BFD-based Link Latency Measurement in Software Defined Networking

Seong-Mun Kim*, Gyeongsik Yang†, Chuck Yoo†, and Sung-Gi Min*
*Department of Computer and Radio Communication Engineering, Korea University, Seoul, South Korea
Email: soulcrime@korea.ac.kr, sgmin@korea.ac.kr
†Department of Computer Science and Engineering, Korea University, Seoul, South Korea
Email: ksyang@os.korea.ac.kr, chuckyoo@os.korea.ac.kr

***Abstract*—5G networks offer various network services based on software defined networking and network function virtualization. However, certain services are sensitive to link latency which is why it is consistently observed to provide high quality services. Previous studies have proposed two approaches to this task: measuring the latency by probe packets and link-layer discovery protocol (LLDP) packets. However, they have several limitations like flow rule preconfiguration, influence of the control plane traffic, and necessity of calibration. In this paper, Bidirectional forwarding detection (BFD) based approach is proposed. The approach measures latency at the data plane with simply implemented echo mode in Open vSwitch. We evaluates and compare the proposed approach to LLDP-based one in terms of single link latency and path latency, and error rate. In addition, we verify that the control plane throughput affects link latency according to the increased number of switches. As a result, the proposed approach can resolve the limitations and provides accuracy link latency.**

I. Introduction

Recently, Software Defined Networking(SDN) and Network Function Virtualization(NFV) have received attention as foundation techniques for 5G network. 5G requires network slicing based on SDN and NVF to provide various services. Several services, such as mission-critical IoT and realtime streaming, are very sensitive to network latency. Thus, network latency monitoring is continuously and accurately required for all links of the network.

Traditional network has an architecture with coupled data and control planes, and distribution of network functions. The architecture has limitations such as network management and open networking. In contrast, SDN is decoupling control and data planes. It provides direct programmability, centralized network management, open standards ,and vendor neutralization. SDN consists of a controller, switches, and a southbound interface for communication between the controller and switches as OpenFlow protocol [1].

There are two proposed approaches to measure link latency. The first approach uses probe packets (Probe) [4]-[8]. A controller inserts Probe with sending timestamp (T_{SEND}) to the switch of the data plane. The inserted switch is denoted SrcSW. Probe is traversed links and one of switches returns it to the controller. The switch returning the Probe to the control plane is denoted DstSW. At that time, the controller is able to calculate the latency $T_{MEASURE}$ ($T_{RECV} - T_{SEND}$). $T_{MEASURE}$ includes the latency $T_{SWtoCTL}$ between the controller and the switches. It also includes OpenFlow message processing latency T_{PROC}. However, Probe approach does not offer the method of measurement $T_{SWtoCTL}$. It makes additional overhead at the control plane because Probe is additionally built and OpenFlow message carries with it in order to inject to the data plane. The important issue is that the entry of flow table should be preconfigured for Probe forwarding.

The second approach uses Link-layer Discovery Protocol (LLDP) [2] packets identical to Probe [10]. A controller periodically sends a LLDP packet included in a PACKET_OUT message (POUT). The LLDP packet carries T_{SEND} in an optional TLV and $T_{MEASURE}$ is calculated. Unlike Probe approach, this approach is able to obtain $T_{SWtoCTL}$ with the help of OpenFlow messages. Thus, T_{LINK} is computed as follows: ($T_{RECV} - T_{SEND} - 2T_{SWtoCTL}$). The advantage of this approach has less overhead at the control plane because it does not need to build an additional Probe. The entry of flow table is not required to forward LLDP packet. However, $T_{MEASURE}$ increases according to the control plane throughput. Increasing the number of switches leads to more overhead and delay at the control plane as the controller handles more control messages. It causes critical error rate to measure link latency. Consequentially, this approach has to need calibration to reduce error rate.

In this paper, we propose a Bidirectional Forwarding Detection(BFD)-based Link Latency Measurement technique. BFD is designed to detect physical link failure and it is supported in Open vSwitch (OVS) which is the most popular software switch [14][3]. However, it is impossible to measure link latency with OVS because OVS only supports asynchronous mode. This being so, we design and implement echo mode in BFD module of OVS. It makes OVS possible to measure link latency at the data plane. Measured link latency included in LLDP packets is forwarded to the controller. The advantage of BFD-based approach is avoidance the influence of overhead at the control plane. Also, the approach provides precise link latency without calibration.

The rest of this paper is organized as follows. Section II summarize the related works. BFD-based link latency measurement is introduced in Section III. In Section IV, we

978-3-901882-98-2

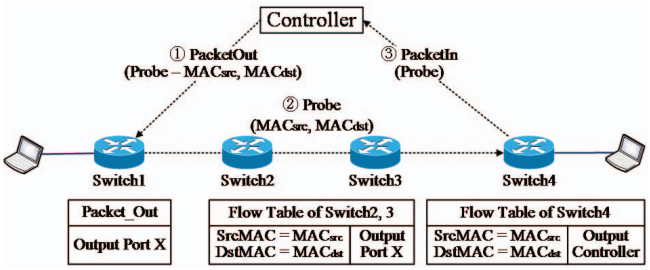

Fig. 1. Example of Probe-based Measurement

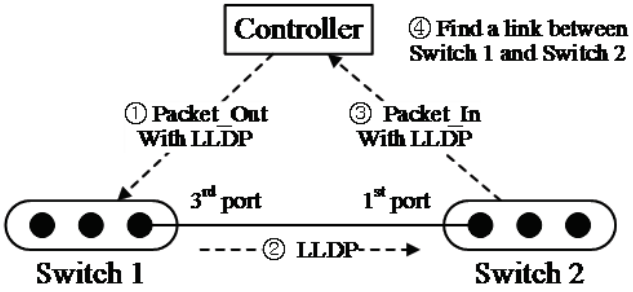

Fig. 2. Topology Discovery in SDN

compare the proposed approach and LLDP-based one through experiments with OVS and Floodlight [15], and Section V concludes this paper.

II. Related Works

A. Probe-based Measurement

Probe-based measurement techniques [4][5] insert a Probe to the data plane to measure the latency of all paths in SDN. In [4], Ethernet frame carries timestamp, and calibration is developed to enhance accuracy. Real data packet is used like Probe to carry timestamp in [5]. This approach targets the network having long route latency to ignore measurement error. Both approaches uses building Probe, while they do not consider overhead at the control plane.

[6] and [7] targets the network with low link latency. They measure path latency, not a single link. A controller inserts a Probe including timestamp T_{SEND} to SrcSW of the path. The Probe is traversed through the path, and DstSW of the path return it to the controller. The controller can get the link latency which is the path latency $T_{RECV} - T_{SEND}$ divided by hop count. In addiotn, [8] is same approach. It monitors continuously the network latency by inserting data packet with sending timestamp. Its measurement frequency is based on the flow table entry which has valid active time. The frequency is unable to be altered.

However, those approaches require the additional flow table entries of the switches on all paths in order to forward Probe as illustrated in Fig. 1. For example, in order to forward Probe, switches on the path need two type flow table entries at least. The controller injects a POUT with the time-stamped Probe. The Probe is sent by the output action of the POUT through the port X (arbitrary port number). Thus, Switch1 does not need a specific flow table entry. But, Switch2 and Switch3 require the specific entries to identify the Probe, assuming that the Probe is identified based on MAC addresses. If the MAC addresses of the Probe are MAC_{src} and MAC_{dst}, the match field of flow table entries of Switch2 and Switch3 are set like (Source MAC Address = MAC_{src}), (Destination MAC address = MAC_{dst}), and (Output Port = X). In case of Switch4, it is the edge switch. The Probe has to be returned to the controller at Switch4. Therefore, Output Port is set Controller [1] different from Switch2 and Switch3. As a result, for transmission Probe, additional flow table entry is set to switches on the path except SrcSW. Probe-based approaches need more resources of the switches due to building Probe and setting the path for Probe.

B. LLDP-based Measurement

In SDN, a controller needs to discover the whole network topology in order to decide how packets are delivered through the appropriate port of a switch. The controller uses LLDP with minor modifications [9]. The topology discovery is described in Fig. 2. The controller periodically sends a LLDP packet included in a POUT to all connected switches. The controller uses POUT message to insert a packet to the data plane. When Switch1 receives POUT with the LLDP packet, Switch1 sends the LLDP packet out to all ports. The LLDP packet is sent from the 3rd port of Switch1. The LLDP packet is traversed a link and reaches Switch2 through the 1st port. Switch2 returns the LLDP packet contained in a Packet_In (PIN) which is used to forward a packet to the controller. At receiving the LLDP packet, the controller is able to find link information between Switch1 and Switch2. As a result, the controller recognizes the whole network topology.

LLDP-based approach uses a LLDP packet like Probe [10]. A controller stores T_{SEND} (ms) in the System Capabilities (SC) TLV option (4bytes) of the LLDP packet when it starts topology discovery. At receiving the LLDP packet, the controller calculates $T_{MEASURE}$ ($T_{RECV} - T_{SEND}$). However, $T_{MEASURE}$ includes $2T_{SWtoCTL}$ which is latency between the controller and SrcSW/DstSW mentioned in Section I. $T_{SWtoCTL}$ is able to be obtained from difference between transmission Barrier Request and reception Barrier Reply messages in [10]. Barrier messages is exchanged when the controller wants to ensure message dependencies have been met or wants to receive notifications for completed operations [1].

Latency accuracy is verified by comparing $T_{MEASURE}$ and PING half RTT. Latency and half RTT are measured with the linear topology consisted of two switches and two hosts. In the simulation, average half RTT is 0.3 ms and $T_{MEASURE}$ is 1.338 ms. The delay between the controller and the first switch is 0.3 ms, and $T_{SWtoCTL}$ between the controller and the second switch is 0.5 ms. Measurement error is 346% ((1.338 âĹŠ 0.3)/0.3 = 3.46) when $2T_{SWtoCTL}$ is not considered. If is considered, T_{LINK} is 0.538ms (1.338 -

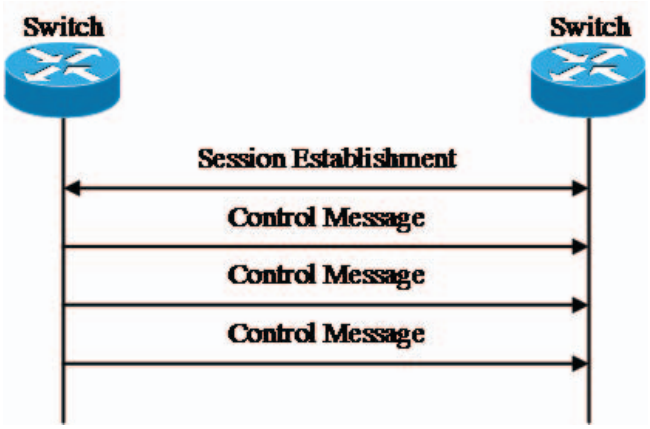

Fig. 3. Asynchonous Mode in Open vSwitch

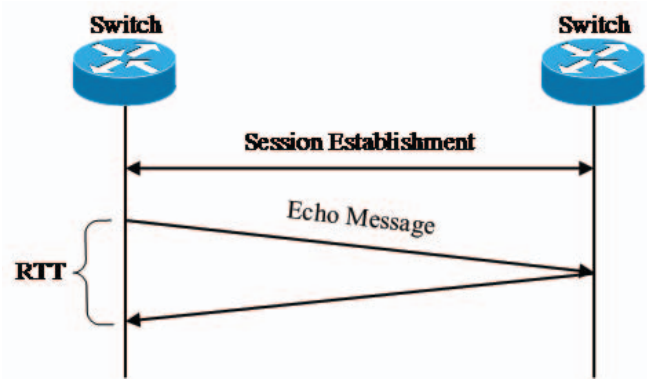

Fig. 4. Echo Mode in Open vSwitch

0.3 - 0.5). Measurement error is 79% ((0.538 âĂŞ 0.3)/0.3 = 0.79). Measurement error is caused by irregular $T_{SWtoCTL}$.

Thus, calibration is required to reduce measurement error. [10] shows that $T_{MEASURE}$ linearly increases according to the number of switches. Linear function is generated as follows:

$$y - c = a(x - 2) + b, \tag{1}$$

where x is the number of switches and y is $T_{MEASURE}$. a, b, and c are obtained through calculation based on PING half RTT. For example, when the simulation is performed, the link delay of Mininet [16] is set 0 ms. In this case, a, b, and c is 35, 770, and 334, respectively. Calibration $T_{MEASURE}$ is expressed as follows:

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

This way, the linear function is generated and it is used to reduce measurement error and to enhance accuracy. However, this approach has a condition that PING RTT is measured at the host. Half RTT is the base line on which to generate the linear function. Moreover, it does not consider irregular link latency according to the network state. If it is considered, this approach would need some interfaces or procedures to reflect dynamically changed PING RTT.

III. BFD-based Link Latency Measurement

In this section, we introduce BFD-based link latency measurement and how to operate it with Echo mode in OVS.

A. Bidirectional Forwarding Detection

Generally, networks are designed having redundant backup links to protect the important applications. Network devices have to quickly forward packets through backup links after detecting link failure. Bidirectional Forwarding Detection (BFD) is proposed for fast detection [11][12]. BFD detects the point-to-point failure of links by regularly sending control messages.

BFD has two operation modes, Asynchronous (Async) mode and Demand mode. In Async mode, network devices periodically forwards control messages to a system. If the system do not receive many control messages, the device should declare happened link failure as shown in Fig.3. Async mode is only implemented in OVS. A system is independent for verification of connections to other system in Demand mode. BFD is used for additional means. The system does not send control messages after BFD session establishment except when the system is not sure that the link connection is active.

Echo mode is used with Aync mode and Demand mode simultaneously. After session establishment, a transmission system sends an echo message and a reception system returns the message to the transmission system as illustrated in Fig. 4. The payload is local matter because echo message is always processed in the transmission system. For the proposed scheme, we implement Echo mode in OVS.

B. BFD Echo Mode in Open vSwitch

BFD is implemented in OVS to detect link failure instead of a well known Connectivity Fault Management (CFM)[13]. OVS periodically sends BFD control messages at a rate negotiated independently in each direction. Each endpoint designates the rate at which it expects receiving control messages, and the rate at which it sends them. In order to detect, OVS uses a detection multiplier of three. It means that an endpoint notifies a link fault if three successive BFD control messages do not reach. In OVS, the implementation of BFD targets to observe the standard [11].

However, OVS does not implement Echo mode. We implement Echo Mode of BFD for the proposed measurement. Echo message transmission frequency is the same as the one of BFD Async mode. After BFD session establishment, echo messages are sent by the monitor thread of the BFD enabled ports. Echo message and BFD control message are registered in the special process list to avoid flow table pipeline processing. Echo message is simply returned to a transmitting OVS or is processed in the OVS. Echo messages are only exchanged between neighbor OVSs. The transmitting OVS is able to measure T_{LINK} from Echo message half RTT.

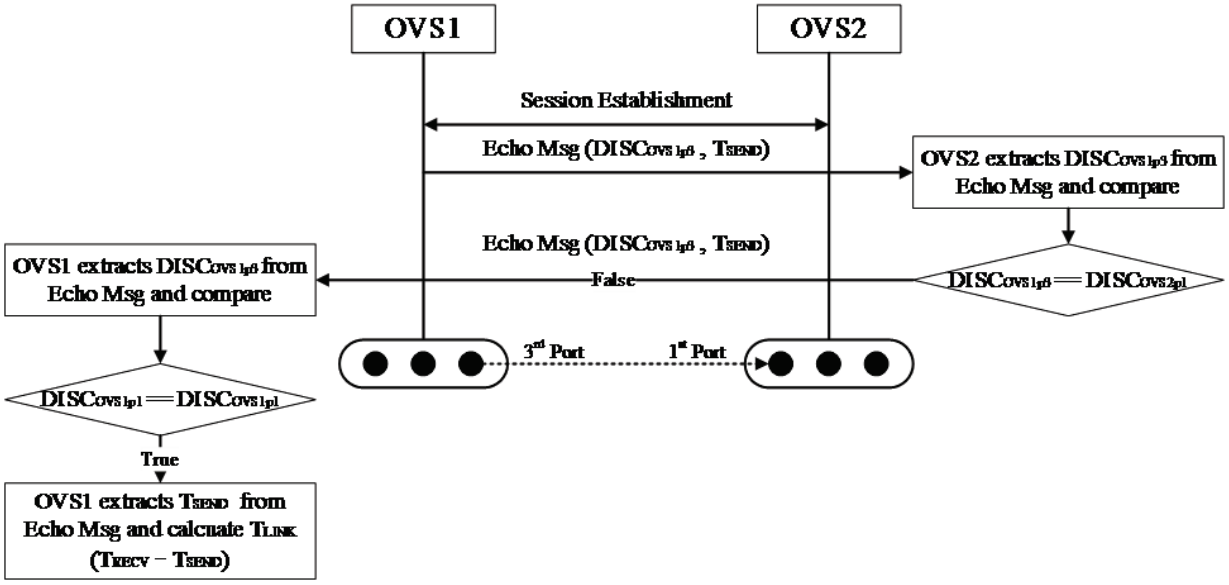

Fig. 5. Echo Mode Operation in Open vSwitch

C. Echo Message Format

Basically, the payload of echo message is local matter. We define Echo message having two parameters, Discriminator (4bytes) and Sending Timestamp (4bytes). In BFD, all ports enabled BFD of OVS have each other different nonzero discriminator (DISC) value generated by the transmitting OVS. OVS is able to classify who transmits echo message based on discriminator. For example, the transmitting OVS compares its own discriminator to the discriminator in received echo message. If they are same, the OVS processes the echo message. If not, the echo message should be returned to the transmitting OVS. Echo message also has to include sending timestamp T_{SEND} in order to calculate T_{LINK} when the transmitting OVS receives the returned echo message. It is noted that regarding the overhead in the data plane caused by echo message exchange, it is 0.00001% per sec for 1G link. Thus, the overhead in the data plane is negligible.

D. Echo Mode Operation in Open vSwitch

Echo mode operation is described as Fig. 5. There are OVS1 and OVS2, and they are adjoining. 3^{rd} Port of OVS1 is connected to 1^{st} Port of OVS2. It is assumed that echo mode is enabled at the ports. The discriminator of 3^{rd} Port of OVS1 and the one of 1^{st} Port of OVS2 are $DISC_{OVS1p3}$ and $DISC_{OVS2p1}$, respectively.

After session establishment, OVS1 transmits a echo message including $DISC_{OVS1p3}$ and T_{SEND} to OVS2. When OVS2 receives the echo message, OVS2 extracts $DISC_{OVS1p3}$ and compare $DISC_{OVS1p3}$ to its own $DISC_{OVS2p1}$. However, they are different from each other. OVS2 returns the echo message to the OVS1. OVS1 extracts $DISC_{OVS1p3}$ and compare $DISC_{OVS1p3}$ to its own discriminator. They are identical discriminators, and OVS1 is able to obtain T_{LINK} (T_{RECV} - T_{SEND}). T_{LINK} is stored to **netdev** structure for the port of OVS.

TABLE I
SERVER SPECIFICATION

CPU	Intel Xeon ES-2650 v3@2.30GHz (10-Core, x86-64) * 1
Memory	64GB (DDR4 ECC 16GB * 4)
NIC	Inte 82599 ES, 10GB
OS	Ubuntu 14.04
Linux Kernel	3.16.0

E. Latency Delivery

T_{LINK} stored in OVS has to be forwarded to the controller. To achieve that, we use a LLDP packet with a SC TLV option as similar concept of [10]. When the controller inserts the LLDP packet to the data plane, it uses a POUT. The POUT carries the LLDP packet in the data field with an output action command. The output action specifies the port which the LLDP packet is transmitted through.

After receiving OpenFlow messages (POUT), OVS decodes the messages to process for the goal of each message. If the reception message is POUT, OVS checks whether the action in POUT is the output action. If it is true, OVS calls **netdev_send** function to send the data included in the data field of POUT.

In **netdev_send** function, OVS saves the link latency to the SC TLV of the LLDP packet before sending the LLDP packet. First, OVS checks whether the link latency is zero to prevent unnecessary processing. If it is not zero, OVS verifies that the data is the LLDP packet. If it is the LLDP packet, OVS examines that the LLDP packet has the SC option TLV. If it has SC TLV option, OVS puts T_{LINK} to the option. The LLDP packet is sent to a neighbor OVS, and the neighbor OVS is forwarding the LLDP included in PIN. The controller can obtain T_{LINK} between the OVS and its neighbor OVS.

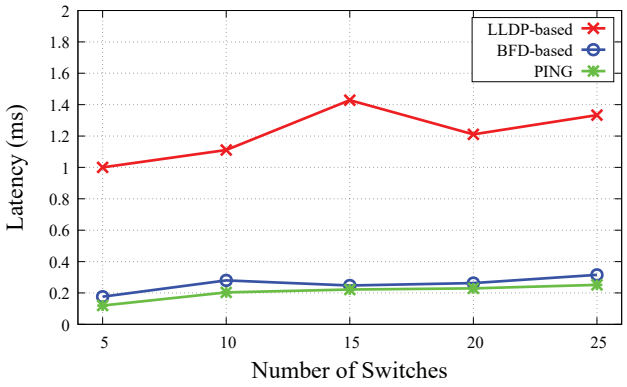

Fig. 6. Single Link Latency (ms)

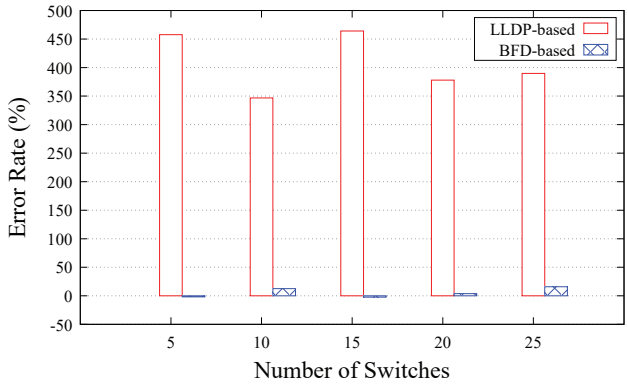

Fig. 7. Single Link Latency Error Rate (%)

IV. Evaluation

In this section, we have experiments with Mininet [16], OVS [14], and Floodlight [15]. Each application is installed in different servers. The specifications of server is shown in Table. I. We compare LLDP-based approach and the proposed approach on the basis of PING half RTT without calibration. The simulation topology is built linearly and the number of switches increases from 5 to 25 by 5. Two hosts are connected to both endpoints.

Basically, LLDP-based link latency measurement approach is implemented in Floodlight version 1.2. Floodlight measures $T_{SWtoCTL}$ when Echo Request/Reply and Features Request/Reply are exchanged between the controller and switches. $T_{SWtoCTL}$ is stored in **Connection Class** managing a connection for the switch. Floodlight stores $T_{SEND} + T_{SrcSWtoCTL}$ to the optional TLV of a LLDP packet and inserts the packet to the data plane. $T_{SrcSWtoCTL}$ is latency between the controller and the switch that the LLDP packet will be inserted. When the LLDP packet is returned to the controller, the controller adds $T_{DstSWtoCTL}$, which is latency between the controller and the switch returning the LLDP packet, to $T_{SEND}+T_{SrcSWtoCTL}$. We implement Echo mode in BFD module of OVS to measure T_{LINK} for the proposed approach. In order to deliver T_{LINK}, SC optional TLV is added to the LLDP packet built in Floodlight. As a result, Floodlight obtains T_{LINK} as follows:

$$T_{RECV} - (T_{SEND} + T_{SrcSWtoCTL} + T_{DstSWtoCTL}). \quad (3)$$

A. Single Link Latency

We measure the single link latency of LLDP-based and BFD-based measurements, and compare them to PING half RTT. Half RTT is the average latency of 5 times measurement. Its single link latency is calculated as (half RTT / total links). Total links includes the number of links between hosts and switches. In case of LLDP-based measurement, the frequency of LLDP packet transmission is set 15sec and the latency is the average value of 3 times measurement of all links. The frequency of echo message transmission is set 1sec and the measured T_{LINK} is delivered to the controller at the period of sending LLDP packet. The single link latency of BFD-based measurement is the average value of all links.

LLDP-based measurement is higher than BFD-based one as shown in Fig. 6. The reason is that the former includes OpenFlow message processing time. Moreover, the overhead of the control plane influences calculating $T_{SWtoCTL}$. In contrast, the latency of BFD-based measurement is proximity to half RTT because this approach is not affected by the overhead of the control plane.

We calculate the error rate of both approaches on the basis of half RTT regarding the single latency as shown in Fig. 7. In case of LLDP-based measurement, error rate is too high. On the other hand, some error rates of BFD-based measurement are lower than 0%, meaning that the single link latency of BFD-based measurement is lower than the one of half RTT. Thus, in theses cases, we can regard that the single link latency of half RTT and BFD-based measurement is almost identical. When the number of switches is 10, 20, and 25, the min and max error rate of BFD-based measurement are 3.75% and 15.83%, respectively. But, real gap is very small such as 0.28ms, 0.18ms, and 1.03ms. As a result, if LLDP-based measurement is used, calibration should be performed to reduce error rate. However, BFD-based measurement shows low error rate without calibration, meaning that the proposed measurement provides high accuracy.

B. Linear Path Latency

In order to measure linear path latency, we set the topology with two hosts and several switches. The path latency of LLDP-based and BFD-based approaches is the sum of link latency between two edge switches connected to the hosts. On the contrary, in case of half RTT, path latency is following as: (half RTT / (total links - 2)), meaning that two links between the hosts and the edge switches are excluded. The path latency of BFD-based measurement is similar with the one of half RTT. However, the path latency of LLDP-based measurement steeply increases according to the increased number of switches due to the overhead affect of the control

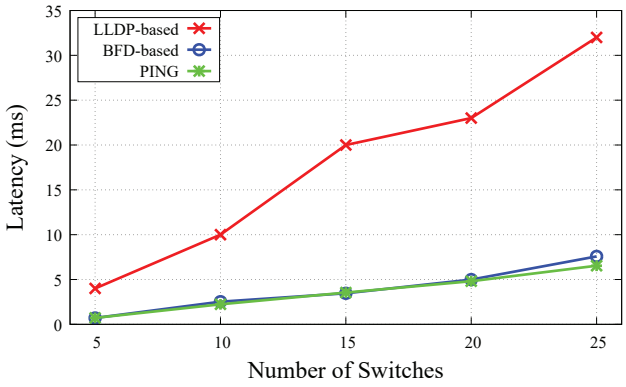

Fig. 8. Linear Path Latency (ms)

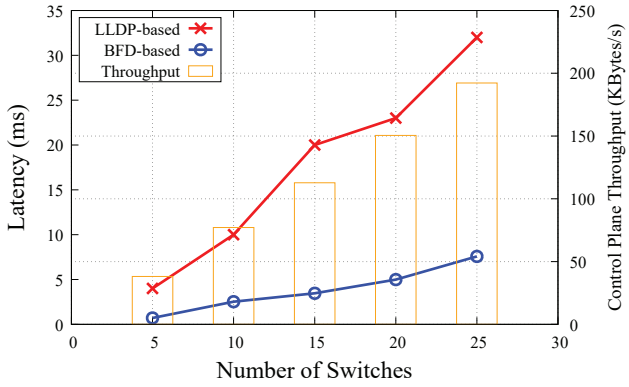

Fig. 9. Control Plane Throughput

plane as shown in Fig. 8. We shall analysis the cause of this situation in Section C.

C. Affect of Control Plane Throughput

In order to verify the affect of the control plane, we measure the throughput of the control plane according to the increased number of switches. The throughput is the average value during the link latency measurement. Fig. 9 shows the path latency of each approach and the throughput of the control plane concurrently. The increase rate of LLDP-based approach is more than the one of the throughput of the control plane. In contrast, the increase rate of BFD-based approach is less than the one of the throughput. It means that LLDP-based measurement is influenced by the throughput of the control plane. According to the increased number of switches, the number of OpenFlow messages increase. The controller frequently sends statistic request messages to recognize the situation of connected switches about flows, ports, groups, meters, etc. Then, the switches response by sending statistics reply messages. More messages influence to the control plane condition, and it makes increasing the workload of the controller. Consequently, LLDP-based approach is not able to provide accurate link latency without calibration.

V. Conclusion

BFD-based link latency measurement is proposed in this paper. Previous approaches, such as Probe-based and LLDP-based measurements, have limitations like the preconfiguration of flow tables to forward Probe, the influence of the control plane throughput, and the necessity of calibration. However, BFD-based measurement resolve the limitations. Echo mode is simply implemented in BFD module of OVS. The measured link latency is delivered by using the LLDP packet. Also, the proposed approach is not influenced by the throughput of the control plane. As a result, BFD-based approach is able to provide accuracy link latency without calibration.

Acknowledgment

This research was supported by the MIST(Ministry of Science and ICT), Korea, under the National Program for Excellence in SW)(2015-0-00936) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] Open Networking Foundation, “OpenFlow Switch Specification Version 1.5.0 (Wire Protocol 0x06),” Dev 2014.
[2] IEEE 802.1AB - 2009, “IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery,” September 2009.
[3] H. Chiu, and S. Wang, “Boosting the OpenFlow control-plane message exchange performance of OpenvSwitch,” Communications (ICC), 2015 IEEE International Conference on, June 2015.
[4] K. Phemius, and M. Bouet, “Monitoring latency with openflow,” Network and Service Management (CNSM), 9th International Conference on, October 2013.
[5] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, “OpenNetMon: Network monitoring in Openflow Software-Defined Networks,” Network Operations and Management Symposium (NOMS), pp. 1-8, May 2014.
[6] V. Altukhov, and E. Chemeritskiy, “On real-time delay monitoring in software-defined networks,” Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), October 2014.
[7] S. D. Sinha, K. Haribabu, and S. Balasubramaniam, “Realtime monitoring of network latency in Software Defined Networks,” IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS), December 2015.
[8] C. Yu, C. Lumezanu, A. Sharma, and H. V. Madhyastha, “Software-Defined Latency Monitoring in Data Center Networks,” Passive and Active Measurement, Springer International Publishing, March 2015.
[9] OpenFlow, Japan Network Information Center (JPNIC) News letter, No 52, pp 38-41, November 2012.
[10] L. Liao, and V. C.M. Leung, “LLDP Based Link Latency Monitoring in Software Defined Networks,” Network and Service Management (CNSM), 2016 12th International Conference on, October 2016.
[11] D. Katz, D. Ward, and Juniper Networks, “Bidirectional Forwarding Detection (BFD),” RFC 5880 (Informational), June 2010.
[12] D. Katz, D. Ward, and Juniper Networks, “Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop),” RFC 5881 (Informational), June 2010.
[13] IEEE 802.1ag - 2009, “IEEE Standard for Local and metropolitan area networks Virtual Networks Amendment 5: Connectivity Fault management,” September 2007.
[14] http://openvswitch.org/, Accessed July 2017.
[15] http://www.projectfloodlight.org/floodlight/, Accessed July 2017.
[16] M. Team, “Mininet: An instant virtual network on your laptop (or other pc),” 2012.