Minimizing Flow Statistics Collection Cost of SDN Using Wildcard Requests

Hongli Xu[1] Zhuolong Yu[1] Chen Qian[2] Xiang-Yang Li[1,3] Zichun Liu[1]
Email: xuhongli@ustc.edu.cn, yzl123@mail.ustc.edu.cn, cqian12@ucsc.edu, xiangyangli@ustc.edu.cn, lzc223@mail.ustc.edu.cn
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Department of Computer Engineering, University of California at Santa Cruz
[3]Department of Computer Science, Illinois Institute of Technology

***Abstract*—In a software defined network (SDN), the control plane needs to frequently collect flow statistics measured at the data plane switches for different applications, such as traffic engineering, flow re-routing, and attack detection. However, existing solutions for flow statistics collection may result in large bandwidth cost in the control channel and long processing delay on switches, which significantly interfere with the basic functions such as packet forwarding and route update. To address this challenge, we propose a Cost-Optimized Flow Statistics Collection (CO-FSC) scheme using wildcard-based requests. We prove that the CO-FSC problem is NP-Hard and present a rounding-based algorithm with an approximation factor f, where f is the maximum number of switches visited by each flow. Moreover, our CO-FSC problem is extended to the general case, in which only a part of flows in a network need to be collected. The extensive simulation results show that the proposed algorithms can reduce the bandwidth overhead by over 41% and switch processing delay by over 45% compared with the existing solutions.**

***Index Terms*—Software Defined Networks, Flow Statistics Collection, Cost, Wildcard, Approximation.**

I. Introduction

A typical SDN consists of a logical controller in the control plane and a set of switches in the data plane [1]. The controller monitors the network and determines the forwarding path of each flow. The switches perform packet forwarding and traffic measurement for flows based on the rules installed by the controller. Since the controller is able to provide centralized control for each flow, an SDN can help to implement fine-grained management and improve the network resource utilization compared with traditional networks [2].

To explore full advantages of centralized control, an accurate global view of flow traffic is instrumental to various applications, such as traffic engineering, QoS routing, and network attack detection. For example, if the controller performs flow routing without accurate flow traffic knowledge, it will often result in lower throughput or load imbalance. With an accurate global view of flow traffic, it helps to improve the route QoS [2] [3], such as maximum throughput, low latency, and high reliability. As another application example, some security attacks, *e.g.* DDoS [4] [5], are often detected by analyzing the changes of flow traffic. Accurate flow statistics also help to detect the attacks and protect the network. Therefore, it is of vital importance to collect accurate flow statistics.

In an SDN, switches are able to measure different per-flow traffic statistics, including packets, bytes or duration, through flow entries. To allow the controller to obtain traffic statistics information, OpenFlow [6] specifies two different approaches for flow statistics collection (FSC) from the switches.[1] One is the *push-based* mechanism [11]. The controller learns active flows and their statistics by passively receiving reports from the switches. However, several factors limit its application in practice. First, it needs some additional requirements on both hardware and software for the push-based FSC, such as counters and comparators to support report triggers [11]. Moreover, different from OpenFlow's specification, most commodity switches do not inform the controller about the measurement of a flow until the flow entry times out. Thus, these switches do not support the push-based flow statistics collection. Second, even though the push-based collection is feasible on SDN switches, when the traffic varies dynamically, the switches will be frequently triggered and send massive number of measurement reports to the controller, causing large cost of control channel bandwidth [12] and the controller's CPU resource [13]. The other is the *pull-based* method: the controller just sends a `Read-State` message (also called FSC request) to retrieve the flow statistics from a switch. Since this mechanism needs no additional requirement on both hardware and software for switches and can control the number of measurement reports, it has been widely used in SDN applications [13] [14] [15]. In this work, we focus on the pull-based FSC method.

Due to flow dynamics, timely collection of flow statistics in an SDN is required for various applications such as traffic engineering [16]. There are two main schemes of pull-based flow statistics collection, *per-flow collection* [14] [17] and *per-switch collection* [13] [15]. However, *we demonstrate that both two schemes may lead to massive cost of the control channel bandwidth and long processing delay on the switches*, especially in dynamic networks. This cost of the control channel and switches will significantly interfere with basic functions such as packet forwarding and route update.

[1]Note that FSC is different from the flow traffic measurement problem in SDNs [7] [8] [9] [10], which studies how switches derive flow statistics. FSC focuses on how switches report the collected flow statistics to the controller.

For per-flow collection, the controller sends a request to a switch for collecting the traffic statistic of exactly one flow. When the controller requires to collect statistics of many flows, it may generate a large number of FSC requests on each switch. These requests will compete with control messages, *e.g.*, rule setup and update, for the downlink bandwidth of the control channel.[2] Hence with more FSC requests, the rule setup and update messages, serving the fundamental SDN tasks, may be delayed, lost, or disordered [11], resulting in data plane faults or inconsistency.

For per-switch collection, the controller sends a request to collect the traffic statistics of all flow entries from a switch. This scheme brings two main disadvantages: 1) It may lead to unacceptable delay. For example, from the experiments on the HP ProCurve 5406zl switch, it takes about one second to collect traffic statistics of about 5600 flows, even when there is no traffic load on the switch [11]. Ordinary packet forwarding experiences significantly lower throughput during this period. Moreover, one second is still too long for many flow schedulers such as Hedera [16] to conduct accurate routing optimization. 2) It may lead to uplink congestion of the control channel because many flow statistics that have no need to collect will also be frequently reported to the controller. Consider the case that only a small part of flows have varied transmission rate, frequent FSC of all flows is obviously a significant waste of uplink bandwidth. For example, it takes 96 bytes for statistics of each flow entry specified by OpenFlow 1.3 [6]. Then the statistics for 16K exact-matched rules on a 5406zl switch would cost 1.53MB. Setting the FSC rate as twice per second would require 24.6Mbps bandwidth on a control link for only one switch. The massive traffic on control links may increase the delay and loss ratio of control commands.

Therefore, *it is an urgent need to design a new solution of FSC with lower control channel and switch cost*, so that basic functions on switches will be less interfered. Our solution is motivated by the following considerations. To avoid long-delay collection on some switches and massive traffic load on control links, we expect to distribute the FSC among all switches. On each switch only the traffic statistics of a subset of flows will be collected. We implement the fast and selective FSC to collect a subset of flows on a switch using the wildcard-based FSC request, which can be successfully implemented using OpenFlow 1.3. The controller will send FSC requests, each of which contains one wildcard rule, to switches. On the switch only flows matching the wildcard rule will be collected and reported to the controller. We design algorithms to minimize the maximum bandwidth/delay cost among all switches and extend our solutions to partial FSC, where only a subset of flows (not all flows) in the network are collected. The main contributions of this paper are:

1) We propose the cost-optimized flow statistics collection (CO-FSC) problem, prove its NP-Hardness, and present a rounding-based algorithm, called R-FSC. R-FSC achieves the approximation factor of f, where f is the maximum number of switches visited by each flow in a network. Moreover, a primal-dual-based algorithm with lower time complexity is also presented.
2) We extend CO-FSC to the cost-optimized partial flow statistics collection (CO-PFSC) problem, and design a rounding-based algorithm for this problem. The approximation factor of the proposed algorithm is also analyzed.
3) The simulation results show that our algorithms help to reduce the bandwidth overhead by over 41% and switch processing delay by over 45% compared with the existing FSC solutions. Moreover, our partial FSC algorithm reduces the cost by 52% compared with FSC while preserving almost the similar application performance.

[2]We use “downlink” to denote the control channel from the controller to switches and “uplink” to denote that from switches to the controller.

The rest of this paper is organized as follows. Section II formalizes the CO-FSC problem. We propose an approximation algorithm and a primal-dual-based algorithm for CO-FSC in Section III. Section IV designs a rounding-based algorithm for CO-PFSC, and analyzes its approximation performance. We report our simulation results in Section V. We review related work in Section VI and conclude the paper in Section VII.

II. Preliminaries

In this section, we introduce the network model and formalize the CO-FSC problem.

A. Network and Flow Models

An SDN typically consists of a logically-centralized controller and a set of switches, $V = \{v_1, ..., v_n\}$, with $n = |V|$. These switches comprise the data plane of an SDN. Thus, the network topology from a view of the data plane can be modeled by $G = (V, E)$, where E is the set of links connecting switches. Besides the data plane links, there is a set of links serving the control channel connecting the switches and the controller. Note that the controller may be a cluster of distributed controllers [18], which help to balance the overhead among these controllers. Since the metric we evaluate is per-switch bandwidth/delay cost, we assume that there is only one controller for simplicity of presentation.

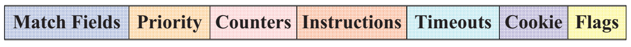

Fig. 1: Illustration of a Standard Flow Entry in a Flow Table.

Each switch has a flow table that performs packet forwarding and traffic measurement. A flow table consists of a certain number of flow entries (also called as rules). A standard flow entry, specified by OpenFlow 1.3 [6], is illustrated in Fig. 1. The match fields and priority together identify a unique entry in the flow table, and the switch measures traffic in the counters field. When a packet arrives at a switch, the header packet will be examined. If there is at least one flow entry that matches the packet, this switch picks the entry with the highest priority and performs the action specified by the instruction field of the entry. Otherwise, the switch reports the header packet to the controller, which shall determine the forwarding path for this flow and setup a sequence of rules to the switches on the path.

B. Advantages of FSC Using Wildcard Requests

Comparing with the previous pull-based FSC methods, wildcard-based FSC has two main advantages. First, using wildcard, the controller can collect statistics of many flows, not just one flow, per request. Thus, the wildcard method can significantly reduce the number of requests and switch overhead compared with per-flow collection method. Second, using wildcard, it is feasible to collect statistics of a subset of flows from a switch. Thus, the wildcard solution is able to distribute the FSC overload among all switches, which helps to reduce the switch cost compared with the per-switch method.

C. Cost-Optimized Flow Statistics Collection (CO-FSC)

Under the general SDN framework, switches report the header packet of each new-arrival flow to the controller. Thus, it is reasonable to assume that the controller knows the existing flows in a network, denoted by $\Gamma = \{\gamma_1, ..., \gamma_m\}$, with $m = |\Gamma|$. Since the forwarding path for each flow is determined by the controller, we also know the flow set, denoted by Γ_i, that passes through each switch v_i. For a flow, if its traffic statistic is gathered, the controller knows its actual number of packets (or traffic intensity). We say that this flow is covered. As the traffic statistic of some flow does not change, this flow has terminated, and the controller will delete the corresponding entry of this flow, and update the current flow set Γ.

Assume that there is a set of wildcards, denoted by $\mathcal{R} = \{r_1, r_2, ..., r_q\}$, with $q = |\mathcal{R}|$. For example, a natural way for setting wildcards is as follows: each wildcard r_j only specifies the destination v_j, and can match all the sources. When the controller sends a Read-State command with wildcard r_j to switch v_i, or we say that wildcard r_j is applied on switch v_i, the switch assembles the flow entries matching with this wildcard into a reply packet, and sends to the controller. Under this case, assume that the covered flow set is denoted by Γ_i^j. From this example, the wildcard rules often satisfy the following two features: (1) The completeness feature, that is, $\bigcup_{r_j \in \mathcal{R}} \Gamma_i^j = \Gamma_i$, $\forall v_i \in V$. (2) The mutual exclusion feature, that is, $\Gamma_i^{j_1} \cap \Gamma_i^{j_2} = \Phi$, $\forall r_{j_1} \neq r_{j_2}$, $\forall v_i \in V$.

When wildcard r_j is applied on switch v_i, a flow set Γ_i^j will be covered/collected, and the cost on switch v_i is denoted by $c(\Gamma_i^j)$. The cost function $c(\Gamma_i^j)$ is usually defined as $c_1 \cdot |\Gamma_i^j| + c_2$, where c_1 and c_2 are constant and determined by different performance metrics, such as bandwidth or delay costs.

- We consider the bandwidth cost as the total bandwidth for FSC of a flow set Γ_i^j. As specified in Openflow v1.3 [6], the bandwidth cost for statistics collection of a set Γ_i^j consists of two parts: (1) the request packet, whose length is 122 bytes; (2) the reply packet, whose length depends on the number of covered flows in Γ_i^j. It is expressed as $l_h + l_f \cdot |\Gamma_i^j|$, where l_h is the length of header packet, and l_f is the length for each flow entry, respectively. According to [15] [6], l_h and l_f are 78 bytes and 96 bytes, respectively. Thus, the bandwidth cost is modeled as $c(\Gamma_i^j) = 96 \cdot |\Gamma_i^j| + 200$ with unit byte.
- We consider the delay cost $c(\Gamma_i^j)$ as the delay for statistics collection of a flow set Γ_i^j. By testing on the HP switch [11], with the increasing number of flows, the delay for flow statistics collection is almost *linearly* increasing, and the increase rate is about $\frac{1000}{5600} = 0.18$ms/flow. When we test on our H3C switch, it takes about 1.4ms and 21ms to collect statistics of one flow and 100 flows using the per-flow and per-switch collection interfaces, respectively. The increase rate is about 0.198ms/flow. Combining the above testing results, the delay cost for Γ_i^j can be approximately modeled as $c(\Gamma_i^j) = 0.19 \cdot |\Gamma_i^j| + 1.21$ with unit ms.

When an FSC event is triggered, the controller will send Read-State commands, each of which contains a wildcard rule, to different switches, so that all the flows can be covered. We should note that the controller may send several commands to one switch per FSC event. As a result, the cost on switch v_i is denoted by $c(v_i)$. Our objective is to minimize the maximum cost among all the switches, that is, $\min \max\{c(v_i), 1 \leq i \leq n\}$, so that no performance bottleneck will happen in SDNs.

Accordingly we formalize the CO-FSC problem as follows:

$$\min \quad \lambda$$
$$S.t. \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} x_i^j \geq 1, & \forall \gamma_k \in \Gamma \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j) \leq \lambda, & \forall v_i \in V \\ x_i^j \in \{0, 1\}, & \forall r_j \in \mathcal{R} \end{cases} \tag{1}$$

where x_i^j denotes whether the controller will send a Read-State command with wildcard r_j to switch v_i or not. The first set of inequalities denotes that each flow will be covered. The second set of inequalities means that the total cost on each switch v_i should not exceed λ. The objective is to minimize the maximum cost on all the switches (or to achieve the cost balancing on switches), that is, $\min \ \lambda$.

Theorem 1: The CO-FSC problem is NP-hard.

We can prove the NP-hardness by showing that the unrelated processor scheduling problem [19] is a special case of CO-FSC. Due to limited space, we omit the detailed proof here.

III. Algorithm Design of CO-FSC

Due to the hardness of CO-FSC, we first design an approximation algorithm using the rounding method for this problem (*Section III-A*), and give performance analysis (*Section III-B*). Then, we present an algorithm based on primal-dual with lower time complexity (*Section III-C*). We also discuss an application of our FSC algorithms (*Section III-D*).

A. A Rounding-Based Algorithm for CO-FSC

This section develops a rounding-based algorithm, called R-FSC, to solve the CO-FSC problem. The proposed algorithm consists of two main steps. As CO-FSC is an NP-Hard problem, the first step will relax the integer program formulation to a linear program as Eq. (2).

$$\min \quad \lambda$$
$$S.t. \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} x_i^j \geq 1, & \forall \gamma_k \in \Gamma \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j) \leq \lambda, & \forall v_i \in V \\ x_i^j \geq 0, & \forall r_j \in \mathcal{R} \end{cases} \quad (2)$$

We can solve Eq. (2) in polynomial time, and obtain the fractional solution, denoted by $\widetilde{x}$. In the second step, the fractional solution will be rounded to the 0-1 solution for flow statistics collection. The set of uncovered flows is denoted by Γ^u, which is initialized as all flows in Γ. We arbitrarily choose an uncovered flow, denoted by γ, from set Γ^u. Then, the algorithm chooses a flow set Γ_i^j, whose $\widetilde{x}_i^j$ is maximum among all these sets containing flow γ, and set $\widehat{x}_i^j = 1$. That is, the controller will send a Read-State command with wildcard r_j to switch v_i. Moreover, we update $\Gamma^u = \Gamma^u - \Gamma_i^j$. The algorithm will terminate until all flows are covered. The R-FSC algorithm is described in Alg. 1.

Algorithm 1 R-FSC: Rounding-based FSC

```
1: Step 1: Solving the Relaxed CO-FSC Problem
2: Construct the relaxed problem as Eq. (2)
3: Obtain a fraction solution $\widetilde{x}$
4: Step 2: Rounding to 0-1 Solution
5: $\Gamma^u = \Gamma$
6: while $\Gamma^u \neq \Phi$ do
7:     Arbitrarily choose an uncovered flow $\gamma$
8:     Choose a flow set $\Gamma_i^j$, whose $\widetilde{x}_i^j$ is maximum among
       all these sets containing flow $\gamma$, and set $\widehat{x}_i^j = 1$
9:     $\Gamma^u = \Gamma^u - \Gamma_i^j$
```

B. Performance Analysis

We analyze the approximate performance of the proposed algorithm. In the second step, we arbitrarily choose an uncovered flow γ. Let Γ^γ denote all the sets that contain this flow. By the second feature of wildcard rules, $|\Gamma^\gamma| \leq f$, where f is the maximum number of switches visited by each flow. Since each flow γ will be covered, $\sum_{\Gamma_i^j \in \Gamma^\gamma} \widetilde{x}_i^j \geq 1$. Assume that a flow set Γ_i^j is chosen in some iteration. It follows $\widetilde{x}_i^j \geq \frac{1}{f}$.

After solving the linear program in the first step of the R-FSC algorithm, we derive a fractional solution $\widetilde{x}$ and an optimal result $\widetilde{\lambda}$ for the relaxed CO-FSC problem. According to the algorithm description, the final cost of switch v_i is:

$$\begin{aligned} c(v_i) &= \sum_{r_j \in \mathcal{R}} \widehat{x}_i^j \cdot c(\Gamma_i^j) \\ &\leq \sum_{r_j \in \mathcal{R}} f \cdot \widetilde{x}_i^j \cdot c(\Gamma_i^j) \leq f \cdot \widetilde{\lambda} \end{aligned} \quad (3)$$

Thus, we can conclude that

Theorem 2: The R-FSC algorithm can achieve the f-approximation for the CO-FSC problem.

C. A Lower-Complexity Algorithm Using Primal-Dual

When a statistics collection event is triggered, we expect that the controller can immediately determine the solution for FSC. The R-FSC algorithm needs to solve the linear program of Eq. (2). Since the number of variables in Eq. (2) mainly depends on the number of flows in an SDN, it may contain a large number of variables for a large-scale network, and it is rather costly in practice to solve such a linear program. Thus, this section designs a primal-dual-based algorithm with lower time complexity for CO-FSC. The primal-dual version of Eq. (2) is given in Eq (4), in which two sets of variables χ and μ denote the first and second sets of constraints in Eq. (2).

Algorithm 2 FSC-PD: FSC based on Primal-Dual

```
1: $\Gamma^u = \Gamma$
2: for each switch $v_i \in V$ do
3:     Compute $\mu_i$ by Eq. (6)
4: while $|\Gamma^u| > 0$ do
5:     for each uncovered flow set $\Gamma_i^j$ do
6:         Increase $\chi_k$ for each uncovered flow $\gamma_k$ to value $\delta_i^j$,
           so that $\sum_{\gamma_k \in \Gamma_i^j} \chi_k = \mu_i \cdot c(\Gamma_i^j)$
7:     Choose a flow set $\Gamma_i^j$ with the minimum value $\delta_i^j$ among
       all the uncovered sets
8:     for each flow $\gamma_k \in \Gamma_i^j - \Gamma^u$ do
9:         $\chi_k = \delta_i^j$
10:    $\Gamma^u = \Gamma^u - \Gamma_i^j$
```

$$\max \quad \sum_{k=1}^{m} \chi_k$$
$$S.t. \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} \chi_k - \mu_i \cdot c(\Gamma_i^j) \leq 0, & \forall i, j \\ \sum_{i=1}^{n} \mu_i \leq 1, & \\ 0 \leq \mu_i \leq 1, & \forall i \end{cases} \quad (4)$$

According to Eq. (4), we design the FSC-PD algorithm for the CO-FSC problem. At the beginning, the algorithm initializes a variable, Γ^u, which denotes the uncovered flow set. To maximize the objective function in Eq. (4), we expect that each variable χ_k can almost grow in line to a value δ. By the first set of inequalities of Eq. (4), we have

$$\sum_{\gamma_k \in \Gamma_i} \chi_k = \sum_{r_j \in \mathcal{R}} \sum_{\gamma_k \in \Gamma_i^j} \chi_k \leq \sum_{r_j \in \mathcal{R}} \mu_i \cdot c(\Gamma_i^j) \quad (5)$$

For simplicity, let $g(v_i)$ be $\sum_{r_j \in \mathcal{R}} c(\Gamma_i^j)$. As we consider the critical case, it follows that $\delta \approx \frac{\mu_i \cdot g(v_i)}{n_i}$, where n_i is the number of flows through switch v_i. Combining with the second inequality of Eq. (4), we set each variable μ_i as

$$\mu_i = \frac{\frac{n_i}{g(v_i)}}{\sum_{v_i \in V} \frac{n_i}{g(v_i)}} \quad (6)$$

For ease description, if there is an uncovered flow in set Γ_i^j, this set is accordingly uncovered. The algorithm mainly comprises a group of iterations. In each iteration, for each uncovered flow set Γ_i^j, the algorithm increases χ_k for each uncovered flow γ_k to value δ_i^j, so that $\sum_{\gamma_k \in \Gamma_i^j} \chi_k = \mu_i \cdot c(\Gamma_i^j)$. We then choose a minimum value, denoted by δ_i^j, which means that Γ_i^j will be chosen for flow statistics collection, or wildcard r_j will be applied on switch v_i. We update the variable χ_k for each flow $\gamma_k \in \Gamma_i^j - \Gamma^u$ as $\chi_k = \delta_i^j$. If all the flows are covered, the algorithm terminates. Otherwise, we continue a new iteration. The FSC-PD algorithm is described in Alg. 2.

Theorem 3: The time complexity of FSC-PD is $O(mnqf)$, where m, n, q and f are the number of flows, the number of switches, the number of wildcard rules, and the maximum number of switches visited by each flow in a network.

Proof: The algorithm first takes $O(mf)$ to initialize each variable μ_i, for all flows on each switch will be processed. Since there are n switches and q wildcard rules, the algorithm consists of $O(nq)$ iterations at most. In each iteration, the algorithm computes the value δ_i^j for each uncovered flow set Γ_i^j, which takes a time complexity of $O(mf)$. Moreover, the algorithm will cost a time complexity of $O(m)$ to update variable Γ^u. Then, the total time complexity of the FSC-PD algorithm is $O(mnqf)$. ■

D. Flow Re-routing using FSC

After flow statistics collection, we can optimize the network performance, such as load balancing, by re-routing flows. The controller determines an elephant flow set, denoted by $\Gamma^e = \{\gamma_1, ..., \gamma_{m^e}\}$, with $m^e = |\Gamma^e|$. We define one as an elephant flow that has transferred at least a threshold number of bytes X. A reasonable value for X is 1-10MB [11]. For each elephant flow $\gamma \in \Gamma^e$, the controller finds the least congested path between the flow's endpoints. To fulfill this function, we use the Dijkstra method to explore a set of the shortest paths of each flow. The algorithm chooses one with the least congestion as a new route of this flow. After determining new routes for all the flows in Γ^e, we execute the route reconfiguration without transient congestion [20].

IV. Extending CO-FSC To the General Case

This section studies a more general case, called partial flow statistics collection. We present the definition of the cost-optimized partial flow statistics collection (CO-PFSC) problem (*Section IV-A*), design an approximation algorithm using the rounding method for this problem (*Section IV-B*), and give performance analysis (*Section IV-C*).

A. Cost-Optimized Partial Flow Statistics Collection

In an SDN, the controller can know the traffic of each flow through direct measurement, *e.g.*, by collecting statistics of all flows in a network. Another way for traffic estimation is to combine the direct measurement (*e.g.*, statistics collection) and inference [21]. The controller can infer the traffic of all flows through statistics information of *partial flows (not all flows)* and link load in a network. Thus, statistics collection of partial flows also benefits for building a global traffic view, with less cost on switches compared with statistics collection of all flows. In this section, we present the partial FSC problem, in which the flow recall ratio is at least a given value $\beta \in (0, 1]$.

The CO-PFSC(β) problem is defined as follows. Similar to CO-FSC, let Γ, V, and $\mathcal{R}$ denote a flow set, a switch set and a wildcard set in an SDN, respectively. When we apply the wildcard $r_j \in \mathcal{R}$ on switch v_i, the controller can obtain the traffic statistics of a flow set Γ_i^j, and its cost is $c(\Gamma_i^j)$, which is defined in Section II-C. The controller will send Read-State commands with wildcards to different switches, so that at least $\beta \cdot m$ flows will be covered, where β is the flow recall ratio requirement, and m is the number of flows in an SDN. The cost on each switch v_i is denoted by $c(v_i)$, and we aim to minimize the maximum cost of all the switches. We give the formulation of the CO-PFSC(β) problem as follows.

$$\min \quad \lambda$$
$$S.t. \begin{cases} z_k \leq \sum_{\gamma_k \in \Gamma_i^j} x_i^j, & \forall \gamma_k \in \Gamma \\ \sum_{\gamma_k \in \Gamma} z_k \geq \beta \cdot m, & \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j) \leq \lambda, & \forall v_i \in V \\ x_i^j \in \{0,1\}, & \forall r_j \in \mathcal{R} \\ z_k \in \{0,1\}, & \forall \gamma_k \in \Gamma \end{cases} \tag{7}$$

where z_k denotes whether the statistics information of flow γ_k is collected ($z_k = 1$) or not, m is the number of flows in a network. The first set of inequalities means that one flow will be covered, if at least one set Γ_i^j containing this flow is collected by the controller. The second set of constraints means that the number of covered flows exceeds $\beta \cdot m$. The third set of inequalities means that the cost on each switch should not exceed λ. The objective is to minimize the maximum cost on all the switches, that is, $\min \ \lambda$.

Note that CO-FSC is a special case of the CO-PFSC(β) problem, with $\beta = 1$. Thus,

Theorem 4: The CO-PFSC(β) problem is NP-hard.

B. An Approximation Algorithm for CO-PFSC(β)

We describe a rounding-based algorithm, called R-PFSC, for partial flow statistics collection. Due to NP-Hardness, the algorithm constructs a linear program as a relaxation of CO-PFSC(β). We formulate the following linear program LP_2.

$$\min \quad \lambda$$
$$S.t. \begin{cases} z_k \leq \sum_{\gamma_k \in \Gamma_i^j} x_i^j, & \forall \gamma_k \in \Gamma \\ \sum_{\gamma_k \in \Gamma} z_k \geq \beta \cdot m, & \\ \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j) \leq \lambda, & \forall v_i \in V \\ x_i^j \geq 0, & \forall v_i \in V, \forall r_j \in \mathcal{R} \\ 0 \leq z_k \leq 1, & \forall \gamma_k \in \Gamma \end{cases} \tag{8}$$

By solving the linear program LP_2, we assume that the optimal solution for LP_2 is denoted by $\widetilde{x}$, and the optimal result is denoted by $\overline{\lambda}$. As LP_2 is a relaxation of the CO-PFSC(β) problem, $\overline{\lambda}$ is a lower-bound result for CO-PFSC(β).

In the second step, the controller will determine which wildcard rules will be sent to each switch for partial flow statistics collection. By solving LP_2, variable $\widetilde{z}_k$ denotes the probability that the statistics information of flow γ_k will be collected. Then, we adapt the second step of R-FSC to determine the flow statistics collection in our algorithm. The set of covered flows is denoted by Γ^c, which is initialized as Φ. We choose an uncovered flow, denoted by γ_k, with maximum $\widetilde{z}_k$. If $\widetilde{z}_k$ is less than β, the algorithm terminates. Otherwise, the algorithm chooses a flow set Γ_i^j, whose $\widetilde{x}_i^j$ is maximum among all these sets containing flow γ_k, and sets $\widehat{x}_i^j = 1$. This means that the controller will send the Read-State command with wildcard r_j to switch v_i. The algorithm will terminate until there have $\beta \cdot m$ covered flows in set Γ^c. The R-PFSC algorithm is described in Alg. 3.

Algorithm 3 R-PFSC: Rounding-based Partial FSC

```
1: Step 1: Solving the Relaxed CO-PFSC Problem
2: Construct a linear program in Eq. (8)
3: Obtain the optimal solution $\widetilde{x}$ and $\widetilde{z}$
4: Step 2: Determining Flow Statistics Collection
5: $\Gamma^c = \Phi$; $z = 1$
6: while $|\Gamma^c| < \beta \cdot m$ and $z \geq \beta$ do
7:     Choose an uncovered flow $\gamma_k$, with maximum $\widetilde{z}$
8:     $z = \widetilde{z}_k$
9:     if $\widetilde{z}_k \geq \beta$ then
10:        Choose a set $\Gamma_i^j$, whose $\widetilde{x}_i^j$ is maximum among all these sets containing the flow $\gamma_k$, and set $\widehat{x}_i^j = 1$
11:        $\Gamma^c = \Gamma^c + \Gamma_i^j$
```

C. Performance Analysis

We first give a famous lemma for performance analysis.

Lemma 5 (Chernoff Bound): Given n independent variables: $x_1, x_2, ..., x_n$, where $\forall x_i \in [0, 1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} x_i]$. Then, $\textbf{Pr}\left[\sum\limits_{i=1}^{n} x_i \leq (1-\epsilon)\mu\right] \leq e^{\frac{-\epsilon^2\mu}{2}}$, where ϵ is an arbitrarily positive value.

Number of Covered Flows Constraint. The R-PFSC algorithm may not fully guarantee that the number of covered flows exceeds $\beta \cdot m$. Under this case, one flow γ_k, with $\widetilde{z}_k \geq \beta$, should be covered. By the second constraint in Eq. (8), $\sum_{\gamma_k \in \Gamma} \widetilde{z}_k \geq \beta \cdot m$, so we expect that more flows (*e.g.*, $\beta \cdot m$) will be collected. We will further observe the ratio of covered flows through extensive simulations in Section V-B. In the following, we analyze the expected number of covered flows for a special case, where z_k is a random variable with uniform distribution from 0 to 1. Let φ denote the event of solving Eq. (8). Variable $\widehat{z}_k$ denotes whether flow γ_k is covered or not. Each flow γ_k will be covered with the probability of $\theta_k = Pr(z_k \geq \beta|\varphi)$. We compute $\mathbb{E}(z_k|\varphi)$ as follows:

$$\begin{aligned}&\mathbb{E}(z_k|\varphi)\\ =&\mathbb{E}(z_k \geq \beta) \cdot Pr(z_k \geq \beta|\varphi) + \mathbb{E}(z_k < \beta) \cdot Pr(z_k < \beta|\varphi)\\ =&\frac{1+\beta}{2} \cdot \theta_k + \frac{\beta}{2} \cdot (1-\theta_k) = \frac{\beta+\theta_k}{2}\end{aligned} \tag{9}$$

Combining the second inequality of Eq. (8), it follows $\sum_{\gamma_k \in \Gamma} \mathbb{E}(z_k|\varphi) = \sum_{\gamma_k \in \Gamma} \frac{\beta+\theta_k}{2} \geq \beta \cdot m$. Thus, $\sum_{\gamma_k \in \Gamma} \theta_k \geq \beta \cdot m$. Then, the expected number of covered flows is:

$$\begin{aligned}&\mathbb{E}[\sum\nolimits_{\gamma_k \in \Gamma} \widehat{z}_k] = \sum\nolimits_{\gamma_k \in \Gamma} \mathbb{E}[\widehat{z}_k]\\ =&\sum\nolimits_{\gamma_k \in \Gamma} Pr(z_k \geq \beta|\varphi) = \sum\nolimits_{\gamma_k \in \Gamma} \theta_k \geq \beta \cdot m\end{aligned} \tag{10}$$

According to the Chernoff bound in Lemma 5, we have

$$\textbf{Pr}\left[\sum\nolimits_{\gamma_k \in \Gamma} \widehat{z}_k \leq (1-\rho)\,\beta m\right] \leq e^{\frac{-\rho^2\beta \cdot m}{2}} \tag{11}$$

where ρ is an arbitrarily constant with $0 \leq \rho < 1$. We make the following assumption:

$$\textbf{Pr}\left[\sum\nolimits_{\gamma_k \in \Gamma} \widehat{z}_k \leq (1-\rho)\,\beta m\right] \leq e^{\frac{-\rho^2\beta \cdot m}{2}} \leq \frac{1}{n^2} \tag{12}$$

As a result, we obtain

$$\rho \geq 2\sqrt{\log n/\beta m} \tag{13}$$

We have the following lemma

Theorem 6: The R-PFSC algorithm will cover $(1-\rho)\beta m$ flows at least for statistics collection, with $\rho \geq 2\sqrt{\log n/\beta m}$ under some special situations.

By our analysis, the number of covered flows will hardly be violated by a factor of $1 - 2\sqrt{\log n/\beta m}$. For example, let n, m and β be 10^3, 10^5 and 0.5, respectively. Obviously, $\log n = 10$, and we set $\rho = 0.09$. In other words, our R-PFSC algorithm will collect statistics information of at least $0.91 \cdot \beta m$ flows.

Bandwidth/Delay Cost Performance. After the first step of the R-PFSC algorithm, we derive a fractional solution $\widetilde{x}$ and an optimal result $\widetilde{\lambda}$ for the relaxed CO-PFSC(β) problem. In each iteration of the second step, assume that the selected uncovered flow is denoted by γ_k. It follows that $\widetilde{z}_k \geq \beta$. If one flow set Γ_i^j is chosen, we know that $\widetilde{x}_i^j \geq \frac{\beta}{f}$. The cost of switch v_i is:

$$\begin{aligned}c(v_i) &= \sum\nolimits_{r_j \in \mathcal{R}} \widehat{x}_i^j \cdot c(\Gamma_i^j)\\ &\leq \sum\nolimits_{r_j \in \mathcal{R}} \frac{f}{\beta} \cdot \widetilde{x}_i^j \cdot c(\Gamma_i^j) \leq \frac{f}{\beta} \cdot \widetilde{\lambda}\end{aligned} \tag{14}$$

Thus, it follows

Theorem 7: The cost on each switch will not exceed $\frac{f}{\beta}$ times of the fractional solution by the R-PFSC algorithm.

V. Performance Evaluation

This section first introduces the metrics for performance comparison (*Section V-A*). Then, we evaluate our proposed algorithms by comparing with the previous methods through extensive simulations (*Section V-B*).

A. Performance Metrics and Simulation Setting

This paper mainly cares for bandwidth/delay cost of FSC for both CO-FSC and CO-PFSC problems. We use the following performance metrics in our numerical evaluation.

1) The maximum bandwidth cost at any time during a run of simulation. As described in Section II-C, for a flow set Γ_i^j, its bandwidth cost is defined $c(\Gamma_i^j) = 96 \cdot |\Gamma_i^j| + 200$ with unit byte.
2) The maximum delay cost on any switch at any time during a run of simulation. As described in Section II-C, for a flow set Γ_i^j, its delay cost is defined $c(\Gamma_i^j) = 0.19 \cdot |\Gamma_i^j| + 1.21$ with unit ms.
3) The algorithm running time. We mainly compare the running time of R-FSC and FSC-PD, both designed for the CO-FSC problem, by changing the number of flows.
4) The ratio of covered flows. Given a recall ratio β, the R-PFSC algorithm may not fully guarantee that the number of covered flows exceeds $\beta \cdot m$. The ratio of covered flow is the number of covered flow by our R-PFSC algorithm divided by the number of all flows in a network.

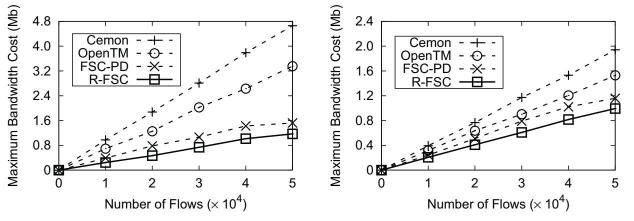

Fig. 2: Maximum Bandwidth Cost vs. Number of Flows for CO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

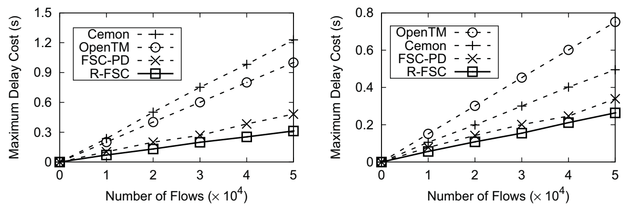

Fig. 3: Maximum Delay Cost vs. Number of Flows for CO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

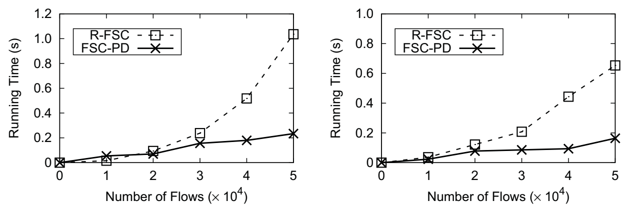

Fig. 4: Comparison of Running Time. *Left plot*: Topology (a); *right plot*: Topology (b).

5) The load balancing factor. As an application example, we regard that the (partial) flow statistics knowledge will benefit to the efficient routing. To measure the route efficiency, the load factor of a link is the traffic load divided by the link capacity. The load balancing factor is the maximum load factor among all links.

We compare the proposed FSC algorithms with the most-related, state-of-the-art work of OpenTM [17] and Cemon [15]. OpenTM and Cemon are typical studies for per-flow and per-switch statistics collection mechanisms, respectively. OpenTM tries to determine the switch using the random method for statistics collection of each flow, so that the bandwidth/delay cost can be reduced. The objective of Cemon is to reduce the total bandwidth cost on the controller in an SDN. The statistics collection of all flows is implemented by combination of per-switch and per-flow mechanisms.

As described above, there are three different schemes for flow statistics collection, per-flow, per-switch, and wildcard-based, respectively. In our SDN platform, the current version of the H3C S5120-28SC-HI switch only supports the per-flow and per-switch statistics collection, and we have implemented two methods on the platform using the RESTful APIs specified by the Opendaylight controller. For example, to obtain the statistic of a flow, we should specify the values of parameters, such as switch-id, table-id and flow-id, in the implementation. Moreover, we have tested the delay for per-flow and per-switch statistics collection on the H3C switch. Unfortunately, since our switches do not support the wildcard-based collection currently, we have not implemented our proposed algorithms on hardwares. In the future version, provided that the hardware can support the wildcard-based statistics collection, we can easily implement our proposed algorithms using RESTful API interfaces by adapting our per-flow and per-switch implementations.

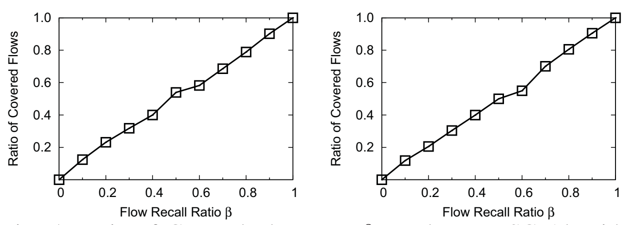

Fig. 5: Ratio of Covered Flows vs. β By the R-PFSC Algorithm. *Left plot*: Topology (a); *right plot*: Topology (b).

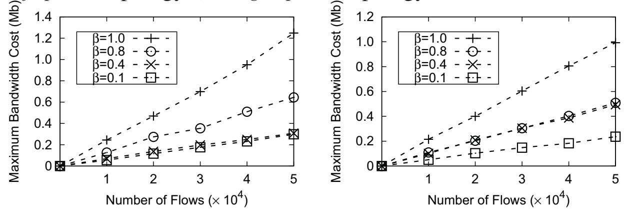

Fig. 6: Maximum Bandwidth Cost vs. Number of Flows for CO-PFSC. *Left plot*: Topology (a); *right plot*: Topology (b).

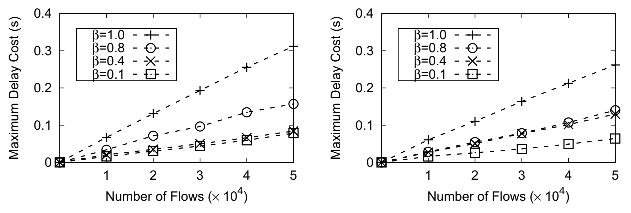

Fig. 7: Maximum Delay Cost vs. Number of Flows for CO-PFSC. *Left plot*: Topology (a); *right plot*: Topology (b).

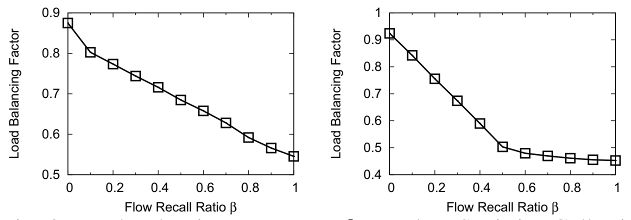

Fig. 8: Load-Balancing Factor vs. β By Flow Statistics Collection and Re-routing. *Left plot*: Topology (a); *right plot*: Topology (b).

B. Simulation Evaluation

1) Simulation Setting: In the simulations, as running examples, we select two practical and typical topologies, one for campus networks and the other for datacenter networks. The first topology, denoted by (a), contains 100 switches, 200 servers and 397 links from [22]. The second one is a fat-tree topology [23], which has been widely used in many datacenter networks. The fat-tree topology has 16 core switches, 32 aggregation switches, 32 edge switches and 192 servers. Due to capacity constraint of our simulation platform, the capacity of each link is set as 100Mbps on both topologies. We execute each simulation 100 times, and average the numerical results. For the flow size, the authors of [11] have shown that less than

20% of the top-ranked flows may be responsible for more than 80% of the total traffic. Thus, we allocate the size for each flow according to this rule.

2) Simulation Results: We run two groups of simulations to check the effectiveness of our proposed algorithms. The first set of simulations observes how the number of flows affects the statistics collection performance, including maximum bandwidth cost and maximum delay cost, of different algorithms on two topologies. From Fig. 2, our proposed algorithms, both R-FSC and FSC-PD, can significantly reduce the maximum bandwidth cost compared with both two methods, especially the per-switch method. That's because, Cemon collects the statistics information of all flows on some switches, which leads to higher bandwidth cost on control links between these switches and the controller. For example, when there are 1000 flows per server on average, and the collection period is $1s$, our FSC-PD algorithm can reduce the maximum bandwidth cost from 19Mbps by Cemon to only 6Mbps. Fig. 2 shows that FSC-PD reduces the maximum bandwidth cost by about 41% and 53% compared with the OpenTM and Cemon methods, respectively. From Fig. 3, our proposed algorithms can significantly reduce the maximum delay cost compared with both per-switch and per-flow collection methods. Note that, for the per-flow method, frequent collection requests lead to serious collection delay. Fig. 3 shows that FSC-PD can reduce the delay cost of all switches by about 52% and 45% compared with OpenTM and Cemon, respectively. From Figs. 2 and 3, we find that our rounding-based R-FSC algorithm performs better than the FSC-PD algorithm based on primal-dual. However, Fig. 4 shows that FSC-PD can save much more running time compared with R-FSC. More specifically, the running time of FSC-PD is only about $1/4$-$1/5$ as that of R-FSC.

The second set of eight simulations observes the different performance metrics of our R-PFSC algorithm by changing the parameter β on two topologies. In these simulations, there are $40K$ flows (*i.e.*, about 200 flows per server) by default in the network. From Fig. 5, we find that, for a given parameter β, the ratio of covered flows by our R-PFSC algorithm mostly exceeds β, while it is very close to β in some cases, *e.g.*, $\beta = 0.6$. This figure shows that our R-PFSC algorithm can satisfy the flow recall ratio in most situations. From Figs. 6 and 7, the maximum bandwidth/delay cost of FSC increases when the flow recall ratio β increases. That's because, with increase of flow recall ratio, statistics of more flows will be collected, which results in a higher cost, including bandwidth cost and delay cost. Both two figures show that the R-PFSC algorithm with $\beta = 0.8$ can decrease the maximum bandwidth/delay cost by about 52% compared with that with $\beta = 1.0$.

As an application example, after (partial) FSC, we can re-route flows using the routing method, as described in Section III-D, for better network performance. There are $40K$ flows in the simulation. We observe the route performance by changing the parameter β. Fig. 8 shows that the load-balancing factor will be reduced with a larger flow recall ratio β (or with statistics knowledge of more flows). For example, R-PFSC at $\beta = 0.8$ can reduce the cost 52% compared with that at $\beta = 1.0$, with increased load-balancing factor only about 5%.

VI. Related Works

Recently, SDN [1] has become an emerging technology for future networks. Most previous works, *e.g.*, [2] [24], assume that the controller knows traffic intensity of each flow to provide efficient route selection in a network. However, the flow traffic intensity is often unknown in advance in many applications, and dynamically changed during flow forwarding.

A related problem with our statistics collection is the flow traffic measurement, and the comprehensive survey can be found in [25]. The previous traffic measurement solutions are mostly implemented through the sampling technique. OpenSample [8] leveraged sFlow packets [26] to provide near-real-time measurements of both network load and individual flows. Yu *et al.* [7] used a sketch-based measurement library to automatically configure and manage resources for measurement activities. The similar sketch-based traffic monitoring method was also studied in [27]. The authors of [10] allocated resources for sketch-based measurement tasks to ensure a user-specified minimum accuracy. Some works [21] studied the rule placement and traffic measurement for an SDN. All the above methods often estimate the flow size with less overhead, which is different from our statistics collection. Note that, our FSC solutions can be combined with traffic measurement methods for different applications.

In a general SDN, each switch counts the traffic of each flow through the counter field in the flow entry. OpenFlow [6] specified two different approaches, push-based and pull-based, for flow statistics collection.

The first one is the push-based collection. FlowSense [28] utilized the PacketIn and FlowRemoved messages, which were sent by switches to the controller when a new flow come in or upon the expiration of a flow entry. Devoflow [11] extended OpenFlow with a new push-based statistics collection mechanism for identifying the elephant flows and re-routing them. However, the push-based mechanism required additional hardware support on switches, or some modification on the packet head (such as sFlow [26]). These requirements might not be fully supported by most commodity switches, which limited the application of the push-based mechanism.

The second one is the pull-based collection, which is simple and has been widely used in many SDN applications. OpenTM [17] was designed for traffic matrix estimation using simple logic for querying flow table counters. The logic was based on keeping statistics for each active flow in the network. The information about active flows was pulled from the switches periodically. OpenNetMon [14] presented an approach and open source software implementation to monitor end-to-end QoS metrics of per-flow, especially throughput, delay and packet loss, in OpenFlow networks. The authors used an adaptive fetching mechanism to pull data from switches where the rate of the queries increased when flow rates differ between samples and decreased when flows stabilized. PayLess [29] focused on the tradeoff between accuracy and network overhead. It provided a flexible RESTful API for flow statistics collection at different aggregation levels. The key advantage of PayLess was that it used an adaptive statistics collection algorithm to attain accurate information in real-time without incurring significant network overhead. The most related works with ours were FlowCover [13] and CeMon [15], which proposed a low-cost per-switch monitoring scheme to support various network management tasks. As a collection event was triggered, the controller collected the statistics information of all the flows in a network. Some applications, *e.g.*, flow re-routing [11], require that the pull-based statistics should be collected frequently enough, which may result in more serious per-switch cost, preventing from packet forwarding on switches.

VII. Conclusion

In this paper, we have studied the efficient FSC mechanisms to reduce the bandwidth cost and processing delay in an SDN. We have proposed to use wildcard-based FSC to avoid the disadvantages of both per-flow and per-switch FSC, and presented several approximation algorithms for both FSC and partial FSC problems. The extensive simulation results show high efficiency of our proposed algorithms. Moreover, partial FSC can significantly reduce the bandwidth/delay cost compared with FSC. Since the delay on the switch may depend on its traffic load, in the future, we will study more practical delay model for flow statistics collection.

Acknowledgement

This search of Xu, Yu and Liu is supported by NSFC with No. U1301256 and 61472383, NSF of Anhui in China under No. 1408085MKL08, and NSF of Jiangsu in China under No. BK20161257. The research of Qian is supported by UC Santa Cruz Startup Grant and NSF grant CNS-1464335. The research of Li is partially supported by NSFC with No. 61520106007, China National Funds for Distinguished Young Scientists with No. 61625205, Key Research Program of Frontier Sciences, CAS, No. QYZDY-SSW-JSC002, NSF ECCS-1247944, NSF CMMI 1436786, and NSF CNS 1526638.

References

[1] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “Nox: towards an operating system for networks,” *ACM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, “Achieving high utilization with software-driven wan,” in *ACM SIGCOMM*, 2013, pp. 15–26.

[3] H. Xu, X. Li, L. Huang, J. Wang, and B. Leng, “High-throughput anycast routing and congestion-free reconfiguration for sdns,” in *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*. IEEE, 2016, pp. 1–6.

[4] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, “Ddos attack protection in the era of cloud computing and software-defined networking,” *Computer Networks*, vol. 81, pp. 308–319, 2015.

[5] Q. Yan, F. R. Yu, Q. Gong, and J. Li, “Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[6] B. Pfaff *et al.*, “Openflow switch specification v1.3.0,” 2012.

[7] M. Yu, L. Jose, and R. Miao, “Software defined traffic measurement with opensketch,” in *the 10th USENIX Symposium on Networked Systems Design and Implementation*, 2013, pp. 29–42.

[8] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, “Opensample: A low-latency, sampling-based measurement platform for commodity sdn,” in *Distributed Computing Systems (ICDCS), 34th International Conference on*. IEEE, 2014, pp. 228–237.

[9] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff, “Sampling and large flow detection in sdn,” in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 345–346.

[10] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, “Scream: Sketch resource allocation for software-defined measurement,” *CoNEXT, Heidelberg, Germany*, 2015.

[11] 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 Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[12] M. Aslan and A. Matrawy, “On the impact of network state collection on the performance of sdn applications,” *IEEE Communications Letters*, vol. 20, no. 1, pp. 5–8, 2016.

[13] Z. Su, T. Wang, Y. Xia, and M. Hamdi, “Flowcover: Low-cost flow monitoring scheme in software defined networks,” in *Global Communications Conference (GLOBECOM)*. IEEE, 2014, pp. 1956–1961.

[14] N. L. van Adrichem, C. Doerr, and F. A. Kuipers, “Opennetmon: Network monitoring in openflow software-defined networks,” in *Network Operations and Management Symposium*. IEEE, 2014, pp. 1–8.

[15] S. Zhiyang, T. Wang, Y. Xia, and M. Hamdi, “Cemon: A cost-effective flow monitoring system in software defined networks,” *Computer Networks*, vol. 92, pp. 101–115, 2015.

[16] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks.” in *NSDI*, vol. 10, 2010, pp. 19–19.

[17] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, “Opentm: traffic matrix estimator for openflow networks,” in *International Conference on Passive and Active Network Measurement*. Springer, 2010, pp. 201–210.

[18] T. Wang, F. Liu, J. Guo, and H. Xu, “Dynamic sdn controller assignment in data center networks: Stable matching with transfers,” in *Proc. of INFOCOM*, 2016.

[19] J. K. Lenstra, D. B. Shmoys, and É. Tardos, “Approximation algorithms for scheduling unrelated parallel machines,” *Mathematical programming*, vol. 46, no. 1-3, pp. 259–271, 1990.

[20] H. Xu, Z. Yu, X.-Y. Li, C. Qian, and L. Huang, “Real-time update with joint optimization of route selection and update scheduling for sdns,” in *IEEE ICNP 2016*, 2016.

[21] Z. Hu and J. Luo, “Cracking network monitoring in dcns with sdn,” in *Proc. IEEE INFOCOM*, 2015.

[22] “The network topology from the monash university,” http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies.

[23] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[24] S. Agarwal, M. Kodialam, and T. Lakshman, “Traffic engineering in software defined networks,” in *IEEE INFOCOM*, 2013, pp. 2211–2219.

[25] A. Yassine, H. Rahimi, and S. Shirmohammadi, “Software defined network traffic measurement: Current trends and challenges,” *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, 2015.

[26] P. Phaal and M. Lavine, “sflow version 5,” *URL: http://www. sflow. org/sflow_version_5. txt, J uli*, 2004.

[27] T. Wellem, Y.-K. Lai, and W.-Y. Chung, “A software defined sketch system for traffic monitoring,” in *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*. IEEE Computer Society, 2015, pp. 197–198.

[28] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, “Flowsense: Monitoring network utilization with zero measurement cost,” in *Passive and Active Measurement*. Springer, 2013, pp. 31–41.

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