LLDP Based Link Latency Monitoring in Software Defined Networks

Lingxia Liao
Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
Email: liaolx@ece.ubc.ca

Victor C.M. Leung
Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
Email:vleung@ece.ubc.ca

***Abstract*—Current latency monitoring approaches for Software Defined Network often use the control plane as the infrastructure to inject time-stamp data packets as probe packets to measure the network latency at a particular time, but suffer from three major issues when the network latency needs to be continuously monitored: 1) the increased control plane's overhead, 2) the feasibility of using data packets as probe packets, and 3) the increasing measurement error using OpenFlow messages to measure the time from the controller to a switch as the network scale grows. To overcome these issues, this paper proposes link latency monitoring using time-stamped Link Layer Discovery Protocol (LLDP) packets, aided by a linear calibration function to reduce errors of measuring switch-controller delays. Time-stamping LLDP packets, which are used to discover the global network topology in SDNs, does not add extra workload to the control plane and the results always reach the controller thus while avoiding measurement failures that might occur in existing approaches. Our linear calibration function can reduce the measurement error to less than 5% of the link latency measured by ping in a network with up to 30 switches and the link latency not less than 1ms.**

I. Introduction

Running latency sensitive applications over current Internet Protocol (IP) network with high user experiences often needs to accurately and continuously monitor the latency of all the routes in the network, and then quickly route the packets away from the busy route. However, current approaches accurately and continuously monitoring the latency of all the routes of an IP network are expensive due to the lack of a dedicated network infrastructure to handle probe packets in active latency measurements [2], or a global clock among network devices in passive latency measurements [1]. Traditional ping utility also cannot be used due to the restriction in accessing to the user hosts in many networks.

OpenFlow [3] is a protocol commonly used to control the data plane in a Software Defined Network (SDN) by the decoupled control plane. Using OpenFlow, the control plane can insert flow entries as well as packets into the OpenFlow switches in the network to program the behavior of the network flows. Therefore, the control plane can provide the network infrastructure dedicated to support active latency measurement by generating probe packets, inserting them to the data plane, and calculating the time difference between receiving and sending a probe packet. This time difference ($t_{measured}$) minus the time delay from the controller to the source switch ($t_{ctltosrc}$) and the time delay from the destination switch to the controller ($t_{dsttoctl}$) represents the latency of a route ($t_{srctodst}$) [4]. However, injecting OpenFlow messages to switches to measure the time delay from the controller to switches cannot give accurate results because the behaviors of the controller and switches are different in processing an OpenFlow message as opposed to a probe packet, and periodically injecting probe packets to the data plane also change the overhead of the control plane and affects the measurement accuracy. More importantly, time-stamping data packets as probe packets [4] [5] can cause measurement failure, because the flow entries for the real data packets will forward the time-stamped data packets to its next hop rather than the control plane. How to correctly generate probe packets to accurately and continuously monitor the latency of all the routes in a SDN without changing the behavior of the real data packets has not been explored in current research.

Observing that existing OpenFlow controllers often insert Link Layer Discovery Protocol (LLDP) [6] packets to the data plane to discover the global network topology, we therefore propose a novel link latency monitoring approach based on LLDP in SDNs to address the above issues (a link represents a route with only two adjacent switches in this paper). Directly measuring the latency of links of a network can form a link latency map to quickly find out a path with the lowest latency for a flow to provide best user experiences for latency sensitive applications. Time-stamping LLDP packets to measure the network latency does not add extra overhead on the control plane, and hence does not reduce the measurement accuracy as it in current proposed latency monitoring approaches. Moreover, LLDP is a protocol used among switches. LLDP packets are not data packets generated and consumed by an application running at a host. Therefore, time-stamping LLDP packets does not require that each switch of the network matches flow entry to these packets and therefore it does not impact the behaviors of the data flows in the network. To minimize error in measuring the controller-switch latency as the network scale grows, we generate a linear function to calibrate the measured time delay from sending to receiving a probe packet over an emulated SDN with up to 30 switches using Mininet [7]. Using an emulated SDN to study the major factors affecting the latency measured by our proposed approach and to evaluate its accuracy is feasible because our measurement is fully done at the control plane. An emulated SDN provides us a scalable network environment to identify the major factors affecting the measurement accuracy, study the possible relationship between the measurement error and these factors, and develop a calibration function to improve the measurement accuracy.

The rest of this paper is organized as follows. The related work is summarized in Section II. The feasibility of using LLDP packets as probe packets is discussed in Section III. LLDP based link latency measurement is proposed in Section IV. Section V introduces the experiments showing the major factors that affect the link latency measured by our LLDP based approach. A linear calibration function is developed in Section VI to facilitate time-cost measurements. Section VII evaluates the measurement accuracy, and Section VIII concludes the paper.

II. Related Work

References [4] and [5] implement the same approach that uses control plane to inject probe packets to switches to monitor the latency of a route for SDNs. The former generates Ethernet frames carrying the time-stamps and develops a calibration constant to improve the measurement accuracy for a physical network with 2 switches, while the latter time-stamps real data packets as probe packets and targets a network with long route latency to ignore the measurement error. Both of them do not consider the potential issue in generating probe packets, and their experiments are done over a network with very limited number of switches, while our proposed approach addresses the issue in generating probe packets and conducts experiments over an emulated network with up to 30 switches. TTL-looping [8] [9] targets a network with low link latency. It sends probe packets to the source switch of a path, and then lets the probe packets loop the path a number of times before finally sending them back to the controller. By calculating the time difference between receiving and sending a probe packet divided by the number of loops, the measurement error caused by the time delays from the controller to the source switch and from the destination switch to the controller can be significantly reduced. However, this approach adds a huge amount of extra workload to the data plane. While it may be used for measuring the latency of a particular route in a network at a particular time, it may not be feasible to continuously monitor the latency of all the routes in a network. SLAM [10] presents an approach to continuously monitor the network latency of a SDN by injecting time-stamped data packets, but its monitoring frequency relies on the active time of a flow entry, while our approach can adjust the monitoring frequency based on the requirements.

III. Feasibility of LLDP Packets as Probe Packets

To discover the topology of a conventional IP network, each layer 2 switch has a particular process that periodically broadcasts LLDP packets to and receives LLDP packets from its peers. However, in a SDN, this process is implemented in the control plane that periodically injects LLDP packets to each OpenFlow switch of the network. When an OpenFlow switch receives a LLDP packet from the control plane, it floods it. When an OpenFlow receives a LLDP packet from its adjacent switch, it packs the received LLDP packet into an OpenFlow packet-in message and sends it back to the controller. This way, a LLDP packet in the network is always forwarded to the controller by a link destination switch to discover a link and hence the network topology.

For the purpose of latency measurement, a probe packet is a time-stamped packet. SDNs can use the control plane to generate, send, and receive probe packets. To ensure a successful latency measurement, each probe packet has to be sent from and received back at the controller. To meet this requirement, a probe packet should not have any matching flow entry stored in the switches along its path.

Most data packets do not meet this requirement because they are usually sent by an application and received by another application. There is no way to ensure that no matching flow entries are stored in the switches along the path. Therefore time-stamping a data packet so that it also serves as a probe packet for latency monitoring purpose may fail as a switch along its path will forward the probe packet to its next hop switch rather than the controller. If we force a time-stamped data packet to be forwarded to the controller by removing the matching flow entry at a switch, the data packets may also be forwarded to the controller, and hence the behavior of the data packets is changed. Using different flow entries to differentiate a data packet and a time-stamped data packet can solve this problem, but cannot be supported by current OpenFlow protocols. Therefore, time-stamping real data packets as probe packets may not be feasible in practice.

Since LLDP is a protocol among switches, the process that discovers the network topology does not involve any application running at hosts. Since no host will send LLDP packets to other hosts in the network, there is no need to setup flow entries for LLDP packets in the network. In fact, current switch and routing applications of OpenFlow controllers never setup flow entries for LLDP packets to ensure that each LLDP packet can be sent back to the controller to discover the global network topology. Therefore, LLDP packets can meet the requirement of probe packets, and it is feasible to time-stamp LLDP packets to continuously monitor the link latency of a network.

IV. LLDP Based Link Latency Monitoring

When we use time-stamped LLDP packets as probe packets, the route latency measurement described in the first section monitors the latency of a link in the network. We formulated this measurement as the following:

$$t_{measure} = t_{ctltosrc} + t_{srctodst} + t_{dsttoctl} \tag{1}$$

$$t_{srctodst} = t_{measure} - t_{ctltosrc} - t_{dsttoctl} \tag{2}$$

Here, we use $t_{measure}$ to represent the delay of the packet seen at the controller from the time it is sent to the time it comes back, $t_{ctltosrc}$ to be the time delay incurred by the packet from the controller to a link source switch, $t_{srctodst}$ to be the time delay incurred by the packet from the source switch to the destination switch of a link, and $t_{dsttoctl}$ to be the time delay incurred by the packet from a link destination switch to the controller.

Our prototype is built on top of the discovery component of current Nox classic project [11]. Particularly, we modify the discovery.py of the discovery component by adding a system capabilities TLV [6] to each LLDP packet before the end TLV for latency monitoring. Since discovery.py uses a timer to periodically send LLDP packets to each port of each switch in the network, when the timer fires, we retrieve the current time, and take the fractional part of the current GTC time and convert it to a 4-byte integer. We store the higher 2 bytes of the integer in the capabilities available field and the lower 2 bytes of the integer in the capabilities enabled field, then send the LLDP packet out. When a time-stamped LLDP packet comes back to the controller, the LLDP input handler function that handles the incoming LLDP packets in discovery.py firstly retrieves the current time, then decodes the incoming LLDP packet to get the time-stamp. The fractional part of the current time minus the time-stamp in the incoming LLDP packet is the $t_{measure}$ in (1) and (2). Since the system capabilities TLV has not been used in topology discovery, using it to carry the time-stamps for latency monitoring does not affect the behavior of network flows. However, a more general way to prototype our LLDP based link latency monitoring is to define an organizationally specific TLV [6] for latency measurement.

In order to measure the time delay between the controller and a switch, our prototype also periodically sends a barrier request to each switch of the network. We use a hash map to store the time-stamp when a barrier request is sent to a switch and calculate the time difference when a barrier reply is received from a switch. Since a barrier request message has no payload, and has to be replied without further decoding, this time difference represents the RTT between the controller and a switch. This way, we have $t_{measure}$, $t_{ctltosrc}$, and $t_{dsttoctl}$, and $t_{srctodst}$ can be calculated from (2).

V. Factors Affecting Measured Link Latency

Any factors that affect the $t_{measure}$, $t_{ctltosrc}$, and $t_{dsttoctl}$ will affect the $t_{srctodst}$. Here, we consider 1) the inaccuracy when measuring $t_{ctltosrc}$ and $t_{dsttoctl}$, 2) the time-stamped LLDP packets injecting frequency, and 3) the number of switches in the network. We conduct several experiments based on an emulated network using Mininet to investigate how these factors affects the monitored latency. We consider a network with light work load at the data plane. A SDN emulated by Mininet has each of its switches virtualized by a virtual switch process. This difference between an emulated SDN and a physical SDN does not really affect our latency measurement, because our measurement is fully done at the control plane. All the experiments in this paper choose a linear topology rather than any others, because emulating a linear topology costs less resource, a computer running Mininet can uses its limited resource to emulate a larger network. Meanwhile, our proposed approach directly monitors the latency of all the links in a network, the topology of the network does not really affect the measured results. With – link TC parameter provided by Mininet to adjust the link latency, an emulated SDN is very suitable to study the factors affecting measurement latency as well as to evaluate the measurement accuracy for our proposed approach.

A. Time Delays between Controller and Switches

To achieve a high measurement accuracy at (2), the $t_{ctltosrc}$ and $t_{dsttoctl}$ have to be accurately measured. Current approaches using control plane to measure the network latency often insert OpenFlow messages to every switch of the network from the controller and calculate the time difference between the sending and reception of each message. Since the behavior of a switch is different when processing an OpenFlow message as opposed to a probe packet, the controller-switch time delay measured by these approaches does not really represent the time delay experienced by a probe packet traveling between the controller and a switch. This difference can create serious measurement errors, especially in a local area network with low link latency.

We conduct an experiment over an emulate linear network with 10 switches. We let the controller inject barrier messages to each switch of the network and calculate the time difference between reception and sending of each message. The average time delay from the controller to the first switch measured by our LLDP based approach is about 0.3 ms, the average time delay from the second switch to the controller is about 0.5 ms, and the time difference between a time-stamped LLDP packet on receiving and on sending is about 1.338 ms. Therefore, the one way link latency between the first and second switches is $1.338 - (0.3 + 0.5) = 0.538ms$. To reduce the influence of the software fluctuation at the control plane, all the results used are the averaged latencies.

We also use "h1 ping -c 1 h2" inside Mininet to test the RTT between the first and second switches; the average half RTT is about 0.3 ms. In fact, the half RTT measured by ping should be larger than the real link latency due to the delay between hosts and switches. However, it is smaller than our measured latency when considering the delay between controller and switches. Using the half RTT measured by ping as the base line, we calculate the measurement error when omitting the time delay from the controller to the switches, the measurement error is 346% ($(1.338 - 0.3)/0.3 = 3.46$), while the measurement error when using barrier messages to measure the time delay from the controller is 79% ($(0.538 - 0.3)/0.3 = 0.79$). Therefore, our measurement exists serious measurement error, though the measurement error can be reduced when the real link latency of a network is increased.

B. LLDP Packets Injection Frequency

Frequently injecting the time-stamped LLDP packets to switches consumes the resources of the controller and the bandwidth of the control channel, and this in turn affects the

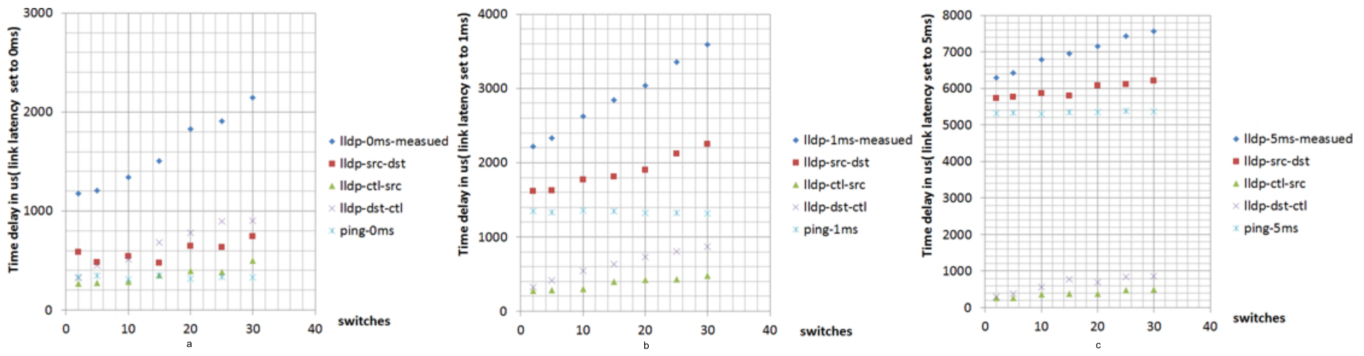

Fig. 1. The $t_{measure}$, $t_{ctltosrc}$, $t_{dsttoctl}$, and $t_{srctodst}$ comparing to the half RTT measured by ping. Fig. a, b, and c are the test results when the link latency is set to 0ms, 1ms, and 5ms, respectively.

TABLE I
LINK LATENCY USING VARIOUS LLDP PACKET INJECTING FREQUENCY

approach	1 packet/s	2 packet/s	10 packets/s
$t_{measure}$	2056 us	2148 us	2187us
$t_{srctodst}$	675 us	744 us	817 us
$t_{srctodstbyping}$	305 us	302 us	310 us

performance of the controller processing OpenFlow messages and probe packets, and hence the link latencies measured by our LLDP abased approach. We conduct an experiment that uses Mininet to emulate a network with 30 switches, and we let the controller inject 1, 2, and 10 time-stamped LLDP packets per second, and measure the $t_{measure}$, $t_{ctltosrc}$, and $t_{dsttoctl}$. We also use "h1 ping -c 1 h2" to measure the RTT between the first and second switches of the tested network. The same as last subsection, we measure multiple times and take the average results. The test results are listed in Table I. It is apparent that as the frequency increases, the $t_{measure}$, $t_{ctltosrc}$, $t_{dsttoctl}$, and $t_{srctodst}$ also increase, while the half RTT measured by ping is almost unchanged. Since the half RTT measured by ping does not involve the control plane, the overhead change in the control plane does not have any influence in the RTT measured by ping. It also implies the overhead change at the data plane does not really change the half RTT measured by ping as well as the latency measurement in our proposed approach. Therefore, the overhead change at the control plane decides the measured latency in our proposed approach.

C. Network Scale

We conduct an experiment that uses Mininet to emulate a linear network with 2, 5, 10, 15, 20, 25, and 30 switches. We use – link TC parameter provided by Mininet to set the link latency to be 0 ms, 1 ms, and 5 ms to emulate a network with very low link latency, a local area network, and a wide area network respectively. We let the controller inject barrier messages and time-stamped LLDP packets to the network with the frequency of 2 packets per second. We calculate $t_{measure}$ and $t_{srctodst}$, and compare them to the half RTT measured by " h1 ping -c1 h2".

As shown in Fig. 1, $t_{measure}$ increases as the number of switches in the network grows, while the half RTT measured by ping forms an horizontal line. Since the controller needs to send time-stamped LLDP packets to each switch in the network, the more switches has a network, the larger is the number of time-stamped LLDP packets that the controller needs to send per second, and the more overhead adds on the controller and the control channel. This increased overhead causes longer time cost for the controller to process an OpenFlow message or a packet. Therefore, $t_{measure}$, $t_{ctltosrc}$, and $t_{dsttoctl}$ increase as the number of switches in the network grows. The number of switches in the network does not affect the half RTT measured by ping because the control plane is not involved in this process.

Since $t_{ctltosrc}$, and $t_{dsttoctl}$ measured by barrier messages are not the real $t_{ctltosrc}$, and $t_{dsttoctl}$ that probe packets use, the $t_{srctodst}$ measured by our LLDP based approach can not keep horizontal like the half RTT measured by ping as the number of switches in the network grows. In fact, the difference between $t_{measure}$ and ping or between $t_{srctodst}$ and ping is increased as the number of switches in the network grows, especially when the link latency is configured to 1ms and 5ms. This implies the measurement error is increased as the network scale grows, and this measurement error can not be reduced using a calibration constant in a network, where the network scale may grow up or shrink down depending on the workload or requirements. However, this measurement error should be reduced by using a linear calibration function, because the $t_{measure}$ values are linear to the number of the switches in the network no matter the link latency is set to 0ms, 1ms, or 5ms. The $t_{srctodst}$ values in Fig. 1 do not form a nice line as the number of switches in the network grows when the link latency is set to 0ms due to the software fluctuation, but they do when the link latency is set to 1ms and 5ms. This implies the smaller the link latency is, the more difficultly this software fluctuation can be reduced (refer to the evaluation section for details).

VI. CALIBRATION

From last section, we know the change in network scale can increase the measured link latency when using our proposed

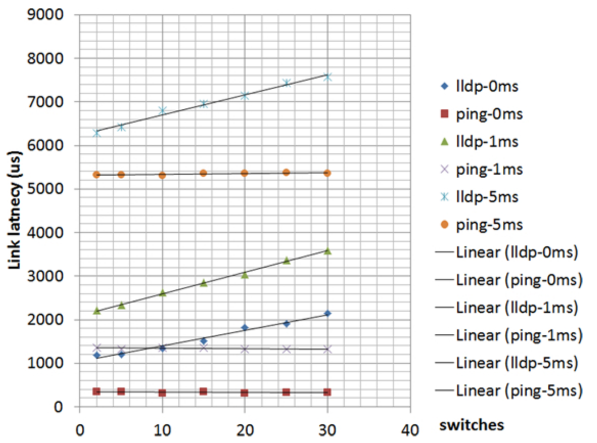

Fig. 2. The one way link latency between two adjacent switches. The latency measured by our LLDP based approach consists of the time delay from the controller to switches.

approach. To improve it, we choose the $t_{measure}$ rather than the $t_{srctodst}$ as the starting value to do the calibration, because the $t_{srctodst}$ measured by our approach includes the measurement error incurred by the controller-switch delay. As shown in Fig. 1, the $t_{measure}$ presents better linear relationship with the number of switches in the network than the $t_{srctodst}$. Our goal is to develop a linear function to calibrate the $t_{measure}$. We choose the $t_{measure}$ and the half RTT of ping from last experiments, as shown in Fig. 2, and only consider the change in network scale in our linear calibration. We measure the latency several times and take the average to reduce the impact of software fluctuation at the control plane.

We notice that the measured delays by our approach in a network when the link latency is set to 1 ms and 5 ms form two parallel lines. However, these two lines are not strictly parallel to the line formed by the measured delays when the link latency is configured to 0 ms. We therefore generate two different calibration functions, one for the network with link latency set to 0 ms, the other for the network with link latency set to 1 ms or 5 ms.

Equation (3) represents the line formed by the measured delays when the link latency is set to 0 ms, where x is the number of switches in the tested network and y is $t_{measure}$ as measured by our LLDP based approach. Using the averaged half RTT measured by ping as the base line, we calculate the parameters a, b, and c to be 35, 770, and 334 respectively. The half RTT measured by ping is slightly larger than a real link latency due the the delay between hosts and switches. However, the calibration function should be able to compensate it. We then generate the calibration function for the network with link latency set to 0ms as shown in (4). Using the same method, we generate the calibration function for the network with link latency set to 1 ms or 5 ms as shown in (5).

$$y - c = a(x - 2) + b \tag{3}$$

$$y_{calibration} = y - 35 * (x - 2) - 770 \tag{4}$$

$$y_{calibaration} = y - 50 * (x - 2) - 890 \tag{5}$$

Given a tested network, parameter c is critical to compute the parameters a and b. Parameter c can be determined by the link latency measured by ping. In practice, we may have to take some sample switches from different vendors, and use the ping utility to test the link latency among them to decide the parameter c. The limitation of our calibration is that using different linear functions to calibrate the results for a network with link latency less than 1 ms and a network with link latency greater than 1 ms. For a network with link latency less than 1 ms, our evaluation in next section shows our calibration can only improve the measurement accuracy to 80%. This suggests that the impact of software fluctuation at the control plane cannot be fully compensated by taking averaged results for such networks. For a network with link latency not less than 1 ms, our calibration can achieve at least 95% measurement accuracy (refer to next section for details). This implies the impact of the software fluctuation at the control plane is reduced by using the average results. Since the work load change in the data plane does not really change the measured latency in a network with light work load, given a certain latency monitoring frequency, the change in network scale is the only factor significantly affecting measured latency, and hence, our calibration function can be used in such a network in practice.

VII. Evaluation

This section evaluates the measurement accuracy of our proposed approach after calibration. We use (4) and (5) to calibrate $t_{measured}$, and compare it to the half RTT measured by ping when the link latency is set to 0 ms and 1 ms/5 ms, respectively, as shown in Fig. 3. The one way link latency measured by our proposed approach after calibration is from 284 us to 407 us, 1249 us to 1332 us, and 5276 us to 5509 us when the link latency is set to 0 ms, 1 ms, and 5 ms, respectively; while the half RTT measured by ping is from 312 us to 350 us, 1320 us to 1360 us, and 5320 us to 5380 us when the link latency is set to 0 ms, 1 ms, and 5 ms respectively. The measurement error is about $\pm$ 70 us, -70 us to -6 us, and -90 us to +170 us comparing to the half RTT by ping when the link latency is set to 0ms, 1ms, and 5ms, respectively. The measurement accuracy comparing to the half RTT measured by ping is greater than 80%, 95%, and 97% when the link latency is set to 0 ms, 1 ms, and 5 ms, respectively.

Though we have taken the average latency to reduce the influence of the software fluctuation, we are still not able to achieve high measurement accuracy when the link latency is set to 0ms. This suggests that the software fluctuation can be the major factor affecting the measurement accuracy for a low link latency network when using control plane to measure the network latency. Since the software fluctuation is caused by the CPU clock speed fluctuation, it is hard to find a pattern for it, and hence hard to calibrate it. However, as the link latency increases, the influence of the software fluctuation is decreased. Our evaluation shows that using our calibration function can reduce the measurement error to less than 5% when the link latency is set to greater than 1ms.

TABLE II
LINK LATENCY MEASUREMENT COMPARISON

Approach	Network	Topology	Switches	link latency set	Calibration	link latency by ping	measurement error
LLDP based	Mininet	linear	up to 30	0ms	calibration function	0.3ms	20%
LLDP based	Mininet	linear	up to 30	1ms	calibration function	1.3ms	5%
LLDP based	Mininet	linear	up to 30	5ms	calibration function	5.3ms	3%
Probe packet [4]	Mininet	linear	2	0, 10ms, 20ms, 30ms	calibration constant	not provided	0.88%
TTL-looping [8]	physical	not linear	4	no set	no calibration	20-30 us	25%

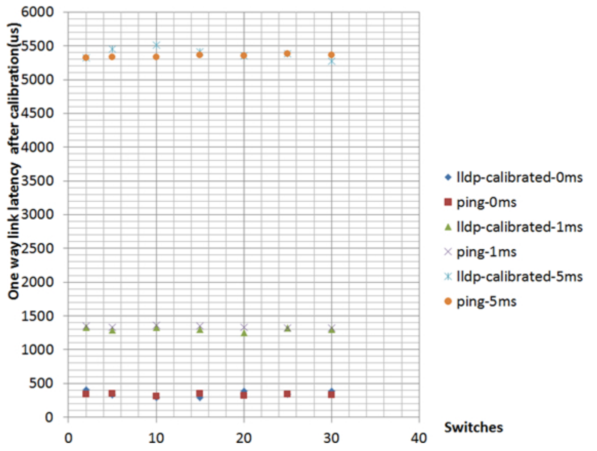

Fig. 3. The one way link latency from two adjacent switches after calibration

We also compare our measurement errors to the ones of some currently proposed approaches, as shown in Table II. The measurement proposed by [4] can achieve up to 99% of measurement accuracy after calibration, while the measurement accuracy of our LLDP based approach after calibration is 80%, 95%, and 97% comparing to the half RTT measured by ping when the link latency is set to 0ms, 1ms, and 5ms, respectively. However, our calibration function fits a network with up to 30 switches, while the calibration constant of the probe packet approach proposed by [4] can only used in a network with 2 switches. The TTL-looping approach proposed by [8] can reduce its measurement error to -7 us compared with 20-30 us' half RTT measured by ping without calibration, while our approach after calibration has a ± 70 us measurement error compared with 0.3 ms of the half RTT measured by ping when the link latency is set to 0 ms. It should be noticed that the one way link latency measured by our approach is 10 times greater than the one measured by TTL-looping, because the TTL-looping approach measures the link latency going through switches' fast channel, while our approach measures the link latency going through the switches' slow channel [12]. The TTL-looping has a measurement inaccuracy up to 25%, while our proposed measurement has a measurement inaccuracy up to 20% comparing to ping when the real link latency in a network is set to 0ms. This implies that using controller to inject probe packets to accurately measure the link latency of a network with the link latency less than 1 ms may not be feasible.

VIII. CONCLUSION AND FUTURE WORK

We have proposed a link latency measurement method for SDNs using LLDP packets. By analyzing the fundamental idea of using the control plane to monitor the network latency of a SDN, we have highlighted three major issues in current SDN link latency measurements, and shown how our LLDP based approach can address them in a network emulated by Mininet with a light workload. We have conducted experiments to show the major factors affecting measured results, and developed a linear calibration function to improve the measurement accuracy. We believe our linear calibration function with reconfigured parameters can be used in a real SDN with link latency not less than 1 ms.

ACKNOWLEDGEMENT

This research work is supported by the Canadian Natural Sciences and Engineering Research Council through grant STPGP 447230.

REFERENCES

[1] Chuck Fraleigh, C Diot, B Lyles, etc. *Design and deployment of a passive monitoring infrastructure*. Evolutionary Trends of the Internet. Springer Berlin Heidelberg, p 556-575, 2001.
[2] A. Caida Ma, *tools: measurement: skitter*. http://www.caida.org/tools/measurement/skitter/
[3] OpenFlow Switch Consortium. *OpenFlow Switch Specification Version 1.0. 0.* (2009). Internet: archive.OpenFlow.org/documents/OpenFlow-spec-v1.0.0.pdf
[4] Kevin Phemius and Mathieu Bouet. *Monitoring latency with openflow*. Network and Service Management (CNSM), 9th International Conference on. IEEE, 2013.
[5] Niels L. M. van Adrichem, Christian Doerr, and Fernando A. Kuipers. *Opennetmon: Network monitoring in openflow software-defined networks*. In Network Operations and Management Symposium (NOMS), pp. 1-8. IEEE.2014
[6] IEEE standards association, *Link Layer Discovery Protocol*. http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf
[7] Bob Lantz, Brandon Heller, and Nick McKeown. *A network in a laptop: rapid prototyping for software-defined networks*. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010.
[8] V. Altukhov and E. Chemeritskiy. *On real-time delay monitoring in software-defined networks*. Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 International. IEEE, 2014.
[9] Sinha Debanshu Sinha, K. Haribabu, and Sundar Balasubramaniam. *Real-time monitoring of network latency in Software Defined Networks*. 2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS). IEEE, 2015.
[10] Curtis Yu, C Lumezanu, A Sharma et al. *Software-defined latency monitoring in data center networks*. Passive and Active Measurement. Springer International Publishing, 2015.
[11] Nicira Networks, *NOX Network Control Platform*. https://github.com/noxrepo/nox-classic
[12] Yu M, Rexford J, Freedman M J, et al. *Scalable flow-based networking with DIFANE*. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 351-362.