FlowSpy:An Efficient Network Monitoring Framework using P4 in Software-Defined Networks

Bowei Guan, Shan-Hsiang Shen
Dept. of Computer Science and Information Engineering, National Taiwan Univ. of Science and Technology, Taiwan
Email: m10415802@mail.ntust.edu.tw, sshen@csie.ntust.edu.tw

***Abstract*—With the rapid development of network technology and growing services running, there is more network traffic on the Internet. To ensure the reliability and security of network, we need to do more effective network monitoring tasks that can help us gain more information for network troubleshooting and malicious traffic detection. Software-Defined Networks (or SDN, for short) provides a flexible platform for the network monitoring and relies on a central controller and switches interact with each other to gain a global view of traffic. However, the computation resources for network monitoring in switches are limited in SDN. Thus, too many monitoring tasks will affect data plane traffic performance. To address this issue, we propose FlowSpy, which is a load balancing network monitoring framework using P4 (Programming Protocol-Independent Packet Processors) programming language in SDN. P4 program can specify how a switch processes packets, that can reduce the overhead of the interaction between data plane and control plane in SDN, and provides more flexibility for monitoring than OpenFlow-based SDN. As compared to existing network monitoring methods, FlowSpy can take more monitoring capacity at each switch to complete more monitoring tasks without any overloaded nodes.**

***Index Terms*—Software-Defined Networking, Traffic Monitoring, P4, Resource Management**

I. Introduction

Based on Cisco Visual Networking Index: Forecast and Methodology (2016–2021) report [5], in 2016, global IP traffic was 1.2 ZB per year or 96 EB (one billion Gigabytes [GB]) per month. By 2021, global IP traffic will reach 3.3 ZB per year, or 278 EB per month. Global Internet traffic in 2021 will be equivalent to 127 times the volume of the entire global Internet in 2005. With growing traffic, DDoS (Distributed Denial of Service) and web application attacks both increase year by year based on Akamai Q4 2017 security report [1]. Network monitoring provides a view of the network status and illustrates network behavior. To ensure the reliability and security of network, we need to do more effective network monitoring tasks that can help us gain more information for network troubleshooting and malicious traffic detection [22]. However, in traditional networks, it is difficult to manage monitoring tasks and assign them flexibly to nodes (switches or middleboxes). Software-defined networking (SDN) [11] is a novel network architecture that provides a flexible network management and improves network programmability. SDN provides a global view of the entire network by a centralized control plane so that it is easier to monitor traffic in networks.

However, there are some challenges that affect the performance of network monitoring in SDN. The authors in FlowRadar [12] point out that the capacity of the most current switches is limited for monitoring. Moreover, networking monitoring affects data plane performance, DevoFlow [8] shows that TCP connection rate drops in half, if controller requests traffic information once per second in OpenFlow-based [14] switches. Thus, it is crucial to efficiently utilize restricted resources to distribute monitoring tasks among switches and satisfy more monitoring tasks.

Mann et al. [13] deploy their monitoring tasks on egress or ingress switches. In their solution, the egress or ingress switches are overloaded easily. Moreover, most of studies [2], [4], [6], [7], [10], [12], [13], [18], [19] on network monitoring in SDN focus on how to collect traffic information form OpenFlow switches or consider how to reduce monitoring bandwidth consumption [23]. However, none of them consider about global view of monitoring policy about how to choose measuring targets to assign monitoring tasks for balancing the load of traffic monitoring.

To address this issue, in this paper, we propose an efficient network monitoring framework in SDN named FlowSpy. The goal of FlowSpy is to assign monitoring tasks to switches evenly which means the load of monitoring tasks at switches is balanced. Moreover, FlowSpy uses control plane resources for monitoring more efficiently to potentially satisfy more monitoring tasks. Moreover, FlowSpy uses P4 [3] program to define the switch pipeline, which is how a switch processes packets, that can reduce the overhead of the interaction between data plane and control plane in SDN and provides more flexibility for monitoring than OpenFlow-based SDN. Moreover, the switch with P4 support can match more packet field than OpenFlow such as TCP SYN flag to detect SYN flooding attacks. To test the efficiency of FlowSpy, we evaluate FlowSpy in Mininet. The results show that FlowSpy utilizes monitoring resources more efficiently and switches can potentially complete more monitoring tasks than existing monitoring methods.

The rest of this paper is organized as follows. In Section II, we provide related studies. Section III presents the system architecture of FlowSpy and describes the core components within the FlowSpy framework. In Section IV we introduce the core functions of FlowSpy and discuss the problem formulation and $FlowSpy.p4$. We do emulation experiments to evaluate the efficiency of FlowSpy and also do simulation experiments for large scale network in Section V. Finally, we conclude this paper along with our future work in Section VI.

978-1-7281-1220-6/19/$31.00 ©2019 IEEE

II. Related Work

Most of studies [2], [4], [6], [7], [10], [12], [13], [18], [19] on network monitoring in SDN focus on how to collect raw traffic information from measuring targets and provide collection mechanisms. NetFlow [6] and sFlow [18] are widely used monitoring tool and both require device support. NetFlow is supported by Cisco, which can record flows and their properties for remote collectors. sFlow provides a network traffic sampling mechanism to export truncated packets, together with interface counters to remote collectors via sFlow agent. Living on the edge [13] implements NetFlow and sFlow inside the vSwitch on the edge of network and compares the performance of their monitoring results. MicroTE [2] tracks network traffic at data center servers and leverages the existence of short term and partial predictable traffic to mitigate unpredictable traffic congestion. Mahout [7] uses the shim layer in the end host to monitor socket buffers and uses OpenFlow-like central controller approach for elephant flow management. HONE [19] presents a software-defined networking platform for joint HOst-NEtwork (HONE) traffic management and provides a query language for ordering data collection across multiple hosts, which can reduce the controller load. PayLess [4] offers an adaptive scheduling algorithm for controller polling switches for monitoring data to reduce control channel communication overheads. Authors in [23] propose a flow monitoring algorithm to minimize monitoring bandwidth consumed. INT [10] is one of implementation of In-band Network Telemetry, that is the collection mechanism by data plane using P4, which provides collection and reporting of network state without generating extra packets.

However, none of them consider about global view of monitoring policy about how to choose measuring targets to assign monitoring tasks to balance the load of traffic monitoring.

III. System Architecture

We aim at providing an efficient monitoring system for switches in SDN. To achieve this and address the aforementioned challenges and limitations, we seek a solution that fulfills the following design requirements.

1) *Efficiency*. Network monitoring framework should have the capability to assign monitoring tasks to switches evenly, and all monitoring tasks can be completed at each switch. In addition, the control plane communicates with data plane should be more efficient to save the control plane resources.
2) *Deployability*. Network monitoring framework should be easy to deploy on SDN architecture. In addition, network monitoring framework should have a more flexible definition of the pipeline that specifies the packet we want to monitor.
3) *Usability*. Network monitoring framework should have a friendly interface for user to input monitoring tasks and then assign tasks to a massive number of switches automatically at the same time. Also, the network monitoring framework should provide a visual presentation of the progress of monitoring task.

We propose a comprehensive framework, FlowSpy, to accommodate our design requirements. FlowSpy is designed according to the SDN 3-planes architecture to balance network monitoring tasks among switches along with a variety of applications for optimization, visualization and integration of network monitoring. We next articulate the core components within the FlowSpy framework.

When a network monitoring system using FlowSpy framework is deployed, the $FlowSpy.p4$ is compiled as a target-specific configuration binary, which contains a set of monitoring tasks processing logic, such as how to trigger alert to controller and counting target packets. The target-specific configuration binary is sent to the switches which interact with controller via Remote Procedure Call (RPC) connection, and it is loaded as run-time monitoring function in P4 target on switch. The more details about $FlowSpy.p4$ will be discussed in Section IV-B.

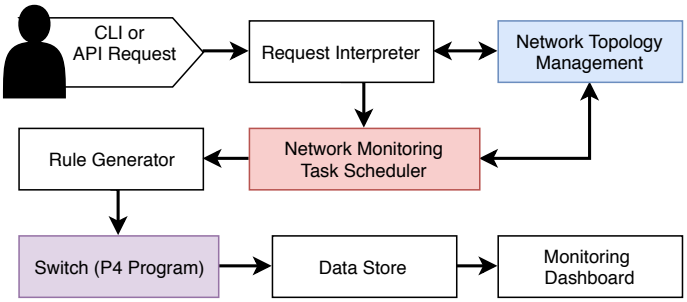


Fig. 1. The Workflow Chart of FlowSpy

The main workflow chart of our system is shown in Figure 1. A user like a network operator can use command line or API requests to send a set of monitoring tasks to FlowSpy, such as capture 90 packets from 10.0.0.1 to 10.0.0.2. Request Interpreter receives the monitoring tasks and asks Network Topology Management for the $Flow\_IDs$ and then translates monitoring tasks to flow-level type. $Rule\ Generator$ uses the analysis result of $Network\ Monitoring\ Task\ Scheduler$ and assigns monitoring rules to switches. Then, monitoring data will be sent to $Collector$ in the controller and passed to $Data\ Store$. Finally, $Monitoring\ Dashboard$ extracts the results provided by monitoring tasks from $Data\ Store$ and provides real-time presentation.

IV. FlowSpy

FlowSpy is divided into control and data planes. The control plane in a central controller determines monitoring task assignment by solving an Integer Linear Programming (ILP) problem, while the data plane in switches executes the monitoring tasks implemented as a P4 program.

A. Problem Formulation

To judge how monitoring tasks assignment is balancing, we choose $Mean\ Deviation$ which is a summary statistic of statistical dispersion or variability. Our goal is to assign monitoring tasks for requested flow to each node evenly and

it subjects to the monitoring and bandwidth capacities. To make the monitoring tasks assignment more balancing, we formulate the monitoring task assignment as an ILP problem and leverage ILP solver to determine the monitoring tasks assignment in FlowSpy. The notations are summarized in Table I and the problem is formulated as follows:

TABLE I
NOTATIONS

Notation	Description
F	A set of flows
N	Total number of nodes (Switches)
$x_{f,n}$	The number of requested monitoring tasks for flow f at node n
R_f	The number of requested monitoring tasks for flow f
C_n	Monitoring capacity at node n
$P_{f,n}$	Whether flow f passes through node n or not
B	A Big Number

$$\min \quad \frac{1}{N}\sum_{n=1}^{N}\left|\frac{1}{N}\sum_{f\in F}R_f-\sum_{f\in F}x_{f,n}\right| \qquad \text{(Origin)}$$

$$\text{s.t.} \quad x_{f,n}\le P_{f,n}\times B, \qquad \forall f\in F,\forall n\in[1,N] \qquad (1)$$

$$\sum_{n=1}^{N}x_{f,n}\le R_f, \qquad \forall f\in F \qquad (2)$$

$$\sum_{f\in F}x_{f,n}\le C_n, \qquad \forall n\in[1,N] \qquad (3)$$

$\sum_{f\in F}x_{f,n}$ means the number of monitoring tasks for flow f assigned to node n and it should be as close as possible to $\frac{1}{N}\sum_{f\in F}R_f$, which is the average number of monitoring tasks that should assigned to each node to satisfy all monitoring requests, in objective function.

The objective function above minimizes the $Mean\ \ Deviation$ of total number of requested monitoring tasks at all nodes. Moreover, the objective function is not linear and cannot be accepted by ILP solver, so it needs to be rewrited. We observe that $\left|\sum_{f\in F}x_{f,n}-\frac{1}{N}\sum_{f\in F}R_f\right|$ is the smallest number Υ that satisfies $\sum_{f\in F}x_{f,n}-\frac{1}{N}\sum_{f\in F}R_f\le\Upsilon$, and we use Constraints 7 and 8 to obtain the linear programming formulation as following:

$$\min \quad \frac{1}{N}\sum_{n=1}^{N}\Upsilon \qquad \text{(ILP)}$$

$$\text{s.t.} \quad x_{f,n}\le P_{f,n}\times B, \qquad \forall f\in F,\forall n\in[1,N] \qquad (4)$$

$$\sum_{n=1}^{N}x_{f,n}\le R_f, \qquad \forall f\in F \qquad (5)$$

$$\sum_{f\in F}x_{f,n}\le C_n, \qquad \forall n\in[1,N] \qquad (6)$$

$$\sum_{f\in F}x_{f,n}-\frac{1}{N}\sum_{f\in F}R_f\le\Upsilon, \qquad \forall n\in[1,N] \qquad (7)$$

$$\frac{1}{N}\sum_{f\in F}R_f-\sum_{f\in F}x_{f,n}\le\Upsilon, \qquad \forall n\in[1,N] \qquad (8)$$

The objective function minimizes the mean of Υ. In Constraint 4, if the flow f doesn't pass through the node n, $P_{f,n}$ should be 0, which means the requested monitoring task should be assigned at the node that flow passed through, otherwise, the number of requested monitoring tasks for flow f in this node n will be less than a big number, that doesn't affect the optimization results. Constraint 5 shows that the number of monitoring tasks for each flow should not less than the number of requested to satisfy the monitoring requests. Constraint 6 is node capacity constraint, which means the monitoring capacity of each node are limited, if the number of monitoring tasks assigned exceeds the capacity of the node, it will affect the capacity of node for packet forwarding and increase the loading of the node.

B. P4 Program for Monitoring

$FlowSpy.p4$ is one of the FlowSpy core functions and relays on $P4\ \ Architecture\ \ Model$ provided by switch's vendor. We can define how a switch parse the packet header of incoming packet in $FlowSpy.p4$. When content extracted match some rules, it will trigger some actions. We can define the match-action behavior in $FlowSpy.p4$. Moreover, we can insert or modify the rules in the tables of switch in runtime via RPC connection.

When the packet arrives at the switch, $FlowSpy.p4$ will parse headers we concern based on the parsing process defined in $FlowSpy.p4$ to extract the packet headers. After $FlowSpy.p4$ verifies the checksum of the packet, the packet is passed to Ingress Pipeline. There are Match-Action rules we defined in $FlowSpy.p4$, and the key of match value, action and action data can be set in real-time by FlowSpy Application via the controller. For example, we monitor the packets from source IP address 10.0.1.1 to destination IP address 10.2.2.2, and expect an alert to be triggered when the number of packets exceeds a certain threshold. We set do_set_alert action in ingress match-action to tag some meta data and update the register of the counter, when the packets arrive the Egress Pipeline. When the value of the counter register exceeds the threshold θ we set, it will trigger $send\_alert$ action, recompute the checksum of the packets, deparser header and send to controller via control port.

The controller and switch within $FlowSpy.p4$ are able to interact with RPC connection in real-time. We can assign monitoring rules to switch by FlowSpy application, such as $Rule\ \ Generator$. We can not only set the match key, action and action data of the tables, but also update the value of counter and register. Then, $FlowSpy.p4$ follows the monitoring rules and sends the monitoring data from the control port of switch to the collector in the controller; then, the statistic data is forwarded to the FlowSpy application for further analysis.

V. Performance Evaluation

In this section, we evaluate the performance of FlowSpy based on a real-world network topology Germany50 derived from the SNDLib website [17] including 50 nodes and 88 links.

A. Implementation

In the control plane, to determine monitoring task assignment, FlowSpay calculates ILP model in Network Monitoring Task Scheduler by using the Gurobi Optimizer [9]. The data plane of FlowSpy is implemented by using $P4_{16}$ [20] program via BMv2 (Behavioral Model version 2, P4 software switch) [21][1] and Mininet. We attempt to evaluate the efficiency of FlowSpy with 3 metrics (in Section V-C) compared with other 3 common monitoring task assignment methods (in Section I). As mentioned in previous Section I, there are 3 terms (the number of monitoring tasks, the capacity of node and the number of monitoring requests) will determine network the monitoring performance, so we make 3 kinds of scenarios in the evaluation (in Section V-B).

B. Experiment Scenarios

We change the following factors to evaluate FlowSpy.

Different Monitoring Capacity of Nodes (or $Capacity$, for short): In this scenario, we assume the capacity of each node is the same. We set $E(R)$ to 30 (packets) with 150 flows selected randomly and evaluate the performance with different $Capacity$.

Different Total Number of target Flows (or $Number\ of\ Flows$, for short): In this scenario, we set $Capacity$ to 300(packets) and $E(R)$ to 90 (packets), and evaluate the performance with different number of flows.

Different Requested Packet Capture Rates for Flows (or $E(R)$, for short): We set the requested packet capture rates for different flows in normal distribution. The requested packet capture rate R is generated following normal distribution with mean $E(R)$ and variance is 5:

$$R \sim \mathcal{N}(E(R), 5). \tag{9}$$

FlowSpy is compared with the following task assignment methods: (1) dst places monitoring tasks in egress nodes, (2) src places monitoring tasks in ingress nodes, and (3) $random$ randomly places monitoring tasks along the paths of flows.

C. Experiment Metrics

The Mean Deviation of work load in nodes (or $Mean\ Deviation$, for short): whether monitoring tasks assignment is balanced as the objective of the problem formulation in Section IV-A.

The Completeness Rate of Monitoring Tasks (or $Completeness\ Rate$, for short): When all requested capture packets for a monitoring task are successfully collected to Data Store, that means the monitoring task is completed. We count how many monitoring tasks are successfully completed.

[1]We choose BMv2 because it's one of the most popular software switches supporting P4 program.

D. Emulation Results

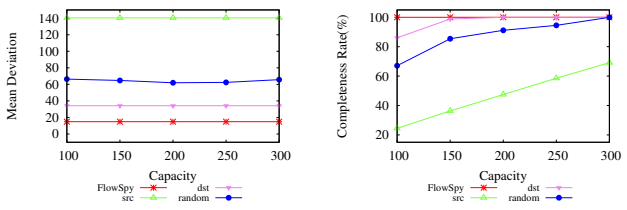
Figure 2(a) shows the mean deviation of work load at nodes with different monitoring capacity provided by the nodes. As the monitoring capacity changes, the mean deviation at all nodes are almost the same with the 4 methods, where $E(R)$ is 30(packets) and 150 flows we requested for monitoring in this scenario, because the target flows are the same and algorithms provide similar assignment with different monitoring capacity. FlowSpy evenly distributes monitoring tasks, so it works better than others. Figure 2(b) shows the completeness rate of monitoring tasks with different monitoring capacity in nodes. When the monitoring capacity is growing, the completeness rates of monitoring tasks provided by ILP model are remaining stable, which is near 100%, and the completeness rates of monitoring tasks also increase with the other three methods with growing monitoring capacity, that means with more monitoring capacity, each node can complete more monitoring tasks.

Figure 2(c) shows the mean deviation of total number of requested monitoring tasks at all nodes with different total number of target flows. When the total number of target flows is growing, the mean deviation at all nodes also increase. We observe that FlowSpy is the most stable and all mean deviation values provided by FlowSpy are less than 50. Figure 2(d) shows the completeness rate of monitoring tasks with the different total number of target flows. As the total number of target flows grows, the completeness rates of monitoring tasks provided by FlowSpy are remaining stable near 100%, and the completeness rates of monitoring tasks decrease with other 3 methods. We observe that more target flows will cause more overloaded nodes, so the completeness rates of monitoring tasks drop in the source edge node method, the destination edge node method and random node method. However, our FlowSpy can effectively distribute the monitoring loading to nodes evenly.

Figure 2(e) and Figure 2(f) show that the mean deviation of the total number of requested monitoring tasks at all nodes and the completeness rate of monitoring tasks with different requested packet capture rates for flows. The mean deviation of the total number of requested monitoring tasks increases with the request packet capture rates, but the increase is not critical in FlowSpy case, because FlowSpy can still distribute the monitoring load even with high request packet capture rates. The higher capture rates also increase the load in the nodes, so the completeness rates drop in src, dst, and $random$ cases. However, FlowSpy can still keep high completeness rates. To summarize the evaluation, FlowSpy outperforms the other three methods by successfully distributing monitoring tasks evenly among nodes.

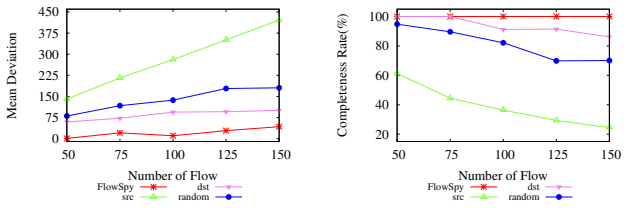
VI. Conclusions and Future Work

With growing network traffic, we need to do more network monitoring to ensure the reliability and security of the network. In this paper, we present FlowSpy, an efficient framework to avoid overloading the network monitoring capacity of each node (switch) by providing better load balancing network



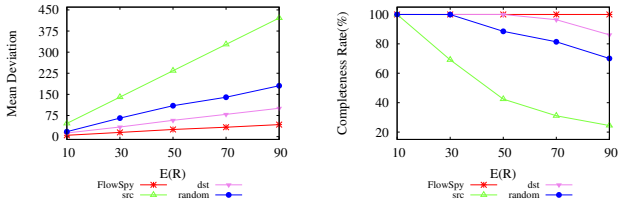
(a) The Mean Deviation of All Nodes with Different Node Capacity

(b) The Completeness Rate of Monitoring Tasks with Different Node Capacity



(c) The Mean Deviation of All Nodes with Different Number of Flows

(d) The Completeness Rate of Monitoring Tasks with Different Number of Flows



(e) The Mean Deviation of All Nodes with Different E(R)

(f) The Completeness Rate of Monitoring Tasks with Different E(R)

Fig. 2. Emulation Results: Mean Deviation and Completeness Rate

monitoring using P4 in software-defined networks. In addition, we do simulation experiments by Python program, also do emulation experiments to implement FlowSpy with real network topology using $P4_{16}$ program via BMv2 software switch and Mininet. The results show that FlowSpy can complete all network monitoring tasks and the load of each node is the most balanced compared to traditional SDN monitoring schemes, without any overloaded nodes. In the future, we will implement and deploy FlowSpy using ONOS controller [15] for support Stratum Project [16].

REFERENCES

[1] Akamai. Q4 2017 State of the Internet / Security Report. https://www.akamai.com/stateoftheinternet-security, 2017. [Online].

[2] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In *ACM CoNEXT '11*, pages 1–12.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[4] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *IEEE NOMS '14*, pages 1–9.

[5] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, 2017. [Online].

[6] Cisco. NetFlow. https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html, 2018. [Online].

[7] A. R. Curtis, W. Kim, and P. Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *IEEE INFOCOM '11*, pages 1629–1637.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *ACM SIGCOMM '11*, pages 254–265.

[9] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. http://www.gurobi.com, 2018. [Online].

[10] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L. J. Wobker, and B. Networks. In-band Network Telemetry via Programmable Dataplanes. *ACM SIGCOMM*.

[11] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[12] Y. Li, R. Miao, C. Kim, and M. Yu. Flowradar: A better netflow for data centers. In *USENIX NSDI '16)*, pages 311–324.

[13] V. Mann, A. Vishnoi, and S. Bidkar. Living on the edge: Monitoring network flows at the edge in cloud data centers. In *COMSNETS '13*, pages 1–9.

[14] 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.*, 38(2):69–74, Mar. 2008.

[15] Open Networking Foundation. ONOS. https://onosproject.org/, 2018. [Online].

[16] Open Networking Foundation. Stratum. https://stratumproject.org/, 2018. [Online].

[17] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *INOC '07*.

[18] sFlow.org. sFlow. https://sflow.org/, 2018. [Online].

[19] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker. Hone: Joint host-network traffic management in software-defined networks. *J. Netw. Syst. Manage.*, 23(2):374–399, Apr. 2015.

[20] The P4 Language Consortium. $P4_{16}$ language specification. https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html, 2017. [Online].

[21] The P4 Language Consortium. Behavioral Model version 2. https://github.com/p4lang/behavioral-model, 2018. [Online].

[22] P. W. Tsai, C. W. Tsai, C. W. Hsu, and C. S. Yang. Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, pages 1–12, 2018.

[23] Z. Yang and K. L. Yeung. An efficient flow monitoring scheme for sdn networks. In *IEEE CCECE '17*, pages 1–4.