group Identifier	Group Type	Counters	Action Buckets

An example as shown in Fig. 4, when the link between switch b and switch d fails, switch b can replace the label-*1* with the label-*2* for correct forwarding. The general approach in our local protection as follows. For a path has link $l(i, j)$ and $l(j, k)$, where i, j and k are nodes in the path. When $l(i, j)$ fails, a typical treatment in our approach is to make a bypass link from i to j replace the original link. Therefore, link failure conditions can be handled locally in this approach. Unfortunately, there is no alternative path label to handle a link failure, the switch will inform the controller to decide how to deal with it.

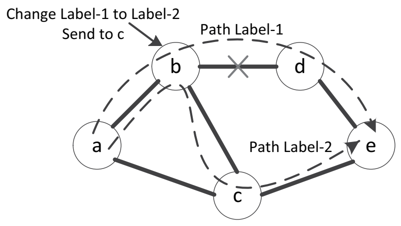

Fig. 4. A demonstration of link failure recovery.

III. ANALYSIS OF PERFORMANCE

A. State distribution

Using the path label routing, as shown in Fig. 5, new state information will be pushed to only the ingress switch, instead of having to distribute states to each switches along the path. Assuming the number of switches in an path to be five, the controller only needs to distribute one-fifth the amount of states distributed if traditional OpenFlow is used.

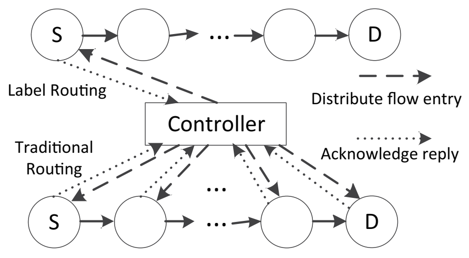

Fig. 5. Schematic diagram of state distributions.

This significant state reduction directly improves the network average latency and reduces the sensitivity of controller placement. The ratio of the state reduction in a path is[10]:

$$state\ reduction = \frac{n-1}{n} \times 100\% \tag{1}$$

where n is the number of switches in an path. The curve of state reduction displays in Fig. 6.

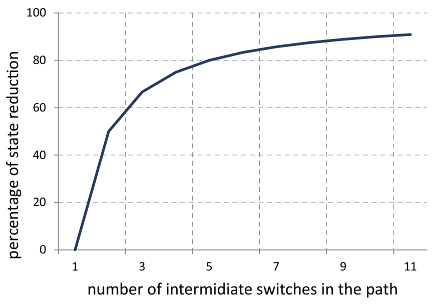

Fig. 6. State reductions with different path lengths.

B. *Sensitivity of controller placement*

As mentioned earlier, from an SDN perspective, the placement of the controller in WAN becomes a major factor in network design as it affects the switch-to-controller latencies. Optimizing the controller placement in wide area software-defined networks is complicated and expensive. In addition when this optimizing procedure is completed only a small percentage of the possible network placements are near optimal. Fortunately, the path label routing can reduce the sensitivity of such a network design parameter.

When the percentage of path label routing large enough, we can consider distributed state reductions of the network equal to that of path label routing analysed above. In this situation, the controller sends state information to ingress switches only. From the perspectives of unknown flows, the time of building paths improves a lot. Therefore, to a certain extent the sensitivity of controller placement reduces. We prove it through simulating in Section IV.

C. *QoS*

Utilizing meter table in OpenFlow [19], we have been able to achieve essential QoS. Table II describes main components of a meter entry in a meter table. Each meter entry is identified by its meter identifier and contains:

• Meter Identifier: a 32 bit unsigned integer uniquely identifying the meter.

• Meter Bands: an unordered list of meter bands, where each meter band specifies the rate of the band and the way to process the packet.

• Counters: updated when packets are processed by a meter.

TABLE II
MAIN COMPONENTS OF A METER ENTRY IN A METER TABLE.

Meter Identifier	Meter Bands	Counters

Per-flow meters enable OpenFlow to implement various simple QoS operations, such as rate-limiting, and can be combined with per-port queues to implement complex QoS frameworks, such as DiffServ [23]. Our hybrid routing approach makes it more practical and convenient. Through establishing dedicated and alternative path labels in advance, it is easy to meet QoS demands.

IV. SIMULATION AND RESULTS

In this section, we try to implement our idea in OPNET simulator ver14.5. The Open Science, Scholarship and Services Exchange (OS^{3}E) [24] has a 34-node SDN across the US to support advanced global scientific research. In this paper, we use OS^{3}E topology to simulate. Fig. 7 shows the simulation topology of OS^{3}E in OPNET.

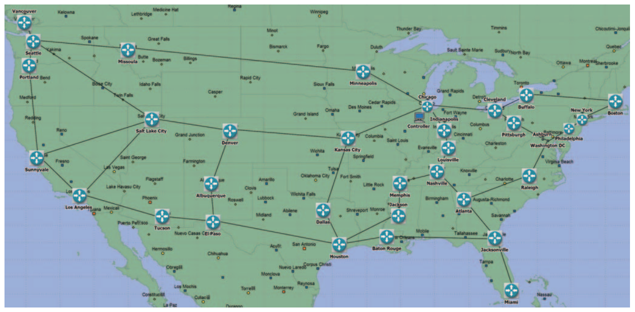

Fig. 7. Simulation topology of OS^{3}E.

Owing to OS^{3}E mainly use Juniper T1600 and MX960 as infrastructures, we assume all port rates of OpenFlow switches are 10GbE in our experiments. Considering a fair estimation as the effect of an error on the propagation delay in a fiber link, the link speed would be minimal the speed of light in Fiber ($2 \times 10^8 m/s$). With respect to the controller, we adopt NOX-MT [25] which can handle 1.6 million requests per second on an eight-core machine with 2GHz CPUs. The processing time of the controller to handle requests obeys negative exponential distribution as shown in the following equation:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

where x is processing time, and $\frac{1}{\lambda}$ is the average processing time.

In order to highlight the influence of conventional routing and path label routing, we make some simplification. All flows transport through shortest paths calculated by the controller with Dijkstra algorithm. The controller distributes different labels for each shortest paths. Different flows between a pair of switches can be forwarded in conventional routing or path label routing decided by the controller. When the controller decide the routing mode of a flow, the controller will send new flow table entries to related switches as described in Section II, and the forwarding path is established. The mixing ratio mentioned below in the simulation refers the proportion of flows adopted path label routing in all flows. Unless otherwise stated, latency of a flow means the interval between the generation and arrival of the first packet of the flow in this paper.

A. *Performance in terms of different mixing ratio*

We firstly test the performances in different mixing ratio with increasing numbers of total flows in the network. If mixing ratio is 50%, an unknown flow has the probability of fifty percent to forward with path label routing. The position of the network controller was chosen in Chicago based on the analysis results from [8] so as to minimize average latency from the network switches to the controller. As shown in Fig. 8, average delay of flows increases while the number of total flows increases due to increasing traffic load. Obviously, it indicates that the larger mixing ratio of path label routing leads to the lower network latency.

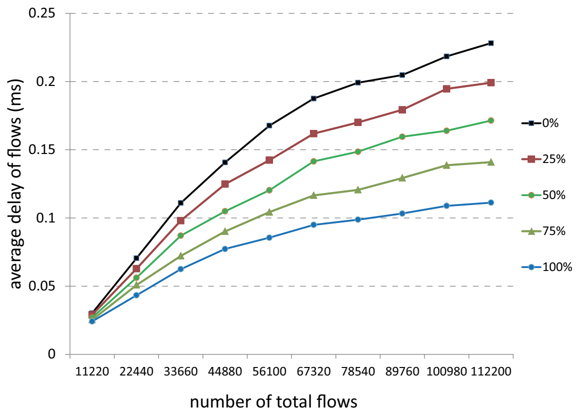

Fig. 8. Latencies in different mixing ratio networks with increasing numbers of total flows.

B. *Impact of different intervals of flow table entry timeout*

The longer the time interval of flow table entry timeout, the more possible switches can handle a new flow locally. The average latency of the network is positively related with the interval. However, the interval cannot be too lager result from the problem of flow table expansion. We can draw a conclusion from the following experiments that path label routing can improve the average latency even when the interval of flow table entry timeout reduces. We measure the network average latencies of flows under the following conditions that the intervals are set to 60 seconds, 120 seconds, 180 seconds, 240 seconds and 300 seconds, respectively. The number of total flows is more than 112 thousands.

Taking the average latencies of the network whose timeout interval equals to 300 seconds as a reference object, the increasing rates of the average latencies with other intervals are displayed in Fig. 9. We can see that increasing rate is negatively correlated with percentage of path label routing.

C. *Performance in terms of different controller placements*

For a network graph $G(V, E)$ where edge weights represent distances, where $d(\nu, s)$ is the distance of the shortest path from node $\nu \in V$ to $s \in V$, and the number of nodes $n = |V|$, the average distances for a placement of controllers $L_{avg}(s)$ is:

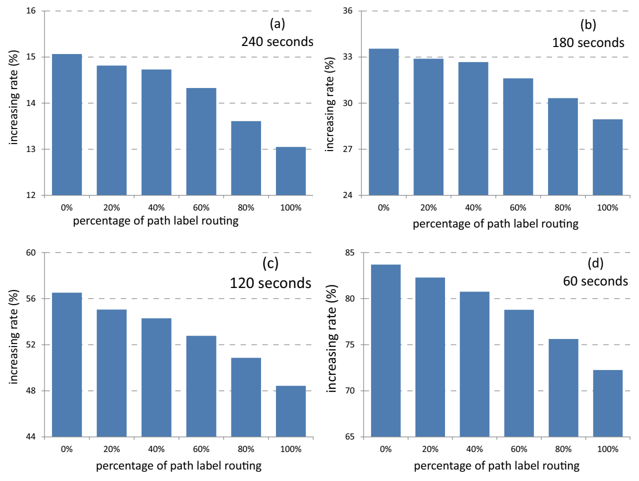

Fig. 9. Increasing rates diagrams, (a) timeout interval is 240 seconds, (b) timeout interval is 180 seconds, (c) timeout interval is 120 seconds, (d) timeout interval is 60 seconds.

$$L_{avg}(s) = \frac{1}{n} \sum_{\nu \in V} d(\nu, s) \tag{3}$$

Fig. 10 exhibits the average distances of all cities in OS^{3}E topology. Chicago has the shortest average distance which means minimum average latency. The result corresponds to [8]. We can see that Vancouver has the largest distance nearly twice as large as the shortest one, and the distance of Denver is the median.

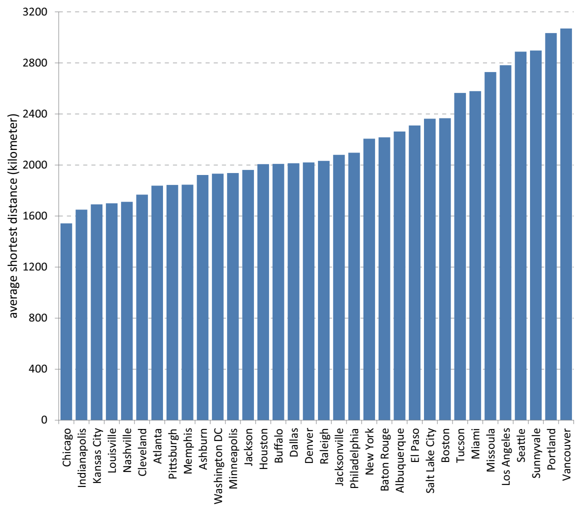

Fig. 10. The average distances for controller placements of cities in OS^{3}E.

We measure the network average latencies of flows with the controller placed in Chicago, Denver and Vancouver, respectively. The number of total flows is more than 112 thousands. Taking the average latency of the network containing 0% path label routing and the controller placement is Chicago as a reference object, the change rates are exhibited in Fig. 11. We can see that as the percentage increases, the average latencies of Denver and Vancouver increase less, even that starting from 40% in Denver the latencies less than the latency of Chicago's. It also indicates that the latencies of controller placement in Vancouver with the percentage of path label routing large enough can less than that in Chicago with 0% path label routing. From the results we can arrive at the conclusion that path label routing is an effective approach reducing the sensitivity of controller placement. It proves again that the approach can bring significant gains in SDN performance in WANs.

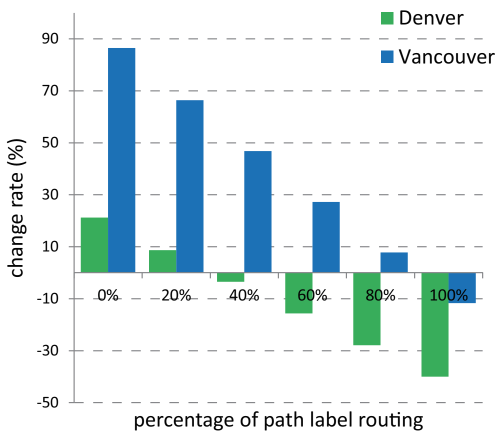

Fig. 11. Change rate comparing to controller placement in Chicago with containing 0% path label routing.

V. Conclusion

In this paper, we propose a path label routing approach in SDN-based WANs and mix it with conventional routing using OpenFlow. Moreover, we propose a workable implement method of path label routing without modifications to the OpenFlow protocol. Finally, we implement our idea in OPNET simulator. Both results of analyses and experiments demonstrate that under the conditions that having enough label resources or most flows traffic involving path labels, the network average latency and state distribution will significantly reduce. In other words, the larger percentage of path label routing in hybrid routing, the better the network performs.

References

[1] R. Callon and E. C. Rosen, "Multiprotocol label switching architecture," 2001.

[2] A. Banerjee, J. Drake, J. Lang, B. Turner, D. Awduche, L. Berger, K. Kompella, and Y. Rekhter, "Generalized multiprotocol label switching: an overview of signaling enhancements and recovery techniques," *Communications Magazine, IEEE*, vol. 39, no. 7, pp. 144–151, 2001.

[3] S. Das, G. Parulkar, and N. McKeown, "Why openflow/sdn can succeed where gmpls failed," in *European Conference and Exhibition on Optical Communication*. Optical Society of America, 2012, pp. Tu–1.

[4] "Software-Defined Networking: The new norm for networks," https://www.opennetworking.org/, [Online].

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] "SDN Solutions Showcase," https://www.opennetworking.org/news-and-events/sdn-solutions-showcase-2015, [Online].

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[8] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[9] P. Ashwood-Smith, "Software defined networking and centralized controller state distribution reduction–discussion of one approach," *IEEE Draft, July*, 2012.

[10] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 2012, pp. 43–44.

[11] R. M. Ramos, M. Martinello, and C. Esteve Rothenberg, "Slickflow: Resilient source routing in data center networks unlocked by openflow," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*. IEEE, 2013, pp. 606–613.

[12] G. T. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. Godfrey, and S. Shenker, "Slick packets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 39, no. 1, pp. 205–216, 2011.

[13] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 1–6.

[14] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[15] "Floodlight OpenFlow Controller," http://www.projectfloodlight.org/, [Online].

[16] "Ryu SDN Framework," http://osrg.github.io/ryu/, [Online].

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[18] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[19] "OpenFlow Switch Specification: Version 1.5.0 (Protocol version 0x06)," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf, 2014, [Online].

[20] 802.1Q, "Local and metropolitan area networks, virtual bridged local area networks," 2003.

[21] 802.1ad, "Ieee draft standard for local and metropolitan area networks, virtual bridged local area networks, amendment 4: Provider bridges."

[22] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A demonstration of fast failure recovery in software defined networking," *Testbeds and Research Infrastructure. Development of Networks and Communities*, pp. 411–414, 2012.

[23] K. Nichols, D. L. Black, S. Blake, and F. Baker, "Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers," 1998.

[24] "Internet2's Advanced Layer 2 Service (Formerly OS3E)," http://www.internet2.edu/products-services/advanced-networking/layer-2-services/, [Online].

[25] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, vol. 54, 2012.