Li Y, Cai ZP, Xu H. LLMP: Exploiting LLDP for latency measurement in software-defined data center networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 33(2): 277–285 Mar. 2018. DOI 10.1007/s11390-018-1819-2

LLMP: Exploiting LLDP for Latency Measurement in Software-Defined Data Center Networks

Yang Li[1], *Student Member, CCF*, Zhi-Ping Cai[1,*], *Senior Member, CCF, Member, IEEE* and Hong Xu[2], *Member, ACM, IEEE*

[1] *College of Computer, National University of Defense Technology, Changsha 410073, China*
[2] *Department of Computer Science, City University of Hong Kong, Hong Kong 999077, China*

E-mail: liyang06200910@163.com; zpcai@nudt.edu.cn; henry.xu@cityu.edu.hk

Received July 31, 2017; revised January 24, 2018.

Abstract The administrators of data center networks have to continually monitor path latency to detect network anomaly quickly and ensure the efficient operation of the networks. In this work, we propose Link Layer Measurement Protocol (LLMP), a prototype latency measuring framework based on the Link Layer Discovery Protocol (LLDP). LLDP is utilized by the controller to discover network topology dynamically. We insert timestamps into the optional LLDPTLV field in LLDP, so that the controller can estimate latency on any single link. The framework utilizes a reactive measurement approach without injecting any probe packets to the network. Our experiments show that the latency of a link can be measured accurately by LLMP. In relatively complex network conditions, LLMP can still maintain a high accuracy. We store the LLMP measurement results into a latency matrix, which can be used to infer the path latency.

Keywords software-defined network (SDN), Link Layer Discovery Protocol (LLDP), latency measurement

1 Introduction

The Open Networking Foundation (ONF), a developing non-profit organization dedicating to the promotion of software-defined network (SDN), defines SDN architecture as the physical separation of the control plane from the data plane in traditional networks. The data plane of the network typically consists of all the network elements such as switches and the routers that forward packets from one network node to another. The control plane is an intelligent controller behind those network elements and decides how to forward packets from one network node to another. The SDN architecture has been applied to data center networks (DCNs) where the data needs to be managed and controlled centrally. In addition, network performance measurement in software-defined DCNs is a fundamental management task of administrators to ensure the smooth operations of the network. Latency is one of the most important factors for network measurement because many interactive requirement services and applications are latency-sensitive, such as online games and multimedia services. The network administrator relies on the measurement results to detect latency violation. Specifically, administrators need real-time fine-grained latency measurements on any segment of a network path to pinpoint faults, hotspots, and troubleshooting. The problem is especially challenging due to the complex topologies and sheer volume of traffic in data center networks. Thus, a latency measurement solution with minimal overhead is preferred for practical deployment.

Traditional link and path latency measurement methods include active measurement and passive measurement. Active measurement techniques[1] frequently take the measures of sending and receiving probe packets (e.g., ICMP requests) from the network edge and measure their response time. Therefore, end-to-end probes[2] cannot measure the latency of path segments between arbitrary network devices. Another ap-

Regular Paper
Special Section on Computer Networks and Distributed Computing
Recommended by CCF ICoC 2017
This work was supported by the National Natural Science Foundation of China under Grant Nos. 61379145, 61501482, 61762033.
*Corresponding Author

proach is to install measurement tools at servers and launch probes from them, such as FlowSense[3] and OFLOPS[4]. This may not be feasible since colocation data centers restrict access to customer servers. Passive measurement[5] captures latency information about paths by measurement tools installed in the network devices. However, these approaches usually incur the overhead of performing real-time local coordination and aggregating measurements captured at many devices. The installation and deployment of passive measurement methods requires high network privileges, and sometimes even needs the support from network operators. In order to passively measure the traffic, we need to install corresponding applications or deploy measurement devices in each node. With the growing size of the network, the cost of the deployment is getting higher and higher. Therefore, it is not feasible to apply the passive measurement method to monitor the large-scale data center networks. Hence it is difficult and expensive to measure link and path latency using existing methods.

In this paper, we propose Link Layer Measurement Protocol (LLMP), a novel approach to measuring path latency in data center networks. LLMP overcomes the limitations of existing work by exploiting the Link Layer Discovery Protocol (LLDP)①, which is widely used in SDN controllers[6] to discover and update network topology. The idea is that we can simply utilize LLDP packets to piggyback timestamp information, and use it to measure latency of any single link segment. In addition, the proposed method uses the echo message to improve the accuracy of the measurement result. LLMP is a reactive scheme and does not inject additional probe packets to the network. All the data packets LLMP② exploits (LLDP packets, packet-out packets, echo messages, etc.) are inherent packets in SDN. It is arguably the first work combining LLDP and echo messages to complete practical work in SDN. Related studies such as OpenNetMon[7] and SLAM[8] are also reactive measurement approaches. However, they cannot measure the latency of path segments between arbitrary network devices, and they mainly depend on generating extra probe packets and sending corresponding matching rules to the target switches, which would consume expensive flow table resources of switches.

In our implementation, the LLDP packet is appended with a customizable field which is called latency-tag to carry the timestamp. When the controller sends the LLDP packet to one switch and retrieves it from another, it can infer the latency of this link by subtracting the time packets taken from the controller to the switches. We deploy our prototype on a physical and an emulated SDN testbed[9] and find LLMP can accurately measure link latency inflations of tens of milliseconds with little overhead. We also record the latency information of links into a latency matrix, which can be used to infer the path latency. Furthermore, if LLMP becomes a basic component for all SDN controllers, the network operator can utilize the measurement results to improve network service quality.

The remainder of this paper is structured as follows. In Section 2, we discuss the design and implementation of LLMP. We perform experiments and analyze the evaluation results of LLMP in Section 3. We also deploy LLMP in a small-scale campus DCN and analyze its real performance in Section 4. Section 5 concludes our work.

2 LLMP Design and Implementation

In this section, we present our measurement solution LLMP, which computes the latency of a single link by sending and retrieving LLMP packets over a specified period of time. Based on the measurement results, the path latency between any two switches can be inferred.

2.1 LLDP (Link Layer Discovery Protocol)

To dynamically discover the topology in SDN, the link discovery service inside controllers takes advantages of the data link layer LLDP to detect links between switches. The LLDP information is sent by controllers from of each of their interfaces at a fixed interval, in the form of an Ethernet frame. Table 1 shows the Ethernet frame format used in LLDP. DA and SA represent the multicast destination MAC address and source MAC address respectively. The ethertype field is set to 0x88CC. Each frame contains one LLDP data unit (LLDPDU). Each LLDPDU which carries the payload of an LLDP frame is a sequence of type-length-value (TLV) structures[10]. Each LLDP frame always includes the following integral mandatory TLVs: chassis ID, port ID, and time-to-live. The mandatory TLVs are followed by any number of optional TLVs. The

①http://www.ieee802.org/1/pages/802.1ab.html, Jan. 2018.

②https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.-0.noipr.pdf, Jan. 2018.

frame ends with a special TLV, named end of LLDPDU in which both the type and the length fields are 0. Table 2 exhibits the structure of an LLDPTLV, in which the actual value can be 0 to 511 octets.

Table 1. Frame Format of LLDP

Field	Value	Length (Byte)
DA	LLDP multicast address	6
SA	MAC address	6
Ether type	0x88cc	2
Data pad	LLDPDU (collection of TLVs)	1 500
FCS	Check digit	4

Table 2. Format of LLDPTLV

Item	Length (Byte)
TLV type	7
TLV information string length	9
TLV actual value of information	0∼511

Fig.1 shows the general link discovery procedures in an OpenFlow-based SDN network③. Here we illustrate only a unidirectional link discovery process for simplicity. The discovery of the reverse link is performed in a similar fashion.

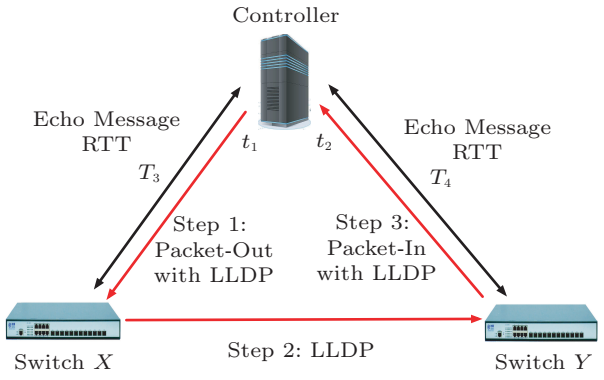

Fig.1. Link discovery procedures of LLDP, and simulated network experimental environment of single link.

Initially, the controller sends a packet-out message to switch X in Fig.1 with the payload of a controller-specific LLDP packet. The payload of the LLDP packet contains DPID (DataPath Identity) and the output port ID of switch X. Upon receiving the LLDP packet, switch X broadcasts it to all other ports. Typically, in an OpenFlow network, this is achieved by iterative transmissions of one LLDP packet to each broadcast enabled port. When receiving the broadcasting LLDP packet sent from switch X, the neighbor switch Y reports the incoming LLDP packet to the controller with the ingress port ID and DPID of switch Y via a packet-in message. When receiving the packet-in message from switch Y, the controller can detect a unidirectional link from switch X to switch Y.

In an OpenFlow-based network environment, the OpenFlow controller takes advantage of above link discovery method to obtain the entire network topology. The typical interval time is 12 seconds (e.g., in Floodlight④). In other words, if there are any changes of network topology, the controller can detect it within 12 seconds. The interval time can also be set to different values according to network management demand.

2.2 Design and Implementation of LLMP

Our algorithm implementation is mainly based on OpenFlow⑤ version 1.3 and LLDP. Now we introduce how to design LLMP using OpenFlow messages and LLDP in our algorithm.

According to the specification of LLDP, there are many reserved TLVs which can be utilized to collect the information of the OpenFlow switches. The LLMP packet is built by embedding the timestamp into the LLDP packet. The timestamp is encoded as a TLV[11]. In order to distinguish from other TLVs, a special type and a special name are set in the TLVs for the proposed LLMP: the type number is set to 10 (because numbers 1∼9 have been occupied by other TLVs, such as chassis ID and port ID) and the name is set to latency-tag. When the LLMP packet encapsulated in a packet-out message is traversed to the switches, we convert the current system time t_1 to string data type and embed it to the TLV's value. Step 1 accomplishes the above process in Fig.1.

When switch X receives the packet-out message containing the LLMP packet, it will broadcast the LLMP packet to all other ports. The next-hop switch Y reports the incoming LLMP packet to the controller via a packet-in message. The procedure is shown in step 2 and step 3 in Fig.1. When receiving the packet-in message from switch Y, the controller can calculate the latency of the LLMP packet in the network by comparing the receiving time with the sending time embedded in the LLMP packet, i.e., $(t_2 - t_1)$. Note that the accuracy of measurement results can be up to the

③http://www.opennetworking.org, Jan. 2018.
④http://www.projectfloodlight.org, Jan. 2018.
⑤https://www.opennetworking.org, Jan. 2018.

nanosecond granularity, but we only consider the millisecond granularity in this work as the millisecond level of precision is enough for most of web applications.

Although we have obtained the time the controller consumes between sending the LLMP packet and receiving the LLMP packet, it does not reflect the exact latency of the packet traverse between switches X and Y. Actually, the total latency is the sum of the latency time of the LLMP packet traversing from the controller to switch X, plus the latency time of forwarding it from switch X to switch Y, and plus the latency time of forwarding it from switch Y to the controller. Hence, in order to improve the accuracy, we must eliminate these influential elements. Skillfully we can take advantage of the echo messages in OpenFlow. One of the functions of echo messages is to measure the end-to-end latency. Hence, we utilize the function to obtain the latency between the controller and the switches. The black lines in Fig.1 show the procedure. First, the controller sends echo request messages to switches. When switches receive the echo request messages, they generate echo reply messages and send them back to the controller. The controller retrieves the echo reply messages and can calculate RTT between the controller and switches (i.e., T_3 and T_4 in Fig.1 are the latencies between the controller and the switches). Note that a one-way delay method can be used to measure the delay between the controller and switches, which would be more accurate if the controller and switches have synchronous clocks. However, in order to get the measurement results of one-way delay, the entire network would require clock synchronization and need install extra measurement devices or applications on all network nodes[12]. Consequently, in order to make LLMP easier to deploy, we adopt the method of RTT. Having tested the real one-way delay and half of RTT in the same experiment conditions, we find that the measurement results of them are almost equal. As a consequence, we ultimately select the half of RTT approximatively as the one-way delay in LLMP.

The controller estimates the actual latency by calculating the interval time T_{LLDP} of the LLMP packet passing through the network, and then subtracting the additional time the LLMP packet consumes in the link between the controller and the switches. The mathematical calculation formula can be directly described as:

$$T_{\mathrm{LLDP}} = t_2 - t_1,$$
$$T_{\mathrm{LLMP}} = T_{\mathrm{LLDP}} - (T_3 + T_4)/2.$$

2.3 Multi-Links Path Segment Inference

The controller periodically launches the LLMP service to discover topology and measure all links latency between switches. The measurement results are stored in the latency matrix, in which every entry is composed of five tuples (source switch DPID, source switch port ID, destination switch DPID, destination switch port ID, and latency value). The information of latency matrix is updated along with the changes of network topology by sending and retrieving LLMP packets.

As a vital part of the SDN architecture, the controller has a global view of the entire network topology which can be used to implement route selection and other network functionalities. The controller can utilize the link latency matrix to infer and estimate the multi-links path segment latency. For instance, given any multi-links path segment, the controller identifies the first and last switches of the path segment and determines the switches and the links in the path segment[13]. Then, the controller carries out a series of actions, searching corresponding links' latency values in the latency matrix and summing these results to get the path segment latency.

The latency matrix can help the controller obtain path segment latency rapidly and the controller utilize the results to re-route the flow forwarding, which can reduce network resource consumption and enhance the efficiency of network operation. In addition, the network operators evaluate network congestion condition relying on the measurement results obtained by LLMP, which we will analyze in detail in Section 4.

3 Experiments and Evaluation

3.1 Experiments

We implement LLMP based on Floodlight and modify the LinkDiscoveryManager module. In all experiments we launch iperf flows to simulate user traffic, and develop a user application to record a packet's arrival and departure time on switches to obtain the accurate one-way delay. We also realize SLAM by preinstalling forwarding rules on the switches along the path, and making the controller generate a series of probe packets delivering along the link or path.

In order to guarantee the feasibility and flexibility of LLMP, we set two experimental conditions.

In the first setting, we collect measurement results of all single links on a physical testbed as shown in Fig.1. The physical testbed consists of three servers

with Intel® Core i5 CPU running stock Ubuntu Server 14.04 LTS and OpenvSwitch. Both switches are linked to the central controller running Floodlight. The three measurement methods (user applications, SLAM and LLMP) are launched simultaneously, and thus we can obtain three types of synchronized measurement results which can be used to evaluate LLMP's accuracy. We perform two sets of experiments to evaluate the LLMP's performance in different network conditions. One is in the regular latency condition and the other is in the high latency condition, i.e., the switches are in the congestion condition. We estimate the latency between the same pair of switches in our testbed, and introduce the background traffic using iperf and simulate the congestion by shaping traffic at a switch in the regular latency condition.

In the other setting, we use Mininet to emulate a larger-scale network again with OpenVSwitch controlled by Floodlight. The links are configured with 20 ms latency and 200 Mbps bandwidth. We select eight typical topologies (Table 3) from Topology Zoo[6], measure the latency of all links by LLMP, and preserve the results in the corresponding latency matrix. We choose two paths each composed by three links in every topology, and infer their latency through the latency matrix. In addition, we use the 4th topology SWITCH as our foundational test topology since we consider this topology to be more representative, choose the path segment composed by different links, and run the user application, SLAM, and LLMP on it respectively.

Table 3. Latency of Different Path Segments

No.	Type	Switch	Link	Path Latency (ms)
1	Sprint	11	17	61.7
2	IBM	15	20	62.2
3	Renater	25	34	62.9
4	SWITCH	21	47	62.5
5	Tinet	32	53	63.1
6	DFN	35	62	63.3
7	TATA	64	88	63.8
8	Viatel	98	115	64.1

3.2 Results and Evaluation

3.2.1 *Accuracy in Single Path Segment*

In the regular latency environment, we run all of the three methods synchronously and obtain three datasets. Fig.2(a) shows the variation tendencies of measurement results, and it is clear to see that the fluctuation of the delay values is not obvious, mainly because the rate of injected traffic is constant. The figure also indicates that the measurement results of LLMP are extremely close to those of user applications, verifying that LLMP can be utilized to detect millisecond-level link latency variations accurately.

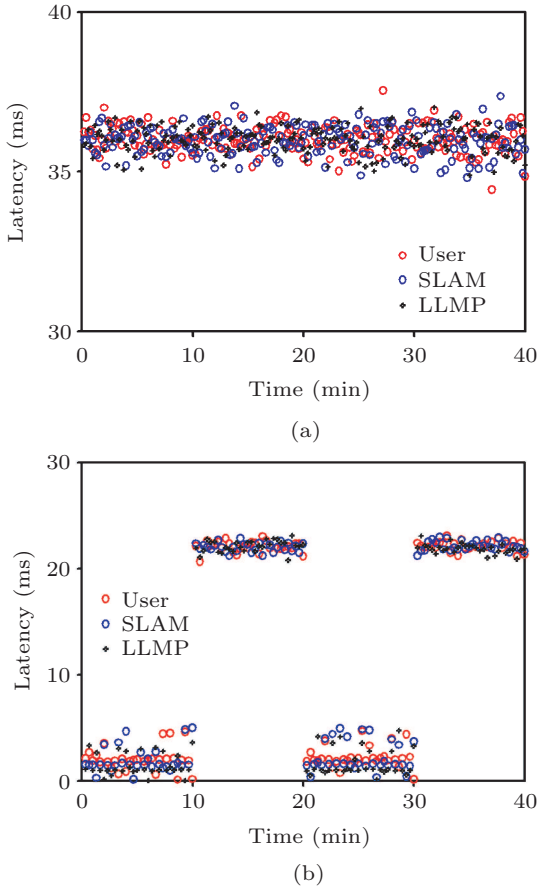

Fig.2. (a) Comparison of LLMP, user application (user) and SLAM with constant traffic. The measurement results of the three methods are pretty similar. (b) Comparison of LLMP, user application and SLAM with bursty traffic. LLMP is able to correctly detect the variations of links' latency.

We compare the dataset of user application against that of LLMP, and the minimum difference of the two datasets is 0.011 ms. Besides, most of the differences are between 0 ms and 1 ms. All statistical measures of the deviations of the two datasets are shown in Table 4. In addition, the measurement results of SLAM have similar accuracy as those of LLMP.

Table 4. Statistical Results of Deviations

Statistical Characteristic	Value
Min-deviation value	0.011 0
Average deviation	0.451 3
Standard deviation	0.684 2
Variance	0.583 5

[6]http://www.topology-zoo.org/dataset.html, Jan. 2018.

3.2.2 Sensitivity to Network Conditions

We use iperf to slightly congest the network, and estimate the measurement accuracy of using LLMP in the presence of bursty traffic. Fig.2(b) shows the variations of link latency along with time since we intermittently introduce bursty traffic. The figure indicates that LLMP has strong adaptability when there is congestion in the network. When the network traffic is normal, measured latency is small; however, when launching iperf, with a sudden rise of network traffic, the measurement results of relevant links commence to increase. Obviously, LLMP can accurately capture the latency variations as well as the other two approaches.

3.2.3 Multi-Links Path Latency Estimation

LLMP can calculate single link path latency, but more realistic network topology is mainly composed by a vast number of path segments and one path segment is usually composed of many links. In order to verify the feasibility of LLMP in the path segments, we use eight distinct topologies from Topology Zoo as shown in Table 3 and run LLMP on them. Then we randomly select two path segments composed by three links from every topology and obtain the path segments' latency information by summing up latency values of corresponding links which are stored in the latency matrix[14]. Table 3 shows the relevant topologies and the calculated delay results. From the data in the table we can observe that with the increasing number of switches and links in different topologies, the measurement results grow slightly.

Figs.3(a) and 3(b) display the distribution of path segments' latency results measured by different methods in topology SWITCH⑦, and we can draw a conclusion that within the same range of error, LLMP shows the equal accuracy with the two other measurement methods.

As a result, the measurement results in the latency matrix can be used to infer the multi-links path segment latency. We compare our proposed LLMP method with SLAM from the aspect of resources consumption as follows. If a network path segment is composed by n switches and $n-1$ links, and in order to get the latency between any two nodes, the controller has to execute M times calculation operations and send Q probe packets to the switches. Considering the difference of bidirectional latency in actual network environment, we obtain the values of M and Q by the following formulas:

$$M = 2\sum_{i=1}^{n-1} i = n(n-1)(n>2), \tag{1}$$

$$Q = 2\sum_{i=1}^{n-1} i(n-i+1) = \frac{n(n-1)(n-2)}{3}(n>2). \tag{2}$$

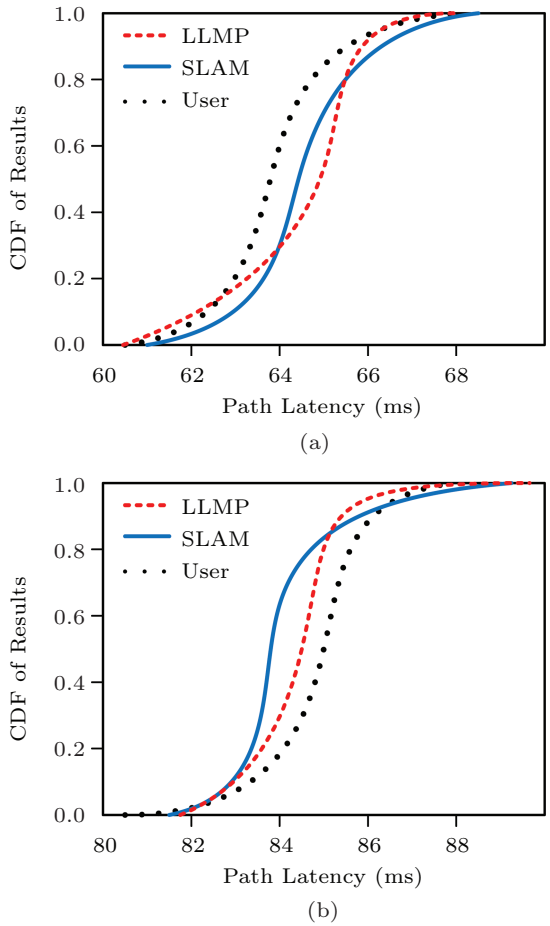

Fig.3. LLMP vs user application (user) and SLAM measurement results' CDFs for paths composed of different numbers of links in SWITCH. (a) Path including three links. (b) Path including four links.

From (1) and (2), we can draw a conclusion that the spatial complexity of injecting probe packets using SLAM is $O(n^3)$. However, LLMP only needs the controller to calculate the delay of $n-1$ links for one path segment, and cutting down the number of calculations of so many path segments' delay is pretty significant, because too many calculation and probe manipulations would consume the computing resources of the controller and the flow table resources of the switches. If

⑦http://www.topology-zoo.org/dataset.html, Jan. 2018.

the controller uses such methods which need preinstall forwarding rules in the switches for probe packets to obtain the delay information of every path segment, the additional resource consumption is inevitable[15]. Even though SLAM has similar accuracy to LLMP, it would be only adapted to a given link or path. If the controller needs to know the delay information of a path segment in specific moment, it could utilize SLAM to measure the given path. However, in most situations, the controller must obtain latency conditions of all links actively. It is infeasible for the controller to reach the destination by SLAM, but it can do that with LLMP. Consequently, for the controller and network operators, it is a better alternative to obtain the links' delay information.

4 Impletemention in a Real Network Environment

We have disposed the controller including LLMP in a small-scale data center networks in our campus network for three months. Fig.4 indicates the network topology of our campus, composed by different types of switches (e.g., core switches, aggregation switches[16]) and a server center. As long as the controller implements network topology discovery, all of the links' latency information can be recorded by LLMP. After analyzing the latency data, we find that most of the links' measurement values keep in a low level, less than 20 ms during a vast majority of time of the day. In contrary, only a very small number of links display high latency scene, over 40 ms, just at a specific moment. As a consequence, we filter out the data less than 20 ms, and detect the congestion condition of links whose delay is over 20 ms to verify whether they are suffering from congestion[17]. High latency does not necessarily imply that the links or path segments are congested. Yet when some other similar characteristics appear on the same link, such as increasing packet loss rate, jitter[18], and throughout degradation[19], the link is highly likely to encounter congestion.

We detect the link congestion condition[20] according to LLMP's measurement results. Fig.5 shows the relation between the possibility of links' congestion and their delay measurement results, as well as their rates. Obviously, the higher the links' delay, the greater the probability of their congestion. When the latency results increase to 80 ms, the links are almost in a congestion state. Thus, we believe that the measurement results of LLMP can be used as suggestions of detecting link congestion. This means that the network operators do not need to observe and detect the network state constantly. They only need to carry out relevant monitoring manipulations when the measurement re-

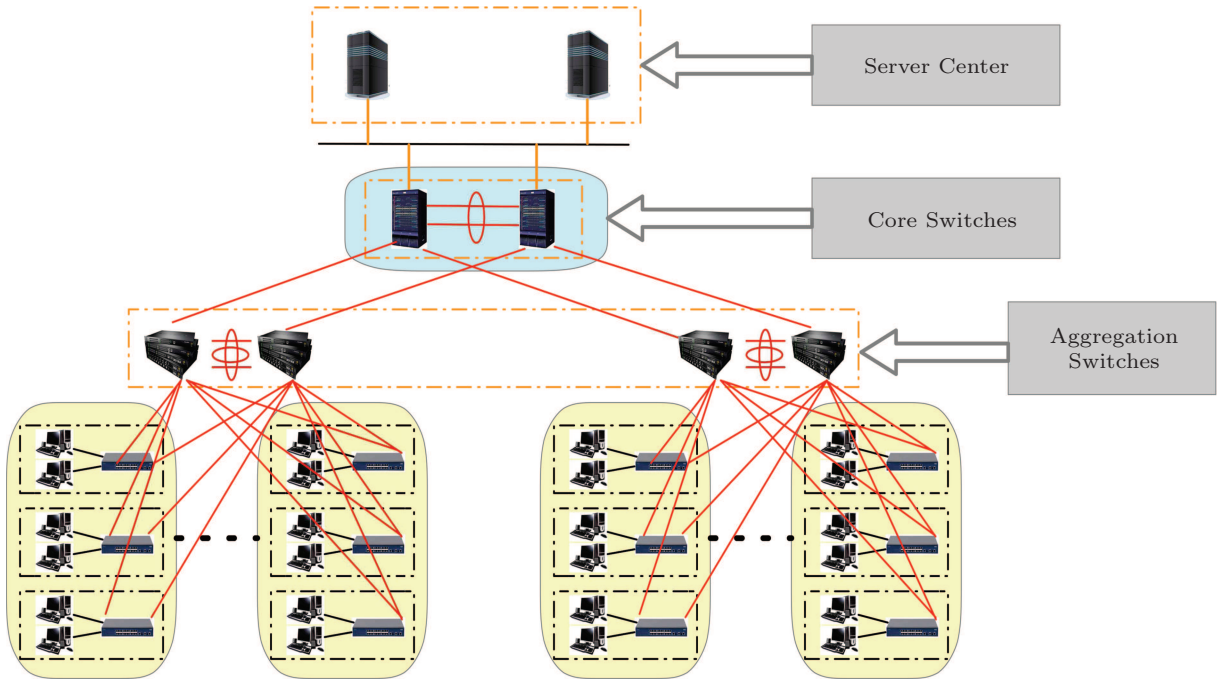

Fig.4. Campus network topology. A typical small-scale software-defined data center runs the controller in the server center and all of the switches support the OpenFlow.

sults obtained by LLMP show abnormal values, which can reduce the onerous and heavy tasks of them.

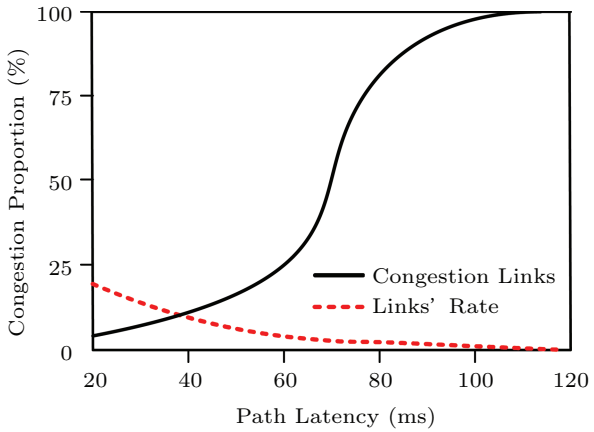

Fig.5. Relation between links' congestion and their latency. Apparently, the higher the delay, the higher the probability of congestion. In addition, high-delay links occupy a very small proportion.

5 Conclusions

We presented LLMP, a link and path segment latency measurement framework for software-defined data center networks. LLMP utilizes timestamps of carefully triggered control messages to measure network latency between any two arbitrary switches. LLMP's measurements are accurate enough to detect latency inflations of tens of milliseconds. The method also can be used to monitor an SDN network for measuring the delay without any extra traffic and can carry out real-time network measurement and monitoring. The administrators can depend on latency results obtained by LLMP to detect the network conditions rather than do some repetitive and boring work.

For future work, we plan to estimate the end-to-end latency utilizing LLMP, and design a more accurate measurement module for the controller, because it is vital important for the controller to gain the links delay information[21]. The proposed LLMP method is possible to be used on network security[22]. We will also consider using LLMP for other network management tasks such as load balancing, anomaly detection and congestion control.

References

[1] Das A, Lumezanu C, Zhang Y *et al.* Transparent and flexible network management for big data processing in the cloud. In *Proc. the 5th USENIX Workshop on Hot Topics in Cloud Computing*, June 2013.

[2] Duffield N G, Grossglauser M. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 2001, 9(3): 280-292.

[3] Yu C, Lumezanu C, Zhang Y *et al.* FlowSense: Monitoring network utilization with zero measurement cost. In *Proc. PAM*, Oct. 2013, pp.31-34.

[4] Rotsos C, Sarrar N, Uhlig S *et al.* OFLOPS: An open framework for OpenFlow switch evaluation. In *Proc. PAM*, Mar. 2012, pp.85-95.

[5] Huang D Y, Yocum K, Snoeren A C. High-fidelity switch models for software-defined network emulation. In *Proc. HotSDN*, Aug. 2013, pp.43-48.

[6] Kreutz D, Ramos F M V, Paulo Esteves Verssimo *et al.* Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 2015, 103(1): 14-76.

[7] van Adrichem N L M, Doerr C, Kuipers F A. OpenNetMon: Network monitoring in OpenFlow software-defined networks. In *Proc. Network Operations and Management Symposium* (*NOMS*), May 2014.

[8] Yu C, Lumezanu C, Sharma A *et al.* Software-defined latency monitoring in data center networks. In *Proc. PAM*, Mar. 2015, pp.360-372.

[9] Cui Y, Xiao S, Liao C *et al.* Data centers as software-defined networks: Traffic redundancy elimination with wireless cards at routers. *IEEE Journal on Selected Areas in Communications*, 2013, 31(12): 2658-2672.

[10] Han K, Hu Z, Luo J *et al.* RUSH: Routing and scheduling for hybrid data center networks. In *Proc. IEEE INFOCOM*, Apr. 2015, pp.415-423.

[11] Narisetty R R, Dane L, Malishevskiy A *et al.* OpenFlow configuration protocol: Implementation for the of management plane. In *Proc. the 2nd GENI Research and Educational Experiment Workshop*, Mar. 2013, pp.66-67.

[12] Mckeown N, Anderson T, Balakrishnan H *et al.* OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2008, 38(2): 69-74.

[13] Dhawan M, Poddar R, Mahajan K *et al.* SPHINX: Detecting security attacks in software-defined networks. In *Proc. NDSS*, Feb 2015.

[14] Yu M, Jose L, Miao R. Software-defined traffic measurement with OpenSketch. In *Proc. NSDI*, Apr. 2013, pp.29-42.

[15] Braun W, Menth M. Software-defined networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet*, 2014, 6(2): 302-336.

[16] Kim H, Feamster N. Improving network management with software-defined networking. *IEEE Communications Magazine*, 2013, 51(2): 114-119.

[17] Chowdhury S R, Bari M F, Ahmed R *et al.* PayLess: A low cost network monitoring framework for software-defined networks. In *Proc. NOMS 2014*, May 2014, pp.1-9.

[18] Gill P, Jain N, Nagappan N. Understanding network failures in data centers: Measurement, analysis, and implications. *ACM SIGCOMM Computer Communication Review*, 2011, 41(4): 350-361.

[19] Gandhi R, Liu H H, Hu Y C *et al.* Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review*, 2015, 44(4): 27-38.

[20] Xie D, Ding N, Hu Y C *et al.* The only constant is change: Incorporating time-varying network reservations in data centers. *ACM SIGCOMM Computer Communication Review*, 2012, 42(4): 199-210.

[21] Zhang H, Cai Z P, Liu Q *et al.* A survey on security-aware measurement in SDN. *Security and Communication Networks*, 2018, doi:10.1155/2018/2459154. (to be appeared)

[22] Xia J, Cai Z P, Hu G *et al.* An active defense solution for ARP spoofing in OpenFlow network. *Chinese Journal of Electronics*, 2018. (to be appeared)

Yang Li received his B.S. degree in computer science from Ocean University of China, Qingdao, in 2014. Then he received his M.S. degree in computer science and technology from National University of Defense Technology, Changsha, in 2017. His research interests are mainly in software-defined network (SDN) and network measurement.

Zhi-Ping Cai received his B.Eng., M.A.Sc., and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT), Changsha, in 1996, 2002, and 2005, respectively. He is a full professor in the College of Computer, NUDT, Changsha. His current research interests include network security and big data. He is a senior member of CCF and a member of IEEE. His doctoral dissertation was rewarded with the Outstanding Dissertation Award of the Chinese PLA.

Hong Xu is an assistant professor in Department of Computer Science, City University of Hong Kong, Hong Kong. His research area is computer networking, particularly data center networks, NFV, and big data systems. He received his B.Eng. degree from The Chinese University of Hong Kong, Hong Kong, in 2007, and his M.A.Sc. and Ph.D. degrees from University of Toronto in 2009 and 2013 respectively. He was the recipient of an Early Career Scheme Grant from the Hong Kong Research Grants Council in 2014. He received the Best Paper Awards from ACM TURC 2017 (SIGCOMM China), IEEE ICNP 2015, and ACM CoNEXT Student Workshop 2014. He is a member of ACM and IEEE.