Hindawi
International Journal of Reconfigurable Computing
Volume 2019, Article ID 7363901, 12 pages
https://doi.org/10.1155/2019/7363901

Research Article

Exposing End-to-End Delay in Software-Defined Networking

Ting Zhang and Bin Liu

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Correspondence should be addressed to Bin Liu; liub@mail.tsinghua.edu.cn

Received 5 September 2018; Accepted 18 February 2019; Published 4 March 2019

Academic Editor: Jim Torresen

Software-Defined Networking (SDN) shows us a promising picture to deploy the demanding services in a fast and cost-effective way. Till now, most SDN use cases are deployed in enterprise/campus networks and data center networks. However, when applying SDN to the large-scale networks, such as Wide Area Network (WAN), the end-to-end delay of packet traversal is suspected to be very large and needs to be further investigated. Moreover, stringent time constraint is the cornerstone for real-time applications in SDN. Understanding the packet delay in SDN-based large networks is crucial for the proper design of switch architecture and the optimization of network algorithms such as flow control algorithms. In this paper, we present a thorough systematic exploration on the end-to-end delay in SDN which consists of multiple nodes, fully exposing the components which contribute to the long delay. We disclose that SDN switches cannot completely avoid the generation of flow setup even in proactive mode and conduct data mining on the probability of flow setup. We propose an analytical model for the end-to-end delay. This model takes into account the impact of the different rule installation time consumption on different switches. Considering the delay in switches contributes a large proportion to the entire delay, we conduct various measurements on the delay of a single switch. Results for the delay at different flow setup rates and with different rule priority patterns are presented. Furthermore, we study the impact on packet delay caused by ternary content addressable memory (TCAM) update. We measure parameters in the delay model and find that if SDN is deployed in all segments of WAN, the delay of packet traversal will be increased up to 27.95 times in the worst case in our experimental settings, compared with the delay in conventional network. Such high delay may eventually lead the end-to-end connections fail to complete if no additional measures are taken.

1. Introduction

SDN research has attracted wide attention in both academia and industry. It proposes to decouple control plane and data plane in routers/switches with the initial purpose of reducing the infrastructure cost and improving the traffic engineering/management performance [1]. SDN enables high level abstraction for the network, targeting at fast service creation/insertion via open Application Programming Interfaces (APIs).

It has obtained success in enterprise/campus networks and data center networks, and researchers are beginning to focus on the deployment of SDN on large-scale networks [2–10]. When applying SDN to the large-scale networks, such as Wide Area Network (WAN), there are still many challenges that should be further investigated and the end-to-end delay of packet traversal is one of them [4–7].

The end-to-end delay of each packet is the sum of the delays experienced at a sequence of intermediate nodes on the way to the destination. Each delay is the sum of two parts, a fixed part such as transmission delay and propagation delay and a variable part such as processing delay and queueing delay at the nodes. We analyze two different scenarios: (1) the conventional networks; (2) the SDN-based networks. In the first scenario, as the routing table is preassigned, the corresponding lookup entry in the forwarding table for every packet is early ready before the packet arrival (otherwise the packet will either be routed by default path or dropped). In the second scenario, rule can be installed in the flow table of a switch in either proactive mode or in reactive mode. Existing WAN deployments do not advocate to employ reactive flow setup [1–3]. However, we disclose that SDN switches cannot completely avoid flow setup even in proactive mode and show two scenarios to further prove this conclusion (see in Section 2.2).

On the other hand, stringent time constraint is the cornerstone for real-time applications in SDN. Packets must be delivered between hosts with guaranteed delay bounds. For instance, in order to leverage short-term traffic predictability, fine-grained traffic engineering approach in data center has a delay budget as short as a few hundred milliseconds. Similarly, when link or switch failures are detected in WAN, SDN management application needs to recompute a new path to bypass failed nodes or links. It has only a strict 50 ms requirement for failure recovery [11]. Long delay will result in traffic congestion, and even packet loss.

In this paper, we expose the end-to-end delay of SDN network which consists of multiple nodes, fully disclosing the delay components which contribute to the long delay. We propose a model for analyzing the end-to-end delay. This model takes into account different time consumption of rule installation on different switches, which is different from the models proposed in previous studies. Models in previous studies [4–7] are based on the assumption that all the switches managed by one controller can receive and install the corresponding rules from the controller at the same time. This assumption actually neglects packet sojourn time in switches when the switches have not finished the rule installation. We conduct experiments on the parameters of this model to get the end-to-end delay. Moreover, we try to discover various factors that affect the delay of both new flows and base traffic. These factors include packet-in messages, TCAM update and rule priority.

Especially, we make the following major contributions:

(1) We propose an end-to-end delay model for SDN which consists of multiple nodes. This model shows the breakdown of packet traversal delay and includes important timing components, recognizing key complexities introduced by the interactions between the separate forwarding plane and control plane in SDN-based network. The interaction introduced in a multi-switch configuration is important and well explained.

(2) We conduct various measurements on the parameters in our delay model and the results show that in some cases, especially when the flow setup is triggered or rule installation on different switches cannot be completed at the same time, the end-to-end delay will increase sharply compared with the delay in the traditional network.

(3) We further decompose the delay of a single SDN switch and show how the packet-in messages and TCAM update prolong the packet sojourn time in an SDN switch.

Measurement results show that as the arrival rate of new flow increases, packet delay will increase significantly and can be incredible high. The packet delay can even reach 2.4654 s when the new flow arrival rate is 5k pps. The high delay is caused by the relatively low processing ability of CPU in the SDN switch. The delay of TCAM update is also high and is influenced by the priority distribution of rules. We compare the delay in SDN-based networks with the delay in traditional networks. In our experimental settings, even the switches are configured as proactive mode, the delay is increased up to 27.95 times in the worst case compared with the delay in traditional networks. The delay in switches contributes a large proportion to the entire delay, and the proportion is more than 97% in some selected source-destination pairs. We conclude that the up-controller-down-switch rule updating mechanism and rule update in TCAM fundamentally determine the over-large delay, which may lead us to rethink the architecture design of SDN in the future.

The rest of this paper is organized as follows. In Section 2, we compare the proactive flow installation with the reactive flow installation and conclude that SDN switches can not completely avoid the flow setup even in proactive mode. In Section 3, we build an end-to-end delay model. In Section 4, we conduct measurements on each parameter. Section 5 reviews the related works. Finally, Section 6 concludes the paper.

2. Motivation

2.1. Proactive Flow Installation versus Reactive Flow Installation. In SDN, when a controller populates flow tables, there are essentially two modes of operation: proactive rule installation and reactive rule installation [12].

In reactive mode, flow table is empty at the beginning. When a packet arrives at the switch, no rule is matched and the packet will be encapsulated as packet-in message and sent to the controller by the data plane CPU in the switch. At the same time, the switch buffers the packet waiting for the controller's instruction. The controller downloads the corresponding rule(s) into the switch. Subsequent packets of the same flow then match the newly installed rule(s) and are transmitted at full line rate, without disturbing the controller.

However, prior study [13] and our measurement (see in Section 4) disclose that initiating flow setup for every incoming new flow, together with TCAM update, increases the delay of packets. What is worse, it also increases the workload of the controller, further affecting the stability of the network.

Existing SDN deployments in WAN such as AL2S on Internet2 do not employ reactive flow setup [2]. The OESS controller provided by Indiana University is similarly deployed across a number of production WANs to provide forwarding free of flow setup latency [3]. Diego Kreutz et al. [1] declared that large-scale SDN deployments should rely on a combination of proactive and reactive flow setup.

In proactive mode, flow table is installed in the switch in advance before the packet arrival. To achieve wire-speed lookup performance, current merchant switches usually use TCAM to store these rules. TCAM can match each incoming packet against all the rules in parallel and output the result in a single clock cycle. However, there may not be enough space in an SDN switch to accommodate all the rules, especially in the large-scale networks, because of the following points:

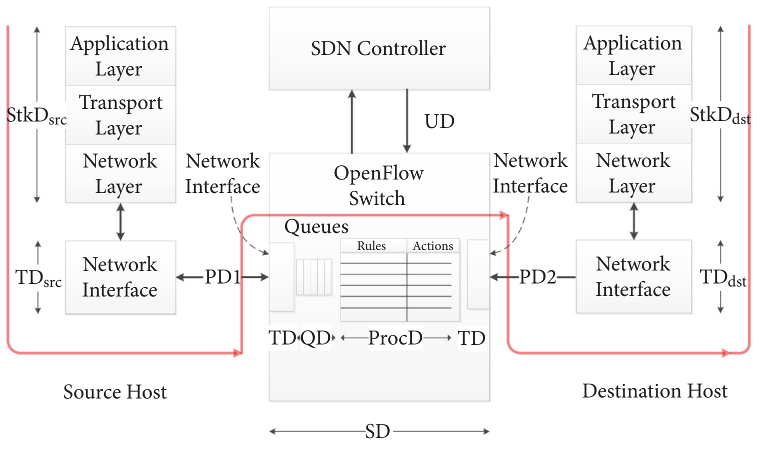

FIGURE 1: The sketch of end-to-end delay breakdown.

(1) TCAM is expensive and has high power consumption [14]. Current TCAM chipsets typically support just a few thousand or tens of thousands of entries [15, 16], which is far from meeting the current network management requirements.

(2) The gap between the significant increase of traffic and the slow increase of TCAM capacity is growing. The scale of flow table is constantly expanding.

(3) SDN table size is also increasing. Take OpenFlow, one of the well-defined open southbound APIs as an example, the number of match fields is 12 in version 1.0, but 45 in version 1.5. This means each rule will take up more storage space.

When TCAM is full and a new rule needs to be inserted, we need to delete the old rule to leave space for the new rule. No matter what replacement algorithm is adopted, it will inevitably cause some active rules to be deleted from flow table. The subsequent packets correspond to these deleted rules have to invoke packet-in generation.

2.2. Motivating Scenario. The following two scenarios highlight that even if the SDN switch is configured in proactive mode, it is still possible that flow setup is invoked.

2.2.1. The Need for Many Fine-Grained Rules. Some researches such as DevoFlow [16] proposed using coarse-grained rules to reduce invoking controller as much as possible. However, this is a dilemma: aggressive use of flow-match wildcards undermines the ability of controller to effectively manage network traffic and also can not make accurate measurement and statistics-gathering. Actually, in order to improve the network utilization and application performance and meet the needs such as security, measurement, and fairness in various network scenarios, more and more traffic management policies are proposed. These policies are usually translated into fine-grained rules (e.g., access control rules, rate limiting rules, and policy routing) [17]. The number of fine-grained rules can probably reach up to hundreds of thousands or even millions [17], which can hardly be stored in the current TCAM. This will inevitably lead to the replacement of rules and flow setup.

2.2.2. Virtual Machine Migration. Due to the CPU and memory constraints and traffic changes, VMs are frequently required to migrate both within and across data centers [18]. However, VMs usually have a tight coupling with the network. For example, network polices such as QoS, ACL, and policy routes, in switches usually, depend on the VMs. Consequently, VM migration often induces the reconfiguration of the network policies. The corresponding switch's rules need to be deleted and this process causes the update of the flow table. Meanwhile, the switch associated with the new server dynamically triggers the rule insertion to redirect the traffic to new physical interface.

3. Delay Breakdown

To understand how the end-to-end delay in SDN impacts the large-scale networks, we should develop a model to analyze the end-to-end delay (shown in Section 3.1). We will further study the delay of SDN switch in Sections 3.2 and 3.3. Based on the single switch delay model, we propose a delay model for multiple switches. This model takes into account the impact of rule installation sequence on different switches.

3.1. End-to-End Delay Model. Figure 1 shows the delay breakdown of end-to-end packet traversal. Packets start from the source host, cross several switches, and finally arrive at their destination host. The total delay includes (1) the protocol processing delay ($StkD$), which is the processing delay associated with the protocol stack of source and destination host; (2) the transmission delay (TD) on the source and destination host; (3) the propagation delay (PD) on transmission medium; (4) the switch delay (SD) caused by packet forwarding in the switches. The end-to-end delay can be presented by the following formula:

$$D = StkD_{src} + TD_{src} + PD + SD + TD_{dst} + StkD_{dst} \quad (1)$$

The end-to-end delay starts from the moment when the application on the source host sends out a message to the socket interface. Then the message is processed by the protocol stack and finally the Network Interface Card (NIC) fetches a packet from the host memory and sends it to the Physical Layer (PHY). The time cost in above procedure is represented by $StkD_{src}$. The receiving process on the destination host is the reverse process of packet sending. $StkD_{dst}$ is the amount of delay which the packet is processed by the protocol stack on the destination host. $StkD_{dst}$ is the time gap between the moment when the NIC starts to copy the packet into memory using DMA and the moment when an application at the destination host receives this packet.

TD_{src} is the delay it takes to push all bits of a packet from the source host to the outgoing link. Similarly, TD_{dst} is the amount of time required to receive all bits of a packet from the link to the destination host. TD can be calculated by the following equation: $TD = s/r$, where s is data size and r is transmission rate. For a 1500-byte packet, for example, if the data is transmitted through the 1 Gbps link, $TD = (1500 * 8)bit/(10^9 bit/s) = 12us$.

PD refers to the delay of propagating packets in the communication media (copper, optical fiber, or wireless channel) on each link. It is proportional to the distance between the source host and the destination host and can be presented by the following formula for each link segment: $PD = l/p$, where l is distance and p is propagation speed. Propagation speed varies in different communication media. The speed of the copper wire is approximately $2.3 * 10^5 km/s$, while being approximately $2.0 * 10^5 km/s$ in fiber-optic cable [19].

SD is the total delay caused by the passing SDN switches, which will be discussed later.

3.2. Single SDN Switch Delay Model. For the purpose of clarity, we will first develop a delay model for a single SDN switch. Figure 2 shows the delay breakdown of flow setup on a single switch S_i. The flow setup can be further divided into the following subprocedures: (1) When the packets arrive in burst, they may not be immediately processed and have to be queued in the buffer. The waiting time is queue delay (q_i). (2) The switch extracts the header from an incoming packet and compares it against the flow table. We name the time cost of this process as processing delay (t_i). (3) If the packet does not match any entry in the flow table, it will be transmitted from the forwarding engine to the host processor in the data plane via PCI/PCIe. The host processor then generates the corresponding packet-in message. The delay is defined as h_i^{up}. (4) Packet-in message is uploaded to the controller via secure OpenFlow channel and this delay is defined as w_i^{up}. (5) The controller parses the packet-in message and generates the control messages including flow-mod and packet-out based upon its local policies. The delay is defined as C.

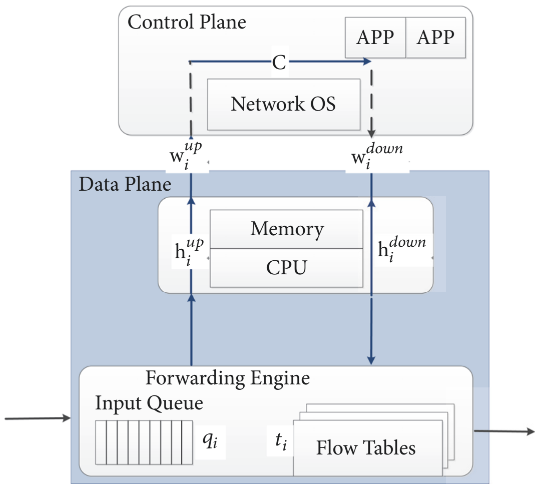

Figure 2: The delay breakdown of flow setup.

(6) The controller message is downloaded to the switch via secure OpenFlow channel. The delay is defined as w_i^{down}. (7) Once receiving the control message, the host processor parses and converts the logic representation of instructions into hardware specific instructions. The instructions are transferred to the forwarding engine of switch via PCI/PCIe. The instructions are executed by the hardware and the new rule(s) are inserted into the search engine (e.g., TCAM). The delay is defined as h_i^{down}. D_i^{setup} is the total delay of the above subroutines and can be expressed as the following formula:

$$D_i^{setup} = q_i + t_i + h_i^{up} + w_i^{up} + C + w_i^{down} + h_i^{down} \quad (2)$$

This kind of up-controller-down-switch rule update operation is, to some extent, similar to the ATM signaling procedure, but more time-consuming. Particularly, in subprocedure 3, if the packet matches an entry in the flow table, the flow setup is bypassed and the delay model of S_i can be simplified as

$$D_i^{bypass} = q_i + t_i \quad (3)$$

3.3. Multiple SDN Switches Delay Model. In this section, we focus on the delay of multiple SDN switches. Compared with the single switch scenario, the delay model of multiple switches is more complicated. When the controller sends out the flow-mod message, the propagation delay between the controller and different switches is different. Moreover, the time consumption of rule installation on different switches is also different, depending on the layout of the flow table in TCAM and the dependencies between the new arrival rule(s) and existing table items in TCAM. These two kinds of delay may affect the order of rule installation on different switches, and may further affect the consistency of the control logic [20].

We discuss the delay in proactive mode and reactive mode, respectively. Although the existing SDN deployments in WAN such as AL2S on Internet2 [2] and OESS [3] do

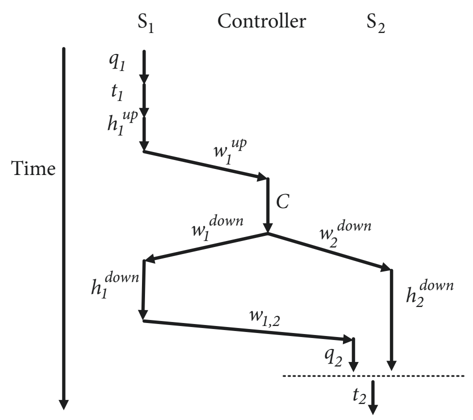

FIGURE 3: Sequence diagram of packet traversal in the example topology. This topology consists of two switches S_1 and S_2.

not employ reactive flow setup, we will still build an end-to-end delay model and make some measurements for purpose of comparison. We still use the variables in Section 3.2. We define new parameters, $D_{i,j}^{reactive}$, which is the delay start from the moment when S_i receives the packet to the moment when the packet leaves S_j in reactive scenario and $D_{i,j}^{proactive}$, which is the corresponding delay in proactive scenario. The parameter $w_{i,j}$ is defined as the propagation delay between S_i and S_j.

3.3.1. Reactive Delay Model. For better explanation, we first study a simple topology which only consists of two switches, S_1 and S_2. These two switches are managed by the same SDN controller.

We use sequence diagram to describe the delay of packet traversal in this topology (shown in Figure 3). In reactive mode, flow table is empty in the beginning. Flow setup is triggered in the ingress switch S_1, and then the controller downloads the corresponding rule(s) into S_1 and all the downstream switch(es), S_2 in this example. The downstream switch(es) are no longer required to trigger flow setup. The delay is $(w_1^{down} + h_1^{down} + w_{1,2} + q_2)$ from the time point at which the controller sends out the rule(s) to the time point at which the packet leaves S_1 and arrives at S_2. Meanwhile, S_2 will spend $(w_2^{down} + h_2^{down})$ on receiving and installing the rule(s). When the packet comes to S_2 and is ready to search, if the corresponding rule(s) have already been downloaded into the TCAM, the packet will not necessarily need to wait in S_2 and the total delay is

$$D_{1,2}^{reactive} = D_1^{setup} + w_{1,2} + D_2^{bypass} \tag{4}$$

Otherwise, the packet has to be suspended until the end of TCAM reorganization in S_2. The total delay can be represented by the following equations:

$$D_{1,2}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + w_2^{down} + h_2^{down} + t_2 \tag{5}$$

The above two equations can be integrated into one equation:

$$D_{1,2}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + t_2 + \max\left(w_1^{down} + h_1^{down} + w_{1,2} + q_2, w_2^{down} + h_2^{down}\right) \tag{6}$$

We define a new parameter M_j. M_j is the delay from the time point at which the controller sends out the rule(s) to the time point at which the packet is ready to search the flow table in S_j. M_j can be represented by the following recursive equation:

$$M_j = \max\left(M_{j-1} + w_{j-1,j} + q_j, w_j^{down} + h_j^{down}\right) \tag{7}$$

The initial value of M_1 is

$$M_1 = w_1^{down} + h_1^{down} \tag{8}$$

Then (6) can be simplified as

$$D_{1,2}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + M_2 + t_2 \tag{9}$$

Similarly, when there are three switches, the total delay $D_{1,3}^{proactive}$ can be represented:

$$D_{1,3}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + M_3 + t_3 \tag{10}$$

Furthermore, we give a generalization of the above model, and the total delay of packet traversal from S_i to S_j can be formulated by the following equation:

$$D_{i,j}^{reactive} = q_i + t_i + h_i^{up} + w_i^{up} + C + M_j + t_j \tag{11}$$

3.3.2. Proactive Delay Model. In proactive mode, as mentioned in Section 2.1, although rules are preinstalled in the switch, packet will still experience table miss and be sent to the controller at a certain probability. The parameter p_i denotes this probability in S_i. So in proactive scenario, the delay of switch S_i, $D_i^{proactive}$ can be calculated as

$$D_i^{proactive} = p_i \cdot D_i^{setup} + (1 - p_i) \cdot D_i^{bypass} \tag{12}$$

In order to derive the end-to-end delay formula, we will first focus on a simple topology which includes two switches S_1 and S_2 managed by the same SDN controller. In this scenario, there are three cases: (1) search in S_1 misses and search in S_2 hits; (2) search in S_1 hits and search in S_2 misses; (3) search in S_1 hits and search in S_2 hits. The probability and corresponding delay of each case are listed in Table 1. So the total delay $D_{1,2}^{proactive}$ in proactive mode can be calculated:

$$\begin{aligned} D_{1,2}^{proactive} &= p_1 \cdot D_{1,2}^{reactive} + (1 - p_1) \cdot p_2 \cdot \left(D_1^{bypass} + x_{1,2} + D_2^{setup}\right) + (1 - p_1) \cdot (1 - p_2) \cdot \left(D_1^{bypass} + x_{1,2} + D_2^{bypass}\right) \qquad (13) \\ &= p_1 \cdot D_{1,2}^{reactive} + (1 - p_1)\left(D_1^{bypass} + x_{1,2} + D_2^{proactive}\right) \end{aligned}$$

Table 1: The probability and corresponding delay of three cases in above example.

Case #	Detail	Probability	Delay
case 1	S_1 misses, S_2 hits	p_1	$D_{1,2}^{reactive}$
case 2	S_1 hits, S_2 misses	$(1-p_1)\cdot p_2$	$D_1^{bypass}+x_{1,2}+D_2^{setup}$
case 3	S_1 hits, S_2 hits	$(1-p_1)\cdot(1-p_2)$	$D_1^{bypass}+x_{1,2}+D_2^{bypass}$

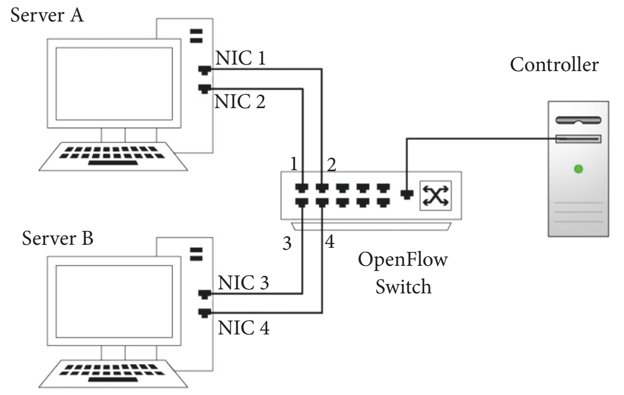

Figure 4: The setup of measurement experiment. Server A is used to generate new flows and server B is used to generate base traffic.

The above equation can be simplified as

$$D_{1,2}^{proactive} = p_1 D_{1,2}^{reactive} + (1-p_2)\left(D_2^{bypass} + x_{1,2} + D_1^{proactive}\right) \quad (14)$$

Similarly, the total delay $D_{i,j}^{proactive}$ can be presented by the following recursive equation:

$$D_{i,j}^{proactive} = p_j \cdot D_{i,j}^{reactive} + \left(1-p_j\right) \cdot \left(D_j^{bypass} + x_{j-1,j} + D_{i,j-1}^{proactive}\right) \quad (15)$$

4. Delay Measurement

4.1. Measurement Methodology. In order to get the delay decomposition for the components in above delay models, we build an experimental testbed to measure the actual time cost. The measurement platform is shown in Figure 4.

We use Floodlight (version 0.9) as the running package in our controller and the hardware configuration of controller is listed in Table 2. The operating system on the controller is Ubuntu 12.04.5 LTS, and kernel version is 3.11.0-26-generic. The SDN switch we use is Pica8 P-3297. The local CPU in control plane is PowerPC P2020. The switch runs PicOS 2.4 operating system and supports OpenFlow v1.0. Server A is used to generate packets for new flows according to different parameters. These new flows will trigger packet-in packets in the switch under testing. Server B is used to generate base traffic. Base traffic refers to the packets that match the preinstalled rule(s) in TCAM. Both server A and server B are installed with two 10 Gbps NICs (Intel Ethernet Converged Network Adapter X520-DA2). Controller is connected to 3297's management port via a 1Gbps port. Clocks on different server are not synchronized. In order to avoid the deviation caused by the time synchronization, we use one NIC to send packets and use another NIC on the same machine to receive packets. We use Ostinato to generate traces with size and characteristic diversity. We deploy Wireshark to capture packets on sending NIC and receiving NIC, respectively.

Table 2: The hardware configuration of the controller.

Item	Controller
Motherboard	Asus P9X79 WS (INTEL C222)
CPU	Intel Xeon E3-1230v3 (4 cores, 3.3GHz)
RAM	DDR3 ECC 32GB (1600MHz)

4.2. SDN Switch Delay Measurement

4.2.1. Impact of Packet-In on Delay. In this part, we conduct the measurement on the packet delay when the packet-in packets are generated. The flow table in switch is empty initially. NIC1 on Server A sends new flows at different speeds to the switch via port 1 and these new flows will trigger corresponding packet-in packets. The switch receives and parses flow-mod messages from the controller and inserts the corresponding flow entries into TCAM. We set all these flow entries' priority equally to isolate the impact of TCAM reorganization. Then the switch forwards these packets to NIC 2. The start point of the packet delay is the moment when NIC 1 begins to send out and the end point is the moment when NIC 2 receives the packet.

We conduct three experiments to show the packet delay with/without packet-in under new flow rate 50 pps, 500 pps, and 5k pps, respectively. The results are present in Figures 5, 6, and 7. In each experiment, the total number of flows is 300 and x-axis refers to flow ID. For a certain new flow arrival rate, the delay of packets that trigger packet-in is dramatically larger than that of packets without packet-in. For example, when the flow arrival rate is 5k pps, the packet delay without packet-in is variable with a mean of 0.0859 ms and the standard deviation of 0.0452, while the mean of

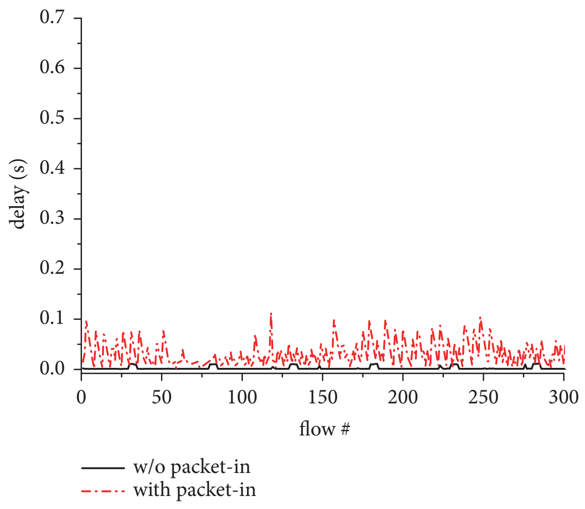

Figure 5: Delay of packets with/without packet-in when new flow rate is 50pps.

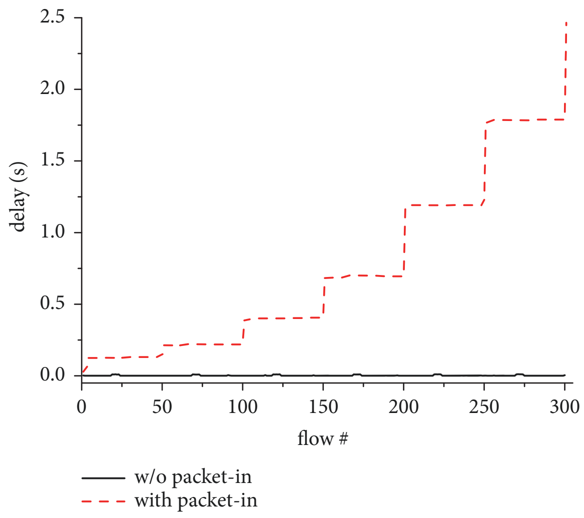

Figure 7: Delay of packets with/without packet-in when new flow rate is 5kpps.

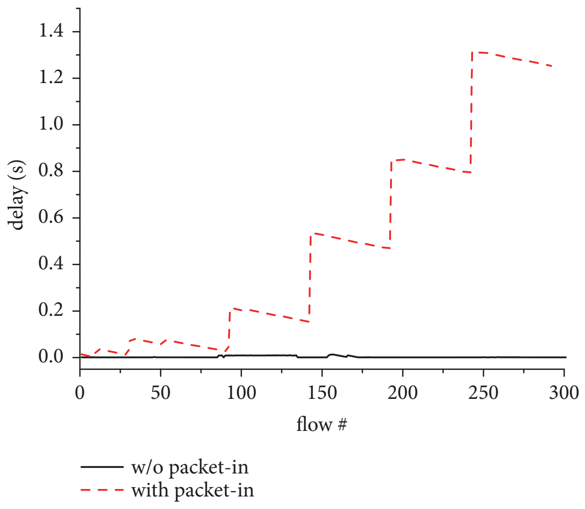

Figure 6: Delay of packets with/without packet-in when new flow rate is 500pps.

the packet delay with packet-in is 0.7415 s and the standard deviation is 0.5954.

We move to the delay of the packets which trigger the packet-in. The packet delay increases with the new flow arrival rate. For example, the average delay of the new flows with 50 pps arrival rate is 32.291 ms while that of the new flows with 5k pps arrival rate is 0.7415 s. The maximum delay is 2.4654 s at 5k pps. We can also find that the delay distribution of the new flows with 50pps arrival rate is different from that of the new flows with the arrival rate of 500 pps and 5k pps. The delay distributions in Figures 6 and 7 present staircase shape. The delay for every 50 packets is increased dramatically.

In order to find the root cause of this phenomenon, we further conduct the following two sets of experiments. On one hand, we use Cbench [21], an SDN controller performance testing tool, to conduct a measurement on the controller. The Cbench is configured as throughput mode. The controller in the experimental topology manages 1000 PCs via one switch. The test is repeated 10 times. Experimental results show that the minimum throughput is 451119.36 responses/s, the maximum throughput is 473804.17 responses/s, and the average throughput is 460435.15 responses/s. On the other hand, we use Wireshark to capture the packets on the controller's NIC. We find that the switch sends the packets to the controller in a batch and the batch size is exactly 50 packets. The average delay of each new flow processed by the controller is about 1-2 ms and does not drastically increased. This means the packet-in generation speed in above experiments does not reach the controller's performance bottleneck. Meanwhile, we list the CPU utilization with/without packet-in under new flow rate of 50 pps, 500 pps, and 5k pps, respectively in Table 3. We can see CPU utilization increases significantly with the increase of the new flow rate, especially when the packet-in packets are triggered.

Based on the above experiments, we can conclude that local CPU in an SDN switch contributes the majority of the delay. Packet-in generation, together with flow-mod message, consumes too much CPU computation resources. When the new flow arrival rate increases, packets have to be buffered and handled in a batch.

4.2.2. Impact of TCAM Update on Delay.

In this section, we will explore the switch delay during TCAM update. At first, the flow table stores only one rule. This rule has the lowest priority and is used to forward the base traffic. NIC 1 sends 1600 new flows to the switch. The packets arrival rate is fixed at 5 pps. The new flows trigger the rule installation; thus the switch will insert the corresponding rule for each flow into the TCAM. Then the switch forwards these packets to NIC 2 guided by the flow table. NIC 3 is used to send the base

Table 3: CPU utilization with/without packet-in under different new flow rates.

Item	New Flow Rate		
	50 pps	500 pps	5k pps
w/o pkt-in	3%-4.1%	12%	13.20%
with pkt-in	24%-30%	36%-48%	35.2%-67%

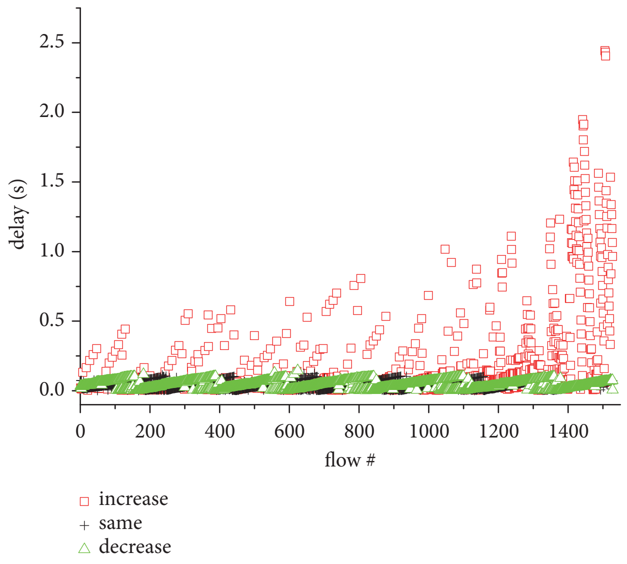

Figure 8: Delay comparison of packets when the priority of rules is in the same order, increasing order and decreasing order, respectively.

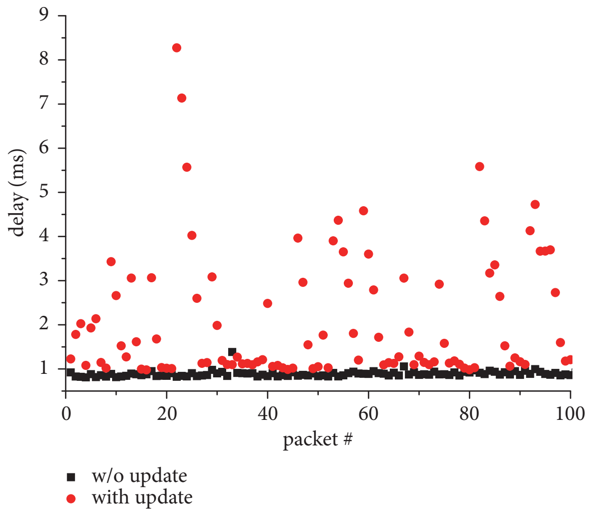

Figure 9: Delay comparison of base traffic with/without TCAM update.

traffic. The base traffic will match the preinstalled rule. NIC 4 on server B receives these packets.

First we focus on how the rule priority affects the switch delay. We carry out three experiments. By modifying Floodlight code, we set the priority of rules in the same order, increasing order and decreasing order, respectively. The rule download sequence can be defined as $r_1, r_2, \ldots r_n$. Denote $r_i.prio$ as the priority of r_i. For the increasing order, $r_i.prio > r_j.prio$, if $i > j$. For the same order, $r_i.prio = r_j.prio$, if $i > j$. For the decreasing order, $r_i.prio < r_j.prio$, if $i > j$. The result is shown in Figure 8. We can see when in the same order and deceasing order, the new flow delay is usually below 100 ms. However, when in the increasing order, the new flow delay shows a sharp increase, especially after the arrival of 1200th new flow.

The root cause behind this phenomenon is the mechanism of rule organization in TCAM. In the design of Pica8 3297 switch, rule with highest priority will always be stored in the lowest address of TCAM. When the rule with higher priority is inserted, its position is already occupied by the rule with relatively low priority. Then the rules that have already been in the TCAM have to be moved one by one to free the position with lower address. These new flows have to be buffered until the end of the TCAM update. If the download rule sequence is in the same order or decreasing order, there will be no entry move or only a few entry moves.

We now study the impact of TCAM update on the delay of base traffic. The start point of the base packets delay is the moment when NIC 3 on Server B begins to send out the packets. The end point of is the moment when NIC 4 receives the packets. The rules are inserted in increasing order. Figure 9 shows the delay comparison of base flow with/without TCAM update. We can see when TCAM update happens, the delay of base traffic is significantly increased. The average delay of base traffic with TCAM update is 2.42 times longer than that of base traffic without TCAM update. Meanwhile, local CPU utilization is increased and the counter of the rule on data plane is also increased. We infer that during the update of TCAM, table lookup on the TCAM is suspended and the base traffic is buffered and directed to the local CPU for table lookup. We can conclude that TCAM update not only causes the increase of the new flow delay, but also increases the delay of the base traffic. The probability of packet loss is greatly increased when more rules need to be moved and the base traffic rate is high. This will potentially reduce the stability of the entire network.

Previous research efforts have been made on TCAM update issues (e.g., [22, 23]). These optimizations try to minimize the number of rule movements at the cost of reducing storage utilization. These algorithms cannot thoroughly avoid the rule update. On the other hand, in SDN scenario, besides topology change and rule update by network administrator, packet-in generation will make the TCAM update more frequent.

Table 4: The detail and the probability of flow setup of each trace.

Name	Source	Link Rate	Start Time	p
trace 1	chicago.dirB	OC-192	20160218-133000	16.87%
trace 2	chicago.dirB	OC-192	20160317-133000	16.57%
trace 3	chicago.dirB	OC-192	20160406-133000	18.56%

Table 5: The tracert results of five sets.

SET	Src IP	Dst IP	TD (us)	Number of hops	Number of ASes	PD (ms)
Pair 1	199.58.208.116(US)	166.111.68.210(China)	0.512	20	3	121.265
Pair 2	199.58.208.116(US)	146.97.137.154(UK)	0.512	11	3	82.005
Pair 3	199.58.208.116(US)	202.53.187.213(New Zealand)	0.512	18	4	136.36
Pair 4	199.58.208.116(US)	155.232.27.78(South Africa)	0.512	16	4	145.355
Pair 5	199.58.208.116(US)	128.100.96.9(Canada)	0.512	10	3	31.36

4.3. The Probability of Flow Setup in Proactive Mode. Before the end-to-end measurement, we conduct data mining on the probability of flow setup in proactive mode, namely, p_i in (12). In order to get this value, we inject the real world traffic [24] to the Pica8-3297. The traces are collected from a high-speed link of a Tier-1 ISP between Chicago and Seattle. The traces are collected from two directions: direction A is from Seattle to Chicago, and direction B is from Chicago to Seattle. The trace used in our experiments is from direction B and the results from direction A are similar. There are three trace sets used in our measurement. Each set lasts 180s.

We use the 5-tuple (source IP, destination IP, source port number, destination port number, and type of the protocol) to identify the flows. The TCAM of our tested switch can accommodate 4000 5-tuple rules. We preinstall 4000 rules into TCAM. When the TCAM is full, we use the commonly used replacement algorithm LRU to replace old rules. Table 4 lists the detail and the probability of flow setup of each trace. From the table, we can find that, in proactive mode, there are still nearly 20% packets that fail to match the rules in switch and need to trigger the flow setup.

Although aggressive use of coarse-grained rules such as range match or wildcard rules may cover more percentage of traffic and thus reduce the probability of flow setup, it restricts the granularity of flow control.

4.4. End-to-End Delay Measurement and Calculation. In this section, we focus on the end-to-end delay in SDN-based networks (in reactive mode and in proactive mode) and compare it with the delay in conventional networks.

When applying SDN, as we known, the enterprise/campus networks or the data center networks can be viewed as the first/last hop in the Internet. In the intermediate MAN/WAN, a packet will jump more hops to route to its destination, crossing many ASes/ISPs. However, due to the fact of policy nonsharing among ISPs, each SDN domain will execute its own forwarding decision independently. This causes every SDN domain produces its own rule(s). Thus the setup packet will always be forwarded to the controller(s) at the ingress switch of each SDN domain to ask for new rule(s) in reactive mode. In proactive mode, the flow setup is triggered with a certain probability in each SDN domain.

In order to get the end-to-end delay, we conduct a variety of experiments on the parameters of equations in Section 3.3. Single test is easily influenced by random factors. In order to make our results more reliable and robust, we have repeated each measurement 10 times and use the average value of each parameter. We use the TraceRT tool to get the path of different source-destination pairs. We randomly select several IP addresses worldwide as the destination IP addresses. By using TraceRT, we get all the IP addresses of the routers passing from source to destination. We can get the AS ID of these IP addresses by using tools in https://iptoasn.com/. We get the value of PD by the following steps: (1) We obtain the geographical locations of these IP addresses by using the tool in https://www.ip2location.com. (2) We use maps.google.com to get the distance between these geographical locations. (3) We get the length of submarine cable in the world by using the tool in https://www.cablemap.info. Although there are slight deviations between the values obtained by these tools and the actual values, we believe the deviations are acceptable.

We set the size of packets which travel from source to destination to be 64 bytes. The packets are sent and received at the end host at 1 Gbps transmission rate. For the end host, protocol stack delay can be accurately measured by adding kernel probes into the source code of the application and kernel. A ping-pong test [19] shows that, for a 64-byte packet, the value of $StkD_{src}$ is at 4159 ns and the value of $StkD_{dst}$ is 7747 ns. Queue delays vary widely in different network environments, so our measurements will not take into account the queue delays in this part.

Table 5 lists the TraceRT results of the five sets. Each set includes source IP address (src IP), destination IP address (dst IP), transmission delay at the end host (TD), number of hops, number of ASes, and total propagation delay (PD).

Till now, we have got the value of each parameter related to the total delay D, including the switch delay (seen in Section 4.2). In particular, the delay of a switch without flow setup, namely, D^{bypass} in (2), ranges from 892.162 us to 11.350

Table 6: The comparison of end-to-end delay in different scenarios.

	Delay in conventional network	Delay in SDN-based network (reactive mode)	Delay in SDN-based network (proactive mode)
Pair 1	128.386 ms	223.504ms-7.709s	153.744ms-1.624s
Pair 2	85.271 ms	176.215ms-7.568s	106.455ms-1.483s
Pair 3	139.785 ms	264.943ms-10.115s	171.930ms-2.042s
Pair 4	160.947 ms	272.153ms-10.142s	179.140ms-2.028s
Pair 5	45.064 ms	124.678ms-7.506s	54.918ms-1.421s

ms; the delay of a switch with flow setup, namely, D^{setup} in (3), ranges from 29.020 ms to 2.465 s (new flow rate=5k pps).

By using the TraceRT tool, we can get the end-to-end delay in current network. Table 6 lists the end-to-end delay in conventional network and SDN-based network (in reactive mode and in proactive mode).

We need to notice that the delay in SDN-based network includes a lower bound and an upper bound. In fact, the delay of most packets is between the upper bound and the lower bound. As can be seen from the table, the summing-up delay in SDN-based network is much larger than the delay in conventional network. Particularly, in proactive mode, the delay is increased by 11.3% in the best case and up to 27.95 times in the worst case in our experimental settings. The delay in switches contributes a large proportion to the entire delay, and the proportion is more than 97% in some selected source-destination pairs. Furthermore, in SDN scenario, packet delay in reactive mode is incredible high, especially in the worst case. This is due to the following reasons: (1) flow setup and TCAM updates cause increased delay; (2) the calculation is based on an assumption. The assumption is that every traversal of an AS boundary needs an independent computation of forwarding rules, which must be executed sequentially, implying additive delays; (3) due to the limitation of experimental conditions, the tested switch we choose, Pica8-3297, is a relatively low performance switch; (4) each AS has to experience a worst case switch delay on the same packet, which is a low probability event.

5. Related Work

There have been several studies related to performance evaluation of the SDN system. Cbench [21] evaluated controllers by generating packet-in events for new flows. The delay mode in Cbench measured the controller's request processing delay under low-load conditions. Bianco et al. [25] conducted experiments on the performance of software-based OpenFlow switch built on an Ubuntu PC and concluded that a Linux based OpenFlow switch was able to offer very good performance. OFLOPS [26] conducted detailed measurements on the OpenFlow-enabled switches from three different switch vendors. Huang et al. [27] also benchmarked three OpenFlow-enabled switches and showed that different implementation dramatically affected the delay and the throughput. However, these previous researches only focused either on controller side or switch side, just from a local view. Our research pays attention to the delay of SDN architecture which consists of multiple nodes. Moreover, compared with these researches, we not only measure the delay of the new flow, but also explore the impact of TCAM update and flow setup on the delay of base traffic.

Model proposed in [28] is the first attempt to analyze the delay of OpenFlow-based networks. The model is based on queueing theory. The switch is modelled as an M/M/1 queue and the controller as an M/M/1-S feedback queue system. However, this model can only be applied to a single switch per controller. Model proposed in [4] is a recent extension of that work. The new model can be applied in the scenario in which a controller is responsible for a number of different switches in the data plane. But these two models did not distinguish between traffic from the controller and other traffic; thus they can not accurately model the interaction between the controller and the switches. Azeem Iqbal et al. [5] proposed a stochastic model for the end-to-end delay in SDN-based networks and performed experiments on three platforms. However, their measurements are based on OVS, a software switch, which is substantially different from many ASIC-based OpenFlow switches. Our experiments are all based on hardware switches. Siamak Azodolmolky et al. [6] utilized deterministic network calculus as a mathematical framework to analytically model the behavior of an SDN switch and its interaction with an SDN controller. Lin et al. [7] exploited the capabilities of the stochastic network calculus method to analyze and evaluate the performance of SDN deployment. The above models [4–7] are based on the assumption that when the flow setup is triggered, all the switches managed by one controller can receive and install the corresponding rules at the same time. Our models take into account the different installation time on different switches. What is more, we also conduct data mining on the probability of flow setup in proactive mode, which is also an important factor to the end-to-end delay.

6. Conclusion

When applying SDN to the large-scale networks, such as WAN, the end-to-end delay of packet traversal is a crucial factor for real-time applications. What is more, understanding the packet delay in SDN-based large networks helps the researchers better design the switch architecture, optimize the network algorithms, and choose appropriate parameters in simulation tools and analytic studies. In this paper, we propose a model to quantify the end-to-end delay composition. This model takes into account the impact of rule installation sequence on different switches. We further decompose the delay of a single SDN switch. We build real experimental testbed to conduct various measurements to figure out how packet-in messages and TCAM update prolong the packet sojourn time in an SDN switch. Measurement results show a prodigious increase in the end-to-end delay compared with the traditional network if no additional measure is taken. Given the stringent time constraint, a proportion of real-time applications may potentially suffer the timeout for connection setup. This, in return, will lead us to rethink feasibility of directly deploying SDN to the large-scale networks.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is sponsored by NSFC (61432009, 61373143, and 61872213) and partially by Huawei Innovation Research Program (HIRP).

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] "Internet2's advanced layer 2 service," 2007, https://www.internet2.edu/products-services/advanced-networking/layer-2-services/.

[3] "Oess: open exchange software suite," 2017, http://globalnoc.iu.edu/sdn/oess.html.

[4] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[5] A. Iqbal, U. Javed, S. Saleh, J. Kim, J. S. Alowibdi, and M. U. Ilyas, "analytical modeling of end-to-end delay in openflow based networks," *IEEE Access*, vol. 5, pp. 6859–6871, 2017.

[6] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An analytical model for software defined networking: a network calculus-based approach," in *Proceedings of the 2013 IEEE Global Communications Conference, GLOBECOM 2013*, pp. 1397–1402, USA, December 2013.

[7] C. Lin, C. Wu, M. Huang, Z. Wen, and Q. Zheng, "Performance evaluation for SDN deployment: an approach based on stochastic network calculus," *China Communications*, vol. 13, supplement 1, pp. 98–106, 2016.

[8] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Exploring source routed forwarding in SDN-based WANs," in *Proceedings of the 2014 1st IEEE International Conference on Communications, ICC 2014*, pp. 3070–3075, Australia, June 2014.

[9] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of SDN in hybrid enterprise and ISP networks," in *Proceedings of the Symposium on SDN Research*, pp. 1–7, ACM, Santa Clara, Calif, USA, March 2016.

[10] H. Luo, J. Cui, G. Chen, Z. Chen, and H. Zhang, "On the applicability of software defined networking to large scale networks," in *Proceedings of the 2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6, China, August 2014.

[11] X. Wen, B. Yang, Y. Chen et al., "RuleTris: minimizing rule update latency for TCAM-based SDN switches," in *Proceedings of the 36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016*, pp. 179–188, Japan, June 2016.

[12] M. Casado, M. J. Freedman, J. Pettit et al., "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1270–1283, 2009.

[13] K. He, J. Khalid, A. Gember-Jacobson et al., "Measuring control plane latency in SDN-enabled switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR 2015*, pp. 25:1–25:6, ACM, June 2015.

[14] K. Zheng, C. Hu, H. Lu, and B. Liu, "A TCAM-based distributed parallel IP lookup scheme and performance analysis," *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, pp. 863–875, 2006.

[15] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: scalable ethernet for data centers," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, pp. 49–60, ACM, Nice, France, December 2012.

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM*, pp. 254–265, ACM, August 2011.

[17] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Scalable rule management for data centers," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, pp. 157–170, USENIX Association, Berkeley, Calif, USA, 2013.

[18] S. Ghorbani, C. Schlesinger, M. Monaco et al., "Transparent, live migration of a software-defined network," in *Proceedings of the the ACM Symposium*, pp. 3:1–3:14, ACM, Seattle, Wash, USA, November 2014.

[19] S. Larsen, P. Sarangam, R. Huggahalli, and S. Kulkarni, "Architectural breakdown of end-to-end latency in a TCP/IP network," *International Journal of Parallel Programming*, vol. 37, no. 6, pp. 556–571, 2009.

[20] M. Canini, D. Venzano, P. Perešni, D. Kostić, and J. Rexford, "A nice way to test openflow applications," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*, p. 10, USENIX Association, Berkeley, Calif, USA, 2012.

[21] "Rob sherwood, kok-kiong yap, cbench: an openflow controller benchmarker," 2017, openflow.org/wk/index.php/Oflops.

[22] Z. Wang, H. Che, M. Kumar, and S. K. Das, "CoPTUA: consistent policy table update algorithm for TCAM without locking," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1602–1614, 2004.

[23] D. Shah and P. Gupta, "Fast updating algorithms for TCAMs," *IEEE Micro*, vol. 21, no. 1, pp. 36–47, 2001.

[24] C. Walsworth, E. Aben, K. C. Claffy, and D. Andersen, "The caida anonymized internet traces," 2016, http://www.caida.org/data.

[25] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: data plane performance," in *Proceedings of the 2010 IEEE International Conference on Communications, ICC 2010*, pp. 1–5, South Africa, May 2010.

[26] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: an open framework for openflow switch evaluation," in *Proceedings of the 13th International Conference on Passive and Active Measurement*, Lecture Notes in Computer Science, pp. 85–95, Springer, 2013.

[27] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proceedings of the 2nd ACM SIGCOMM Workshop*, pp. 43–48, ACM, Hong Kong, August 2013.

[28] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Proceedings of the 2011 23rd International Teletraffic Congress, ITC 2011*, pp. 1–7, USA, September 2011.