Software-Defined Label Switching: Scalable Per-flow Control in SDN

Nanyang Huang*, Qing Li†, Dong Lin‡, Xiaowen Li†, Gengbiao Shen†, Yong Jiang*†,
*Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, Shenzhen, China
†Graduate School at Shenzhen, Tsinghua University, Shenzhen, China
‡Huawei Technologies Co., Ltd

***Abstract*—Deploying Software-Defined Networks (SDNs) faces various challenges, and one of them is to implement per-flow control while preserving data plane scalability. Due to the limited rule storage space of commodity SDN switches, achieving flexible control and having a low-latency data plane with a low storage cost are often at odds. Unfortunately, existing SDN architectures fail to implement per-flow control efficiently: they either incur extra delays to packets or pose high storage burden to switches. In this paper, we propose Software-Defined Label Switching (SDLS) to achieve both data plane scalability and per-flow control. SDLS combines central control with label switching to reduce storage burden while maintaining per-flow control. SDLS introduces software switches into the data plane and manages the network in regions for scalability. SDLS is OpenFlow-compatible and employs a hybrid data plane to provide efficient flow setups. We evaluate SDLS by comparing with the state-of-the-art SDN architectures and show that SDLS can rival the best on the latency performance while reducing the number of flow entries and overflows by more than 47%.**

I. Introduction

Compared with traditional networks which perform destination-based aggregate routing, Software-Defined Networks (SDNs) enable network administrators to centralize network control logic and perform fine-grained, per-flow traffic management. As forwarding elements, switches do not know network states or policies. They rely on flow entries distributed by a central controller to match and forward incoming packets, and the controller manages the network by collecting statistics from the switches.

However, implementing flexible per-flow control and having a low-latency data plane with a low storage cost are two potentially conflicting goals. This dilemma stems from the limited storage space of commodity SDN switches. Modern SDN switches are typically hardware switches employing TCAM to store rules in flow tables. TCAM-based flow tables could achieve parallel matching with multiple fields to provide fast, flow-based packet forwarding. However, TCAM is small, expensive and power-hungry compared with SRAM, leading to a limited storage space of only thousands of rules in switches [1]. Even the high-end Broadcom Trident2 chipset only supports 16k forwarding rules [2]. Therefore, pre-deploying all flow entries is infeasible because the flow table size is much smaller compared with the number of flows in the network.

To solve the problem of limited storage space, a conventional flow management architecture like OpenFlow [3] chooses to let the switch redirect the first packet of every unknown flow to the controller, which reactively installs matching flow entries in the switch flow table on demand. However, this flow setup mode poses a high latency to the first packet, and the delay is an order of magnitude higher than the matching and forwarding time of switch ASIC [4], [5]. As a result, this long flow setup time becomes a performance bottleneck for latency-sensitive applications (e.g., instant messaging) and short-lived flows, which directly impacts network QoS in data centers [6]. For large flows, although the average packet delay might be low, the slow flow setup process increases response time for users and degrades quality of experience (QoE). Taking video streaming as an example, the join time of a video has a pronounced impact on the total play time at the viewer level, and users will quickly abandon video sessions if QoE is poor [7]. Moreover, packets of a large flow may experience several times of flow setups due to flow table updates, leading to data plane performance oscillation. Users may suffer from frequent buffering events when watching videos, which will significantly reduce their future engagements.

Therefore, one fundamental challenge when deploying SDN is to *implement per-flow control while preserving data plane scalability*. Many previous works try to resolve this dilemma by redesigning an SDN architecture and introducing new elements into the data plane, but none of them can achieve both per-flow control and a low-latency, low storage cost data plane. Some augment the ability of switches to support some control functions, such that the switch could process flow setup requests locally and keep packets in the data plane [1], [4], [5]. Nevertheless, these schemes require switch data plane or control plane modifications [4], [5], or fail to address the limited TCAM space problem [5], or are unable to provide per-flow control [1], [4]. Some proposals decouple the data plane into *edge and core*, where the edge employs general software for traffic control and the core performs simple packet forwarding [8], [9], [10], [11]. Edge-core design introduces software switches to provide flexible traffic control and large flow tables, but it poses extra delays to packets which leads to long flow completion time (FCT) for large flows.

In this paper, we propose Software-Defined Label Switching (SDLS), a flexible and scalable SDN architecture. The design of SDLS aims at both reserving the benefits of per-flow control of SDN, and achieving a low-latency data plane with a low storage cost. On one hand, we use the controller to flexibly

Corresponding author: Qing Li (li.qing@sz.tsinghua.edu.cn).

978-1-5386-2542-2/18/$31.00

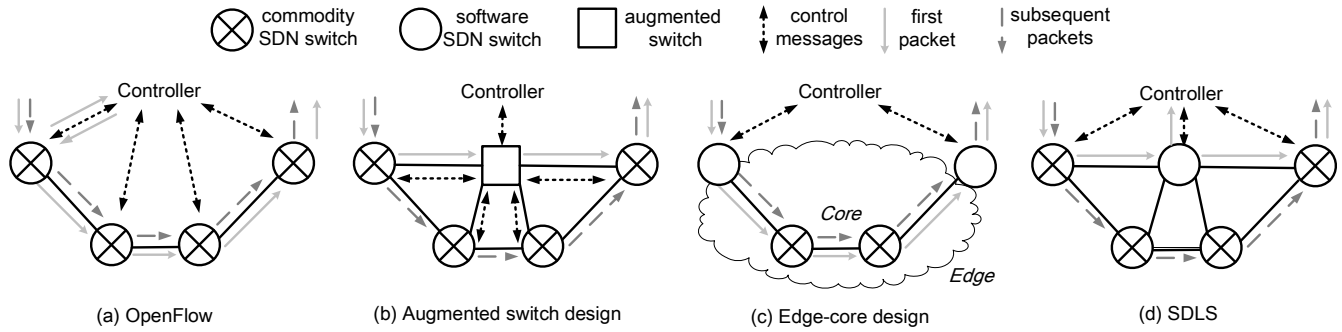

Fig. 1. SDN architectures: (a) OpenFlow. (b) Augmented switch design (DIFANE). (c) Edge-core design (Fabric). (d) SDLS.

deploy global policies to manage traffic, and keep the data plane simple and programmable. On the other hand, we adopt label switching technique to implement packet matching and forwarding, which significantly reduces the number of rules in the switch flow table.

The design of SDLS consists of three key elements: *label switching*, *regional management*, and a *hybrid data plane*. 1) We use label switching to reduce the storage load while maintaining data plane programmability and per-flow control. Combining OpenFlow with label switching allows us to decouple the process of matching and forwarding. 2) We perform regional management to scale the data plane, where the data plane is divided into regions and each region contains a software switch and makes forwarding decisions independently. 3) To provide efficient flow setups, we employ a hybrid data plane and propose a parallel packet processing technique to keep packets in the data plane. A basic SDLS implementation requires no modification to the OpenFlow specification, switch data plane or control plane design.

In summary, the main contributions of this paper are:

- We propose an OpenFlow-compatible SDN architecture SDLS to achieve a low-latency, low storage cost data plane with per-flow control. SDLS combines central control with label switching, manage the data plane in regions and is efficient in handling flow setup requests.
- Simulation compares SDLS with state-of-the-art SDN architectures, and the results show that SDLS rivals the best on data plane latency with a lower storage cost. For example, SDLS could deliver packets with a low delay and reduce the number of flow entries and overflows by more than 47%.

The rest of the paper is organized as follows. Section II discusses the strengths and weaknesses of current SDN designs and reveal why they fail to achieve a high performance data plane. Section III describes the design, architecture and related algorithms of SDLS. Section IV presents the evaluation methods and results, Section V discusses some optimization methods and related works, and Section VI concludes the paper.

II. Exploring SDN architectures

Centralizing control brings both benefits and costs. Although the control plane has a global vision and the data plane becomes more programmable, it is difficult for SDN switches to seek a balance between generality and efficiency. In this section, we introduce how conventional SDN architectures (e.g., OpenFlow) suffer from high flow setup delay, and two proposals which try to resolve this dilemma but fail to implement per-flow control efficiently.

A. OpenFlow

An OpenFlow network removes all control functions from switches; thus every forwarding decision is made by the controller. In Fig. 1(a), an SDN switch receives the first packet of a new flow and forwards the packet to the controller. The controller matches the packet, makes forwarding decisions and installs corresponding entries in the switches. Subsequent packets hit the entry, and the switch forwards them at line rate.

This flow setup process poses a high latency to the first packet and leads to many performance problems. Commodity SDN switches have wimpy CPU and limited control bandwidth. Consequently, the process of encapsulating/decapsulating packets and communicating with the controller pose high latency to the first packet. The round-trip time between switch CPU and the controller is nearly four-order-of-magnitude higher than forwarding delay of switch ASIC [4]. For short and latency-sensitive flows, the high delay of the first packet impacts the performance of throughput and FCT, degrading QoS. For large flows, the response time becomes higher and packets may experience several times of flow setups due to flow table updates. The controller is constantly involved in flow setups and rule updates, resulting in data plane performance oscillation.

Although a simple and programmable data plane is future proof for it accelerates network upgrading and deployment, we conclude that the control plane should put some knowledge (control policies) and logic (forwarding decisions) back to the switches for the efficiency of the data plane.

B. Augmented switch design

Many proposals augment SDN switches to support larger storage space [1], dual stack [2], more control functions [4], etc., and some of them require modifications to either switch data or control plane, which may bring difficulties for building or upgrading the network. We choose DIFANE [5] as a representative because a DIFANE network can enable per-flow control, has good latency performance and only needs control plane modifications for a fraction of switches.

In Fig. 1(b), the controller pre-installs flow entries in augmented switches (they are called authority switches in DIFANE), and requires commodity switches to forward the first unknown packet to them. The controller installs wildcard rules in commodity switches to inform which augmented switch they should send unknown packets to. The augmented switch receives and forwards the packet to the egress switch, and installs the matching flow entry into the commodity switches. By doing so, all packets are processed in the data plane. Commodity switches in the fast path forward subsequent packets to avoid the augmented switch to become a performance bottleneck.

Ideally, in a DIFANE network, only the first packet experiences path stretch and additional latency. Nevertheless, this design is not efficient since the controller still requires every switch to store a considerable number of exact matching rules to support per-flow control; thus the switch flow table may need to be constantly updated. If the flow table becomes full, the controller needs to remove some existing rules before inserting new entries, and we refer to this situation as *overflow*. Overflow should be avoided because updating TCAM is a slow process and results in a large number of control messages between the controller and the switch [1], [12]. The speed of network updates determines the agility of the control loop, and a slower update implies a longer period which congestion or packet loss occurs [12]. To make things worse, overflow also brings vulnerability to the switch and leads to security problems [13].

C. Edge-core design

The idea of the edge-core design is to move complex control functions to the network edge and leave the core a clean slate. It has been adopted in the design of SDN [8], datacenter network [14], cellular network [9], Internet exchange point (IXP) [10] and Content Delivery Network (CDN) [11].

We illustrate how an edge-core SDN works by taking Fabric [8] as an example (Fig. 1(c)). The controller pre-deploys *all* control policies in software access switches which run on commodity servers in the edge. Software switches have more powerful CPU and larger storage space compared to commodity hardware switches, but their packets forwarding latency is also higher [1], [15]. The core is composed of hardware switches which match and forward packets according to MPLS labels. Thus for every packet, an ingress software switch finds a matching entry, pushes a corresponding label, and forwards it to the core. The core forwards the packet by its label. Finally, the packet leaves the network from an egress software switch which pops the label.

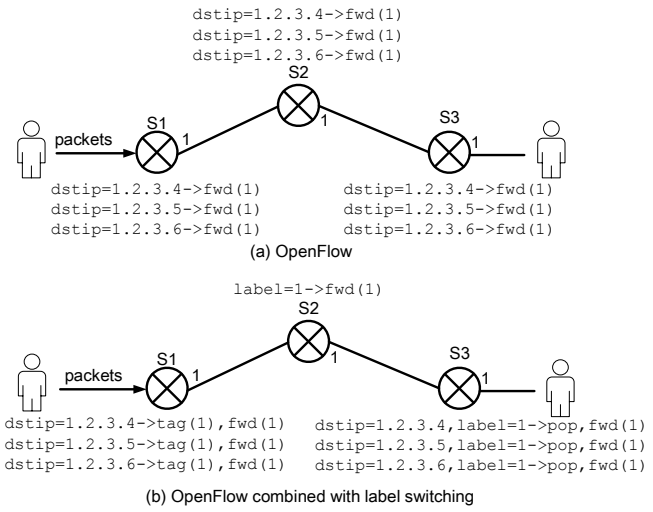

Fig. 2. (a) Rule deployment scheme of OpenFlow, where 9 OpenFlow entries are required. (b) Rule deployment scheme of OpenFlow combined with label switching, where 6 OpenFlow entries and 1 label rule are required.

In an edge-core network, switches in the core become simple packet forwarders, and the edge switches make all the routing and access control decisions. The edge-core design significantly reduces the number of flow entries by forwarding packets based on labels. Meanwhile, the controller is free from handling flow setup requests. However, it requires to replace all the access switches and limits the ability to deploy forwarding decisions within the core [16]. We conclude the edge-core design is not efficient enough since *every* packet has to traverse software switches which add extra delays. The design choice of splitting the data plane into two parts inevitably lets software switches become a bottleneck of latency performance and leads to long FCT for large flows.

III. Software-Defined Label Switching

As shown in the previous section, the state-of-the-art SDN architectures are far from satisfactory. They either add extra delays or require a large number of flow entries. This section presents the Software-Defined Label Switching SDN architecture, as shown in Fig. 1(d), where we combine central control with label switching to build a scalable SDN with per-flow control.

A. Label switching

Our first key insight in SDLS is using label switching to reduce the storage load while maintaining data plane programmability and per-flow control. As the number of match field grows from 12 to 41, OpenFlow is becoming increasingly complex. While OpenFlow is suitable for building a heterogeneous network, where the network supports multiple protocols and implement fine-grained per-flow control, we can still go a step further by combining OpenFlow with label

switching to decouple the process of matching and forwarding. Although a general packet processing design like P4 [17] may provide more programmability, it requires a thorough change to existing switch designs. In contrast, our design of SDLS is compatible with existing OpenFlow switch.

We believe that OpenFlow is more suitable for packet "classifying" and label switching is the right choice for packet forwarding. Leveraging label switching would provide a more general and simpler data plane, and gains the benefit of fewer flow entries. 1) The controller compiles high-level control policies into OpenFlow and label rules. Packets are aggregated and forwarded by labels, and we only need OpenFlow for switches to inspect the packet headers and push a corresponding label to indicate its destination. 2) Packets sharing the same path could be forwarded by one label, which significantly reduces the number of flow entries since we no longer need to deploy flow entries switch-by-switch. The controller only needs to install one rule to push the label in the ingress switch, and another to pop the label in the egress switch. We could still collect per-flow statistics from ingress or egress switches using OpenFlow primitives without losing the fine-grained control over the network.

To manage N flows in the network sharing the same path with a length of d, a conventional OpenFlow deployment scheme would require dN OpenFlow rules to match and forward packets. But if we combine OpenFlow with label switching, we would need $2N$ OpenFlow rules for pushing/popping labels and d rules for label switching. Fig. 2 provides an example. (We write rules in Pyretic [18] language.) Besides, this decoupling of matching and forwarding offers convenience for policy updating, which now would require only to modify the OpenFlow rules in the access switches and leave label rules unaffected.

In SDLS, to be compatible with OpenFlow, we still implement label switching using OpenFlow rules; thus we need to transform the original rules to a new set of OpenFlow entries as shown in Fig. 2. We use MPLS label as the match field to implement label switching, but we do not limit our architecture to this particular label switching technique. Note that our design significantly differs from MPLS networks and MPLS-assisted SDN designs. In traditional MPLS networks, switches announce and broadcast their labels to let others create forwarding entries. However, in SDLS we employ the controller to distribute, assign and collect labels, which provides us with strong consistency and globally optimal control to support QoS policies. Unlike existing MPLS-assisted SDN designs, SDLS does not divide the data plane into core and edge or builds fabric. Instead, we manage the data plane in regions, which allows every switch to perform label pushing/popping and switching, and eventually enables us to deploy routing policies everywhere instead of end-to-end.

B. Regional management

Our second key insight in SDLS is performing regional management to scale the data plane. Managing SDN data plane as a whole may not be the optimal solution: deploying

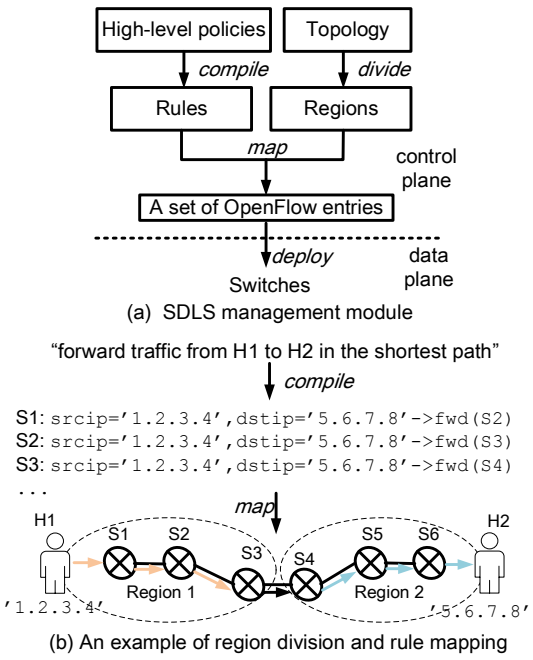

Fig. 3. (a) SDLS management module in the controller. (b) An example of region division and rule mapping.

flow entries switch-by-switch incurs a high storage cost while dividing data plane into edge and core results in poor scalability. Instead, we focus on another design choice: regional management. Our idea to manage SDN data plane in regions is inspired by the success of hierarchical SDN control plane [11], [19]. In a hierarchical control plane, local controllers are responsible for frequent and urgent local events such as flow setups or failure recoveries, and a global controller performs high-level policy distributing and traffic engineering. Decision-making processes of each local controller are independent; thus control load of the global controller is distributed, and network dynamics can be quickly processed rather than waiting for the global controller to react.

Similar to the hierarchical SDN control plane, we design SDLS to implement regional control for a low-latency, low storage cost data plane. Firstly, a local control unit handles flow setups to accelerate the processing of local events. Secondly, the controller distributes control policies in regions, which reduces the number of flow entries for each switch. Finally, a properly designed regional control scheme is scalable, since packets are processed independently in each region without interfering with the others.

To implement regional management, the controller runs an SDLS management module to perform high-level policy compilation, region division, rule mapping and deployment. As shown in Fig. 3(a), the SDLS module divides the data plane into some regions based on network topology and compiles policies into rules, which are then mapped to a set of OpenFlow entries and installed in the underlying switches.

The controller compiles high-level policies into low-level

rules with match field, priority, and related actions. High-level names are substituted with network addresses such as IP addresses and ports for packet matching as shown in Fig. 3(b). When receiving a flow setup request, the controller parses the packet header and finds a matching rule and a corresponding path of switches. We refer to this path as the *fast path*. A fast path could be the shortest path between hosts, or a special route to implement service chain or QoS guarantees. Flow-based management is the key to an OpenFlow QoS framework which matches against packet headers and detects flows requiring QoS guarantees [4]. Nevertheless, the total number of flow entries after compilation corresponding to network flows could be numerous. Even employing label switching, it is infeasible to store all rules in commodity OpenFlow switches. Techniques such as hashing are not promising for wildcard OpenFlow rules [5], and replacing all switches with software switches is less-efficient and results in delays.

Instead, we divide the data plane into regions based on network topology. To be specific, we select k locations to place software switches and get k regions. Region R_i contains a software switch s_i and a number of commodity hardware switches. The hardware switch x belongs to region $R(x)$ containing the nearest software switch:

$$R(x) = \text{argmin}_{R_i}\, d(x, s_i), 1 \leq i \leq k \tag{1}$$

$d(a, b)$ indicates the latency of fast path from switch a to b. These switches are responsible for handling flows entering, leaving or passing the region. In each region, the software switch processes local flow setup requests and keeps packets in the data plane. The controller also generates coarse-grained wildcard flow entries to represent each region, and install them in the software switches to let the switch forward unknown packets to the nearby region. In this way, we distribute the storage load so that no switch has to store all the flow entries. The number of regions is determined by the network administrator and could be flexible.

The placement of software switches determines how the data plane is divided and is similar to the placement of controllers [20]. The placement would affect the latency performance of the data plane since unknown packets in a region would traverse its software switch before they enter another region or reach the destination. In this paper, we minimize the sum of the latency caused by the flow setup process. For a region $R_i(V_i, E_i)(1 \leq i \leq k)$, where V_i is the set of switches, E_i is the set of links, and $N_i = |V_i|$, we place software switch at s_i such that for any hardware switch $x, x \in V_i$, the sum of the total latency:

$$\sum_{i=1}^{k} latency(R_i) = \sum_{i=1}^{k} \sum_{x \in V_i} d(x, s_i) \tag{2}$$

is minimized. We note that this is the k-median problem, and we adopt the k-median clustering algorithm to efficiently find the region division and software placements.

Unlike previous solutions which directly install compiled rules in the fast path, we map the original rules to a set of

Algorithm 1 Deploy flow entries in the fast path

Input: packet pkt.

```
Find matching entries, destination dst and corresponding fast path p for pkt
for segment s in p do
    Get ingress switch ingress and egress switch egress
    Get label l which corresponds to egress
    Install an entry in ingress, which pushes label l and forwards the packet to egress
    if s is the last segment then
        Install an entry in egress, which pops the label and forwards the packet to dst
    else
        Get ingress switch next of the next segment
        Install an entry in egress, which pops the label and forwards the packet to next
    end if
    if no path from ingress to egress then
        Install entries in this path, which forward packets by label l
    end if
end for
```

OpenFlow rules based on region division. The match field of the new rule contains MPLS label to perform label switching as we illustrate in Fig. 2. If the fast path traverses several regions, the controller cuts it into *segments* by region. For rules to be installed in the switches of different regions, the controller assigns MPLS labels independently. Therefore MPLS labels have meaning only within the region, and routing decisions in each region would not interfere with others. In Fig. 3(b), the fast path $[S1, S2, S3, S4, S5, S6]$ is cut into two segments: $[S1, S2, S3]$ in Region 1 and $[S4, S5, S6]$ in Region 2. Packets are forwarded based on different labels in two regions (represented by two colors in the figure). For each region, the controller allocates and collects labels by managing a ring buffer, so that outdated labels can be reused. We will discuss the details of how to deploy these new OpenFlow entries in Section III-C.

Benefited from regional management, we are also able to handle network dynamics gracefully. To deal with policy changes and rule updates, the controller may update the rule mapping from the compiled rules to the OpenFlow entries rather than perform per-switch configuration. To handle software switch failure and avoid a single point of failure, we could let its region fall back to the conventional OpenFlow network, where switches forward unknown packets to the controller, without affecting routing decisions in other regions.

C. Hybrid data plane

We introduce software switches into the data plane to provide local control for efficient flow setups. Software switches like Open vSwitch [21] run on commodity servers and can store larger flow tables and provide flexible packet matching. In SDLS, software switches are responsible for processing unknown packets in their regions to handle flow setup requests

locally, but they are not placed in the network edge like previous proposals.

To handle new flows in the network, the controller first changes default actions of switches and pre-installs wildcard flow entries in the software switches to inspect and forward unknown packets. The controller installs a rule with the lowest priority to change the default action such that the switch pushes a label and forwards the packet to the software switch in the region.

At first, the ingress switch has no knowledge of the incoming packets and forwards them to the software switch, which matches the packets with pre-installed entries to decide which switch it should forward the packet to, or whether it should forward the packet to another region. The software switch swaps the label of the packet and forwards it to the egress switch, which pops the label and sends the packet to the host or a switch in another region. During this phase, the packet traverses the software switch and experiences path stretch and delay, and we refer to this path as the *slow path*.

A key design challenge in SDLS is to avoid software switches to become a performance bottleneck since the packet processing rate of software switches is slower than hardware switches. Therefore, our goal is to guide the packets from the slow path to the fast path as soon as possible. We propose a parallel packet processing technique to handle flow setup requests in the data plane without modifying OpenFlow. When receiving a packet, despite forwarding it to the specified port of the matching entry, the software switch makes a copy and forwards it to the controller *simultaneously*. The controller receives this packet and installs corresponding flow entries to establish the fast path. By doing so the process of packet forwarding and parsing are separated and conducted in parallel. We implement this feature in OpenFlow by using group tables to forward packets to multiple output ports.

The controller deploys the rules in the fast path according to Algorithm 1. The controller installs matching flow entries by segments, and in each segment, the controller installs a label pushing rule and a label popping rule. The path from ingress to egress switch only needs to be established once, since packets traversing the same path could share the same label. After the deployment of flow entries, the ingress switch could forward the subsequent packets directly to the egress switch in the fast path.

Compared with the conventional rule deployment scheme, this flow setup time is shorter for two reasons. 1) The switch does not buffer the packet or sends the packet to the controller. In contrast, packets are forwarded in the slow path while the controller processes flow setup request at the same time. 2) The controller does not need to install flow entries switch-by-switch. Instead, it only installs rules in the ingress and egress switch of each segment.

IV. Evaluation

In this section, we present our simulated evaluation of SDLS comparing with the state-of-the-art SDN architectures. We evaluate their performances on the data plane to show how SDLS achieves a scalable SDN with per-flow control.

A. Simulation methodology

To evaluate the performance of various SDN architectures, we implement an event-based platform to simulate behaviors of the controller and switches. The simulator measures per-packet delay and captures the real-time flow entry number of every switch in the network. The controller communicates with the switches via OpenFlow protocol to realize flow entry deployments and updates.

We model the packet forwarding behaviors in several aspects to measure the data plane performances such as per-packet delay, flow table occupancy, etc. 1) Hardware and software switches employ different mechanisms to forward packets. Hardware switches use TCAM to match the packet header with all rules at the same time at line rate, but the software switches can process packets at only tens of Gbps [1]. Therefore, we model the hardware switch referring to HP ProCurve 5406zl [22], and model the software switch based on Open vSwitch, with a forwarding delay of 5 μs and 35μs respectively, and we note that this 7x difference is similar to observations made by other OpenFlow switch evaluation works [15], [23]. 2) The limited switch-controller channel bandwidth and weak switch CPU pose large delay for flow setups. The bandwidth available between the 5406zl and the controller is only 17Mbps and a round-trip time will delay a flow for at least 4 ms [4]. 3) Modern hardware commodity switches can support several thousands of wildcard or exact matching OpenFlow rules [1], [4], [15]; therefore we model the TCAM with a limited size of 3000 flow entries [15]. We do not limit the flow table size of software switches. Lastly, since we are interested in the data plane performance, we ignore the packet processing time of the controller.

We use two real-world ISP topologies `germany50` and `brain` for evaluation [24]. The `germany50` topology has 50 nodes and 88 links, and the `brain` topology has 161 nodes and 166 links. We model the links as 1Gbps and nodes as switches attached by some servers. For each topology, we have a corresponding traffic demand matrix, but we prefer to enable per-flow management. Therefore we map a real-world trace from the MAWI working group of the WIDE project [25] to our topologies. In this paper, we define a flow as a set of packets sharing the same 5-tuple. The source and destination IP address of every flow in the trace are mapped to the addresses of servers in the topology. Besides, we generate synthetic trace based on flow size distributions to reveal the influence of traffic property to the data plane performance.

During the simulation, we compare the performance of the following four SDN architectures:

- *OpenFlow*: In an OpenFlow network, all flow setups are handled by the controller, and there are only hardware switches in the data plane.
- *DIFANE*: DIFANE represents the augmented switch designs. There are augmented switches and hardware

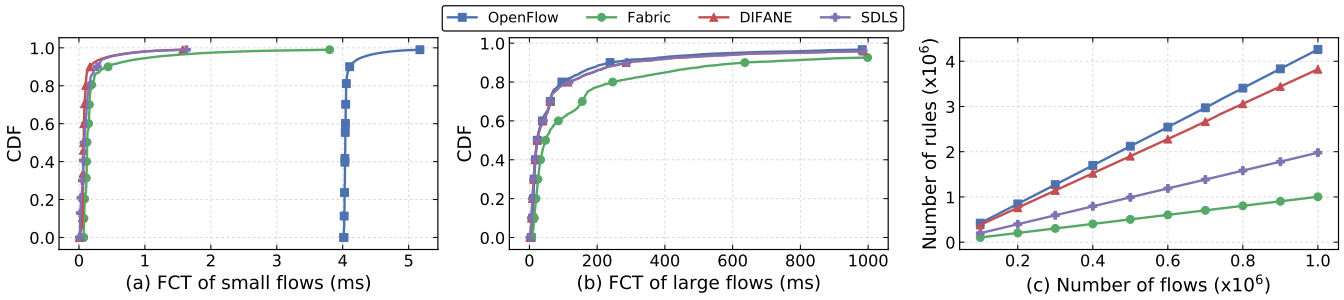

Fig. 4. Data plane performances of SDN architectures in `germany50` topology with unlimited size flow tables.

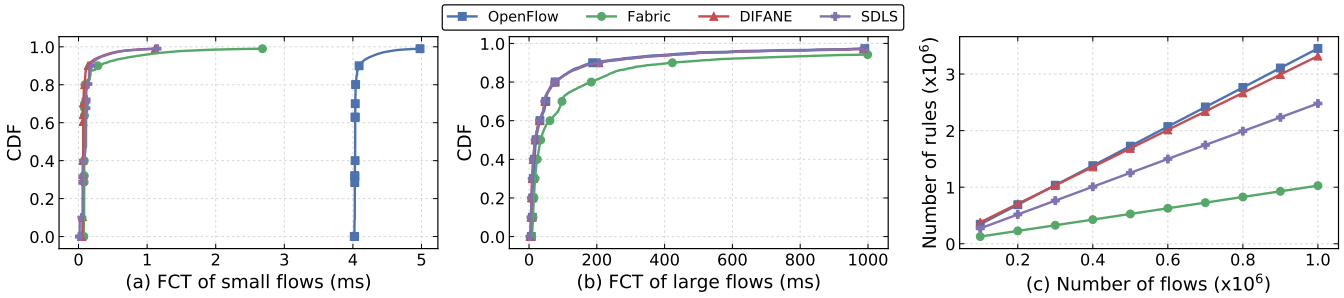

Fig. 5. Data plane performances of SDN architectures in `brain` topology with unlimited size flow tables.

switches in the data plane, and we implement the functions of an augmented switch by modifying the switch control plane, which allows it to communicate with other switches directly. We do not limit the flow table size of the augmented switch. According to the topology sizes, we set the number of augmented switch as 3 in `germany50` and 9 in `brain` [5].

- *Fabric*: Fabric represents the edge-core design. The data plane contains software switches in the network edge and hardware switches in the core. A Fabric network places all per-flow control policies in the software switches; therefore the number of software switches in Fabric is determined by the topology and is different from that of DIFANE and SDLS.
- *SDLS*: SDLS contains both hardware and software switches, and makes no modifications to either switch data or control plane. For comparison, we set the same number of software switches as DIFANE.

For each architecture, we require the controller to implement per-flow control by installing exact matching rules to forward packets in their fast path (shortest path from the source host to the destination). There are three key metrics in our evaluation. 1) FCT measures how the flow latency is affected by different forwarding behaviors of SDN architectures. 2) Average packet delay measures the overall latency of the data plane. 3) Flow table usage indicates the TCAM resource required for each design to implement per-flow control.

We perform simulations under two scenarios. In the first scenario, all the switches have unlimited size flow tables, and we measure the total number of flow entries in the data plane to see how many TCAM entries each design needs. In the second scenario, the hardware switch flow table size is limited to 3000, and we measure the number of overflows to show the storage efficiency.

B. Performance

1) Unlimited flow table size: Fig. 4 and 5 show the 99^{th} percentile FCT CDFs and flow entry number of four architectures in two topologies. We classify the flows as small and large according to their sizes with a threshold of 100KB [26]. In both topologies, the OpenFlow network has the highest FCT for small flows due to its slow flow setup process. However, OpenFlow performs well for large flows with unlimited size flow tables since most packets are forwarded in their fast path by hardware switches. Both SDLS and DIFANE achieve a shorter FCT compared with Fabric, which poses extra delays to packets. The FCT of Fabric also exhibits a heavy-tailed feature because every packet has to be processed by software switches. Thus, for large flows the software switches become a performance bottleneck.

As the number of flow increases, the number of rules in all designs grows linearly. OpenFlow has the largest number of flow entries: the number is over 4.2×10^6 in `germany50` and 3.4×10^6 in `brain`. DIFANE also poses a high storage burden to the switches, where the numbers are more than 3.8×10^6 and 3.3×10^6 in two topologies. In contrast, the numbers of rules in SDLS are much smaller than that of OpenFlow and DIFANE, which are lower than 2×10^6 in `germany50` and 2.5×10^6 in `brain`, indicating a 47% and

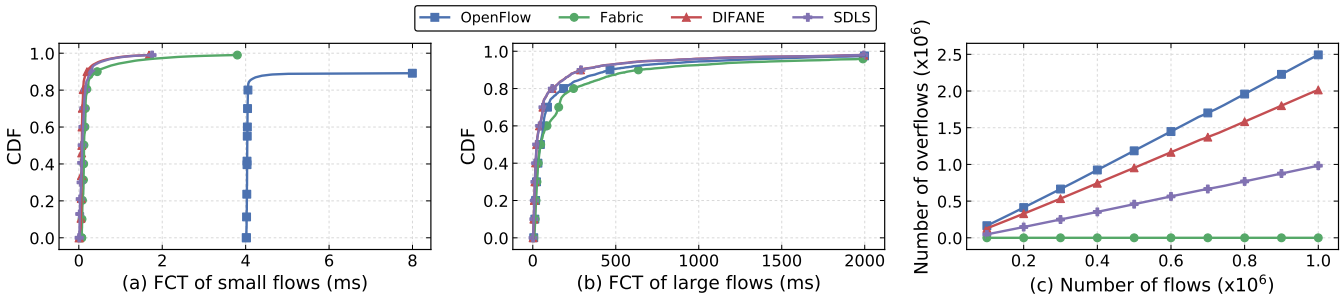

Fig. 6. Data plane performances of SDN architectures in `germany50` topology with limited size flow tables.

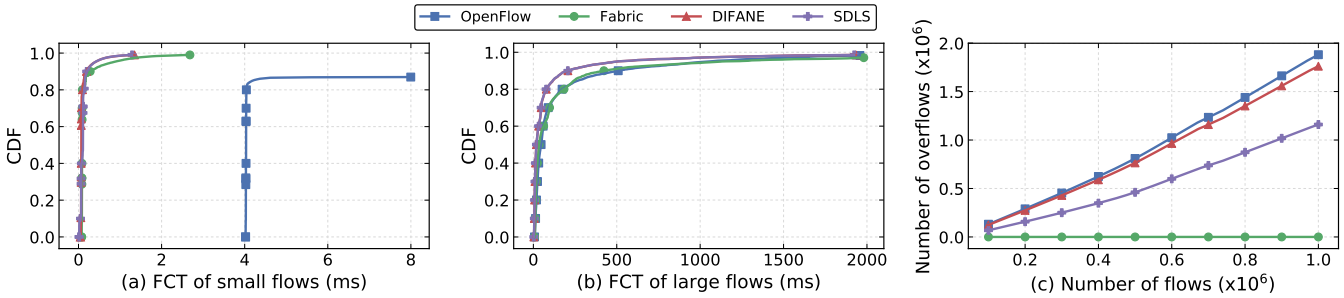

Fig. 7. Data plane performances of SDN architectures in `brain` topology with limited size flow tables.

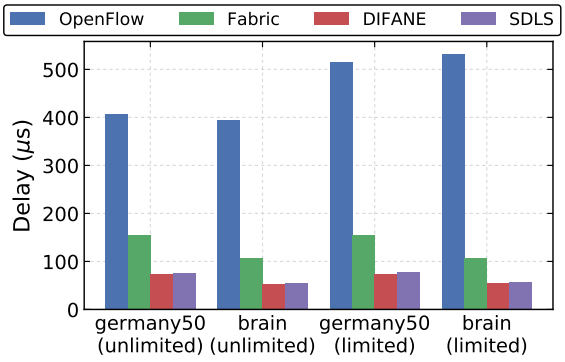

Fig. 8. Average packet delay in `germany50` and `brain` topology with unlimited and limited size flow tables.

a 50% reduction. The reason is that SDLS aggregates network flows by labels while OpenFlow and DIFANE have to deploy exact matching entries switch-by-switch. Although Fabric has the lowest number of rules, we note that it has a high FCT and is not able to implement forwarding decisions within the network core.

Fig. 8 shows the average packet delay. In SDLS and DIFANE, subsequent packets are forwarded by hardware switches in the fast path; therefore they have the shortest average packet delay. The packet delay of OpenFlow is affected by its flow setup process and is the highest among all (at least 5 times higher than SDLS and DIFANE). The delay of Fabric is 2 times higher than SDLS and DIFANE due to its software switches.

2) Limited flow table size: If the switches have limited size flow tables, the controller has to remove flow entries actively if the flow table becomes full. Therefore a flow may experience several times of flow setups. As shown in Fig. 6 and 7, the limited size increases the FCT, especially for large flows. We find OpenFlow delivers poor FCT performances for both small and large flows because of its constant rule updates and flow setups. OpenFlow also has the largest number of overflows. SDLS and DIFANE have the shortest FCT, but the number of overflows of SDLS is much smaller than DIFANE. For example, in `germany50` SDLS has fewer than 1×10^6 overflows whereas DIFANE has more than 2×10^6. The number of overflows rises as the number of flow increases except for Fabric, which has no overflow since the software switches have a large enough storage space to handle all the rules.

Similarly, the average packet delays of four designs increase, compared with that of unlimited size flow tables. OpenFlow experiences the largest increase of 26.8% due to its long flow setup process, while others have a moderate rise of approximately 1%.

3) Effect of traffic property: We generate synthetic trace to reveal the effect of traffic property. To be specific, we generate two traffic files by tuning parameters in the flow size distributions [27]. One trace contains mostly burst small flows with a small flow size of less than 100KB, and another contains large, long-duration flows (>1MB). Fig. 9 shows the 99^{th} percentile FCT CDFs. We use `germany50` topology and set the flow table size as unlimited.

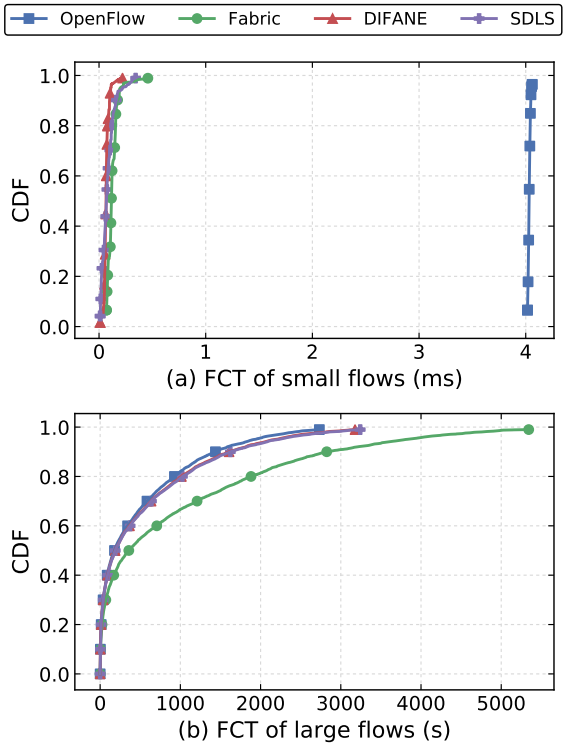

Fig. 9. FCT of SDN architectures for synthetic workloads

For small flows, DIFANE performs the best and SDLS has a slightly larger FCT because it involves the controller in the process of rule deployment. OpenFlow suffers from slow flow setups for small flows, but for large flows OpenFlow has the shortest FCT. The reason is that OpenFlow data plane contains only hardware switches, but for SDLS, DIFANE and Fabric, the fast path of a flow may traverse software switches, which leads to extra delays. Finally, Fabric has a much higher FCT for large flows, compared with other architectures.

V. Discussion

1) Promote efficiency by local control plane: SDLS provides an OpenFlow-compatible solution to build a scalable SDN with per-flow control. However, the long latency between the controller and switches still limits the performance of the data plane. It will further promote the efficiency of SDLS if we can add a local control plane to the software switches to deploy flow entries. Modifying software switch control plane would be easier and more flexible. Once receiving unknown packets, the software switch matches the packets with pre-installed wildcard rules and the local control plane generates the exact matching flow entries. Software switches could install these rules directly in the hardware switches in the fast path. Consequently, the flow setup process becomes faster since the controller no longer needs to take part in handling unknown packets. Actually, the controller could allocate its resource to perform other control missions like global TE. Similarly, for edge-core designs like Fabric, enabling core switches to perform fast local actions like load balancing could significantly improve the latency performance [14].

2) Hybrid networking: Hybrid networking relies on combining central control of SDN and traditional routing protocols to forward packets. A hybrid network consists of a central controller and some or no SDN switches. Panopticon [16] implements a partial SDN with both SDN and legacy L2 switches in the data plane. The controller uses SDN-controlled ports to enforce forwarding policies. However, this approach loses the ability to inspect and control every flow in the network. Another approach uses a controller to inject and override the routing decisions specified by OSPF [28] or BGP [29]. In this way, we preserve the robustness of distributed routing protocols and achieve flexibility by central control. For MPLS networks, with the help of central control and a global vision, the control plane of MPLS-TE or MPLS VPNs will be much easier to implement [30]. In this paper, we combine OpenFlow with label switching to realize both flexible control of SDN and efficient packet forwarding of MPLS, and we expect to extend SDLS to legacy MPLS networks or hybrid SDN networks in our future works.

VI. Conclusion

SDN provides a global vision for administrators to manage the network and implement per-flow control. However, because of the limited flow table size, the existing SDN architectures face various performance problems such as high packet delay, poor scalability or high storage burden. In this paper, we proposed SDLS, an OpenFlow-compatible SDN architecture to achieve both scalability and per-flow control. SDLS combines OpenFlow with label switching which allows us to reduce the number of flow entries in the network and decouple packet forwarding from specific routing protocols. We manage the network in regions and employ a hybrid data plane to provide more timely reactions to local events such as flow setups. Our evaluation results showed that SDLS achieves a low-latency data plane with a low storage cost, and beyond that, SDLS could be adopted to implement a more flexible and efficient traffic management framework.

Acknowledgment

The research is supported by the National Natural Science Foundation of China under grant No. 61625203, the National Key R&D Program of China under grant No. 2017YFB0803202 and 2016YFC0901605, the R&D Program of Shenzhen under grant No. JCYJ20170307153157440.

References

[1] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks," in *Proc. ACM SOSR*, Santa Clara, CA, USA, 2016.

[2] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable Software-Defined Networking through Hybrid Switching," in *Proc. IEEE INFOCOM*, Atlanta, GA, USA, 2017.

[3] 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.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Proc. ACM SIGCOMM*, Toronto, Ontario, Canada, 2011.

[5] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-based Networking with DIFANE," in *Proc. ACM SIGCOMM*, New Delhi, India, 2010.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, New Delhi, India, 2010.

[7] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," in *Proc. ACM SIGCOMM*, Toronto, Ontario, Canada, 2011.

[8] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A Retrospective on Evolving SDN," in *Proc. ACM HotSDN*, Helsinki, Finland, 2012.

[9] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and Flexible Cellular Core Network Architecture," in *Proc. ACM CoNEXT*, Santa Barbara, California, USA, 2013.

[10] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "Sdx: A software defined internet exchange," in *Proc. ACM SIGCOMM*, Chicago, Illinois, USA, 2014.

[11] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain *et al.*, "Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, 2017.

[12] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic Scheduling of Network Updates," in *Proc. ACM SIGCOMM*, Chicago, Illinois, USA, 2014.

[13] R. Kloti, V. Kotronis, and P. Smith, "OpenFlow: A Security Analysis," in *Proc. IEEE ICNP*, Goettingen, Germany, 2013.

[14] S. Ghorbani, Z. Yang, P. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro Load Balancing for Low-latency Data Center Networks," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, 2017.

[15] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation," in *Proc. Springer PAM*, Heidelberg, Berlin, Germany, 2012.

[16] D. Levin, M. Canini, S. Schmid, F. Schaffert, A. Feldmann *et al.*, "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks," in *Proc. USENIX ATC*, Philadelphia, PA, USA, 2014.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[18] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker *et al.*, "Composing Software-Defined Networks," in *Proc. USENIX NSDI*, Lombard, Illinois, USA, 2013.

[19] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. ACM HotSDN*, Helsinki, Finland, 2012.

[20] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," in *Proc. ACM HotSDN*, Helsinki, Finland, 2012.

[21] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open vSwitch," in *Proc. USENIX NSDI*, Oakland, CA, USA, 2015.

[22] HP ProCurve 5400 zl switch series, http://www.hp.com/hpinfo/newsroom/press_kits/2010/HPOptimizesAppDelivery/E5400zl_Switch_Series_Data_Sheet.pdf.

[23] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proc. IEEE ITC*, San Francisco, CA, USA, 2011.

[24] SNDlib, http://sndlib.zib.de/home.action.

[25] MAWI Working Group Traffic Archive, http://mawi.wide.ad.jp/mawi/.

[26] I. Cho, K. Jang, and D. Han, "Credit-Scheduled Delay-Bounded Congestion Control for Datacenters," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, 2017.

[27] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's Law for Traffic Offloading," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 16–22, 2012.

[28] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central Control Over Distributed Routing," in *Proc. ACM SIGCOMM*, London, United Kingdom, 2015.

[29] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering Egress with Edge Fabric: Steering Oceans of Content to the World," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, 2017.

[30] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNs with OpenFlow," in *Proc. ACM SIGCOMM*, Toronto, Ontario, Canada, 2011.