Efficient Performance Diagnosis in OpenFlow Networks Based on Active Measurements

Megumi Shibuya, Atsuo Tachibana and Teruyuki Hasegawa
KDDI R&D Laboratories, Inc.
Saitama, JAPAN
{shibuya, tachi, teru}@kddilabs.jp

***Abstract*—To detect performance degradation and identify causal links promptly in OpenFlow networks, this paper presents the design of a scheme to actively measure the performance of all physical links from a single measurement point based on the controllable feature of individual traffic flow provided in OpenFlow. In the scheme, the measurement paths from a measurement box (called a "beacon") are calculated to comprehensively cover all links and the probing packets are transmitted along elaborately defined routes in cooperation with the OpenFlow controller and multiple switches. Diagnosing the link performance of an entire OpenFlow network with a single beacon is expected to reduce operational costs, such as deployment and maintenance costs of beacons. In addition, reducing the number of flow-entries for active measurements on OpenFlow switches is considered to save the resources of OpenFlow switches. The effectiveness and feasibility of our solution are demonstrated through model emulations.**

Keywords- OpenFlow; Active measurement; Network Diagnosis

I. INTRODUCTION

Software Defined Networking (SDN) architectures that offer flexible traffic control based on programmable network functionalities have been studied. OpenFlow [1] is a major example of the SDN architectures used to control the traffic flows of multiple switches from a centralized controller. In order to provide high quality services to customers over OpenFlow networks, it is important for network operators to manage the networks in a reliable and efficient manner, and performance characteristics and states of network links should be measurable and tractable.

Unfortunately, standard management methods, such as periodically monitoring all network-internal devices using the Simple Network Management Protocol (SNMP), are not always effective for promptly detecting network performance degradation. For example, it is not easy to detect performance degradation of packet delay by analyzing the set of status and event information monitored at individual devices inside the network. In addition, the failure of SNMP-based detection is potentially caused by bugs in router/switch software and network misconfiguration [2]. Accordingly, measuring link performance by actively probing the network is important as an alternative and/or complementary method to detect performance degradation.

With the aim of identifying the root cause of performance degradation in conventional networks, a variety of network diagnostic tools that estimate link-by-link performance characteristics, such as distribution of delays, loss rate and bandwidth, have been developed by exploiting the ICMP responses generated by routers to specially crafted probing packets. However, the probing packets are usually transmitted along pre-defined paths (the shortest paths in most cases), and hence, to comprehensively monitor the entire network, multiple measurement boxes (hereafter, called "beacons") need to be distributed [3][4]. Therefore, it is not always cost efficient from the viewpoint of network operators [5].

In this paper, we propose a diagnosis scheme that utilizes the controllable feature of individual traffic flow provided in OpenFlow networks and conducts an existing active measurement tool to comprehensively monitor the link performance in the OpenFlow network by using a single beacon in cooperation with the OpenFlow controller and multiple switches. In the proposed scheme, each OpenFlow switch maintains some exclusive flow-entries dedicated to active measurements in addition to the user's data traffic flows so that measurement paths (flows) cover all links in the network. According to the flow-entries at each switch, the probing packets are forwarded to the designated physical links and eventually return to the beacon. The beacon estimates link-by-link performance characteristics by analyzing the difference between probing packets returned from different switches based on an existing (*traceroute*-like) measurement technique.

The contributions of this work are two-fold. First, we present the design of a diagnosis scheme to actively measure the performance of all physical links from a single measurement point in the OpenFlow network. Note that we currently use a traceroute-like measurement technique, but we believe we could use other active measurement tools without difficulty. Second, we propose a method to reduce the number of exclusive flow-entries for active measurements on OpenFlow switches by aggregating multiple entries into a single entry by applying common packet header options to the probing packets. Because these excursive flow-entries themselves consume the limited resources of flow table memory of the OpenFlow switch, the overhead flow-entries should be minimized. The feasibility and effectiveness of the proposed method are verified through emulations with Trema [6].

This paper is organized as follows. Section II shows related work on the existing network measurement techniques. Section III explains the proposed measurement scheme on how to define the measurement paths using only one beacon. In Section IV, we evaluate and discuss the effectiveness and the feasibility of the proposed scheme through the emulations. Finally, we conclude the work in Section V.

II. Related Work

A variety of network diagnostic tools that estimate link-by-link network characteristics, such as loss rate, distribution of delays, and bandwidth, along a network path have been extensively developed by sending probing packets over ICMP or specially crafted TTL-limited probing packets, and exploiting the responses generated by routers along the measurement paths (e.g., traceroute command, *Pathneck* [3], *cing* [7], and *BFind* [8]). Network tomographic approaches have also been proposed to infer the performance characteristics of the network interior accurately and/or in identifying degraded links solely based on the correlation among multiple end-to-end path measurements (e.g., a survey [9]). In these studies aiming at conventional networks, each probing packet is assumed to pass though pre-defined paths (the shortest paths among sender and receiver beacons in most cases) and hence, multiple beacons are required to comprehensively monitor the entire network. To effectively monitor large-scale networks, good placement of beacons (a minimal set of beacons) and measurement paths among them is considered with some operational constraints (e.g., [10][11]). For example, in reference [10], selecting optimal sets of active measurement paths to cover all the links is considered with the measurement load constraints and installation costs. However, to the best of our knowledge, few researches have been conducted on the design and development of efficient solutions to promptly detect performance degradation and identify causal links in OpenFlow networks.

III. Proposed Method

A. Calculating Measurement Paths

Unlike conventional Layer 2 and 3 switches, the OpenFlow switches can define *flows* based on the combination of Layer 1, 2, 3, and 4 information such as the ingress port of switches, the MAC address, IP address and port number involved in the incoming frame/packet header fields, and *actions* on how the switches handle the received packet/frame (flow), such as packet forwarding to the designated links (switch ports) and packet header rewriting.

By utilizing the controllable feature of each traffic flow provided in OpenFlow networks, in this paper, we propose a scheme that calculates the measurement paths to comprehensively cover all links from a single beacon and set the calculated paths along which the probing packets are forwarded to multiple switches in cooperation with the OpenFlow controller. In the proposed scheme, the packet header is rewritten at the designated switches to enable the probing packets to be returned to the beacon for the estimation of the performance of individual links. Consequently, the flow-entries for performance measurements at OpenFlow switches define three types of actions, packet-forwarding in the direction from the beacon to other switches (hereinafter called the “forward direction”), sending back received packets to the ingress links and forwarding packets in the direction from other switches to the beacon (hereinafter called “return direction”). The measurement paths are calculated as follows. Here, we suppose the detailed physical topology of the network is given.

1. Suppose a single beacon is connected to a switch in an OpenFlow network that contains N links (switch ports) $\{l_1, l_2, ..., l_N\}$. The shortest paths $\{r_1, r_2, \ldots, r_N\}$ from the beacon to all links are calculated by using an existing graph search algorithm (e.g., Dijkstra algorithm) based on a certain routing criterion (e.g., the number of hops from the beacon). Here, note that the shortest paths are not calculated based on switch-level topologies but link-level (i.e., switch port-level) topologies. In the case where there are multiple shortest paths to a link due to Equal-Cost Multi-Path (ECMP), one of them is selected. The calculated paths are adopted as the measurement paths in the forward direction.
2. In order to cover uncovered links, additional measurement paths are calculated. Suppose an uncovered link l_a between two switches each of which is connected by links l_b and l_c $(b \neq c)$ in the forward direction paths, respectively, we compose measurement path r'_a by combining either of the two shortest paths r_b or r_c, and the uncovered link l_a. For example in Fig. 1, r'_5 is composed of r_3 (l_1-l_2-l_3) and l_5 as illustrated by the red dotted arrow in Fig. 1.
3. Against each path in the forward direction, the measurement path in the return direction is calculated as the path includes the same links and switches in reverse order. The paths in both the forward direction and return direction are combined together as a (round-trip) measurement path. As a result, the set of measurement paths that cover all links in the network is calculated.

B. Setting Measurement Paths in OpenFlow networks

As explained in Section III-A, in OpenFlow networks, traffic flows can be flexibly defined based on the combination of multiple packet headers, and hence there are many ways to define the flows for the performance measurements. As a simple example, we can define the flows by using two-tuple fields in the packet header, destination port number and destination address. An example setting of flow definition and actions on switches are as follows.

1. The beacon sends probing packets that have different destination port numbers toward every target link (i.e., switch ports that are connected to target links) that will send back the probing packets.
2. When an OpenFlow switch receives a probing packet, the switch identifies which flow-entry is matched for the probing packet based on the destination port number, and applies pre-defined

regarding actions on how the switches handle the received packet. If the probing packet is identified as the traffic flow for performance measurements in the forward direction, the probing packet is forwarded to the designated link toward the next switch according to the pre-defined action in the flow-entry table.

3. If the probing packet is identified as flows to be sent back to the beacon based on the destination port number, the switch forwards the probing packet to the ingress link. In order to make other switches distinguish the directions of the probing packets (i.e., forward direction and return direction), the switch changes the destination address of the probing packets to a certain unique address (e.g., the IP address of the beacon). Note that packet rewriting and packet-forwarding are defined by a single flow-entry at OpenFlow (version 1.3) switches.
4. As a result, the probing packets in the return direction are identified by the destination address of the probing packets at any switch and the packet-forwarding action in the return direction is conducted at multiple switches along the measurement paths.
5. As a result, the beacon observes all the returning probing packets from all individual links and estimates link-by-link performance characteristics by analyzing the difference between probing packets as the traceroute command does. Note that we currently use a simple estimation technique of link-by-link performance, but we believe that other sophisticated techniques are widely applicable to our scheme.

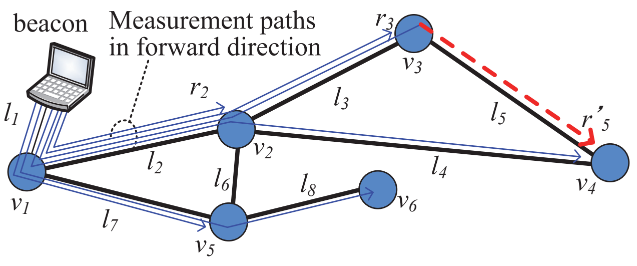

Figure 1. The calculation of the measurement paths.

C. Aggregation of Measurement Flows

In OpenFlow networks, setting a large number of flow-entries at a switch may cause problems where flow-entries dedicated to user traffic flows cannot be added due to the limitation of the maximum number of flow table entries [12]. In addition, the performance of packet processing at a switch may be degraded [13][14][15]. Therefore, it is important to minimize the number of flow-entries for performance measurements in each switch to the extent possible.

In the proposed scheme, flow-entries for performance measurements kept at switches are classified into three types, packet-forwarding in the forward direction, sending-back the received packets toward the beacon, and packet-forwarding in the return direction. However, in this section, we focus on reducing the number of flow-entries of measurement paths in the forward direction. This is because (i) measurement paths in the return direction can be easily aggregated by setting a common header option in all returning probing packets. For example in Section III-B, each switch can define flow-entries for multiple paths in the return direction by a single flow-entry based on the same identifier (e.g., the destination address) of a flow and the same packet-forwarding action. (ii) sending-back action can be merged with packet-forwarding action in the current implementation of OpenFlow switches.

Aiming at reducing the number of flow-entries at switches in the network, we propose a solution that utilizes unused header fields of probing packets. By setting the common header option to multiple measurement flows that define the same action, we aggregate multiple flow-entries into a single flow-entry. For example in Fig. 2, if we set the same source port number to all the probing packets that pass through link l_3 in the forward direction, i.e., r_3 and r'_5, the two flow-entries for these flows can be merged by a single flow-entry at switch v_2, i.e., switch v_2 can identify the flow by using the source port number and apply the same action to forward the packet to link l_3. Here, it should be noted that the same aggregation of the flows is applicable to other switches that a same set of aggregated flows traverse. Namely, the number of flow-entries at switch v_1 is also reduced in Fig. 2.

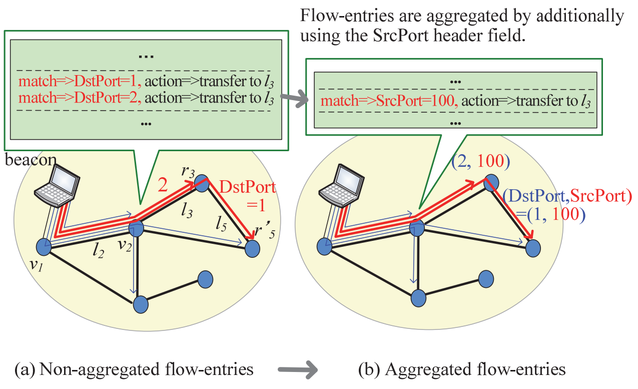
Flow-entries are aggregated by additionally using the SrcPort header field.
...
match=>DstPort=1, action=>transfer to l_3
match=>DstPort=2, action=>transfer to l_3
...
...
match=>SrcPort=100, action=>transfer to l_3
...
beacon
2
r_3
DstPort =1
l_3
l_5
r'_5
l_2
v_2
v_1
(2, 100)
(DstPort,SrcPort) =(1, 100)
(a) Non-aggregated flow-entries
(b) Aggregated flow-entries

Figure 2. Flow aggregation of the flow-entries.

The problem of finding optimal sets of flows to reduce the number of flow-entries over the network is formulated as follows. Suppose the monitored OpenFlow network consists of M switches $\{v_1, v_2, ..., v_M\}$ and N links (switch ports) $\{l_1, l_2, ..., l_N\}$, and the number of the flow-entries at v_i is expressed by F_i (i=1, 2, …, M). As the number of flow-entries at a switch may affect the performance of the packet processing and the number of available flow-entries for user data traffic, in this paper, we focus on minimizing the maximum number of flow-entries at all switches in the network as expressed in (1).

$$\text{Minimize} \quad \max\{F_1, F_2, \ldots, F_M\} \tag{1}$$

Here, F_i (i=1, 2, …, M) is calculated by the sum of the number of flows that pass in v_i as formulated by (2). In (2), $l_j \in v_i$ means that link l_j is connected to switch v_i in the forward direction.

$$\text{s.t.} \quad \forall i, \quad F_i = \sum_{l_j \in v_i} f_j \tag{2}$$

In addition, suppose the number of flows that pass through link l_j in the forward direction is expressed by f_j, the relationship among f_j (j=1, 2, …, N) can be formulated as linear equation (3).

$$\text{s.t.} \quad [f_1, f_2, \ldots, f_L]^T = \boldsymbol{R} \cdot [f_1, f_2, \ldots, f_L]^T + [1, 1, \ldots, 1]^T \tag{3}$$

where **R** is a matrix (referred to as *routing matrix*) that associates the number of input flows and the number of output flows across the switches by reflecting the topology of the measurement paths. Routing matrix **R** = (a_{jk}) encodes whether link l_j is connected with l_k across a switch, i.e., a_{ij} takes a value of 1 if link l_i is connected with link l_j across a switch, and 0 otherwise. Note that the second clause on the right side in (3) represents the number of flows that will be sent back to the beacon.

Furthermore, in (4), we introduce the binary functions b_j (j=1, 2, ..., N), that indicate if all flows that traverse each link l_j are aggregated. Note that because the aggregation of flows is applied to reduce the flow-entries at switches, candidate sets of the aggregated flows pass through the common links. The total number of flows can be expressed by using b_j as shown in (5). In (5), f_j becomes 1 if the value of b_j is 1 and f_j otherwise.

$$\text{s.t.} \quad b_j = \begin{cases} 1 & \textit{all flows that traverse link } l_j \\ & \textit{are aggregated} \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

$$f_j = (1 - b_j) f_j + b_j \tag{5}$$

Next, other constraints are considered. In our aggregation scheme, since we assume that multiple flow-entries are aggregated at switches based on the same header values of incoming probing packets, the values are not changed by any switches while the probing packets traverse the network in the forward direction. This is because changing the header values for the flow aggregation requires setting the exclusive flow-entries whose number is the same as the number of individual flows to be changed, and thus it contradicts our goal of reducing the number of flow-entries on measurement paths. In other words, when we use a single packet header field for the aggregation, measurement paths in the forward direction can be aggregated at most once. The constraints are formulated by (6) with the number of available header fields for the aggregation H.

$$\text{s. t.} \quad \forall k, \quad \sum_{r_j \subseteq r_k} b_j \le H \tag{6}$$

where $r_j \subseteq r_k$ means that two paths r_j and r_k are in the inclusion relationship.

The above equations, (1)-(6) are expressed as an integer programing problem. The optimal solutions (the minimized number of max$\{F_1, F_2, \ldots, F_N\}$ and b_j (j=1, 2, …, L)) that achieve the minimization are calculated by a general programing solver (e.g., IBM ILOG CPLEX [16] and Gnu Linear Programming Kit (GLPK) [17]). After calculating the solution, all measurement flows traversing the relevant links are aggregated at each link l_j of b_j=1. A unique value is set to the header field in the aggregated probing packets. The new aggregated flow-entries are set in substitution for the existing flow-entries at the relevant switch in cooperation with the OpenFlow controller.

IV. EVALUATION AND DISCUSSION

A. Experimental Setup

In order to evaluate the effectiveness and feasibility of our proposal, we set up experimental virtual networks by using *Trema* (v0.3.19) [6] of the OpenFlow emulator. Since the current Trema does not have the functions of accurately capturing packets and emulating the performance degradation, such as packet loss and large packet delay, we set up OpenFlow networks over two physical Linux servers (Servers 1 and 2) and two physical links between them as shown in Fig. 3. The experimental network is configured as follows.

- One host as a beacon was set up on server 2 by using Kernel-based Virtual Machine (KVM) [18]. All probing packets exchanged by the host are captured at the physical network interface between Servers 1 and 2 by using *tcpdump* command. We implemented the functions to collect the statistical data of each measurement flow at all links such as the number of received packets, average packet delay and loss rate on individual links, by analyzing recorded logs in Trema and packet captured data by tcpdump.
- To emulate the performance degradation on OpenFlow networks, we distributed OpenFlow switches over two physical servers and caused 5% packet loss or 20ms packet delay at the two physical links overlaid by the experimental OpenFlow network by using a network emulator [19].
- In order to measure packet delay in the millisecond order, we modified Trema to output the minimum logs. As a result, we confirmed that the packet transfer delay at each switch was 0.4ms on average. The specification of the two servers is as follows: CPU: Intel Xeon E5-2420 @ 1.90 GHz, memory: 18GByte, OS: Ubuntu v12.04.2).

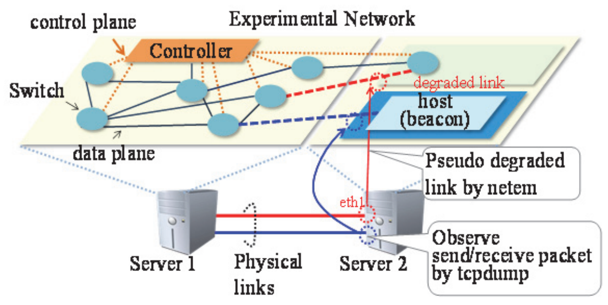

Figure 3. Experimental setup.

We used three mesh topologies and three tree topologies based on six random topologies of the Waxman model and BA model by BRITE [20] as shown in Table I. Here, we set the maximum number of switches to 300 due to the limitation of the current version of Trema. On each topology, we roughly classified all switches into three types (root-, center- and edge-positioned switches) based on the degree of each vertex (i.e., the number of connected links) and connected a beacon (host) to a switch of each type and repeatedly conducted emulations, i.e., we conducted three emulations on each topology.

TABLE I. EXPERIMENTAL NETWORK TOPOLOGY MODELS

No.	#Switches	#Links	Topology
1	10	18	Waxman, Mesh
2	20	21	BA, Tree
3	100	198	BA, Mesh
4	150	601	Waxman, Mesh
5	200	201	Waxman, Tree
6	300	300	BA, Tree

B. Results

1) Network measurement coverage

First, in order to verify the effectiveness of our proposed method, the host continuously sent the probing packets (64 Byte UDP, 100ms constant interval) to estimate the performance of all links in the network. At each link, the average packet delay and loss rate are estimated every ten seconds. The results are summarized as follows.

- First, we confirmed that all links were comprehensively covered by measurement paths and all probing packets were returned to the beacon based on the collected statistical data.
- In the case where we caused 5% packet loss and 20ms packet delay on a targeted link by using a net emulator, the beacon successfully detected the performance degradation on the link. The estimation error of loss rate and packet delay was approximately 10% and 15%, respectively. Note that the estimation accuracy depends on the measurement techniques and evaluation with a more sophisticated measurement method is a future study consideration.
- We confirmed that the results of six network topologies and three types of positions of the beacon's connection (totally 18 emulations) were almost the same.

2) Reduction of the number of flow-entries

To evaluate the effectiveness of the flow aggregation explained in Section III-C, we investigated the maximum number of flow-entries at a switch in the network in three cases; in the case where no flow-entries are aggregated (Case 1), in the case where flow-entries of measurement paths in the forward direction are aggregated (Case 2), and in the case where flow entries of measurement paths in both the forward direction and return direction are aggregated (Case 3). Figure 4 shows the ratio of the flow-entries between Cases 1 and 2, and between Cases 1 and 3, on different topologies and different positions of a switch connected by the beacon (root-, center- and edge-positioned switches). Through all emulations, the maximum number of flow-entries was successfully reduced compared with Case 1. In particular, in the case where the beacon is connected to a center-positioned switch on topology 6, the maximum number of flow-entries was reduced from 1201 to 276 (the reduction ratio was 0.78).

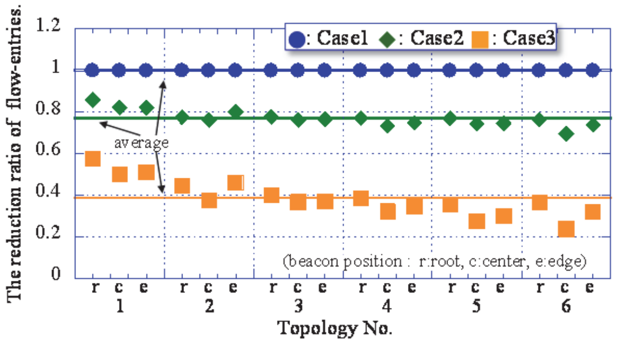

Figure 4. The ratio of flow-entries.

3) Computation time

We evaluated the average computation time of the calculation of measurement paths through emulation experiments. To calculate a set of measurement paths to cover all links on topology 6 by using the Dijkstra algorithm, our method took about 0.45s. In contrast, the computation time to calculate the optimal solution of the flow aggregation was approximately 0.1s. As a result, the computation time was sufficiently short for practical use on a network with 300 switches.

C. Elimination of Redundant Probing Packets

Sending probing packets itself is unsolicited and may cause network congestion especially on a large scale. Therefore, probing packets should be sufficiently light. Especially in the case where we utilize measurement tools that send not only probing packets but also load packets to find a bottleneck link and/or estimate link-by-link available bandwidth, it is important to minimize the number of probing packets. For example, BFind [8] detects bottleneck links by injecting a considerable volume of load traffic into the network path, and at the same time, conducts traceroute to monitor the RTT changes to all links on the path. Pathneck [3] detects the location of the bottleneck by sending probing packet trains including load packets, and the bottleneck link is inferred by measuring the per-hop train length. Note that these tools usually estimate the link-by-link performance based on the per-path measurement.

To reduce the network load of probing packets, the beacon should remove redundant packets. For example in Fig. 1, switch v_1 is shared by many measurement paths toward the same switch r_2, r_3 and r'_5. In this case, if we can utilize the same load packets toward l_5 for measuring the performance of l_2 and l_3, the load packets to measure the performance of l_2 and l_3 can be eliminated. Therefore, clustering the load packets to be sent based on the route, and eliminating the duplicated load packets is a promising approach to reduce the number of probing packets.

To evaluate this approach, we conducted preliminary experiments by using the six topologies shown in Table I. We investigated the number of probing packets in two cases. In one case (Case 1), all performance measurements are separately conducted per path but redundant packets are not eliminated. In the other case (Case 2), redundant packets are eliminated after clustering measurement paths based on their inclusion relationship. Table II presents the results. As can be seen, in each topology, the number of probing packets was reduced by almost half, and we confirmed that the elimination method based on paths' clustering has considerable potential to reduce the traffic load of probing packets of the per-path measurements. Note that the reason why the reduction ratio is near 0.5 is that most of the measurements on a measurement path can be used for the calculation of the performance of two adjacent links based on the difference with the other measurements on the same path.

TABLE II. NUMBER OF PROBING PACKETS

Topology No.	#Probing packets		Reduction ratio (Case2 / Case1)
	Case1	Case 2	
1	34	18	0.53
2	40	21	0.53
3	394	198	0.50
4	1198	601	0.50
5	400	201	0.50
6	598	300	0.50

V. CONCLUSION

In this paper, we proposed a diagnosis scheme to actively measure the performance of all physical links from one measurement point in OpenFlow networks. In addition, we tackled to reducing the number of flow-entries at OpenFlow switches by aggregating multiple flow-entries into a single entry by applying common header options to probing packets, in order to save the resources of OpenFlow switches. The optimization solution is calculated by solving the integer programing problem. Through emulation with the Trema emulator, we confirmed the proposed method has considerable potential to efficiently measure the performance of all links in the OpenFlow networks. Furthermore, we consider other measurement techniques and/or operational constraints, such as the measurement load of networks in OpenFlow networks, and conducting more large-scale experiments in future work.

REFERENCES

[1] Open Networking Foundation, http://www.openflow.org/.

[2] M. Roughan, T. Griffin, Z. M. Mao, A. Greenberg, and B. Freeman, "IP forwarding anomalies and improving their detection using multiple data sources," Proc. ACM SIGCOMM, September 2004.

[3] N. Hu, L. Li, Z. Mao, P. Steenkiste, and J. Wang, "Locating Internet bottlenecks: Algorithms, measurements, and implications," Proc. ACM SIGCOMM, September 2004.

[4] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "User-level Internet path diagnosis," Proc. ACM SOSP, October 2003.

[5] A. Tachibana, S. Ano, and M. Tsuru, "A large-scale network diagnosis system based on user-cooperative active measurements," Int. J. Space-Based and Situated Computing, Vol. 3, No. 2, pp.69-82, 2013.

[6] Trema, http://trema.github.io/trema/.

[7] K. Anagnostakis, M. Greenwald, and R. Ryger, "Cing: Measuring network internal delays using only existing infrastructure," Proc. IEEE INFOCOM conference, 2003.

[8] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area Internet bottlenecks," Proc. SIGMETRICS '03, June 2003.

[9] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: Recent developments," Statistical Science, Vol. 19, No. 3 pp.499-517, 2004.

[10] Y. Zhao, Z. Zhu, Y. Chen, D. Pei, and J. Wang, "Towards efficient large-scale VPN monitoring and diagnosis under operational constraints," Proc. IEEE INFOCOM 2009, pp.531-539, April 2009.

[11] D. Ghita, H. Nguyen, M. Kurant, K. Argyraki, and P. Thiran, "Netscope: Practical network loss tomography," Proc. IEEE INFOCOM, March 2010.

[12] N. Matsumoto, M. Hayashi, and I. Morita, "LightFlow: Leveraging combination of hash and wildcard tables for high performance flow switching in large number of flow entries," USENIX 2013.

[13] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," Proc. PAM 2010, pp.201-210, April 2010.

[14] N. Varris, and J. Manner, "Performance of a software switch," Proc. HPSR 2011, July 2011.

[15] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," Proc. ICC, pp.1-5, IEEE 2010.

[16] CPLEX, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[17] GLPK, http://www.gnu.org/software/glpk/.

[18] KVM, http://www.linux-kvm.org/page/Main_Page.

[19] S. Hemminger, "Network Emulation with NetEm," http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[20] A. Medina, A. Lakhina, I. Matta, and J. Byers. "BRITE: Boston University Representative Internet Topology Generator," http://cs-pub.bu.edu/brite/index.html.