Segment Routing in Hybrid Software-Defined Networking

Ziqiang Li, Liusheng Huang, Hongli Xu, Gongming Zhao
Department of Computer Science and Technology
University of Science and Technology of China
Hefei, China
e-mail: {lzqrush, zgm1993}@mail.ustc.edu.cn, {lshuang, xuhongli}@ustc.edu.cn

***Abstract*—Software-defined networking (SDN) decouples the network into different layers, bringing many new attributes over traditional network. But SDN also comes with its own set of challenges and limitations. Combining the mature traditional network with new benefits of SDN, hybrid SDN has attracted significant attention. On the other hand, segment routing keeps partial routing information within packet header so that switch can directly dispatch without calculate routing path or looking flow entry table. In this paper, we first propose a mechanism which integrates segment routing into hybrid SDN that can reduce the needs of flow entries of SDN switch. Then we formulate a routing algorithm for this mechanism. Our algorithm considers the balance of traffic load and reduces the needs of flow entries in each switch. Simulation results show that the performance of our mechanism significantly reduces the number of flow entries than compared solution, and achieves better load balance compared with previous routing protocols.**

Keywords-software defined networking (SDN); openolow; hybrid SDN; segment routing; traffic engineering

I. INTRODUCTION

A. Hybrid SDN

Because of the flexible, manageable and responsive to rapid changes, software-defined networking has attracted significant attention from many researchers. Those qualities of SDN enable network designers to directly control network through programmable open standardized interface called OpenFlow [1]. SDN promises the ease of network design, operation and management and therefore, breaks through the bottleneck to the acceleration towards the age of cloud computing that networks are evolving to.

With the architecture of decoupling the control framework from data plane, SDN can obtain a global view of entire network and provide centralized control over complementing network function virtualization (NVF). SDN's inherit advantages unlock the new potential of the network fabric.

But along with the promising feature, inevitably, SDN itself still remains many unresolved challenges. According to OpenFlow protocol, the SDN attribute implies that an SDN switch requires a larger sized flow table than that of a traditional switch for storing the same number of flows [2]. Flow tables are implemented by TCAM (Ternary Content Addressable Memory), which is small, costly and energy-hungry [3]. And all the switches are reported to controller, every decision is made by controller and then controller distributes rules to all the switches. This results significant burden to controller and the link between switches and controller.

On the other hand, compared with newly born networking paradigm, traditional network has evolved through decades. Most protocols and techniques have been tested by large-scale network in a long period of time. With its mature functionalities and proven success, conventional network is more attractive to the network operators.

So summing up the advantages of both SDN and traditional network while mitigating their respective challenges into Hybrid SDN seems one better choice. Targeting to these concerns, OpenFlow protocol v1.3.0 in [1] proposed a new hybrid switch mode called OpenFlow-hybrid. OpenFlow-hybrid switches support both OpenFlow operation and normal Ethernet switching operation. Such as traditional L2 Ethernet switching, L3 routing and Qos processing.

Reference [4] classifies hybrid SDN into four types, topology-based, service-based, class-based and integrated hybrid SDN. In the first three hybrid models, traditional network node and SDN node complement each other by controlling disjoint parts of node forwarding information. From perspective of control plan, the control power is divided into two parts, one part is distributed on the traditional switch, and the rest belongs to SDN controller, which can cause severe inconsistency [4]. On the contrary, integrated hybrid SDN only has one kind of node, but contains two working modules, OpenFlow protocol and traditional network protocol. It is responsible for all the network services, and uses traditional network protocols as an interface to node. Controller can get full access to both situation, have entire network status and control network traffic. However, Hybrid SDN contains two kinds of traffic: Normal Ethernet traffic and OpenFlow traffic. The one big problem of hybrid SDN is the co-exist of normal Ethernet traffic and OpenFlow traffic. Normally, two kinds of traffic either share one link, or preserve certain percentage of link capacity respectively.

B. Segment Routing

Restricted by high cost of TCAM, many solutions have been proposed and one of them is segment routing (SR) [5]. Usually, traditional switch stores the entire network topology and then forwarding packet based on computed path

978-1-5090-3822-0/17/$31.00 ©2017 IEEE

according to this network topology. For every new flow, Openflow switch intercepts flow packet header and reports to the controller, then controller computes the forwarding path, generates forwarding rules and distributes to all switches. Either way, switch has to keep a large amount of forwarding information. The key idea of SR is to keep partial forwarding information within the packet rather than store in switch. This approach can largely reduce the network information status that is stored in each switch and hence minimizes the needs of TCAM for Openflow switch. As the number of forwarding rules become smaller, the complexity of maintaining a large-scale forwarding rules is much eased.

Taking advantage of the global view of SDN, SR can be perfectly implanted into SDN. Controller calculates routing path, divides the path into several segment, generates segment routing entry and distribute routing rules down to switches.

The contributions of this paper are as follows. First, we propose a hybrid segment routing mechanism. This mechanism integrates segment routing into Openflow-hybrid mode, which becomes a new hybrid SDN model. Second, we formulate the traffic engineering problem targeting this new hybrid SDN model as an integer linear program, and one algorithm is designed to solve this problem.

The rest of the paper is organized as follows. In Section II, we summarize related work. In Section III, we present the hybrid segment routing mechanism in details. In Section IV, we present the traffic engineering targeting proposed hybrid SDN model. In Section V, we evaluate system performance. Finally, we conclude the paper in Section VI.

II. Related Work

The authors in [6] concerned that today's SDN system still remain a large set of unsolved challenges and keeps high deployment cost, most enterprise are willing to incrementally deploy SDN element. The co-exist of SDN element and traditional network device resulting in a transitional form of networks a hybrid SDN (H-SDN). Authors look into traditional network and SDN, analyze the routing protocols, and proposed a new routing protocol for H-SDN, resolves the problem of SDN traffic passing through traditional network element. Then authors modeling the traffic engineering problem in two modes of H-SDN: barrier mode and hybrid mode. Reference. [7] proposed a novel label switching mechanism by rewriting mac address field with "shadow" MAC address. Each shadow mac address is associated with one spanning tree to achieve traffic engineering. And whole network uses multiple spanning trees rooted at each destination. In this case only few path are available. With the overhead in mac address rewriting, limited path can affected throughput performance.

The authors in [8] raises the issue that even though classical traffic engineering methods can achieve the optimal routing based on single traffic matrix, but it not handle unexpected traffic changes, and propose an approach named hybrid routing. The key idea of this approach is to select small fraction of nodes pairs. The traffic between selected node pairs using explicit routing forwarding, and rest of traffic using destination-based routing. All the explicit routing entries are stored in fast but expensive TACM table and using cheap but large SRAM table to keep destination-based routing entries. Incoming packet will be first matched by TCAM table. When hit one entry, router then apply the action which was specified by that entry. If misses, the router continue to match entries in the SRAM table like normal destination-based routing protocol. Authors try to take advantage of both routing approaches to achieve load balancing.

SecondNet in [9] uses port-switching based source routing (PSSR) to forwarding packet, implement with MPLS. But MPLS increases header overhead because each label length is 4 bytes. Ref. [3] proposed a routing algorithm based on segment routing, authors limit the length of MPLS through hop count constraint. JumpFlow in [10] uses SR method, but it applied into whole network. This caused overhead of egress switch to rewrite the packet. Our method select part of the traffic, the rest of network can still choose normal Ethernet or OpenFlow pipeline.

III. Design of Hybrid Segment Routing Mechanism

We target OpenFlow-hybrid SDN. The objective of hybrid SDN is to manage the co-exist of normal Ethernet traffic and OpenFlow traffic. To ensure this circumstance, we adopt the hybrid mode of OpenFlow switch [1]. The goal of this paper is to design a forwarding mechanism that can achieve balanced flow table usage and load balancing through SR based on Hybrid SDN, denote as HSR. In our considered OpenFlow-hybrid SDN, network traffic shared one link between switches and each switch contained two layer: Normal Ethernet Switching Layer (NESL) and OpenFlow Switching Layer (OFSL). In NESL, Open Shortest Path First (OSPF) is the most commonly used routing protocol to achieve optimized traffic. In OFSL, after studied the current internet protocol, we found that there exists many unused place in packet header (VLAN identifier, etc.). Thanks to the open programmable of SDN, we can put forwarding information into those space, then switch based on those information can decide which port should be forward to. In this paper, forwarding information simply means forwarding port number and so called port-based segment routing. Controller will determine which new flow is using this approach and install classify entries into switch. When one new flow is using this approach, controller calculate the flow path and install related rules into switch along the path. Switch at the beginning of each segment insert the list of forwarding port number into packet header. The field of routing segment is depicted in Fig. 1.

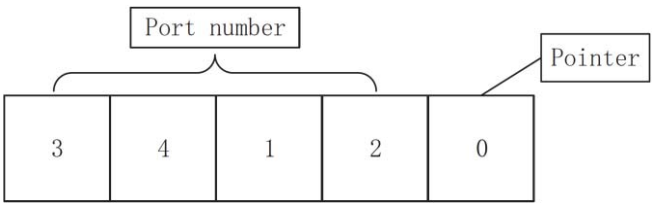

Figure 1. Examples of forwarding field in packet header

Assuming that the length of each segment is 4. After 4 ports is the offset which indicate the port number of switch should be used. At the end of entry actions, the offset should increase. After four switches, the fifth switch will contain a rule specify for this flow or one destination to load next segment route path and the offset is set to 0. The last action of this rule is to forward packet to one port number at this loading node. So it is clear that packet header do not have forwarding information of loading node, but contained within those rules. The purpose of this design is maximally reduces forward information on each packet. We should note that the length of segment can be optimized according to different network size.

Not all flow are using port-based segment routing, real network is composed of many kinds of traffic. On the prospective of quantity of flows, the majority are short-lived with light traffic [11]. The quantity of those flow is large but traffic bytes are relatively small compared with elephant flows and have limited influence on network. When large amount of new short-lived with light traffic hit the network, switches need to communicate with controller to request forwarding rules. This process add additional delay on the entire network. It is wise to direct these flow through NESL, for the purpose of minimizing the number of forwarding rules. On the contrary, elephant flow are dominate in traffic volume although their number may be relatively small [11], so it is wise to steer those traffic into OPSL and using port-based forwarding method for better control of network. Also controller can choose any flow to use segment routing mechanism according to different traffic engineering circumstance.

A. *Overview of HSR Mechanism*

In OpenFlow, OpenFlow-compliant switch comes in two types: OpenFlow-only, and OpenFlow-hybrid. OpenFlow-only switches support only OpenFlow operation. In those switches all packets are processed by the OpenFlow pipeline. And the switch can only handle OpenFlow traffic, which mean many function of traditional switch is not available. But for OpenFlow-hybrid switch, at every packet header, there is one field indicating this packet is whether openflow packet or traditional packet. For example, switch may use the VLAN tag to decide whether to process the packet using Openflow pipeline or the other.

In our designed mechanism, for every new flow, controller should decide whether this flow goes to NESL or OFSL, and whether using segment routing, then distribute one rule to inform ingress switch injecting classification tag into every packet belongs to this flow. We denote this rule as tag rule.

In Fig. 2, we demonstrate the main process of our designed HSR mechanism. When packet entering a switch, it will be handled by classification process. The classification process is to check specific packet header tag. The classification tag divides traffic into two parts. One part goes to NESL, and the rest goes to OFSL. If the packet is classified to normal flow, it will be forwarded to normal Ethernet switching operation. Otherwise, The OpenFlow operation catch this packet and use OpenFlow pipeline to handle it. Next we will discuss these two layers in detail.

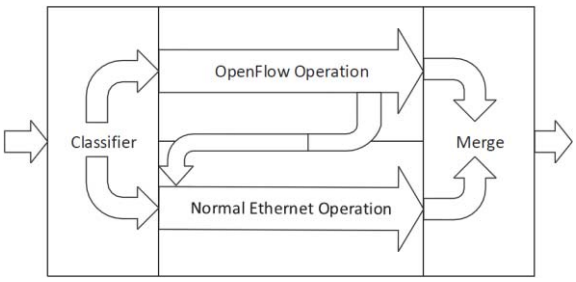

Figure 2. Examples of forwarding field in packet header

B. *Normal Ethernet Switching Layer*

Incoming packet through NESL has two sources. One is packet forwarded by adjacent switch, another is forwarded by OpenFlow pipeline within the switch. NESL identify flows by their destinations and forward them to corresponding outgoing links along the shortest paths that are calculated via OSPF protocol. NESL do not aware of the difference between two sources packet and treat SDN packet as normal packet. NESL traffic is assumed to be optimized via standard link-state routing protocols such as Open Shortest Path First Traffic Engineering (OSPF-TE).

C. *OpenFlow Switching Layer*

We integrate port-based segment routing into this layer. Port-based segment routing uses unused tag within packet header to store forwarding information. For example, the next header in IPv6 protocol is used to packet header extension. So data structure which depicted in Fig. 1 can be injected into those spaces.

We discuss OFSL from control plane and data plane:

Control Plane: The control plane is implemented via extensions to a centralized SDN controller. All the new flows are controlled by network controller. Here we only discuss the control procedure of traffic which using segment routing. Other traffic is standard OpenFlow procedure.

First, controller sent the tag rule into ingress switch of this flow. Then controller calculate the path of the requests, get the port number of switch along the path that need to be forward to and split the list of port number into several segment. The length of each segment of hop path is set to a reasonable number, this paper is set to 4. The switches in each beginning of segment list have the responsibility of loading segment list into packet, other switch just forward packet.

The controller sees switches into two types: loading nodes and forwarding nodes. Forwarding node simply get the port number, which this packet will be forwarded to, based on the pointer. On the basis of forwarding node, loading node have one more function that is loading routing segment into packet header. The specific rule of loading segment list is denoted as loading rule.

So it is obvious that every node can be loading node and forwarding node at the same time from the perspective of

different flows. Controller can select any flow or any sub-path of specific traffic to use port-based routing.

For the ingress switch, the last action of tag rule is to match the flow entry table which contains loading rules to loading the first segment list.

Data Plane: Two types of nodes in SDN Routing procedure: loading nodes and forwarding nodes all have been pre-installed a default forwarding action, denote as DF action. The functionality of this default action is simply extract the forwarding port number which stored in packet header, then forwarding this packet through given port number.

When packet is classified as OFSL packet, switch will first check packet for segment routing tag. When this packet is using segment routing, switch will compare the pointer with the length of segment list. If pointer number is smaller, DF action will be execute on this packet. If pointer number exceed the length of segment list, means packet needs reload segment list. So packet will be matched by flow table which contains loading rules and reload the segment list.

According to [1], OpenFlow pipeline contains multiple flow tables, each flow table contains a certain amount of flow entries. And the number of tables within OpenFlow pipeline are in ascending order, starting at 0. All the incoming packet will be firstly matched against flow entries of flow table 0. Packets are matched by one or more table. First, packet will be match against table 0 for segment routing tag. If hit, indicates packet needs to load next segment list, then goes to the table which contains all the loading rules. If missed, means this packet is standard OpenFlow packet. Loading rule have two actions: loading routing segment list and forwarding packet through given port.

When packet arrives at loading node, the forwarding information of this node is given by loading rule rather than insert into packet header. Switch find out that the pointer is bigger than given length, so start to match against the flow table which contains all loading rules.

Fig. 3 demonstrates how data plane works. Host a sends flow to Host b. S1 reports to Controller. And controller decide this flow using segment routing protocol. Then controller informs ingress switch to inject OFSL and segment routing protocol tag into this flow, and calculates the path. The length of routing path is 7, so should split into 2 segments. Controller then install one loading rules in switch s1 and s6 for loading segment routing list. When the rest packets of this flow reach its ingress switch s1, first it will be tagged OFSL tag and SR tag, then match all the loading rule. Matched by installed loading rule, according to action 1, switch load routing list {2, 3, 4, 3} into packet header, and pointer is set to 0. According to action 2, this packet is forwarding through port 1. For the next four hop, switch simply execute DF action. When packet arrived s6, pointer is exceed length of segment list. Then s6 extract packet information and look up the flow entry table contains all loading rules. Because the length of rest path is small than 4, so the extra field is 0. After s7, packet is reached its destination. So this flow only takes total two flow rules among seven hop.

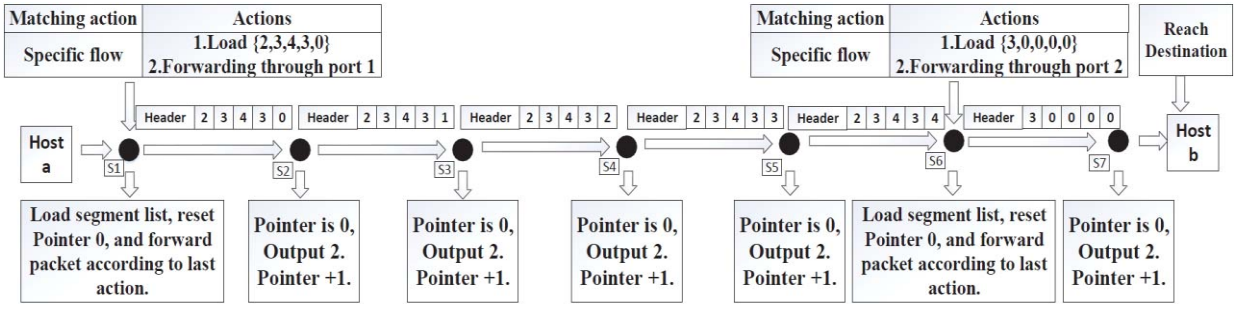

Figure 3. Examples of segment routing in SDN

IV. TRAFFIC ENGINEERING IN HYBRID SDN

A. Network Model

We formulate the hybrid network as a directed graph $G(V, E)$, where V is the set of switch nodes with $n = |V|$, E is the set of directed link in the network. Switches in network are working under hybrid mode and network links are shared by Normal protocol traffic and OpenFlow traffic. The whole network traffic is divided into two types: port-based traffic (using segment routing mechanism) and background traffic(standard Openflow traffic and normal Ethernet traffic).

Link $e \in E$ has a capacity $c(e)$, contain both port-based traffic and background traffic. The background traffic $b(e)$ on each link e is readily retrieved or estimated by the controller. Between any host pair may exist many micro-flow, those flow all have same source address and destination address.

Let γ denote one flow in the network, the flow set Γ denotes all the flow within network. We assume the traffic demand of flow $\gamma \in \Gamma$ is denoted by $f(\gamma)$, and use P_γ to represent a feasible path set from source to destination for each flow$\gamma \in \Gamma$.

Variable $y_\gamma^p \in \{0,1\}$ denotes whether flow $\gamma \in \Gamma$ is using path $p \in P_\gamma$. Let $I(\gamma, v)$ be a binary value for this problem. If switch v have one loading entry for flow γ, $I(\gamma, v) = 1$, otherwise $I(\gamma, v) = 0$. Let T^v denote as the number of loading rules that one switch can store. We should know that T^v is much smaller than switch flow table capacity.

Finally, we try to formalize the segment routing traffic engineering (SR-TE) problem into a non-linear program as (1).

$$\min \lambda$$

$$s.t.\begin{cases} \sum_{p\in P_\gamma} y_\gamma^p \le 1, & \forall \gamma \in \Gamma \\ \sum_{\gamma\in\Gamma}\sum_{p\in e: p\in P_\gamma} I(\gamma,v)\bullet y_\gamma^p \le T^v, & \forall v \in V \\ \sum_{\gamma\in\Gamma}\sum_{e\in p: p\in P_\gamma} y_\gamma^p f(\gamma) + b(e) \le \lambda c(e), & \forall e \in E \\ y_\gamma^p \in \{0,1\}, & \forall p \in P_\gamma, \gamma \in \Gamma \end{cases} \quad (1)$$

B. Algorithm Description

It can be proofed that (1) is an NP-hard problem. So it is hard to solve this problem optimally. We first present one approximation algorithm for (1), called RRSD. The algorithm are shown in Table I. On the purpose of solving the problem formalized in (1), the algorithm constructs a linear program as a relaxation of the SR-TE problem. Back to SR-TE problem, each flow is assumed that only has one routing option. By relaxing this assumption, traffic of each flow is allowed to be divided through a path set P_γ. The relaxed SR-TE is expressed by (2).

$$\min \lambda$$

$$s.t.\begin{cases} \sum_{p\in P_\gamma} y_\gamma^p \le 1, & \forall \gamma \in \Gamma \\ \sum_{\gamma\in\Gamma}\sum_{p\in e: p\in P_\gamma} I(\gamma,v)\bullet y_\gamma^p \le T^v, & \forall v \in V \\ \sum_{\gamma\in\Gamma}\sum_{e\in p: p\in P_\gamma} y_\gamma^p f(\gamma) + b(e) \le \lambda c(e), & \forall e \in E \\ y_\gamma^p \ge 0, & \forall p \in P_\gamma, \gamma \in \Gamma \end{cases} \quad (2)$$

TABLE I. RRSD ALGORITHM

Algorithm 1 RRSD: Rounding-Based Route Selection and Deployment
1: **Step 1: Solving the Relaxed SR-TE Problem**
2: Obtain Traffic demand $f(\gamma)$ for $\gamma \in \Gamma$
3: Obtain link capacity $c(e)$ for $e \in E$
4: T^v is the number of remaining flow entries on v
5: Calculate the feasible path set P_γ for $\gamma \in \Gamma$
6: Construct a linear program in (2)
7: Obtain the optional solution y
8: **Step 2: Route Selection for Load Balancing**
9: Derive an integer solution y_γ^p by randomized rounding
10: **for** each flow $\gamma \in \Gamma$ **do**
11: **for** each routing path $p \in P\gamma$ **do**
12: **if** $y_\gamma^p = 1$ **then**
13: Appoint a path p for flow γ

With the algorithm, we first set system parameter, calculate the path set for each flow. To ease the algorithm, we only consider the feasible path with minimum hop count, because there may have exponential number of feasible paths for each flow. Then since (2) is a linear program, we can solve it in polynomial time with a linear program solver.

V. PERFORMANCE EVALUATION

In this section, we evaluate our proposed mechanism through a network simulator.

A. Simulation Settings

As running examples, we select two practical topologies with different network sizes. The first topology is for campus networks, denoted by (a), containing 100 switches, 200 servers and 398 links from [12]. The second one is the famous fat-tree topology [13], denoted by (b). Our fat-tree topology containing 16 core switches, 32 aggregation switches, 32 edge switches and 128 servers. For both topologies, each link has a uniform capacity, 10Gbps.

To evaluate how well our scalable routing algorithm performs, we compare with three other reference methods. One is the Open Shortest Path First (OSPF). Another is modified version of OSPF. Each flow will calculate three routing path, one of which is shortest path. And randomly choose one path, denote as OSPF_Random. The last one is the Equal-Cost Multi-path Routing (ECMP) method. Each flow will be routed into one of the three paths.

B. Simulation Evaluation

Flow Table Requirement Comparison: First set of simulations compared two solutions in terms of the consumption of different table entries under network (a) and (b). The results are shown in Fig. 4 and Fig. 5, where the horizontal axis represents the number of flows. With the increasing number of flows in network, there are more and more elephant flows. As result, the maximum/average numbers of forwarding rules (or required flow entries) are increased for both OSPF and HSR. However, our proposed HSR mechanism needs much less flow entries on switches than OSPF. HSR can reduce the required flow entries by about 60% compared with OSPF. That is because our hybrid segment routing only need one rule for each segment of routing path.

Route Performance Comparison: The next set of simulations is to evaluate the traffic engineering performance of four methods (OSPF, OSPF_Random, ECMP and HSR). First, from Fig. 6, it is clear that OSPF has the worst performance.

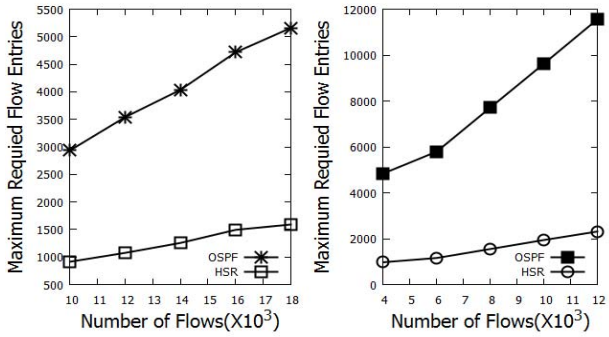

Figure 4. Maximum number of flow entries on one switch. Left plot: Topology (a); Right plot: Topology (b)

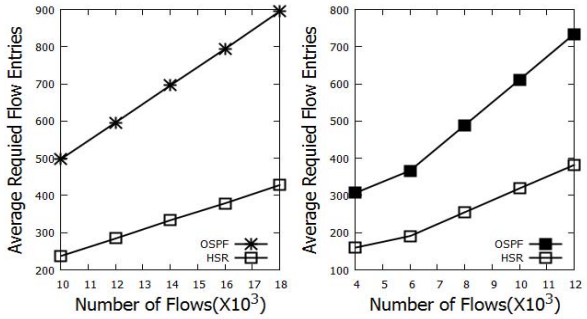

Figure 5. Average number of flow entries on switch. Left plot: Topology (a); Right plot: Topology (b)

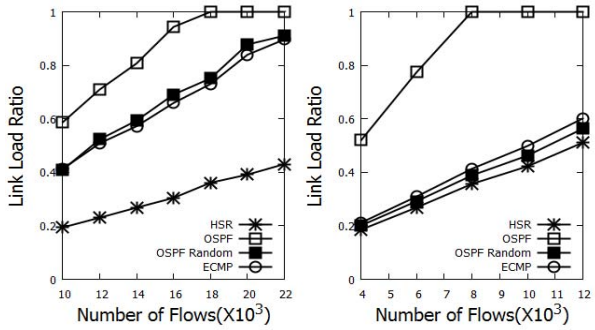

Figure 6. Link Load Ratio. Left plot: Topology (a); Right plot: Topology (b)

That is because when all flows using the shortest path, few switches will be heavily loaded, this situation can severely degenerate network performance. Second, when given three routing options, randomly choosing one is much better than default shortest path first algorithm, and ECMP is slightly better than OSPF_Random, about 15%. Third, our proposed HSR is much better than other three methods. In other words, our method can increase the network performance (link load ratio) by about 60% with the OSPF method using the same number of flow entries for both network (a) and (b).

But we can also notice that our method in topology (b) has little improvement. That is because in fat-tree topology, several top level core switches have to deal with all the traffic. Since all switches have same capacity, when top level switches come to its process limitation, the link load is higher than lower level switches.

VI. CONLUSIONS

This paper proposed a mechanism that integrate segment routing into hybrid SDN. Using unused field within packet header to store forwarding information, switch can classify packet through different Layers (NESL, OFSL). Controller decides whether each flow adopt segment routing or other process. For the traffic engineering part, this paper formulates one routing algorithm to obtain perfect network performance.

The performance results indicate that HSR can highly reduce the number of switch entries, and traffic engineering targeting some network could achieve better performance than several traditional algorithm. But under specific kind of network topology, this method still needs improvement.

ACKNOWLEDGMENT

This paper is supported by the National Science Foundation of China under No. 61472383 and 61472385.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[2] K. Kannan and S. Banerjee, Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 439–444. [Online]. Available: http: //dx.doi.org/10.1007/978-3-642-35668-1_32

[3] M.-C. Lee and J.-P. Sheu, “An efficient routing algorithm based on segment routing in software-defined networking,” Computer Networks, vol. 103, pp. 44 – 55, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128616300871

[4] S. Vissicchio, L. Vanbever, and O. Bonaventure, “Opportunities and research challenges of hybrid software defined networks,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 70–75, 2014.

[5] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, “The segment routing architecture,” in 2015 IEEE Global Communications Conference (GLOBECOM). IEEE, 2015, pp. 1–6.

[6] J. He and W. Song, “Achieving near-optimal traffic engineering in hybrid software defined networks,” in IFIP Networking Conference (IFIP Networking), 2015. IEEE, 2015, pp. 1–9.

[7] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, “Shadow macs: Scalable label-switching for commodity ethernet,” in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN ’14. New York, NY, USA: ACM, 2014, pp. 157–162. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620758

[8] J. Zhang, K. Xi, M. Luo, and H. J. Chao, “Load balancing for multiple traffic matrices using sdn hybrid routing,” in 2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR), July 2014, pp. 44–49.

[9] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, “Secondnet: a data center network virtualization architecture with bandwidth guarantees,” in Proceedings of the 6th International COnference. ACM, 2010, p. 15.

[10] Z. Guo, Y. Xu, M. Cello, J. Zhang, Z. Wang, M. Liu, and H. J. Chao, “Jumpflow: Reducing flow table usage in software-defined networks,” Computer Networks, vol. 92, pp. 300–315, 2015.

[11] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The nature of data center traffic: measurements & analysis,” in Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference. ACM, 2009, pp. 202–208.

[12] “The network topology from the monash university,” http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies.

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” SIGCOMM Comput. Commun. Rev., vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967