This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

# An Efficient and Accurate Link Latency Monitoring Method for Low-Latency Software-Defined Networks

Lingxia Liao, *Student Member, IEEE*, Victor C. M. Leung, *Fellow, IEEE*, and Min Chen, *Senior Member, IEEE*

***Abstract*— This paper proposes a novel latency monitoring method for software-defined networks (SDNs) called LLDP-looping, which uses LLDP packets injected repeatedly in the control plane to determine latency between switches. It provides accurate and continuous latency monitoring without involving any dedicated network infrastructure, while avoiding potential measurement failures that can occur in the existing method of timestamping data packets as probe packets, and overcoming the major factors that decrease the measurement accuracy in many existing methods for monitoring SDN latency. We formulate an optimization problem to enable LLDP-looping to minimize its workload on both control and data planes, and propose a novel greedy algorithm to solve this problem efficiently. Evaluations over the tree-based network topologies demonstrate that LLDP-looping can effectively minimize its overhead, and provide measurement accuracy higher than 90% against the round trip time measured by Ping over an SDN with link latency as small as 0.05 ms. The advantages of LLDP-looping can be realized with minimal modifications to SDN switches and this technique can be generalized to other networking scenarios.**

***Index Terms*— Latency monitoring, minimum vertex cover, OpenFlow, software-defined network (SDN).**

## I. Introduction

THE explosive growth in cloud computing and Internet deployment for latency-sensitive applications has created a massive increase in demand for predictable quality of service (QoS) and quality of experience (QoE), and effective differentiated network services [1]. This increase heightens the needs for measurement technologies to provide network managing with effective tools for monitoring network latency to maintain the QoS/QoE of applications. Such monitoring capability is especially critical over low-latency networks of data centers hosting latency-sensitive applications, such as online gaming/searching/banking, in which several distributed components communicate with others across network devices to generate quick responses for a large number of concurrent users [2], [3].

The latency of a route[1] within a low latency Internet Protocol (IP) network can be accurately and automatically measured using Ping-based utilities. However, Ping-based utilities typically rely on dedicated servers or user hosts running particular monitoring applications (i.e., monitoring points) to repeatedly inject Internet Control Message Protocol (ICMP) packets to enable accurate and automatic latency monitoring over the entire network. To accurately monitor all possible routes within a network, such an arrangement requires a large number of monitoring points to inject a large number of ICMP packets into the network, which could significantly impact the network performance. Therefore, this approach is not suitable to efficiently and continuously monitor the latency of all the routes of networks. Many active or passive latency monitoring approaches proposed in the current literature have provided mechanisms to reduce their resource usage, but they are not really resourced-efficient since either a dedicated network infrastructure has to be built to handle probe packets for active latency measurements [4]–[6], or a global clock has to be deployed to synchronize the time of all the devices in a network for passive latency measurements [7], [8]. The latency monitoring methods based on the ITU-Y.1731 protocol [9] can work in both active and passive modes. Although a global clock is not required in the passive mode, management end points (MEPs), which constitute a dedicated infrastructure, are required in both modes.

Software-defined networks (SDNs) have decoupled control and data planes. With the primitives for injecting OpenFlow [10] messages or data packets into the data plane, the control plane can generate probe packets, insert them into the data plane, and calculate the time difference between receiving and sending a probe packet as $t_{\text{measured}}$. The control plane can further generate OpenFlow messages, insert them into the switches, and calculate half the time difference

Manuscript received February 5, 2018; revised May 27, 2018; accepted May 28, 2018. This work was supported in part by the National Natural Science Foundation of China (Grant 61671088), the Fundamental Research Funds for the Central Universities (HUST: 018KFYXKJC045), and the Canadian Natural Sciences and Engineering Research Council. The Associate Editor coordinating the review process was Huang-Chen Lee. *(Corresponding author: Victor C. M. Leung.)*

L. Liao is with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: liaolx@ece.ubc.ca; vleung@ece.ubc.ca).

V. C. M. Leung is with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada, and also with Zhejiang Gongshang University, Hangzhou 310018, China (e-mail: vleung@ece.ubc.ca).

M. Chen is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China (e-mail: minchen2012@hust.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIM.2018.2849433

[1]This paper uses the route to represent the path between the source and destination switches of a flow and the link to represent the path between two adjacent switches.

0018-9456 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

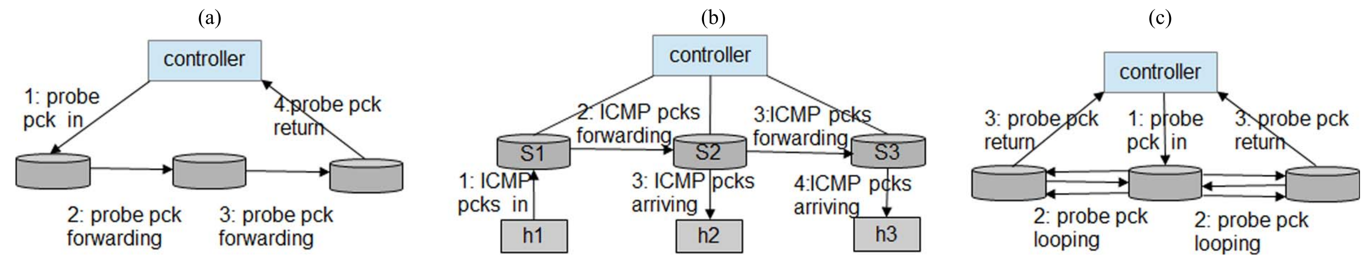

Fig. 1. Latency monitoring in SDNs. (a) Method based on (1). (b) Method based on RTT subtraction. (c) LLDP-looping.

between receiving and sending an OpenFlow message, represented as $t_{\text{con-to-src}}$ and $t_{\text{dst-to-con}}$. The route latency represented as $t_{\text{src-to-dst}}$ can be calculated by

$$t_{\text{src-to-dst}} = t_{\text{measured}} - t_{\text{con-to-src}} - t_{\text{dst-to-con}}. \quad (1)$$

Equation (1) provides an active measurement of latency without the use of dedicated infrastructure, as shown in Fig. 1(a). However, it cannot achieve high accuracy over low-latency networks in which the link latency can be 0.1 ms or even smaller [11] and where many latency-sensitive applications are being run. This is because the different behaviors of a switch in processing data packets, probe packets, and OpenFlow messages cause systemic errors in measuring $t_{\text{measured}}$, $t_{\text{con-to-src}}$, and $t_{\text{dst-to-con}}$ in (1), and performance fluctuations in the controller cause further measurement errors [12]–[14]. We can increase the weight of the real route latency in $t_{\text{measured}}$ to achieve higher measurement accuracy in low-latency networks, by modifying the above as $t_{\text{src-to-dst}} = t_{\text{measured}}/\text{looptimes}$ and using a large value for looptimes, which denotes the number of times each probe packet is forced to loop in the data plane. However, this approach incurs a huge extra workload in the network and potentially changes the network performance and scalability [15], [16]. More importantly, all these approaches do not provide mechanisms to deal with potential latency measurement failures caused by timestamping data packets as probe packets, or minimize the workload caused by latency measurements that might impact the network performance and bias the measurement results. Consequently, although these approaches obviate the need for a dedicated network infrastructure, they may not be suitable to monitor the latency of all the links or routes of an SDN in a continuous manner.

Some other approaches over SDNs such as granular round-trip time monitoring infrastructure (GRAMI) [17] follow the typical strategy used by many active latency monitoring approaches proposed for conventional IP networks [18]–[20]. In these approaches, one or multiple dedicated servers are deployed over the network to send probe packets repeatedly to carefully selected hosts. The round trip times (RTTs) of a link can be directly measured by a probe packet or the subtraction of two RTTs. As shown in Fig. 1(b), the RTT of link $S2$–$S3$ is the subtraction of the RTT of route $h1$–$h3$ and the RTT of route $h1$–$h2$. By carefully selecting destinations of probe packets and locations deploying the servers, the latency of all the routes of a network can be monitored using a minimal number of probe packets without causing any potential measurement failures. Several mechanisms have been proposed to determine the locations of the servers, to generate the probe packets and to determine their destinations. While these approaches may provide accurate and continuous latency measurements of the latency of all the links or routes of an SDN, the dedicated servers that inject probe packets and collect measurement results do contribute extra capital and operation costs for latency monitoring. On the other hand, the SDN control plane can work as a helper to determine the destination of probe packets and install flow entries for probe packets. However, methods that take advantage of the control plane to completely avoid the use of dedicated servers while efficiently, accurately, and continuously monitoring the latency of all the routes of an SDN have not been fully explored in the current research.

To fill the gap, we propose a novel latency monitoring method called LLDP-looping for low-latency SDNs. Without the need for any dedicated server, LLDP-looping uses the control plane to repeatedly inject LLDP packets into switches. Since LLDP is used for topology discovery, LLDP-looping combines latency monitoring with topology discovery. Two types of LLDP packets are used: LLDP packets with the extra type-length-value (TLV) structure are used as probe packets and injected into some selected switches for both latency monitoring and topology discovery, and normal LLDP packets without the extra TLV are injected into the other switches for topology discovery only. To determine the switches into which the control plane should inject probe packets, LLDP-looping views a network as an undirected graph and formulates a vertex cover problem (VCP) [21] to find a minimum vertex cover of the graph. This cover consists of the switches that need to receive probe packets from the control plane. LLDP-looping injects probe packets into these switches, forcing each packet to loop over the link three times, calculates the RTTs of all the links at the switches, and sends them back to the controllers for latency monitoring as well as topology discovery, as shown in Fig. 1(c).

Timestamping LLDP packets as probe packets and injecting them repeatedly into the SDN via the control plane allows the RTTs of all the network links to be continuously tracked without deploying any dedicated server, metering any end host, or installing any dedicated flow entry into switches for latency monitoring. LLDP-looping completely avoids measurement failures that could occur in the current approaches that use timestamped data packets as probe packets, and provides a very high accuracy in latency monitoring by determining the RTT of links at switches in a manner that overcomes almost all the major factors that impact the measurement accuracy in the currently proposed latency monitoring methods

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

for SDNs. The formulated VCP minimizes the overhead of LLDP-looping in both control and data planes, and the proposed greedy algorithm that always picks the switches near to the network edge can quickly find an optimal solution of the VCP over tree-based topologies. By topology discovery with latency monitoring capability, LLDP-looping not only reduces the extra workload added on the network while keeping the existing logic in topology discovery module almost unchanged, but also minimizes the modification to the software of both controllers and switches so that LLDP-looping can be easily applied over SDNs.

LLDP-looping has been prototyped on Nox C controllers [22] and Open vSwitch switches [23], and evaluations show that it can provide an accuracy of better than 90% (compared to latency measured using a Ping utility) in network latency measurements over a large-scaled network with link latency as small as 0.05 ms. The greedy algorithm that we propose can always minimize LLDP-looping's overhead in both the control and data planes over tree-based topologies in much lower computation complexity than an exhaustive search [21]. To the best of our knowledge, the proposed LLDP-looping is the only latency measurement method with a very low resource usage and high accuracy, which can continuously monitor the latency of all the links of an SDN, even for links with a very low latency, without deploying any dedicated infrastructure.

Although this paper presents an extension of the conference paper [14] and follows the same strategy where the timestamped LLDP packets are repeatedly injected by the control plane as probe packets, it proposes a new method to select switches for packet injections and determine link latency at these switches, which is lacking in the previous work [14]. The major contributions of this paper are threefold: 1) a highly accurate latency monitoring method is proposed for low-latency SDNs, which does not require deployment of dedicated network infrastructure or installing flow entries dedicated for latency monitoring; 2) a VCP is formulated to minimize the overhead of the proposed latency monitoring method; and 3) a greedy algorithm is proposed to compute an optimal solution of the VCP over tree-based network topologies.

The rest of this paper is organized as follows. Related work and major issues are summarized in Section II. LLDP-looping is proposed in Section III, prototyped in Section IV, and evaluated in Section V. Conclusions are drawn in Section VI.

## II. Related Work and Major Issues

### A. Latency Monitoring Methods Over IP Networks

In [4]–[6], the methods for active latency measurements over conventional IP networks were proposed. These approaches inject probe packets into the network and rely on geographically distributed beacons to perform trace routes in large-scaled networks, while our LLDP-looping works over SDNs and no distributed beacons are needed. IPMON [7] and COLATE [8] are the methods for passive latency measurements over conventional IP networks. They need a global clock to synchronize the timing of different network devices, and a latency monitoring unit at each network device is installed to capture sample packets or timestamp the caught packets. Approaches based on the ITU-Y.1731 can work actively or passively without a global clock, but they all require the use of a dedicated infrastructure consisting of the MEPs. TIMELY [24] uses network interface cards to measure the RTT of a route. However, it does not provide a mechanism to efficiently and continuously monitor the latency of all the links over the entire network. Scmon [20] uses a single monitoring point to inject probe packets into the network such that the link latency of the whole network can be monitored. It computes the latency of a link by subtracting the latency of two routes to reduce the overhead added by latency monitoring. However, its effectiveness has not been demonstrated over data center networks. In contrast, the proposed LLDP-looping can use the controller that has been deployed over an SDN as the monitoring point. It minimizes overhead by carefully selecting the switches where the probe packets need to be injected, and its effectiveness over data center networks has been demonstrated in this paper.

### B. Latency Monitoring Methods Over SDNs

Yassine *et al.* [25] have reviewed the state of the arts of latency measurements for SDNs. The approaches proposed by Adrichem *et al.* [12] and Phemius and Bouet [13] directly implement active latency monitoring based on (1). The approach in [13] achieves high measurement accuracy by finding a calibration constant, but has only been shown to work in a static network with two switches under a light workload. The approach in [12] targets an SDN with link latency greater than 1 ms for high measurement accuracy. In our previous work [14], we proposed an approach based on (1). With the aid of a linear calibration function, this approach can achieve a latency measurement accuracy of better than 80% over low latency and static SDNs. Time to live (TTL)-looping [15] provides high measurement accuracy over low-latency networks but this method is not directly based on (1). Unlike the above-mentioned methods that rely on controllers to repeatedly inject probe packets, software-defined latency monitoring [11] is based on (1) but relies on flow entry timeout messages to trigger a route latency measurement. On the other hand, GRAMI [17] uses dedicated servers that are distributed over the network to inject ICMP packets as probe packets.

Many potential issues exist in the above-mentioned approaches when they are used to monitor the latency of all the routes of an SDN continuously. Our previous work [14] identifies that timestamping data packets as probe packets, as in the measurement methods proposed in [12] and [13], can suffer from potential measurement failures, because current OpenFlow protocols lack mechanisms to differentiate the timestamped data packets from the nontimestamped ones. A flow entry forwarding a data packet to its intended destination will route a timestamped one to the same destination and incur a measurement failure, while forcing timestamped data packets to go back to the controller at each destination switch will send all the nontimestamped ones to the controller as well and cause a routing failure. Although ICMP packets can be

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

used as probe packets to avoid these potential measurement failures, ICMP-based methods such as GRAMI or Ping-based utilities have to rely on dedicated servers or user hosts, causing extra capital and operation costs. Also, ICMP-based methods need to install flow entries dedicated to latency monitoring in switches, which could lead to shortages of flow table space in switches. To address these issues, in our previous work [14], we proposed to timestamp LLDP packets as probe packets. Since LLDP is an open link layer protocol among switches and used by network devices to broadcast their identities and capabilities, many current SDN controllers implement a process that periodically injects LLDP packets to and receives LLDP packets from its supervised switches for network topology discovery [22], [26]. Since this process does not involve any application running at the hosts and no host will send LLDP packets to other hosts in the network, there is no need to set up flow entries for LLDP packets in switches. LLDP packets are guaranteed to be sent back to the controllers. By using timestamped LLDP packets as probe packets [14], we completely avoid the needs to utilize user hosts, deploy dedicated servers, or install dedicated flow entries in switches for latency monitoring, and thus eliminate the causes of potential measurement failures described earlier. Therefore, the LLDP-looping method proposed in this paper follows the same strategy.

Another issue in the above-mentioned approaches is their inability to achieve a high latency monitoring accuracy. Methods based on (1) can incur large systemic errors and measurement errors over low-latency SDNs. The systemic errors are mainly caused by different channels of a switch used in forwarding probe packets and data packets because these methods use probe packets injected by controllers via the control channel to measure the latency of data packets going through data channels. An OpenFlow switch has a control channel and a data channel. The data channel can work in the OpenFlow mode or the normal mode. The control channel is used to forward a flow injected by controllers or a flow sent by another switch or a host when the flow entry of the flow has not been installed in the flow tables. The OpenFlow mode of the data channel is used to forward a flow sent by another switch or a host when the flow entry of the flow has been installed in the flow table but not cached in the application-specific integrated circuit (ASIC), and the normal mode is used to forward a flow sent by another switch or a host when the flow entry of the flow has been cached in the ASIC. Since probe packets are injected by controllers while data packets are sent by another switch or a host, probe packets are always forwarded over the control channel of switches while data packets are forwarded by the switches using the normal mode most of the time when the flow entries have been installed. This difference incurred a systemic error of 1400% over our test bed (connected by 1-Gbit/s Ethernet switches) by comparing the RTT of different ICMP packets of Ping, where the first ICMP packet was forwarded using the OpenFlow mode due to no matching flow entry existing in the switch fabrics, and the following ICMP packets were forwarded by switches using the normal mode since the flow entries had been loaded into the switch fabrics by the first ICMP packet. The RTT of the first ICMP packet was about 0.6 ms and the RTT of following ICMP packets was about 0.04 ms over our test bed. This huge systemic error makes it difficult for methods based on (1) to achieve a high accuracy over SDNs with low link latencies without calibration.

The measurement errors of the above-mentioned approaches based on (1) are mainly incurred by the fluctuation of the CPU clock speed of the controller. In our experiment, we found this fluctuation could cause jitters of hundreds of milliseconds in measuring $t_{\text{src-to-dst}}$ using (1), while the real $t_{\text{src-to-dst}}$ measured by Ping was 0.02 ms with 10-$\mu$s jitter over our test bed under a light workload. The jitter was much larger than the real link latency, indicating that the measurement error incurred by the fluctuation was much larger than the real link latency. Since this fluctuation is highly dependent on the CPUs of controllers and tends to be random, it is difficult to reduce the measurement errors by calibration over low-latency SDNs. This is why the calibration method developed in [13] cannot be generally applied in any other SDN. Also, increasing the scale of an SDN increases the measurement errors because having more switches in an SDN increases the overhead of the controllers, leading to a longer delay in processing a probe packet or an OpenFlow message at the controllers [14]. This longer delay increases $t_{\text{measured}}$, $t_{\text{con-to-src}}$, $t_{\text{dst-to-con}}$, and $t_{\text{src-to-dst}}$ measured by approaches based on (1), while the real $t_{\text{src-to-dst}}$ measured by Ping remains almost unchanged due to the unchanged switch state. Regardless the linear calibration function proposed in [14] based on the averaged $t_{\text{measured}}$ accommodates the change of network scales, the averaged $t_{\text{measured}}$ hides the real performance change of SDNs, making the calibration ineffective over dynamic SDNs. TTL-looping is based on $t_{\text{src-to-dst}} = t_{\text{measured}}/\text{looptimes}$ rather than (1). While it does not need calibration, it nevertheless adds a huge overhead in the data plane to achieve higher measurement accuracy.

Since the systemic and measurement errors of methods based on (1) are extremely hard to mitigate over low-latency SDNs, and using controllers to determine the latency of SDNs is the root of these errors, the LLDP-looping method proposed in this paper uses the data plane instead of the control plane to determine the latency and thereby improves the measurement accuracy over low-latency SDNs. Similar to LLDP-looping, GRAMI determines the network latency over the data plane, but it does not demonstrate a high measurement accuracy over low-latency SDNs while our LLDP-looping does.

## C. Approaches to Reduce Overhead of Latency Monitoring

Minimizing the overhead of latency measurements is crucial for an active latency monitoring method to continuously monitor the latency of all links or routes of a network without affecting the measurement metrics and biasing the results of measurements. Many latency measurement approaches for SDNs proposed in the literature do not provide any mechanism to reduce such an overhead [12], [13], [15]. Given a network with $n$ links or routes, using these approaches to monitor the latency of every link or route of this network will cause the control plane to inject $n$ probe packets per measurement

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

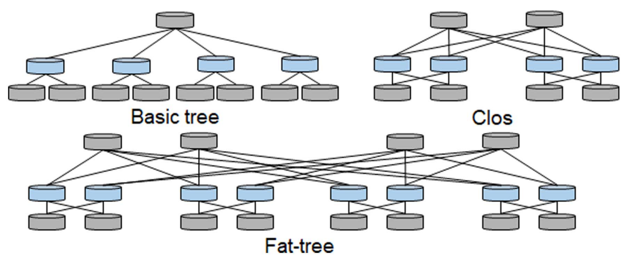

Fig. 2. Three tree-based topologies.

TABLE I
SWITHES AND LINKS IN THE THREE TREE-BASED TOPOLOGIES

| Topology | Core | Aggregation | Leaf | Links |
|---|---|---|---|---|
| basic tree | 1 | $m$ | $m(m-1)$ | $m^2$ |
| fat free | $m^2/4$ | $m^2/2$ | $m^2/2$ | $m^3/2$ |
| clos | $m/2$ | $m$ | $m$ | $m^2$ |

TABLE II
OVERHEAD OF CURRENT APPROACHES

| Ports per switch | Edge switches | Scale | Packet injected or returned | Bandwidth usage |
|---|---|---|---|---|
| 24 | 288 | medium | 6912 | 0.13% |
| 48 | 1152 | large | 55296 | 1.06% |
| 72 | 2592 | large | 186624 | 3.58% |
| 96 | 4608 | massive | 442368 | 8.49% |

iteration, because the latency of each link or route has to be measured by one probe packet. This increases the overhead of the control plane and consumes the bandwidth of the control channel of the control plane.

We calculate the bandwidth of the control channel consumed by these approaches over a full fat-tree topology, since it has short latency and is widely used by current data centers to achieve lower latency [27]. Three typical full tree-based topologies are shown in Fig. 2, and their basic information is listed in Table I. We let the control plane timestamp Ethernet frames (24 bytes) as probe packets and inject one probe packet per link per second, and the full bandwidth of the control channel is 1 Gb/s. As listed in Table II, the bandwidth usage of the control channel can increase from 0.13% to 3.58% and 8.49%, when the number of edge switches increases from 288 to 2592 and 4068, respectively. Since full fat-tree networks consisting of 288, 2592, and 4608 edge switches are the representative of the networks of medium, large, and massive data centers [28], respectively, increasing the number of switches in a large-scaled SDN can significantly increase the bandwidth of the control channel consumed by these approaches in monitoring the latency of the entire network. This is especially troublesome when some other bandwidth-consuming control plane applications such as topology discovery have to run simultaneously with latency monitoring over a dynamical network.

Many effective latency monitoring methods over conventional IP networks proposed in the literature provide mechanisms to reduce the overhead of latency monitoring. These mechanisms are often based on the idea that estimating the RTT by subtraction of two RTTs can monitor the latency

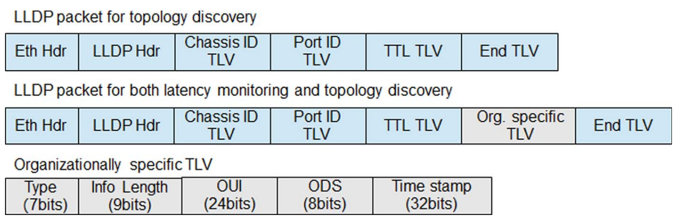

Fig. 3. LLDP packet structure.

of more routes with fewer probe packets injected. However, these mechanisms have to address the issue that the path of probe packets cannot be controlled. Breitbart *et al.* [18] and Aubry *et al.* [20] addressed this issue by implementing source routing over a conventional IP network. GRAMI [17] addressed this issue by installing flow entries at switches along the path of probe packets over SDNs. The proposed LLDP-looping addresses this issue by hardcoding the behavior of switches in processing the received LLDP packets.

Subtraction of two RTTs can reduce the number of probe packets for latency monitoring, but it creates an optimization problem regarding the best locations of servers to inject probe packets into the network such that the number of probe packets used to monitor the latency of all the links or routes of a network can be minimized. This problem can be formulated as a well-known facility location problem (FLP) as proposed in [17] and [18] and solved by some heuristic algorithms since an FLP is NP-hard. Since LLDP-looping injects LLDP packets to a set of switches and lets each LLDP packet flood a switch, the overhead minimization problem needs to be formulated differently, as a VCP instead of an FLP, to minimize the number of LLDP packets used for latency monitoring.

## III. LLDP-LOOPING

We propose an efficient and accurate latency monitoring method called the LLDP-looping to continuously monitor the latency of all links of low-latency SDNs. By keeping the track of the latency of all the links of an SDN and adding all the link latencies along a route, the route latency between arbitrary switches can be calculated. LLDP-looping combines latency monitoring with topology discovery by timestamping LLDP packets as probe packets. Two types of LLDP packets are used by LLDP-looping: those consisting of an extra TLV are used as probe packets for latency monitoring and topology discovery, while those without the extra TLV are used for topology discovery only. The extra TLV is an organizationally specific TLV (10 bytes) that stores the timestamp of a link, as shown in Fig. 3, where the organizationally unique identifier and organizationally defined subtype have to be carefully configured to avoid conflicting with existing organizationally specific TLVs. LLDP-looping lets the control plane inject probe packets to a selected set of switches but inject normal LLDP packets to the others. Each probe packet is forced to loop around a link for three times, and the RTT of the LLDP packet is calculated at the switch, as shown in Fig. 1(c).

### *A. Determination of Latency at Switches*

To facilitate discussion, we call the switch that receives an LLDP packet injected by the control plane as the link

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

source switch, and the switch that receives the LLDP packet forwarded by the link source switch the link destination switch.

A latency measurement of LLDP-looping starts with a controller injecting a probe packet with TTL $= 4$ at a link source switch, which is processed in the control channel function of the switch and flooded to all the active ports of the switch so that it is forwarded to all the link destination switches downstream of these ports. Consider one of these link destination switches. As this is the first time that the injected probe packet arrives at this link destination switch, the switch uses its datapath function to decrement the TTL of the arriving probe packet, timestamp the current time to the extra TLV, and sent the packet back to the link source switch from which the injected probe packet has just been forwarded. When the probe packet arrives at its link source switch again, the link source switch uses its datapath function to process the received probe packet by decrementing TTL and returning it back to the link destination switch that has just forwarded the probe packet. Finally, when the probe packet comes back to the link destination switch for the second time, the switch uses its datapath function to retrieve the timestamp from the extra TLV, compute the RTT of the link, timestamp the RTT in the extra TLV, and send it back to the controller via the control channel.

Since each switch can be a link source switch or a link destination switch when handling different probe packets, it uses the TTL of a probe packet to determine its role. TTL $= 4$ in an arriving probe packet indicates that the switch is a link source switch and this packet has been injected by the control plane. TTL $= 3$ in an arriving probe packet indicates to the switch that is a link destination switch and should loop the probe packet back to the link source switch. TTL $= 2$ in an arriving probe packet indicates that the switch is a link source switch that has previously received this packet and should loop the packet back to the link destination switch. TTL $= 1$ in the arriving probe packet indicates that the switch is a link destination switch that is receiving this packet for the second time and should compute and put the RTT in the TLV and return the probe packet back to the controller. By checking the TTL of a probe packet, LLDP-looping dynamically determines the role of a switch with respect to the received probe packet, generalizes the behavior of each switch in processing the probe packets, and minimizes the modification of a switch software for latency monitoring.

The RTT of a probe packet can be determined at its link source switch or destination switch. LLDP-looping chooses to use the link destination switch rather than the link source switch to determine the RTT of the probe packet because: 1) it leaves the normal topology discovery procedure in both switches and controllers unchanged, whereby a normal LLDP packet is also forwarded to the controller by the link destination switch and 2) it confines the modification of the switch software to the datapath functions because a link destination switch only involves its datapath while a link source switch has to involve both its control channel and datapath to process each received probe packet (because the probe packets received by a link destination switch are always sent by a link source switch, while the probe packets received by a link source switch can be sent by either a link destination switch or a controller).

Since the link latency is determined at switches, the fluctuation of controller's CPU clock speed and the change of the network scale that significantly impact the measurement accuracy over low-latency SDNs are completely avoided, and the measurement accuracy is significantly improved. LLDP-looping also completely avoids potential measurement failures in current SDN latency monitoring methods that timestamp data packets as probe packets while enabling continuous latency monitoring without affecting the forwarding of data packets. It simplifies and generalizes the modification of software at both controllers and switches, and can be readily applied over SDNs.

## *B. Overhead Minimization*

LLDP-looping monitors the latency of all network links by using the control plane to inject probe packets to each switch of the network. This process increases the overhead of the control plane due to the CPU time and bandwidth of the control channel consumed by the continuous injection of probe packets. This overhead can be reduced because injection of probe packets to all the switches (each acting both as a source and a destination of some link) causes the latency of each full-duplex link to be measured twice. Therefore, LLDP-looping should carefully select a set of switches into which the control plane injects the probe packets such that each link of the network has at least one probe packet traversing it in either direction, and both the number of probe packets generated for latency monitoring and the number of switches used to flood probe packets are minimized simultaneously.

Therefore, we consider a network as an undirected graph $G = (V, E)$, where $V$ and $E$ are the set of switches and full-duplex links of the network, respectively, and $S \subseteq V$ is the set of switches selected for probe packet injection. The degree of each $v \in V$ (the number of neighbors of the node $v$) is given by $d(v)$. In a measurement iteration, since the control plane injects a probe packet to each of the selected switches, which then floods the received packet to its active ports, the number of probe packets injected by the control plane equals the number of selected switches, and the number of probe packets flooded by a selected switch equals the number of neighbors of the switch. Therefore, we formulate an optimization problem with objectives to simultaneously minimize the number of switches in $S$(OBJ2) and the total degrees of switches of $S$(OBJ1) such that at least one probe packet traverses each link of the network (CON1) to minimize the overhead of LLDP-looping on both the control and data planes.

However, OBJ2 can be discarded because LLDP-looping combines latency monitoring with topology discovery such that a switch will receive LLDP packets from the control plane regardless of whether it is selected to receive probe packets for latency monitoring or not. If a switch is not selected for latency monitoring, it receives LLDP packets injected by the control plane for topology discovery; otherwise, it receives LLDP packets with an extra TLV for both latency monitoring

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

and topology discovery. Therefore, minimizing the number of switches selected for probe packet injection does not reduce the number of LLDP packets injected by the control plane, and hence, the formulated problem is simplified to a typical VCP that finds the minimum vertex cover to optimize the OBJ1 under the constraint of CON1. OBJ1 and CON1 are formulated in the following equations, respectively:

$$\text{OBJ1: } \text{Min} \sum_{v \in S} d(v) \tag{2}$$

$$\text{CON1: } \forall \{u, v\} \in E : v \in S \vee u \in S. \tag{3}$$

A VCP is NP-complete and cannot be solved in polynomial time. The fastest exhaustive algorithm [21] that can find the minimum cover of a VCP has a computation complexity of $O(1.2738^K + K|V|)$, where $K$ represents the maximal size of the vertex cover. A typical way to efficiently find an approximately optimal solution of a VCP is to use greedy techniques. However, greedy techniques do not guarantee a minimal cover. Considering low-latency networks of data centers that often have tree-based topologies, as shown in Fig. 2, and a VCP over a tree-based topology can be solved in polynomial time, while a typical greedy algorithm that always picks the switches with the most number of neighbors in the network cannot yield a minimum cover of our VCP [29]; therefore, we propose a novel greedy algorithm that can quickly find the minimum cover of our VCP over tree-based network topologies.

Particularly, our greedy algorithm maintains an adjacency matrix $A$ $(n*n)$ for a network with $n$ nodes (switches). We let $a_{ii}$ be the element located in row $i$ and column $i$ of $A$ and representing the degree of node $i$, and $a_{ij}$ $(i \neq j)$ be the element of $A$ located in row $i$ and column $j$ with the possible value of 1 or 0, respectively, representing the existence of an edge between nodes $i$ and $j$ or not. Unlike the greedy algorithm that always picks a vertex with the largest degree from the graph [29], our algorithm starts by picking a vertex $v$ with the lowest degree, then finds all the neighbors of $v$ and adds the neighbors to the set $S$. Then, the algorithm removes $v$ and its neighbors and all the edges associated with them from the graph by computing a new adjacent matrix $A^k$. Each element of $A^k$ remains unchanged relative to $A$ except that: 1) the degrees of $v$ and its neighbors are set to 0 in $A^k$, representing the removal of $v$ and its neighbors from the graph; 2) the elements representing the degree of the neighbors of neighbors of $v$ in $A^k$ are decreased by 1; and 3) the elements representing the edges that have an incident vertex with $v$ or any of its neighbors are set to 0 in $A^k$, indicating that the edges associated with $v$ or any of its neighbor are removed from the graph. Finally, the algorithm evaluates $A^k$. If each element of $A^k$ is 0, the algorithm returns the set $S$ as the minimum vertex cover; otherwise, it picks another vertex with the lowest degree from the remaining graph, finds the neighbors of that vertex and adds them to $S$, and updates $A^k$ until each element of $A^k$ is 0. If there are more than one vertex in the remaining graph that has the same lowest degree, our algorithm computes the degree difference of each vertex ($\text{dif}_{\text{low}}$) using the degree of the vertex in $A$ minus the one in $A^k$, and picks the one with the smaller $\text{dif}_{\text{low}}$ to minimize

**Algorithm 1** Pseudocode of Our Greedy Algorithm

```
1: INPUT: The $n * n$'s adjacent matrix $A$
2: OUTPUT: The set S
3: $S \leftarrow \{\}$, $A^k \leftarrow A$
4: while $A^k \neq 0$ do
5:     $a_{low} \leftarrow n$, $dif_{low} \leftarrow n$
6:     for $i = 1; i++; i \leq n+1$ do
7:         if $a_{low} > a^k_{ii}$ then
8:             $a_{low} \leftarrow a^k_{ii}$, $dif_{low} \leftarrow a_{ii} - a^k_{ii}$, $v \leftarrow i$
9:         end if
10:        if $a_{low} == a^k_{ii}$ and $dif_{low} > a_{ii} - a^k_{ii}$ then
11:            $dif_{low} \leftarrow a_{ii} - a^k_{ii}$, $v \leftarrow i$
12:        end if
13:    end for
14:    for $i = 1; i++; i \leq n+1$ do
15:        if $a^k_{iv} == 1$ and $i \neq v$ then
16:            $S \cup \{i\}$
17:        end if
18:    end for
19:    for vertex v and each element in S do
20:        $u \leftarrow element$, $a^k_{uu} \leftarrow 0$
21:        for $i = 1; i++; i \leq n+1$ do
22:            if $i \neq u$ and $a^k_{iu} == 1$ then
23:                $a^k_{iu} \leftarrow 0$, $a^k_{ui} \leftarrow 0$, $a^k_{ii} \leftarrow a^k_{ii} - 1$
24:            end if
25:        end for
26:    end for
27: end while
```

the number of duplicated edges incurred by flooding the picked node and the number of probe packets generated by all the picked nodes. The detail of our greedy algorithm is shown in Algorithm 1.

Given the same $K$ defined in the second last paragraph, which presents the maximum number of switches that can be selected. The computation complexity of the proposed algorithm is $O(K|V|)$ $(|V| = n)$, which is lower than $O(1.2738^K + K|V|)$, the computation complexity of the fastest exhaustive algorithm proposed by [21]. Although the proposed algorithm cannot guarantee to find a minimum vertex cover for an arbitrary network topology, it can always pick the aggregation switches of the tree-based network topologies as shown in blue in Fig. 2. Since the tree-based network topologies are widely used in current data centers, this algorithm minimizes the number of probe packets used by LLDP-looping to monitor the latency of all the links of many low-latency networks. In a dynamically changing network environment where the network topology needs to be continuously discovered to accommodate the possible changes in the states of network nodes and links, the proposed greedy algorithm allows the set of switches for probe packet injection to be selected automatically in the presence of topology changes. Obviously, the optimal set of switches can be selected offline for any given network topology and configured manually for the network, but this approach would have difficulty adapting to dynamic network conditions in a timely manner.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

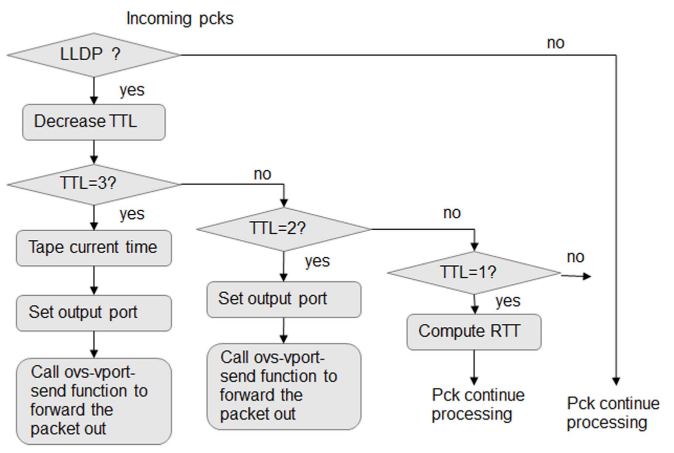

Fig. 4. Flowchart of our LLDP-looping.

## IV. Prototyping

We prototype LLDP-looping on a Nox C controller and Openvswitch switches. Our prototype implementation only adds 36 lines of codes in the *ovs–vport–receive* function of the *vport.c* file of Openvswitch. For each received packet, as shown in Fig. 4, our prototype first checks the type of the packet and decreases the TTL of an LLDP packet, and then checks the value of the TTL. If TTL = 3, indicating the switch is a link destination switch that receives the LLDP packet for the first time, the switch timestamps the current time to the LLDP packet, sets the output port number as the in-port number of the packet, and calls the *ovs–vport–send* function to directly forward the timestamped LLDP packet back to the link source switch without further processing the packet (since an LLDP packet injected by the control plane to a switch does not go through the *ovs–vport–receive* function of the switch, TTL = 3 means a link destination switch rather than a link source switch). If TTL = 2, implying that the switch is a link source switch and the current LLDP packet is just looped back from a link destination switch, the switch sets the output port number as the in-port number of the received packet, and calls the *ovs–vport–send* function to directly forward the LLDP packet back to the link destination switch without further processing the packet. If TTL = 1, meaning that the switch is a link destination switch that is receiving the looped LLDP packet for the second time, the switch calculates the RTT of that LLDP packet, writes the RTT to the appropriate TLV field of the LLDP packet, and processes the LLDP packet normally using the *ovs–vport–receive* function. In all other cases, the switch processes the LLDP packet normally. For any LLDP packet that needs to be processed normally at a switch, since there is no matching flow entry for it in the switch, the LLDP packet will be finally forwarded to the control plane. Since the LLDP packets used for topology discovery only have TTL initialized to 1, when the LLDP packets arrive at their link destination switches, the value of TTL will decrement to 0 and the switches will forward the packets to the controllers directly. In this way, our prototype only slightly modifies the *ovs–vport–receive* function of the *vport.c* file to enable latency monitoring and topology discovery simultaneously over an SDN, without affecting any other part of an Openvswitch.

Since the existing Nox C controller consists of a discovery component that periodically injects LLDP packets to switches for network topology discovery, we slightly modify this module to maintain two types of LLDP packets, one that has an organizationally specific TLV and the other that has not. We set the TTL of the LLDP packets with organizationally specific TLV to 4 so that the packets can monitor link latency as well as discover topology, while configuring the TTL of the LLDP packets without organizationally specific TLV to 1 for topology discovery only. We implement our greedy algorithm as a function of the discovery component, and fork a new thread to determine the switches that should receive the probe packets from the control plane periodically. In each iteration of LLDP-looping, the control plane injects LLDP packets with organizationally specific TLV to those switches selected by our algorithm, and injects LLDP packets without organizationally specific TLV to the other switches. The implementation makes sure that all the LLDP packets are returned to the control plane to be correctly decoded. In this way, LLDP-looping can be implemented with minimal modifications to the software of both controllers and switches. The modifications are also generalized and can be readily implemented in any other controllers and switches of an SDN.

Although current SDNs often deploy physical switches, which software or firmware can only be updated by their manufacturers, increasingly academia and companies are constructing their SDNs with customizable soft switched to meet their special requirements for network security, QoS, and innovations as the technologies of virtual switches and NetFPGAs mature. A data center with thousands of network devices or even more may choose physical switches for deployment, considering network performance and maintenance. However, as a significant customer, the architect or operator of a data center can specify additional functions for the software of their network devices, which vendors are required to customize to meet their special requirements.

## V. Evaluations

We ran our LLDP-looping prototype over SDNs emulated using Mininet [30] and investigated the major factors affecting measurement errors and measurement accuracy before and after calibrations. We estimated the workload generated by our LLDP-looping and measured the flow setup rate of the control plane to show how our LLDP-looping does not impact the performance and scalability of the control plane. We compared our LLDP-looping to the major approaches in SDN latency monitoring proposed in this paper and analyzed how LLDP-looping can extend the same high monitoring accuracy to physical SDNs.

### *A. Factors Affecting Measurement Error*

Since LLDP-looping measures the RTT of an LLDP packet in the data plane, the performance fluctuations of controllers caused by fluctuations of their CPU clock speed do not affect the measured latency. The fluctuations in CPU clock speeds of the switches still lead to jitters in latency measurements, which reflect the real fluctuations in network latency that a

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

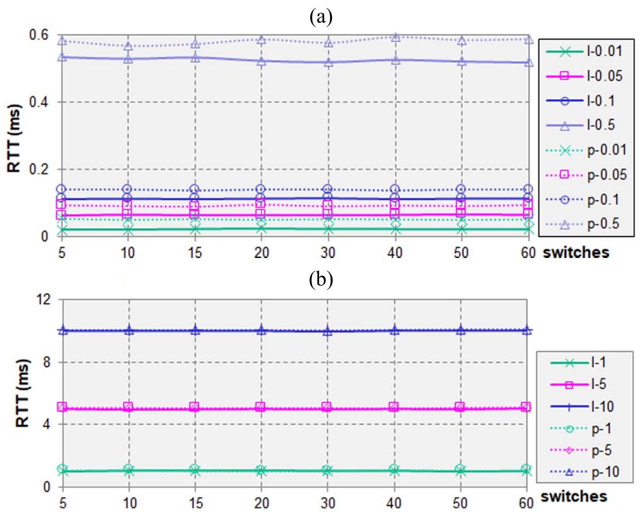

Fig. 5. RTT measured by LLDP-looping (l-) and ping (p-) under various network scale. (a) RTT with link latency set to 0.01–0.5 ms. (b) RTT with link latency set to 1–10 ms.

data packet would experience in a network. Results show that changes in the network scale, the LLDP packet injecting frequency, and the workload do not impact the measurement error of LLDP-looping either.

Fig. 5(a) and (b) shows the comparison of the RTTs measured by LLDP-looping and the Ping utility over a linear network with 5, 10, 15, 20, 30, 40, 50, and 60 switches with the link delays set to 0.01, 0.05, 0.1, 0.5, 1, 5, and 10 ms. The linear networks were emulated by Mininet. Each switch was connected to one host. Since the network topology does not affect the accuracy of LLDP-looping, we emulated linear topologies to facilitate construction of networks with larger scales in the computer emulation. Since changing the number of switches in the network does not change the overhead of switches, the RTTs measured by LLDP-looping and Ping remain almost unchanged as the number of switches in the emulated SDN is varied, indicating that the change of the network scale does not affect the measurement accuracy of LLDP-looping and Ping.

Furthermore, Fig. 6(a) shows the link latency measured by LLDP-looping and Ping over an emulated linear SDN with 10 switches with each link delay configured to 0.1 ms, and the packet injecting frequencies configured to 1, 2, 5, 10, 20, 50, and 100 per second. The results show that the RTTs measured by LLDP-looping do not change with the LLDP injection frequency (with roughly 10-$\mu$s jitter caused by the switch performance fluctuation), because changing the frequency of LLDP packet injection does not really change the overhead of the switches. Fig. 6(b) shows the link latency measured by LLDP-looping and Ping over an emulated linear SDN with five switches and links configured to 10 Mbits/s with 0.1-ms link delay. We ran Iperf [31] to generate workloads of 0, 4, and 8 Mbits/s for 120 s. The results show that the RTTs measured by LLDP-looping and Ping follow the same pattern. Furthermore, as time progresses, the measured RTTs first increases until the output port queue becomes congested, and then decreases with some fluctuation until the output port

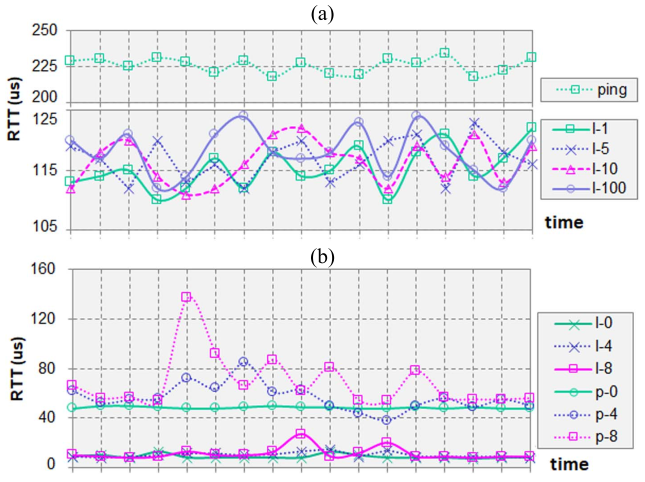

Fig. 6. RTT measured by LLDP-looping (l-) and ping (p-) under various (a) packet injection frequency and (b) workload.

queue becomes empty. This pattern represents a real change in network state when the workload is generated by Iperf, and suggests that LLDP-looping can reliably monitor the link latency under a light workload while the network state changes in real time.

## B. Measurement Accuracy

Since the RTTs measured by LLDP-looping and Ping are both greater than the link latency configured, and the RTTs measured by LLDP-looping were less than the ones measured by Ping [as shown in Fig. 5(a) and (b)] due to the delay between hosts and switches included in the latter, LLDP-looping achieved higher accuracy in latency monitoring than Ping when the configured link latency was used as a baseline. However, Ping is the traditional approach used by network administrators to monitor network latency. Therefore, we used the RTT measured by Ping as a baseline to calculate the measurement error of the experiment [as shown in Fig. 5(a) and (b)]. As shown in Fig. 7(a), the measurement error was around −58%, −30%, −18%, −8%, −3%, −0.4%, and −0.1% when a link delay was configured to 0.01, 0.05, 0.1, 0.5, 1, 5, and 10 ms, respectively. The measurement error does not increase as the network scale grows. If an acceptable measurement error compared to the baseline is 10%, our proposed LLDP-looping cannot provide acceptable measurement accuracy in a network where the baseline is less than 1 ms, which is the RTT measured by Ping when the link delay is set to be less than 1 ms.

Since the delay between hosts and switches contributed the major part of measurement errors, we calculate this delay using the RTT measured by Ping minus the RTT measured by LLDP-looping ($C_{\text{cali}} = \text{RTT}_{\text{ping}} - \text{RTT}_{\text{LLDP-looping}}$) to calibrate the latency measured by LLDP-looping ($\text{RTT}_{\text{LLDP-looping-cali}} = \text{RTT}_{\text{LLDP-looping}} + C_{\text{cali}}$). Theoretically, this delay is constant when the overhead of each switch and the host in the network remains unchanged. This is why, in our experiments, the difference between the RTT measured by Ping and the RTT measured by LLDP-looping remains almost the same when the link delay was configured to a particular value, as shown

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

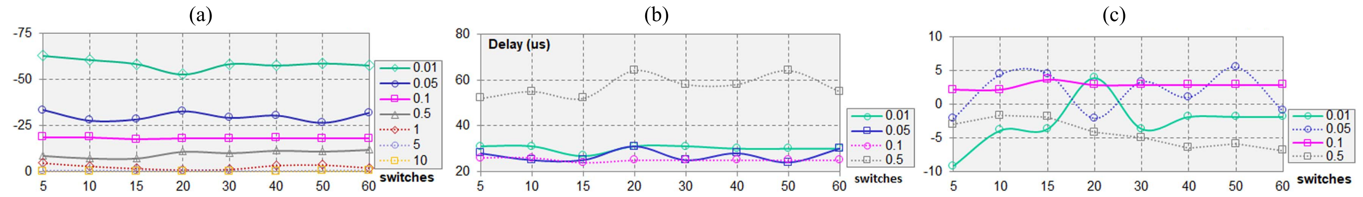

Fig. 7. Link latency measurement error comparing to the half RTT measured by Ping. The labels represent the test results with various link delay. (a) Measurement error before calibration (%). (b) Delay between hosts and switches. (c) Measurement error after calibration (%).

in Fig. 7(b). However, this difference slightly changed when the configured link delay is varied, which is most probably caused by the different overhead added by the link delay configuration process in Mininet when the link delay was set to various values, and it cannot happen in a physical network. Based on this consideration, we apply a calibration constant of 29 $\mu$s that is the difference of the RTTs measured by LLDP-looping and Ping when the link delay is configured to 0.01 and 0.05 ms. The measurement error after calibration is shown in Fig. 7(c).

It is apparent that the measurement error could be reduced to less than 10% of the baseline after calibration when the link delay is set to be not less than 0.01 ms. It implies that when the RTT measured by Ping is not less than 0.05 ms (0.05 ms is the RTT measured by Ping when the link delay is set to 0.01 ms), our proposed approach can achieve accurate latency monitoring if the acceptable measurement error is 10%.

We also tested the real RTT of our test bed (two servers connected through an Ethernet switch with 1-Gb/s ports) using Ping. The RTT was found to be around 0.2 ms and greater than 0.05 ms obtained in our experiments. It suggests that our evaluation emulates an SDN with lower link latency than a low-latency network such as the local area network our lab, and the measurement accuracy of LLDP-looping estimated by the evaluation can apply to many low-latency networks.

### C. Evaluation of the Proposed Greedy Algorithm

We consider the three full tree-based topologies with $m$-port switches, as shown in Fig. 2, wherein the aggregation switches of each topology form the minimum vertex cover of the formulated VCP. We consider $m = 4$, 8, 16, 24, and 48. Table III gives the comparison of the total number of switches (a) and the total degrees of these switches (b) calculated by our proposed greedy algorithm (A1) to the ones computed by the exhaustive algorithm (A2) proposed in [21] and the greedy algorithm (A3) proposed in [29]. The results show that our greedy algorithm and the exhaustive algorithm can always find the aggregation switches of each topology, while the normal greedy algorithm has a worse case that finds a cover 1.5$\times$ of the minimum cover and the total degrees of the cover is 1.5$\times$ of the total degrees of the minimum cover over the fat tree and clos topologies, although the cover computed by the normal greedy algorithm over the basic tree topology is very close to the minimum cover computed by the other two algorithms. The computation complexity of our algorithm is $O(Km^2)$ using $O(K|V|)$ provided in Section III-B for all the three tree-based topologies, while the computation complexity

TABLE III
ALGORITHM COMPARISON OVER THE TREE-BASED TOPOLOGIES

| Top | m | a(A1&A2) | a(A3) | b(A1&A2) | b(A3) |
|---|---|---|---|---|---|
| basic | 4 | 4 | 4-5 | 16 | 16-20 |
| | 8 | 8 | 8-9 | 64 | 64-72 |
| | 16 | 16 | 16-17 | 256 | 256-272 |
| | 24 | 24 | 24-25 | 576 | 576-600 |
| | 48 | 48 | 48-49 | 2304 | 2304-2352 |
| fat | 4 | 8 | 8-12 | 32 | 32-48 |
| | 8 | 32 | 32-48 | 256 | 256-384 |
| | 16 | 128 | 128-192 | 2048 | 2048-3072 |
| | 24 | 288 | 288-432 | 6912 | 6912-10368 |
| | 48 | 1152 | 1152-1728 | 55296 | 55296-82944 |
| clos | 4 | 4 | 4-6 | 16 | 16-24 |
| | 8 | 8 | 8-12 | 64 | 64-96 |
| | 16 | 16 | 16-24 | 256 | 256-384 |
| | 24 | 24 | 24-36 | 576 | 576-864 |
| | 48 | 48 | 48-72 | 2304 | 2304-3456 |

of the exhaustive algorithm is $O(1.2738^K + Km^2)$ using $O(1.2738^K + K|V|)$ provided in [21].

Therefore, the more ports a switch has, the more aggregation switches of a full tree-based network can consist of, and the larger $K$ can be. Also, the growth of the network scale increases the computation complexity of our greedy algorithm and the exhaustive one but the complexity of the exhaustive one increases much faster than ours.

### D. Measurement Overhead on Control Plane

Since LLDP-looping combines latency monitoring with topology discovery, while topology discovery is one of the basic functions that a controller has to implement, LLDP-looping only adds a limited amount of extra overhead on the control plane. To evaluate it, we let LLDP-looping inject one LLDP packet per switch per second, and computed the extra number of LLDP packets and the extra number of bytes injected by LLDP-looping for topology discovery with latency monitoring capability, using the baselines calculated by running a normal topology discovery module for topology discovery. An LLDP packet without the extra TLV dedicated for latency monitoring is 34 bytes long, and the extra TLV used for timestamping RTT adds 10 bytes. LLDP-looping injects LLDP packets with the extra TLV to the aggregation switches of a tree-based topology for both latency monitoring and topology discovery, and injects normal packets to other switches for topology discovery only. As listed in Table IV, LLDP-looping does not inject any extra LLDP packets to the data plane for latency monitoring.

However, LLDP-looping consumes more bandwidth in the control channel of controllers, since each probe packet that is

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE IV
OVERHEAD ADDED BY LLDP-LOOPING OVER TREE-BASED TOPOLOGIES

| Topo-logy | Extra packets injected | Extra bytes injected | Extra packets per link | Extra bytes per link |
|---|---|---|---|---|
| basic | 0 | $10m$ | 2 | $10 + 2 * 44$ |
| fat | 0 | $10m^2/2$ | 2 | $10 + 2 * 44$ |
| clos | 0 | $10m$ | 2 | $10 + 2 * 44$ |

injected to the intended selected switches has an extra TLV that costs 10 more bytes. Suppose the control plane injects one packet per switch per second to an SDN with 1-Gb/s links. The calculated bandwidth consumed by LLDP-looping for latency monitoring and for topology discovery over the three tree-based topologies with 4-port, 8-port, 16-port, 24-port, 48-port, 72-port, and 96-port switches, respectively, are shown in Fig. 8(a) and (b). It is apparent that the more ports a switch have, the more aggregation switches the network could have, and the more probe packets the control plane needed to inject. The bandwidth of the control channel consumed by LLDP-looping for both latency monitoring and topology discovery is increased as the number of ports of a switch increases. Compared to the full bandwidth of the control channel (1 Gb/s), the bandwidth usage of LLDP-looping for latency monitoring over the basic tree and clos networks constructed by switches with up to 96 ports is negligible. Although LLDP-looping over a full fat-tree network can generate much larger overhead over the control channel than the other two types of tree-based networks, the bandwidth usage of LLDP-looping can be limited to be lower than 6.5% of the full bandwidth of the control channel when the fat-tree network is constructed with switches having up to 72 ports, as shown in Fig. 8(a).

It should be noted that a fat-tree network constructed by 72-port switches can have up to 6480 switches (5 $m^2/4$) and connect up to 93 312 hosts ($m^3/4$), which is representative of a network for a very large data center. If more hosts need to be connected into a fat-tree network, 96-port switches may have to be used. Using LLDP-looping over a fat-tree network constructed by 96-port switches with 1-Gb/s links for topology discovery can consume about 28% of the bandwidth of the control channel [as shown in Fig. 8(b)], although among them, only about 15.5% of the bandwidth is used for both latency monitoring and topology discovery [as shown in Fig. 8(a)] and about 3.5% of the bandwidth is dedicated for latency monitoring (because the proposed LLDP-looping combines latency monitoring with topology discovery, and for each 44 bytes injected, only 10 bytes is dedicated for latency monitoring). To reduce the bandwidth usage of LLDP-looping, such a fat-tree network may have to upgrade to 10-Gb/s links or use distributed control plane consisting of multiple controllers, each of which runs an LLDP-looping instance to monitor the link latency within a network partition. To monitor the latency of links between two network partitions, some modifications on the proposed LLDP-looping may have to be done.

To further evaluate the overhead of LLDP-looping in the control plane, we conducted an experiment to measure the flow

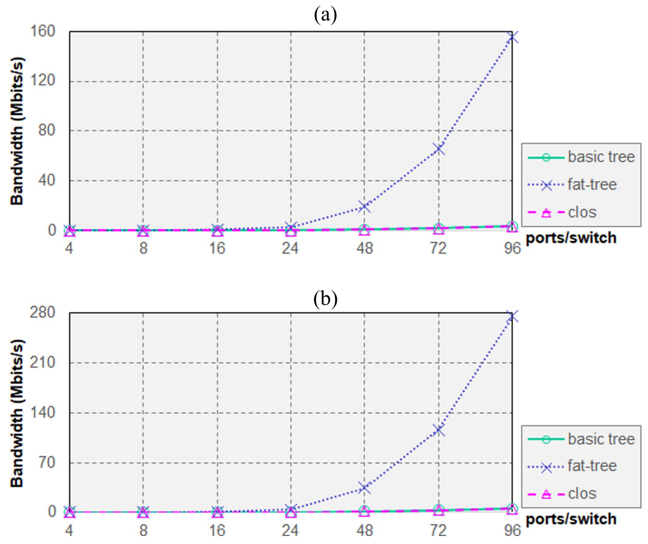

Fig. 8. Bandwidth of control channel consumed by LLDP-looping. (a) Bandwidth usage of LLDP-looping for latency monitoring. (b) Bandwidth usage of LLDP-looping for topology discovery.

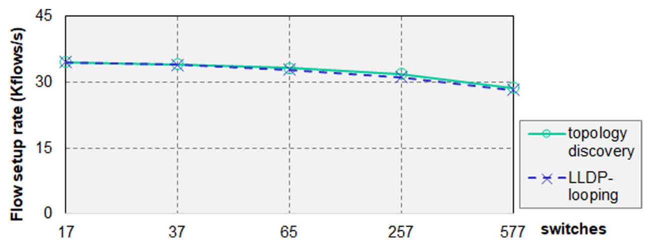

Fig. 9. Flow setup rate over basic tree networks with various switches.

setup rates of the Nox C controller under the normal topology discovery procedure and LLDP-looping. The flow setup rate of a controller is the number of flows the controller can set up per second. It is one of the key scalability metrics of SDNs, and increasing the overhead of the control plane will decrease the flow setup rate. We used a test bed with three computers, one of which emulates using Mininet a basic tree topology constructed by 4-port, 6-port, 8-port, 16-port, and 24-port switches, with the total number of switches of the network set to 17, 37, 65, 257, and 577. The second computer runs a Nox C controller to manage the SDN emulated by the Mininet, and the third computer runs Cbench [32] to measure the flow setup rates of the Nox C controller with normal topology discovery and with LLDP-looping. We chose the basic tree rather than the fat tree and clos for evaluation because a full basic tree topology can have a large network scale that can be emulated by a single computer. We set up the controller to inject 10 LLDP packets per switch per second to simulate a network that may have a larger scale and lower packet injection frequency. The results are shown in Fig. 9, where it is apparent that LLDP-looping does not decrease the flow setup rate of the networks, although the flow setup rate of the controller decreases as the scale of a network grows. Therefore, the extra bandwidth consumed by LLDP-looping dedicated for latency

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

monitoring does not affect the performance and scalability of the control plane.

However, the flow setup rate of a controller is decreased by running the topology discovery process, and this decrease increases as the number of switches in the network increases. Since the measured flow setup rate is indicative of the real flow setup rate that a controller can provide for data packets, this reduced flow setup rate impacts the performance and the scalability of the control plane. Depending on the computation capacity of a controller and how quickly a network requires its data packets to be forwarded, a large-scale network may have to use a distributed control plane consisting of multiple controllers. The proposed LLDP-looping can be run at each controller of the distributed control plane to monitor the latency and discover the topology of the network partition managed by a controller.

### *E. Measurement Overhead on Data Plane*

In LLDP-looping, each injected LLDP packet with the extra TLV to loop the targeted link three times (two times for latency monitoring and one time for topology discovery), LLDP-looping added $(10 + 2 * 44)$ more bytes on a link (10 more bytes for the extra TLV and two more loops for RTT measurement), as listed in Table IV. The extra bandwidth consumed by LLDP-looping over a link is 7.84 Kb/s when the latency of each link is updated 10× per second. This bandwidth usage over a 1-Gb/s link is negligible. Since LLDP-looping limits each link of the network to have exactly one probe packet looping over each in each measurement cycle, each link of the network has the same low overhead consumed for latency monitoring in LLDP-looping, while some links near the monitoring points in GRAMI or many other approaches based on RTT subtraction suffer from a larger workload because these links are shared by many routes and traversed by many probe packets.

### *F. Measurement Accuracy and Workload Comparisons*

We compared LLDP-looping with some currently proposed approaches, as listed in Table V. Opennetmon [12] and the approach in [13] directly implement (1), but the former targets a network with link latency greater than 1 ms and no latency monitoring accuracy is provided, and the latter can achieve measurement accuracy of up to 99% after calibration, but the calibration only works for a particular static network with only two switches. Our previous work presented in [14] achieved measurement accuracy better than 80% over a low-latency network with up to 30 switches after calibration, but the calibration only works over static networks. TTL-looping [15] provided measurement accuracy of about 75% for a network with only four switches by looping a probe packet 1024× over a link, which adds a huge overhead on the data plane. GRAMI [17] provided measurement accuracy greater than 90% for an emulated network with three switches, with the link latency set to 20 ms. LLDP-looping achieved measurement accuracy greater than 97% and 90% in networks with link latency greater than 1 and 0.05 ms, respectively. Among the listed approaches, LLDP-looping is the only one that provides measurement accuracy greater than 90% without being impacted by the network scale over low latency and dynamical SDNs.

Among all listed approaches, GRAMI and LLDP-looping perform RTT measurements and provide optimization mechanisms to minimize the total number of probe packets used to monitor the latency of all the links of a network, while the others are one-way latency measurements without any optimization mechanism provided. The number of probe packets injected by GRAMI is determined by the number of monitoring points that GRAMI needs over a tested SDN, while the topology of the tested SDN decides the number of monitoring points that GRAMI needs and how these monitoring points should be located over the entire network. However, the number of probe packets injected by LLDP-looping is decided by the size of the minimum vertex cover of an SDN, and the control plane is used by LLDP-looping to handle probe packets. Since LLDP-looping adds latency monitoring capability to the existing topology discovery function, which is one of the basic tasks of a controller, the real overhead of LLDP-looping consumed for latency monitoring is very low.

All the one-way latency monitoring approaches listed in Table V loop a probe packet once over a link except TTL-looping that loops a probe packet 1024 times. However, since GRAMI and LLDP-looping are RTT-based methods, in these two methods, a probe packet loops over a link for two times, and one more looping time is needed by LLDP-looping for topology discovery. Similar to the listed one-way latency monitoring methods, our LLDP-looping does not need to deploy any network infrastructure and install any flow entries dedicated to latency monitoring. Therefore, LLDP-looping is the only continuous latency monitoring method that can efficiently and accurately monitor the latency of all the links or routes of a low-latency SDN, without deploying any dedicated infrastructure and consuming extra flow tables.

### *G. Extending to Physical SDNs*

*1) Measurement Accuracy Impacted by Emulated Networks:* Since the time measured by an emulated switch may include an extra time cost spent by the underlying operating system in processing tasks for other emulated switches, the link latency measured by LLDP-looping and Ping over emulated networks is not accurate. To accurately estimate the measurement accuracy of LLDP-looping over an emulated network, we use the configured link latency as a baseline and recompute the measurement accuracy of LLDP-looping, because the configured link latency represents the real delay controlled by the underlying operation system that each LLDP packet has to bear before it is sent out by a port no matter the extra time cost is included or not. We therefore consider the same test scenarios as shown in Fig. 5(a) and (b), and calculate the measurement inaccuracy of those test scenarios, as shown in Fig. 10(a), where the measurement inaccuracy is about 120%, 28%, 14%, 5%, 0.4%, 0.08%, and 0.004% when the configured link delay is 0.01, 0.05, 0.1, 0.5, 1, 5, and 10 ms, respectively, and the measurement error does not increase as the network scale grows. Similarly, if we consider the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE V

LINK LATENCY MEASUREMENT COMPARISON, $n$ AND $r$ ARE THE NUMBER OF LINKS AND THE SIZE OF THE MINIMUM VERTEX COVER OF A NETWORK, RESPECTIVELY, AND $m$ IS THE NUMBER OF MONITORING POINTS OF GRAMI

| **Approach** | **Switch** | **Delay set(ms)** | **Probe packet** | **Cali-bration** | **Measure-ment error** | **Loops** | **Flow entries** | **Bytes injected** | **Injected by** |
|---|---|---|---|---|---|---|---|---|---|
| LLDP-looping | 60 | 0.01-0.5 | LLDP pcks | yes | -9% - 5% | 2 | no | $10r$ | controller |
| LLDP-looping | 60 | 1-10 | LLDP pcks | no | -3% - 0 | 2 | no | $10r$ | controller |
| approach [14] | 30 | 0/1/5 | LLDP pcks | yes | 20%/5%/3% | 1 | no | $34n$ | controller |
| approach [13] | 2 | 0/10/30 | Eth.frame | yes | 1% | 1 | no | $24n$ | controller |
| Opennetmon [12] | 4 | 1/5 | Eth.frame | no | | 1 | no | $24n$ | controller |
| TTL-looping [15] | 4 | 0 | Eth.frame | no | 25% | 1024 | no | $24n$ | controller |
| GRAMI [17] | 3 | 20 | ICMP pck | no | $\leq$ 10% | 2 | yes | $74m$ | hosts |

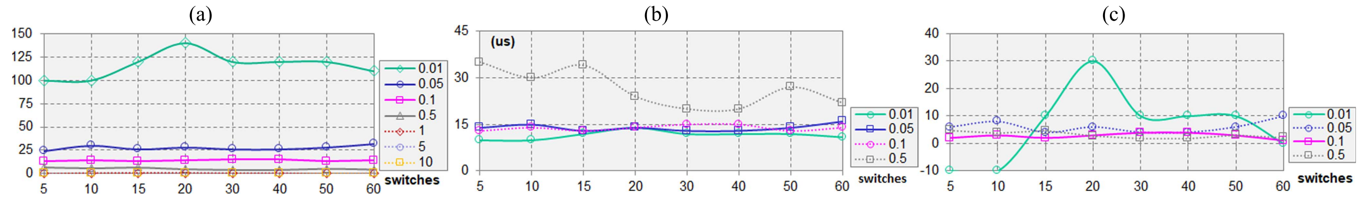

Fig. 10. Link latency measurement error comparing to the configured link latency. The labels represent the test results with various link delay. (a) Measurement error before calibration (%). (b) Subtraction of measured and configured link latency. (c) Measurement error after calibration (%).

acceptable measurement error be the 10% of the baseline, LLDP-looping cannot provide acceptable measurement accuracy in a network where the baseline is less than 0.1 ms without calibration. Since the link latency measured by LLDP-looping does not vary much as the scale of a network increases for each configured link latency, as shown in Fig. 10(b), we utilize a constant of 11 $\mu$s to calibrate the link latency measured by LLDP-looping. The measurement inaccuracy after calibration is shown in Fig. 10(c). It is apparent that LLDP-looping can reduce its measurement error to be less than 10% over a network with link latency configured as 0.05, 0.1, and 0.5 ms. LLDP-looping can also reduce its measurement inaccuracy to be not greater than 10% over a network with link latency configured to 0.01 ms in most of the test scenarios except the one that consists of 20 switches. The increased measurement inaccuracy in that particular scenario implies that some extra cost used by the underlying operating system to process tasks for other emulated switches may have been added into the latency measured by LLDP-looping.

In fact, the real link latency of an emulated network is the sum of the delay in both processing a packet and sending the packet out from a port (the configured link latency) at a virtual switch. Since the real measurement inaccuracy of LLDP-looping should use the real link latency of an emulated network as the baseline, the real measurement inaccuracy of LLDP-looping over emulated networks should be lower than the one calculated earlier, and the real measurement inaccuracy of LLDP-looping over physical networks should be even lower than the real measurement inaccuracy of LLDP-looping over emulated networks, because the real link latency measurement error of LLDP-looping over physical networks will not include the extra time cost incurred by the emulated network. In this sense, the measurement inaccuracy calculated earlier is pessimistic compared with the real measurement inaccuracy of LLDP-looping over physical networks; i.e., the same or even higher measurement accuracy of LLDP-looping can be achieved over physical networks compared to the presented results.

*2) Measurement Accuracy Impacted by Virtual Switches:* When using LLDP-looping to monitor the latency of an SDN with a light workload, the CPU speed fluctuation at the switches contributes the major part of the measurement jitter and decreases the measurement accuracy because LLDP-looping determines latency in the data plane. In the evaluation, we used an emulated SDN with a number of soft switches (the Openvswitch) controlled by a Nox C controller. Since an Openvswitch is run on a host computer, its CPU clock speed is higher than that of a physical switch, and the jitter generated by our emulated SDN is larger than that generated by a physical SDN. Thus, the evaluation over emulated SDNs incurred a larger measurement error than that over a physical SDN. However, we have emulated SDNs with an even lower link latency than most of the physical SDNs; i.e., the RTT measured by Ping over our emulated SDNs is as small as 0.05 ms, while the RTT of our test bed consisting of two hosts connected by a 1-Gb/s switch was 0.2 ms. Moreover, the measurement accuracy decreases as the real link latency in the network decreases. These observations suggest that the proposed LLDP-looping can achieve even higher measurement accuracy over a physical SDN than over our emulated SDNs.

*3) Using LLDP-Looping Over Physical Networks:* The controller that injects the probe packets cannot always directly connect to the intended selected switches. When using LLDP-looping to monitor the link latency of an SDN with an out-of-band control plane where an extra infrastructure is dedicated for the connection between switches and controllers, the probe packets injected by the controller can be routed to the intended selected switches using conventional switching or routing algorithms over the management infrastructure. Accordingly, routing the injected probe packets to the selected switches will not impact the data plane and impede the use of LLDP-looping to monitor the link latency of the data plane.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

However, in an SDN with hundreds of thousands of hosts, an out-of-band management infrastructure may suffer from a large packet forwarding latency, because the infrastructure that connects all the switches of the SDN to their controllers can generate a huge number of forwarding rules and incur large rule lookup delays for management packets [33], which can increase the flow setup latency of the control plane. On the other hand, if the control plane is distributed, the synchronization latency of the control plane may be significant. These scalability issues will be investigated in our future research. When using LLDP-looping to monitor the link latency of an SDN with an in-band control plane, where the same infrastructure is used for the data plane and the interconnection of controllers and switches, the probe packets injected by the controller can be routed to the intended switches using various ways dependent on the bootstrapping process of the switches [34]. However, forwarding injected probe packets to the selected switches over a large-scale SDN with an in-band control plane will increase the overhead of the corresponding switches and links, since all the probe packets and data packets have to share the same infrastructure. In order to enable LLDP-looping over an SDN with an in-band control plane, a controller placement problem that finds the best location of the controller over the entire network such that the overhead incurred by probe packets can be minimized has to be solved. This problem will be addressed in our future research.

The proposed LLDP-looping mechanism is designed to monitor the latency of physical links of a network. It cannot directly monitor the latency of virtual links over a virtual network, because a virtual link can be a logical link consisting of multiple physical links, and LLDP packets used as the probe packets in LLDP-looping can only reach the very first physical link along a logical link. However, once the latency of every physical link of the network has been measured by LLDP-looping, the latency of a virtual link can always be computed from the measurement results.

## VI. Conclusion

Low-latency networks have to monitor their latency continuously, efficiently, and accurately to maintain the satisfactory QoS/QoE of latency-sensitive applications. In this paper, we have proposed LLDP-looping, which serves this purpose in the context of SDNs by using the control plane to inject LLDP packets into the data plane and calculating the link latency at the switches by forcing an LLDP packet to loop a link three times. It guarantees a successful latency measurement, and completely avoids the systemic and measurement errors in most of the current measurement approaches. Compared with RTTs measured by Ping, in an SDN with link latency greater than 1 ms, LLDP-looping achieves a measurement accuracy of better than 97% without calibration, whereas in an SDN with link latency as low as 0.05 ms, the measurement accuracy is better than 90% with calibration. We have minimized the overhead of LLDP-looping in both the control plane and data plane by formulating a VCP to find the minimum vertex cover of the network. We have further proposed a novel greedy algorithm to solve the VCP and minimize the number of LLDP packets looped at each link for latency monitoring over tree-based network topologies. We have evaluated the performance of LLDP-looping over an emulated network and results show that LLDP-looping can effectively improve measurement accuracy, reduce overhead in both the control plane and data plane, and optimize resource usage when the latency of all SDN links need to be continuously monitored. Although the implementation of LLDP-looping method requires modifications of the controller and switch software, we have implemented a prototype to demonstrate that the modifications are minimal and can be easily integrated into any controller or switch of an SDN.

## References

[1] R. Kumar *et al.*, "End-to-end network delay guarantees for real-time systems using SDN," in *Proc. IEEE Conf. Real-Time Syst. Symp.*, May 2017, pp. 1–12.

[2] Y. Chen, R. Mahajan, and B. Sridharan, "A provider-side view of Web search response time," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 243–254, 2013.

[3] T. Flach, N. Dukkipati, and A. Terzis, "Reducing Web latency: The virtue of gentle aggression," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 159–170, 2013.

[4] T. Eylen and C. F. Bazlamacci, "One-way active delay measurement with error bounds," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 12, pp. 3476–3489, Dec. 2015.

[5] J. Fabini and M. Abmayer, "Delay measurement methodology revisited: Time-slotted randomness cancellation," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 10, pp. 2839–2848, Oct. 2013.

[6] M. Lee, N. Duffield, and R. R. Kompella, "Not all microseconds are equal: Fine-grained per-flow measurements with reference latency interpolation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 27–38, 2010.

[7] C. Fraleigh *et al.*, "Design and deployment of a passive monitoring infrastructure," in *Evolutionary Trends of the Internet* (Lecture Notes in Computer Science), vol. 2170, S. Palazzo, Ed. Berlin, Germany: Springer, 2001, pp. 556–575.

[8] M. Shahzad and A. X. Liu, "Accurate and efficient per-flow latency measurement without probing and time stamping," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3477–3492, Dec. 2016.

[9] Telecommunication Standardization Sector of ITU. *OAM Functions and Mechanisms for Ethernet Based Networks*. Accessed: Mar. 28, 2018. [Online]. Available: https://www.itu.int/rec/T-REC-Y.1731

[10] OpenFlow Switch Consortium. (2009). *OpenFlow Switch Specification Version 1.0.0*. [Online]. Available: Online:archive.OpenFlow.org/documents/OpenFlow-spec-v1.0.0.pdf

[11] C. Yu *et al.*, "Software-defined latency monitoring in data center networks," in *Passive and Active Measurement* (Lecture Notes in Computer Science), vol. 8995, J. Mirkovic and Y. Liu, Eds. Cham, Switzerland: Springer, 2015.

[12] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow software-defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp.*, May 2014, pp. 1–8.

[13] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," in *Proc. 9th Int. IEEE Conf. Netw. Service Manage.*, Oct. 2013, pp. 122–125.

[14] L. Liao and V. C. M. Leung, "LLDP based link latency monitoring in software defined networks," in *Proc. 12th Int. IEEE Conf. Netw. Service Manage.*, Oct. 2016, pp. 330–335.

[15] V. Altukhov and E. Chemeritskiy, "On real-time delay monitoring in software-defined networks," in *Proc. 1st Int. IEEE Conf. Sci. Technol. Conf. (Mod. Netw. Technol.)*, Oct. 2014, pp. 1–6.

[16] S. D. Sinha, K. Haribabu, and S. Balasubramaniam, "Real-time monitoring of network latency in software defined networks," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst.*, Dec. 2015, pp. 1–3.

[17] A. Atary and A. Bremler-Barr, "Efficient round-trip time monitoring in OpenFlow networks," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[18] Y. Breitbart, C.-Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz, "Efficiently monitoring bandwidth and latency in IP networks," in *Proc. 12th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, vol. 2, Apr. 2001, pp. 933–942.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[19] M. Shibuya, A. Tachibana, and T. Hasegawa, "Efficient performance diagnosis in OpenFlow networks based on active measurements," in *Proc. ICN*, 2014, p. 279.

[20] F. Aubry, D. Lebrun, S. Vissicchio, M. T. Khong, Y. Deville, and O. Bonaventure, "SCMon: Leveraging segment routing to improve network monitoring," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[21] J. Chen, I. A. Kanj, and G. Xia, "Improved parameterized upper bounds for vertex cover," in *Mathematical Foundations of Computer Science* (Lecture Notes in Computer Science), vol. 4162, R. Královič and P. Urzyczyn, Eds. Berlin, Germany: Springer, 2006, pp. 238–249.

[22] G. Nude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[23] Open vSwitch. *An Open Virtual Switch Internet*. Accessed: Apr. 24, 2016. [Online]. Available: http://openvswitch.org

[24] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, 2015.

[25] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrum. Meas. Mag.*, vol. 18, no. 2, pp. 42–50, Apr. 2015.

[26] J. Medved, R. Varga, A. Tkac, and K. Gray, "OpenDaylight: Towards a model-driven sdn controller architecture," in *Proc. IEEE 15th Int. Symp.*, Jun. 2014, pp. 1–6.

[27] A. Hammadi and L. Mhamdi, "A survey on architectures and energy efficiency in data center networks," *Comput. Commun.*, vol. 40, pp. 1–21, Mar. 2014.

[28] Data Center Institute of AFCOM. *How is a Mega Data Center Different From a Massive One?* [Online]. Available: http://www.datacenterknowledge.com/archives/2014/10/15/how-is-a-mega-data-center-different-from-a-massive-one

[29] K. L. Clarkson, "A modification of the greedy algorithm for vertex cover," *Inf. Process. Lett.*, vol. 16, no. 1, pp. 23–25, 1983.

[30] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, p. 9.

[31] C. H. Hsu and U. Kremer, "IPERF: A framework for automatic construction of performance prediction models," in *Proc. Workshop Profile Feedback-Directed Compilation*, 1998, pp. 1–10.

[32] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Passive and Active Measurement* (Lecture Notes in Computer Science), vol. 7192, N. Taft and F. Ricciato, Eds. Berlin, Germany: Springer, 2012.

[33] R. N. Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, 2009.

[34] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Automatic bootstrapping of OpenFlow networks," in *Proc. 19th IEEE Workshop Local Metrop. Area Netw.*, Apr. 2013, pp. 1–6.

**Lingxia Liao** (S'13) received the bachelor's degree from Tsinghua University, Beijing, China, and the master's degree from The University of British Columbia (UBC), Vancouver, BC, Canada, where she is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering.

She was a Science Researcher with the Computer Science Department, UBC. She was a Research and Development Director and the General Manager of high-performance computing with the Inspur Group, Jinan, China. She was with the Computer and Network Industry, China. Her current research interests include software-defined networking, network function virtualization, network virtualization, network measurement and optimization, distributed systems, cloud computing, and next-generation network.

**Victor C. M. Leung** (S'75–M'89–SM'97–F'03) received the B.A.Sc. (Hons.) and the Ph.D. degrees in electrical engineering from The University of British Columbia (UBC), Vancouver, BC, Canada, in 1977 and 1982, respectively.

He attended graduate school at UBC, on a Canadian Natural Sciences and Engineering Research Council Postgraduate Scholarship. From 1981 to 1987, he was a Senior Member of Technical Staff and a Satellite System Specialist with MPR Teltech Ltd., Burnaby, BC, Canada. In 1988, he was a Lecturer with the Department of Electronics, Chinese University of Hong Kong, Hong Kong. In 1989, he returned to UBC as a faculty member, where he is currently a Professor and the TELUS Mobility Research Chair in Advanced Telecommunications Engineering with the Department of Electrical and Computer Engineering. He has co-authored over 1100 journal/conference papers, 40 book chapters, and co-edited 14 book titles. His current research interests include broad areas of wireless networks and mobile systems.

Dr. Leung is a Registered Professional Engineer in the Province of British Columbia, Canada. He is a Fellow of the Royal Society of Canada, the Engineering Institute of Canada, and the Canadian Academy of Engineering. He was a Distinguished Lecturer of the IEEE Communications Society. He provided leadership to the organizing committees and technical program committees of numerous conferences and workshops. He was a recipient of the several best paper awards, the APEBC Gold Medal as the Head of the 1977 graduating class in the Faculty of Applied Science at UBC, the IEEE Vancouver Section Centennial Award, the 2011 UBC Killam Research Prize, the 2017 Canadian Award for Telecommunications Research, the 2018 IEEE ComSoc TGCC Distinguished Technical Achievement Recognition Award, the 2017 IEEE ComSoc Fred W. Ellersick Prize, the 2017 IEEE Systems Journal Best Paper Award, and the 2018 IEEE ComSoc CSIM Best Journal Paper Award. He is serving on the Editorial Boards for the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE ACCESS, *Computer Communications*, and several other journals. He served on the Editorial Boards for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS—Wireless Communications Series and Series on Green Communications and Networking, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE WIRELESS COMMUNICATIONS LETTERS, and *Journal of Communications and Networks*. He has been a Guest Editor of many journal special issues.

**Min Chen** (M'07–SM'09) has been a Full Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, since 2012, where he is currently the Director of the Embedded and Pervasive Computing Laboratory. He was an Assistant Professor with the School of Computer Science and Engineering, Seoul National University, Seoul, South Korea, where he was also a Post-Doctoral Fellow. He was a Post-Doctoral Fellow with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada. His Google Scholars citations reached 12 500+ with an h-index (55) and his top paper was cited 1300+ times. His current research interests include cyber physical systems, IoT sensing, 5G networks, mobile cloud computing, software-defined network, healthcare Big Data, media cloud privacy and security, body area networks, emotion communications, and robotics.

He is the Chair of the IEEE Computer Society Special Technical Communities on Big Data. He was a recipient of the Best Paper Award from the IEEE ICC 2012, the IEEE IWCMC 2016, and the IEEE Communications Society Fred W. Ellersick Prize in 2017.