Latency Monitoring in Software-Defined Networks

Alaa M. Allakany
Graduate school of Information Science and Electrical Engineering, Kuyshu University, Japan & Faculty of Science, Kafrelsheikh University, Kafrelsheikh, Egypt.
744 Motooka, Nishiku, Fukuoka City, 819-0395.
Japan
alaa_83moh@yahoo.com

Koji Okamura
Research Institute for Information, Technology, Kyushu University.
744 Motooka, Nishiku, Fukuoka City, 819-0395.
Japan
oka@ec.kyushu-u.ac.jp

ABSTRACT

Latency in a network is an important parameter that can be utilized by a variety of applications which required QoS policies. Recently, methods for monitoring latency have been introduced. Most of these methods monitor end-to-end path delay (per path) by sending probes requests along the path. These methods led to redundant work and network overhead, which resulting from monitoring multiple paths between each pair of nodes. Moreover, end-to-end probes cannot monitor the delay on path segments (per link) between arbitrary network devices. However, measuring per link delay is challenging. In this paper, we propose a method to measure per link delay in real-time to efficiently apply QoS policies, our method does not require any complementary support from the switching hardware and can avoid redundant work and network overhead. We validate our method using the popular Mininet network emulation environment with Pox controller.

CCS CONCEPTS

• **Networks** → Network monitoring

KEYWORDS

Software Defined Networks, OpenFlow, Pox controller, QoS.

1 INTRODUCTION

Recently, there is a fast growth in the applications that have strict Quality of Service (QoS) requirements. Management these applications require accurate and timely monitoring statistics of network resources.

CFI'17, June 14-16, 2017, Fukuoka, Japan
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5332-8/17/06...$15.00
http://dx.doi.org/10.1145/3095786.3095791

Researchers have proposed new ideas for managing these applications, but, these ideas often include nonstandard aspects that required change in IP networks. It is difficult to incorporate these changes into IP networks since the devices in these networks don't allow changes to be made in their software systems.

Software Defined Networking (SDN) is introduced as new technology with the following features that can cover the limitation of IP networks: 1) Control and data planes are separated from each other. So, network devices no longer have control functionalities. 2) Control plane is moved to an external entity called controller, and 3) data plane, it is used to forward coming data based on pre-install flow in flow table. In SDN the controller and switch can communicate over the OpenFlow protocol [1]. By taking the advantages of OpenFlow that enables controllers to query for statistics and inject packets into the network the monitoring system can measure QoS parameters in SDN.

There are several proposals for monitoring QoS parameters in SDN, they mostly solve the problems of e.g. bandwidth utilization [2–4], packet loss ratio [4], packet delay per path [4,5], and route tracing [6]. The latency is measured in these methods by end-to-end delay of path between two individual devices, often, these methods cannot calculate delay on path segments (per link) between arbitrary network devices. However, per link delay measurement can have significant importance for both service provider and application perspectives.

In this paper, we present a new monitoring methods that is a POX OpenFlow controller modules enabling accurate online monitoring of link delay. Our method can reduce network overhead, avoid duplicate work, also we can calculate the delay on path segments between arbitrary network devices to support QoS polices. The remainder of this paper, is structured as the following: In section 2, we give a short background and discuss existing monitoring techniques used in SDN. Section 3, introduce our proposal method. Experimentally evaluated discusses in section 4. Section 5 concludes this paper.

2 BACKGROUND AND RELATED WORK

Each application running on a network infrastructure may has different requirements. End-to-End delay is one of these requirements. Unicast applications that required end-to-end delay or least cost delay path from source to destination can be calculated using existing monitor methods by measuring per path delay [4,5], but these methods cannot avoid network overhead (bandwidth, CPU). As shown in Fig. 2, to calculate the least cost delay path between source node (1) and destination node (4) we have to measure delay for all available paths between this pair of nodes. So, measuring delay per path cannot avoid network overhead. Finding MST required driving a weighted graph that shows each link delay in the network. To derive a weighted graph we have to calculate per link delay. Therefore, the existing monitoring method (per path delay) lacks for supporting QoS multicast routing with end-to-end delay constraints. Generally, knowledge about link delay over the network would benefit many users and operators of network applications.

With OpenFlow, it becomes easy to pick up switch and per-flow statistics into a centralized point. Many researches use this advantage to monitor network traffic. In [2] the authors propose a monitoring method to use only the mandatory OpenFlow messages to monitor the bandwidth utilization in the network. The authors in [3] proposed an algorithm (MonSamp) that use the Flow-Stats-Req message in OpenFlow to poll the interface and flow counters in the switches for bandwidth measurement, but this method suggests decreasing the sampling rate when the traffic load is high and that can reduce the accuracy of measurement.

In [4] the authors proposed a novel mechanism called OpenNetMon, this method offers a solution for packet lost and per path delay monitoring. To measure path delay, the controller estimates the complete path delay by calculating the difference between the packet's departure and arrival times, subtracting with the estimated latency from the switch-to-controller delays. But this method still uses per path delay measure that led to redundant work and network overhead which resulting from monitoring multiple paths between each pairs of nodes.

In order to reduce network overhead for monitoring link delay to support QoS unicast and multicast applications with end-to-end delay constraints, we present this paper. Our main contributions to propose a new monitoring method for measuring delay per each link in OpenFlows networks. Taking the advantage of the features that OpenFlow offers, we present our solution based on the following steps: 1) we modify the Dijkstra's shortest path algorithm [7] to derive a tree that covers all links in the network with minimum hop in each path the source of this tree is the monitoring point MP as shown in Figure 3. 2) After deriving this tree we divided it into different levels starting form level 1 to N, as shown in Figure 3,4. the nodes in level 1 are the nearest to MP and the paths from the MP to these nodes consist of only one hop and the nodes in level N have the path consist of N of hops. 3) At this point we measure the path delay for all paths in this tree at each level. 4) Finally, the link and switch delay can be derived by calculating the difference between each two level, for example, difference between level 2 and level 3 result the delay for links between these two levels. Then, the data relative links delay send to the controllers for Traffic Engineering (TE) purposes. In our methods we define the delay between two switches as the time it takes a packet to travel form the output interface of the first switch to the output interface of the second switch.

3 Design

In order to measure link delay on real-time, we present our monitoring method based on OpenFlow, we will use POX controller. In the following we present in details the steps of our method and discuss the possibilities that OpenFlow introduces in the latency measurement. Figure 1, shows the modules of proposed architecture for monitoring link delay, mainly, *Topology Discover*, *Tree Construction* and *Latency Mentoring*:

1- ***Topology discover module:*** this module uses Link Layer Discovery Protocol (LLDP) to discover network topology.
2- ***Tree construction module:*** we proposed this module to derive a tree that covers all links in the network and minimizes the total number of hops in each path.
3- ***Latency monitoring module:*** we use this module for monitoring path delay for all paths in each level in this tree

Tree construction module and Latency monitoring module are the two basic modules in this method. Using the results of these two modules, we can calculate link delay and using the resulting data for QoS polices. In the next sections, we explain more details about each module and how we can calculate the delay per link.

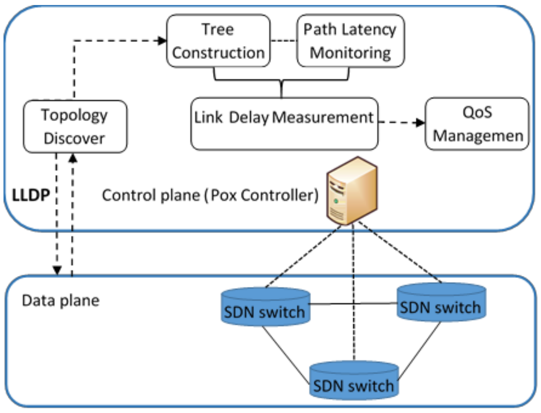

Figure 1: Proposed Architecture.

3.1 Topology discover module

We use this module to discover the topology of the network. By using the information resulting from this module we can build up the network topology graph G (V, E), where the node set V corresponds to the switches and the edge set E corresponds to the links. Then, we send the data relative to the topology graph G to tree construction module to build up the tree.

3.2 Tree construction module

In order to measure link delay, this module will construct the tree that covers all links in the network. Dijikstra shortest path algorithm [7] can be used to derive Minimum Spanning Tree (MST) that cover every node in the network. In our method, we will modify Dijikstra shortest path algorithm to derive the tree that cover every link in the network and minimize the total links in each path. Figure 3, shows the tree derived by this algorithm from the network topology of Fig. 2.

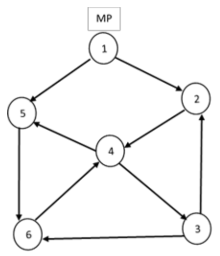

Figure 2: Network topology.

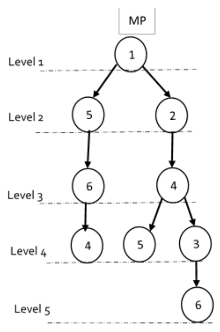

Figure 3. tree covering all networks links

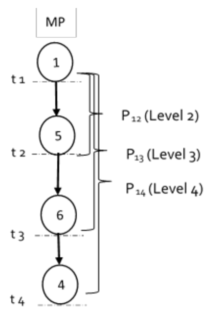

Figure 4. example of sub-paths from MP to node 4 shows arrival time at different levels

The modified Dijikstra shortest path algorithm avoid repeating any node in the same path, but, the nodes may be repeated in different paths of the tree. This, because we use this tree to derive individual paths at each level to measure path delay by sending probes request along these paths. Fig. 3, shows that node number 4 is repeated twice in the tree, but in different paths.

Table 1, shows the individual sub-paths at each level of the tree on Figure 3. This tree have three main paths (path1, path2 and path3) and each path has it's sub-paths at different level. In this table P_{ij} represents a sub-path, i represents the path number and j represents level number, this means that P_{13} represents sub-path derived from path1 at level 3: $(1 \rightarrow 5 \rightarrow 6)$.

If there are sub-paths repeated more than one time, we will avoid monitoring repeated sub-paths in order to avoid network overhead, for example, sub-paths (P_{32} and P_{33}) and sub-paths (P_{22} and P_{23}) are the same, in this case the total sub-paths that will be monitored are 8 sub-paths according to Table 1. The advantage of using this algorithm is that, it can minimize total number of paths and links in the tree that will be used for monitoring the network. Therefore, we can reduce network overhead.

3.3 Path Latency module

In this module we will use the advantages of OpenFlow's to inject packets into the network for measuring path delay. The output data of previous module (*Tree construction module)* will be used as inputs for this module. At every sub-path derived from the tree, we regularly inject packets at the first switch, such that probe packet travels exactly the same path, and have the last switch send it back to the controller. To calculate the departure time at first switch and arrival time at last switch in the path, we calculate the controller to first switch delay by calculating $RTT_{C}\rightarrow S$ from controller to switch, also from last switch to controller we calculate $RTT_{S}\rightarrow C$. Then we use Equation 1 and Equation 2 to calculate departure time at first switch and arrival time at last switch respectively. The controller estimates the complete path delay by calculating the difference between the packet's departure time at first switch and arrival time at last switch.

$$T_{First-switch} = T_{send} + (RTT_{C \to S})/2 \tag{1}$$

$$T_{Last-switch} = T_{arrival} - (RTT_{S \to C})/2 \tag{2}$$

Here T_{Send} represent the time of sending probe packet form the controller and $T_{arrival}$ represent the arrival time of the packet to the controller.

```
Modified Dijkstra's Algorithm
Input: G = (V,E), MP (monitoring point)
Output: a tree covering all link in the network
1:  T={MP};d[MP]←0; d[u]←∞ and pred[i]←nil for each u≠MP, u€V
2:  insert u with key d[u] into the priority queue Q, for each u€V
    while (Q ≠ ∅)
3:   j ← Extract-Min(Q)
4:  for every node i, i €T and i is adjacent to j
        -   alt = d[j] + ew(j, i)     // ew(j, i) is edge weight =1 for all links
        -   if alt < d[i] then
        -   d[i] ← alt
        -   pred[i] ← j // set i as a child node of j
        -   if link (i,j) ∉ T, add the link into T
5:  return T
```

Figure 5: Dijikstra Algorithm.

Steps of our proposed method	
1:	Using Dijikstra algorithm calculate the minimum tree that covering all network links //Figure 5
2:	Divide this tree into different level starting at the monitoring point MP. // Figure 3
3:	For each level on the tree derive sub-paths of this level. // Table 1
4:	Calculate delay for all sub-paths using method describe in *Path Latency module*.
5:	By subtracting the time between each two level we can calculate the links delay between these two levels.
6:	Derive the weighted graph of the network for QoS polices.

Figure 6. Steps of our proposed method.

3.4 Link Delay Measurement

The output of both *Tree Construction module* and *Path Latency module* will be used to calculate link delay. We will discuss how we calculate link delay using example shown in Figure 4. This Figure represent path 1 from the constructed tree Figure 3, and it has three sub-paths (P_{12}, P_{13} and P_{14}), at levels ($level_2$, $level_3$ and $level_4$) respectively. In this method t_1 represents the departure time at MP, (t_2, t_3 and t_4) represent arrival times of probe packet at last switch of (P_{12}, P_{13} and P_{14}) respectively. To calculate the link delay between switch 1 and 5, we subtract t_2 and t_1 (t_2-t_1) and to calculate the link delay between switch 5 and 6, we subtract t_3 and t_2 (t_3-t_2) and so on. After calculating all links delay in the network we can derive weighted direct graph of the network topology. Then, we store this weighted graph into the controller for applying QoS polices. The advantages of our proposed monitoring method are: 1) Avoiding redundant work and network overhead resulting from measuring multiple paths for each pair of switch, this because we use algorithm that can minimize the total paths and links in each path to derive the link delay. 2) Measuring latency in real time for each individual link (per link) not per path delay, then, we can easily derive a weighted network graph. 3) By deriving a weighted graph of the network we can support most of optimization algorithm in unicast and multicast routing. On the other hand, per path delay can't support most of multicast application QoS polices. Our method is summarized in Fig. 6.

Table 1: sub-paths for each path at different levels

Paths in the tree		Path1	Path2	Path3
		(1 → 5 → 6 → 4)	**(1 → 2 → 4 → 5)**	**(1 → 2 → 4 → 3 → 6)**
Sub path at each level	Level 2	P_{12} (1 → 5)	P_{22} (1 → 2)	P_{32} (1 → 2)
	Level 3	P_{13} (1 → 5 → 6)	P_{23} (1 → 2 → 4)	P_{33} (1 → 2 → 4)
	Level 4	P_{14} (1 → 5 → 6 → 4)	P_{24} (1 → 2 → 4 → 5)	P_{34} (1 → 2 → 4 → 3)
	Level 5			P_{35} (1 → 2 → 4 → 3 → 6)

4 EXPERIMENTAL RESULTS

In this section, we test our method with OPENNETMON [4] mechanism. Our method is implemented as OpenFlow controller modules, we use POX controller and Mininet to emulate the network. We used random partial connected topology generated using the waxman generator provided by BRITE. We assume that all link in the network have same link capacity of 20Mb/s. the topology of 10 switch and 24 link is greeted. With each switch we suppose there are only one host is connected. We suppose that each host in the network will send request to each source (Two sources h1 and h2) for requesting the data, then the controller calculated Least Delay Path between the source and destination. We tested network overhead resulting of both method.

Table 2, network overhead in our method and OPENNETMON

	Number of probe packet To monitor network	Number of links in paths that will used by probe packets
LINK-MON	15	43
OPEN-NET-MON	36	116

Table 2 shows the network overhead resulting from sending Probe packet to measure delay in the network. We compare the total number of installed paths (i.e., the total number of probe packet that will send from controller to monitor the network) to forward probe packets and the total number of flows installed on OpenFlow switches to forwarded these packets (i.e., number of links along the paths that will be used to forwarded probe packet in each method). This table shows that our method install 15 paths and thus it need only 15 probe packets to measure all links delay in the network, moreover, the total number of links in these paths are 43 links, this means smaller number of packet will travel along the network to discover all links delay. In the other side OPENNETMON install 36 path (i.e., the controllor need to 36 probe packets along these paths) and the total number of links on these paths are 116 links.

5 CONCLUSIONS

In order to reduce network overhead for monitoring link delay to support a verity of application that required QoS polices, we proposed, a new method for monitoring link-based delay. Our method modify the Dijkstra's shortest path algorithm to derive minimum tree that covering all link in the network to minimize the total number of links to be monitored. Then, we can calculate the link delay based on our purposed idea. Our method can avoid redundant work and network overhead resulting from monitoring multiple paths for each pair of switch. Measuring latency in real time for each individual link. Supporting most of applications that required unicast and multicast QoS polices. On the other side most existing method measure delay per path, that results network overhead.

REFERENCES

[1] Nick M., et al. 2008. Openflow: Enabling innovation in campus networks. SIGCOMM Computer Communnication Review, 38(2):69–74, Mar. 2008.

[2] Yu C., Lumezanu C., Zhang Y., Singh V., Jiang G., and Madhyastha H. 2013. Flowsense: Monitoring network utilization with zero measurement cost. In Passive and Active Measurement, volume 7799 of Lecture Notes in Computer Science, pages 31–41. 2013.

[3] Raumer D., Schwaighofer L., and Carle G. 2014. MonSamp: A distributed SDN application for QoS monitoring. In Federated Conference on Computer Science and Information Systems (FedCSIS), Sept. 2014.

[4] Adrichem N. van, Doerr C., and Kuipers F. 2014. Opennetmon: Network monitoring in openflow software-defined networks. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–8, May 2014.

[5] Phemius K. and Bouet M. 2013. Monitoring latency with openflow. In 9th International Conference on Network and Service Management (CNSM), pages 122–125, 2013.

[6] Agarwal K., Rozner E., Dixon C., and Carter J. 2014. SDN traceroute: Tracing sdn forwarding without changing network behavior. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, pages 145–150, 2014.

[7] Dijkstra E. 1959. "A note on two problems in connexion with graphs," Numerische mathematik, vol. 1, no.1, 1959, pp. 269-2.