PathMon: Path-Specific Traffic Monitoring in OpenFlow-Enabled Networks

Ming-Hung Wang, Shao-You Wu, Li-Hsing Yen, and Chien-Chao Tseng
Dept. Computer Science, National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

***Abstract*—Software-defined networking (SDN) aims to provide open interfaces, such as OpenFlow protocol, that enable programmable flow-based network management. Meanwhile, OpenFlow-enabled switches provide rich statistical information for specific flows, tables, and ports which facilitates various network management tasks. However, current traffic monitoring techniques collecting statistics from existing flow entries do not provide the flexibility of querying path-specific flow statistics at any aggregation levels. In this paper, we proposes PathMon which encodes flow and path information as tags to provide that flexibility and corresponding link-to-link correlations for further anomaly detection or traffic engineering tasks. Our implementation runs on commodity switches which are compliant to OpenFlow 1.3 specification and requires only a few monitoring entries for each switch along monitored paths and un-tag entries for each edge switch.**

***Keywords*—Software-defined networking, traffic engineering, network monitoring, OpenFlow.**

I. Introduction

Traffic monitoring is essential to network operation and management. Network statistics constantly collected from networking devices can help probe health and performance, as well as facilitate various management tasks such as anomaly detection, load balancing, accounting and billing. In this paper, we are interested in traffic monitoring in the context of OpenFlow-enabled Software-Defined Networks (SDNs).

SDN is an emerging networking technology that provides flexibility and programmability in network management [1, 2]. In SDN, control plane is decoupled from data plane. The control plane is carried out by a logically centralized machine called controller, which sends instructions using protocol like OpenFlow to install flow entries to switches. Switches as the main entities of data plane perform data forwarding tasks following the instructions given by the controller. This centralized design makes it more flexible and much easier to control the behavior of the underlying network infrastructure. Network administrators using control applications running on the controller can perform different traffic-engineering functions, such as rerouting and dropping packets pertaining to some specific traffic. Meanwhile, OpenFlow-enabled switches provide rich statistical information for specific flows, tables, and ports.

There exist some traffic monitoring approaches to the collection of data plane statistics in SDN. These approaches either proactively poll switches using `FlowStatisticsRequest` for active flows [3] or reactively get notified by `FlowRemoved` massages upon flow expiration [4]. A common characteristic of these approaches is that only statistics of existing flow entries can be acquired. This is fine if network administration needs can be met with the statistics provided by the existing flows. However, different management tasks may demand statistics at different aggregation levels. For instance, a load balancing scheme may requires flow-specific statistics per unit of time, while a billing system may require monthly usage data for each user. In that case existing approaches do not provide adequate or accurate information. For example, it is usually sufficient for a switch to forward packets by installing flow entries that match on ingress port and destination MAC address of incoming packets. In that case if we want to query traffic information of a specific TCP flow that traverses through a series of links, the obtainable flow statistics that are specific to links are not adequate.

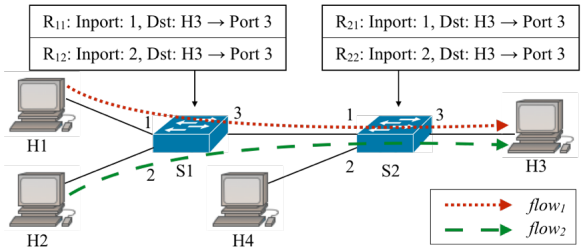


Fig. 1. Example of querying statistics from flow entries in each switch along the designated path may count in irrelevant flows from other paths.

Fig. 1 shows a scenario where we are interested in statistics of $flow_1$ from hosts H1 to H3. The flow traverses the link from H1 to Port 1 of S1, followed by the link from Port 3 of S1 to Port 1 of S2, and then the link from Port 3 of S2 to H3. Suppose that only relevant flow entries for these three links (i.e., R_{11} and R_{21} in Fig. 1) are created. We cannot obtain an accurate statistic for $flow_1$ with current monitoring approaches because the statistic of flow entry R_{21} will count in both relevant statistics of $flow_1$ and irrelevant statistics of $flow_2$ from another network path.

To resolve this problem, PathMon specifically inserts a separate set of flow entries called *monitoring entries* into every switch along a path to be monitored. With PathMon, administrators can query flow statistics at any aggregation level. Since monitoring entries exactly match flows of interest, irrelevant flows are excluded from statistics reported

by switches. In addition to per flow statistics, PathMon also calculates *link-to-link correlation* which can facilitate network management tasks such as loop detection, load balancing, and virtual machine placement. The details of use scenarios will be explained in the following sections.

The remainder of this paper is organized as follows. We first review existing monitoring techniques for SDN. Section III gives an use scenario and describes the design of PathMon in details. Section IV presents a case study for loop detection and numerical results of our experiments. The last section concludes this paper.

II. Related Work

Traffic monitoring approaches fall into two categories: the first measures performance metrics such as throughput, packet loss, delay and jitter by active probes, while the second collects a set of primitive statistics from networking devices by using either proprietary interfaces [5] or open protocol like OpenFlow [2]. In this paper, we focus on the second approach for SDN.

OpenNetMon [6] provides a way to collect per-flow metrics such as throughput, delay and packet loss by polling edge switches at an adaptive rate to reduce network and switch CPU overhead while optimizing measurement accuracy. Moreover, OpenNetMon provides monitoring necessary to determine whether QoS parameters between each pair of end hosts are actually met. The monitored results are then delivered to traffic engineering methods for the generation of appropriate paths.

FlowSense [4] is a push-based approach to monitoring performance changes by receiving notification carrying flow statistics from switches. This passive way to collect flow statistics is to minimize the total number of control messages. However, its estimation may become inaccurate under certain flow conditions.

PayLess [3] focuses on an adaptive sampling algorithm for which polling frequency is variable and adaptive to measured throughputs. Thus, monitoring overhead could be minimized while avoiding its estimation from sacrificing the accuracy. Furthermore, PayLess is designed as a flexible framework for developing a wide range of network monitoring applications.

III. PathMon

PathMon enables the query of path-specific flow statistics at any aggregation level. Administrators can specify an arbitrary monitoring target, a flow F traversing through interested paths NP, by a set of matching fields and a path expression. Fig. 2 gives an example monitoring target:

$$F ::= tcp.dst = 80 \quad \text{and}$$

$$NP ::= 201:3\ .*\ (203:1 \mid 205:1 \mid 207:1)\,,$$

where dot symbol (.) represents a single physical link in the network topology and $digit_{SwitchID} : digit_{IngressPort}$ identifies a network link by an ingress port of a switch; F specifies HTTP traffic and NP specifies a monitored path starting from the link connected to ingress port 3 of switch 201, followed by zero or more links, and ending at one of the following links: the one connected to ingress port 1 of switch 203, the one connected to ingress port 1 of switch 205, or the one connected to ingress port 1 of switch 207. PathMon also provides link-to-link correlation for each link pair, which is defined as $(\text{statistic}_{\text{subsequent-link}} \div \text{statistic}_{\text{preceding-link}})$. Table I shows an example matrix of link-to-link correlations for the monitored path. In this example, $\text{link}_{201:3}$-to-$\text{link}_{203:1}$ correlation=37% means 37% of traffic from $\text{link}_{201:3}$ flows through $\text{link}_{203:1}$. The matrix helps administrators observe traffic distribution among different links.

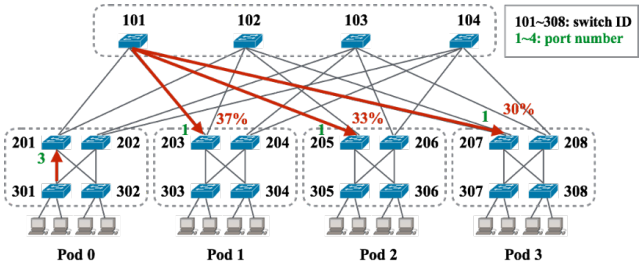


Fig. 2. Example of querying path-specific flow statistics and corresponding link-to-link correlations using PathMon.

TABLE I
Matrix of link-to-link correlations

Subsequent / Preceding	$\text{Link}_{203:1}$	$\text{Link}_{205:1}$	$\text{Link}_{207:1}$
$\text{Link}_{201:3}$	37%	33%	30%

A. Goal and Requirements

Our goal is to acquire flow statistics for a specific path as well as the corresponding link-to-link correlations, with as few monitoring entries as possible. This goal can be achieved by meeting the following requirements:

- **Non-invasive:** Network behaviour determined by forwarding entries should not be affected by monitoring entries. This means that the only function done by monitoring entries is to match flows of interest, update corresponding flow statistics, and then deliver the flows to forwarding tables for further forwarding processing. The forwarding path should not be altered by monitoring entries.
- **Arbitrary granularity:** Administrators should be able to specify flows of interest at any aggregation levels. The granularity of monitored flows depends on monitoring entries rather than existing forwarding entries.
- **Accuracy:** Path-specific flow statistics should not count in flow statistics from other irrelevant paths.
- **Flexible path monitoring:** A monitoring target may consist of either loose paths or strict paths originated from a common root switch as illustrated in Fig. 2 and 4.

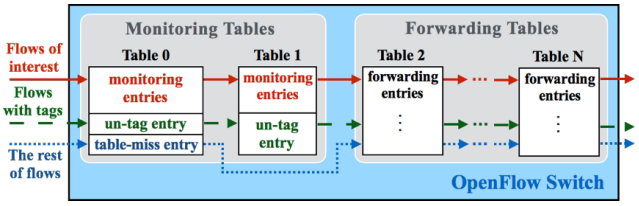


Fig. 3. Table organization and entry arrangement.

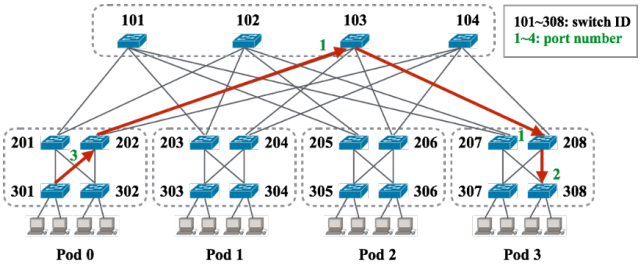


Fig. 4. Example of monitoring a strict path.

B. Design

When administrators designate a monitoring target, PathMon translates it into a set of *monitoring entries* which are inserted into switches along the monitored paths. Monitoring entries encodes flow and path information as tags for each flow of interest. Tagged flows which divert from the designated path become irrelevant flows. PathMon pre-installs two *un-tag entries* to each edge switch to strip off tags from irrelevant flows. Without this action, packets of these flows will usually get dropped out. In addition to monitoring and un-tag entries, a *table-miss entry* is also installed into every switch for directing all irrelevant flows that bear no tag to existing forwarding entries.

To store different types of entries, flow tables in a switch are grouped into *monitoring* and *forwarding tables*. Because incoming flows should be checked for monitoring before being forwarded, monitoring tables must precede forwarding tables as illustrated in Fig. 3. The first two flow tables called monitoring tables are reserved for monitoring and un-tag entries. Forwarding entries which are not installed by PathMon are placed in the rest of flow tables. In other words, monitoring and forwarding tables are started from Table 0 and Table 2, respectively.

C. Table Configuration for Monitoring

Given a monitoring target, PathMon determines configuration for monitoring tables in each switch along the designated path. The entrance switch, immediate switches, and exit switches associated with the path demand different monitoring entries. In addition, PathMon also pre-installs un-tag entries into each edge switch. We take a monitoring target: $F ::= tcp.dst = 80$ and $NP ::= (202:3)\,(103:1)\,(208:1)\,(308:2)$ as shown in Fig. 4 as an example for table configuration.

Configuration for entrance switch: Fig. 5a shows entry arrangement for the first switch along the monitored path. The main functions of Table 0 in the entrance switch include classifying incoming flows into interested and non-monitored, assigning a unique VLAN tag to each interested flow, and directing non-monitored flows to the first forwarding table.

In OpenFlow, a specific flow is defined as a set of matching fields. However, switches and middleboxes may modify header fields. In that case, monitoring entries in the subsequent switches may fail to match flows of interest. For this reason, PathMon assigns a unique tag as flow ID to each flow of interest.

The table-miss entry directs the non-monitored flows to the first forwarding table. The monitoring entry matches all HTTP traffic with $tcp.dst = 80$ and $ingress\ port = 3$, pushes a unique VLAN tag 100 as flow ID, pushes a VLAN tag 3 indicating the current ingress port, and then directs the flow to the first forwarding table. As the result, flow of interest contains stacked VLAN tags and is forwarded according to existing forwarding entries. When tagged flow arrives at an immediate switch, monitoring entries that match both tags and the designated ingress port effectively exclude irrelevant flows from consideration.

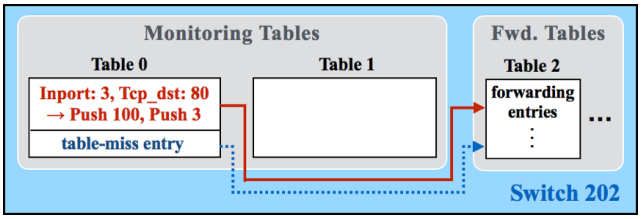
Configuration for immediate switch: As shown in Fig. 5b, the monitoring entry in Table 0 matches all incoming flows traversing $\text{link}_{202:3}$ first and $\text{link}_{103:1}$ next, pops outer VLAN tag, encodes the value of removed VLAN tag into the *metadata* pipeline field, and then directs the flow to Table 1. After that, the monitoring entry in Table 1 can further filter out all irrelevant flows without VLAN tag 100; thus, only the flow with VLAN tag 100 and traversing $\text{link}_{202:3}$ first and $\text{link}_{103:1}$ next can be matched by the monitoring entry in Table 1; the monitoring entry then pushes a VLAN tag 1 indicating the current ingress port into the flow and directs it to the first forwarding table for further processing.

The flow now carries an outer VLAN tag 1 and an inner VLAN tag 100. That is, the outer VLAN tag will be replaced with the current ingress port by each switch along the path. Similarly, switch 208 processes all incoming flows as switch 103.

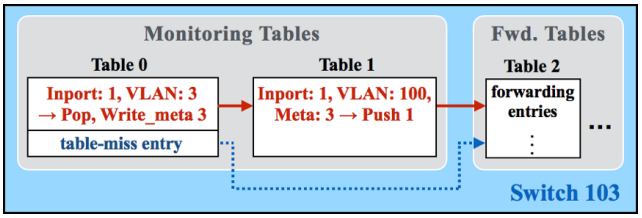
Configuration for exit switch: The function of monitoring entries in the exit switch is almost the same as that of the monitoring entries in the immediate switches. The only difference is that the monitoring entry in Table 1 pops the inner VLAN tag 100 from flow of interest rather than pushes a VLAN tag. These monitoring entries ensure that the stacked VLAN tags are stripped off from the flow of interest.

Configuration for edge switch: In this example, the exit switch 308 happened to be an edge switch. These un-tag entries are used to strip off stacked VLAN tags from irrelevant flows which divert from the designated path. To capture packets with any tag, both un-tag entries match a special tag value `OFPVID_PRESENT` defined by OpenFlow specification as shown in Fig. 5c.

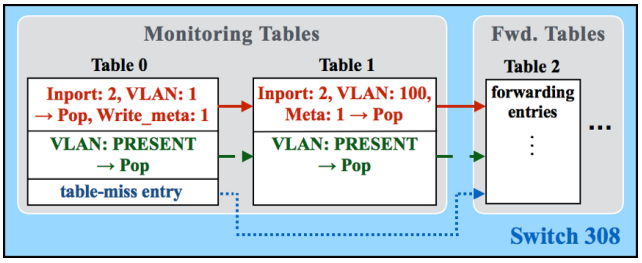
Number of entries for monitoring: PathMon requires the first two flow tables on every switch for monitoring.



(a) The entrance switch



(b) The immediate switch



(c) The exit switch

Fig. 5. Table configuration.

The additional flow entries can be categorized into three entries: monitoring entry, un-tag entry, and table-miss entry. As mentioned above, each monitoring target requires a monitoring entry in the entrance switch and two monitoring entries on the remaining switches along the path. That is, if a monitored path is of N hops, the required monitoring entries are $(2 \times N - 1)$. The total number of monitoring entries required for M monitored paths is therefore

$$(2N_1 - 1) + \cdots + (2N_M - 1) = 2\left(\sum_{i=1}^{M} N_i\right) - M,$$

where N_i is the number of links in the ith monitored path for which i ranges from 1 to M. Besides, each switch needs a table-miss entry to direct non-monitored flows to the first forwarding table and every edge switch requires two un-tag entries to strip off tags. In general, the total number of additional entries can be formulated as

$$2\left(\sum_{i=1}^{M} N_i\right) - M + S + 2E,$$

where S is the total number of switches and E is the total number of edge switches.

IV. Evaluation

We evaluate the current PathMon prototype implementation by conducting a case study and microbenchmarks in mininet environment [7], a container-based network emulator. The case study verifies the effectiveness of our tool for loop detection while benchmarks evaluate the network overhead imposed by PathMon. PathMon is implemented as a module for the Floodlight [8] exposing a set of REST APIs to add/remove/inquire monitoring targets, to collect path-specific flow statistics, and to acquire the corresponding link-to-link correlations. Table II gives information about the emulated network and hardware specification.

TABLE II
Experimental environment

Emulated network		Hardware specification	
Controller	Floodlight v1.1	CPU	Intel Core i5-2400 3.1GHz×4
Emulator	Mininet 2.2.1	Memory	7.5GB
vSwitch	CPqD [9]	OS	Ubuntu 14.04 LTS x64

A. Loop Detection

Since multiple control applications, services or administrators can install forwarding entries independently and dynamically in SDN, various unexpected behaviours including forwarding loops might arise due to many reasons such as incorrect control logic in the control plane, conflicts among flow entries installed by different control applications. To emulate a forwarding loop in a data center network, we inserted network traffic according to traffic pattern reported in previous studies [10–13] into a fat-tree topology with four pods, as shown in Fig. 6; a simple load balance application was used to determine a path for each destination address every five seconds (the hard timeout of forwarding entries was set to five seconds).

Assume that administrators are concerned with forwarding behaviour of IPv4 traffic on the path $NP ::= (202:3)\,(103:1)\,(208:1)\,(308:2)$. PathMon collects statistics and updates corresponding link-to-link correlations every ten seconds. In this experiment, we created a forwarding loop as shown in Fig. 6 at $t = 20s$. Fig. 7 shows that the loop can be observed from link-to-link correlations because $\text{link}_{202:3}$-to-$\text{link}_{103:1}$ correlation exceeds 100% and grows rapidly; likewise, $\text{link}_{103:1}$-to-$\text{link}_{208:1}$ correlation and $\text{link}_{103:1}$-to-$\text{link}_{308:2}$ correlation degrade to zero gradually. The reason is that packets in a loop dominates the statistics related to the loop gradually.

The result confirms the effectiveness of PathMon for loop detection. PathMon allows administrators to set a range for each correlation and generate a notification when a correlation becomes out of range. Without PathMon, administrators need manually probe and analyse per-flow statistics from suspected switches to localize faulty flow entries.

B. Monitoring Overhead

Because PathMon pushes two VLAN tags to flows of interest, increased size of packet may affect end-to-end latency and throughput. We evaluate monitoring overhead by measuring

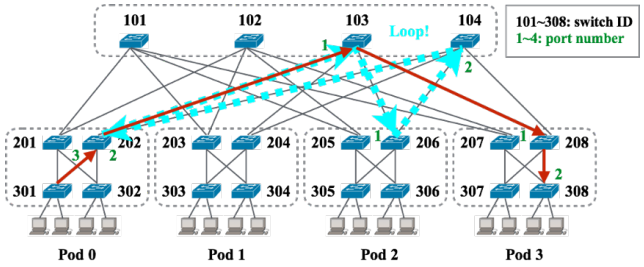


Fig. 6. Example of a forwarding loop consisting of four links: $\text{link}_{103:1}$, $\text{link}_{206:1}$, $\text{link}_{104:2}$, and $\text{link}_{202:2}$.

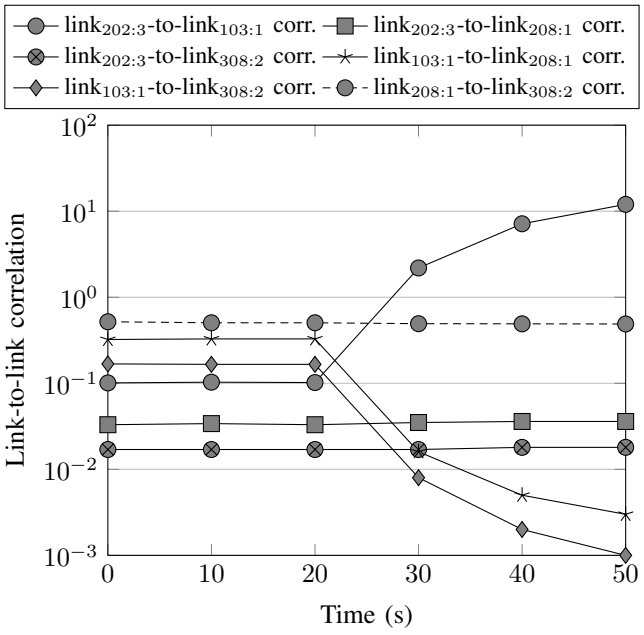


Fig. 7. Example of loop detection using link-to-link correlations of the monitoring target: $F ::= eth.type = 0x0800$ and $NP ::= (202:3)\,(103:1)\,(208:1)\,(308:2)$.

round-trip time (RTT) and throughput between hosts H1 and H2 connected by a linear topology with eight switches. The transmission rate of network links was limited to 10 Mbps. The monitored path includes all switches.

Impact on latency: We measured the average RTTs from H1 to H2 by ping with and without PathMon while disabling all background traffic. Because the size of an ICMP request is 98 bytes, the transmission delay can be ignored. The result shows the average RTTs with and without tagging are 2.1 ms and 2.96 ms, respectively; PathMon introduces 41% overhead on average.

Impact on throughput: We used iperf3 to generate unidirectional UDP traffic from H1 to H2 for which flow size and transmission rate were 100 MB and 10 Mbps, respectively. We measured throughputs for three different packet sizes: 500, 250 and 125 bytes which resulted in 209715, 419430 and 838860 packets, respectively. The result (Table III) shows that PathMon degraded throughput from 9.22 Mbps to 9.16 Mbps (0.65%) for packet size 500 bytes, from 9.22 Mbps to 9.09 Mbps (1.41%) for packet size 250 bytes, and from 8.29 Mbps to 8.14 Mbps (1.81%) for packet size 125 bytes. The ratio of throughput degradation was disproportionate to packet size. Tagging consumes no more than 1.81% of throughput at the extreme setting.

TABLE III
AVERAGE THROUGHPUT

Size/Number	Original/Mbps	PathMon/Mbps
500/209715	9.22	9.16
250/419430	9.22	9.09
125/838860	8.29	8.14

V. CONCLUSION

Monitoring health and performance metrics about networks is an essential function for enterprise network operators, data center network operators, and ISPs. In contrast to existing traffic monitoring tools for SDN, our approach decouples flow statistics of interest from existing flow entries in networks; PathMon employs a set of monitoring entries pushing/popping tags to enable path-specific flow monitoring with a minimal number of flow entries, whereas existing monitoring approaches do not provide this kind of monitoring. Furthermore, PathMon provides link-to-link correlations for each monitored path which can facilitate many network management tasks. Overall, our approach provides more accurate and detailed statistics which enhance the visibility of traffic distribution among different paths.

ACKNOWLEDGMENT

This work was supported in part by Ministry of Science and Technology, Taiwan, under Grants MOST 104-2221-E-009-021-MY3 and MOST 104-2622-8-009-001.

REFERENCES

[1] "Software-defined networking: The new norm for networks," White paper, Open Networking Foundation, April 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–9.

[4] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *Proceedings of the 14th International Conference on Passive and Active Measurement*, ser. PAM'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 31–41.

[5] "sflow," http://www.sflow.org/.
[6] N. van Adrichem, C. Doerr, and F. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–8.
[7] "Mininet," http://mininet.org/.
[8] "Floodlight," http://www.projectfloodlight.org/floodlight/.
[9] "Cpqd," https://github.com/CPqD/ofsoftswitch13.
[10] C.-Y. Lin, C. Chen, J.-W. Chang, and Y. H. Chu, "Elephant flow detection in datacenters using openflow-based hierarchical statistics pulling," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, Dec 2014, pp. 2264–2269.
[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62.
[12] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 202–208.
[13] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280.