Minimizing Flow Statistics Collection Cost Using Wildcard-Based Requests in SDNs

Hongli Xu, *Member, IEEE*, Zhuolong Yu, Chen Qian, *Member, IEEE, ACM*, Xiang-Yang Li, *Fellow, IEEE*, Zichun Liu, and Liusheng Huang, *Member, IEEE*

***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, QoS 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 and a cost-optimized partial flow statistics collection (CO-PFSC) scheme using wildcard-based requests, and prove that both the CO-FSC and CO-PFSC problems are NP-hard. For CO-FSC, we present a rounding-based algorithm with an approximation factor f, where f is the maximum number of switches visited by each flow. For CO-PFSC, we present an approximation algorithm based on randomized rounding for collecting statistics information of a part of flows in a network. Some practical issues are discussed to enhance our algorithms, for example, the applicability of our algorithms. Moreover, we extend CO-FSC to achieve the control link cost optimization FSC problem, and also design an algorithm with an approximation factor f for this problem. We implement our designed flow statistics collection algorithms on the open virtual switch-based SDN platform. The testing and extensive simulation results show that the proposed algorithms can reduce the bandwidth overhead by over 39% and switch processing delay by over 45% compared with the existing solutions.**

***Index Terms*— Flow statistics collection, cost, delay, wildcard, rounding.**

Manuscript received March 19, 2017; revised June 5, 2017 and August 5, 2017; accepted August 29, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Uhlig. Date of publication September 21, 2017; date of current version December 15, 2017. This work was supported in part by the NSFC under Grant 61472383, Grant U1301256, and Grant 61472385, and in part by the Natural Science Foundation of Jiangsu Province in China under Grant BK20161257. The work of C. Qian was supported by the NSF under Grant CNS-1701681. The work of X.-Y. Li was supported in part by the NSF under Grant ECCS-1247944, Grant CMMI 1436786, and Grant CNS 1526638, and in part by the National Science Foundation of China under Grant 61520106007. Some preliminary results of this paper were published in the Proceedings of IEEE INFOCOM 2017 [1]. *(Corresponding author: Hongli Xu.)*

H. Xu, Z. Yu, Z. Liu, and L. Huang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China (e-mail: xuhongli@ustc.edu.cn; yzl123@mail.ustc.edu.cn; lzc223@mail.ustc.edu.cn; lshuang@ustc.edu.cn).

C. Qian is with the Department of Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: cqian12@ucsc.edu).

X.-Y. Li is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: xiangyang.li@gmail.com).

Digital Object Identifier 10.1109/TNET.2017.2748588

I. INTRODUCTION

A TYPICAL SDN consists of a logical controller in the control plane and a set of switches in the data plane [2]. 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 route/management control for each flow, an SDN can help to implement fine-grained management and improve the network resource utilization compared with traditional networks [3].

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, network attack detection, and network management. For example, if the controller performs flow routing without accurate flow traffic knowledge, it will often result in lower throughput or load imbalance. With the help of an accurate global view of flow traffic, it can improve the route QoS [3]–[5], such as maximum throughput, low latency, and high reliability. As another application example, some security attacks, *e.g.*, DDoS [6], [7], 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 [8] specifies two different approaches for flow statistics collection (FSC) from the switches.[1] One is the *push-based* mechanism. The controller learns active flows and their statistics by passively receiving reports from the switches. However, several factors limit its application in practice. First, the existing push-based mechanism does not inform the controller about the behavior of a flow before the entry times out, as a result, push-based statistics are not currently useful for flow scheduling. Second, to achieve efficient flow scheduling, DevoFlow [13] needs some additional requirements on both hardware and software for the push-based FSC, such as counters and comparators to support report triggers. But most commodity switches do not support the statistics report triggers. Third, even though the push-based

[1]Note that FSC is different from the flow traffic measurement problem in SDNs [9]–[12], which studies how switches derive flow statistics. FSC focuses on how switches report the collected flow statistics to the controller.

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 [14] and the controller's CPU resource [15]. 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 [15]–[17]. 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 [18]. There are two main schemes of pull-based FSC, *per-flow collection* [16], [19] and *per-switch collection* [15], [17]. 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 [20].

For per-flow collection, the controller sends a request to a switch for collecting the traffic statistic of exactly one flow. In some situations, the wildcard flow entries may be installed, which will be discussed in Section III-C. 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 [13], 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 may lead to unacceptable collection 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 [13]. Ordinary packet forwarding experiences significantly lower throughput during this period. Moreover, one second is still too long for many flow schedulers such as Hedera [18] to conduct accurate routing optimization. For the cheap and less capable network equipment, 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 [8]. Then the statistics for 16K exact-matched rules on a 5406zl switch would cost 1.54MB. 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 that on each switch only the traffic statistics of a subset of flows (not all flows in the per-switch collection) will be collected. We implement the fast and selective FSC on a switch using the wildcard-based FSC requests, which can be successfully implemented using OpenFlow 1.3. The controller will distribute 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) and cost-optimized partial flow statistics collection (CO-PFSC) problems, and prove the NP-hardness.
2) For CO-FSC, we present a rounding-based algorithm, called R-FSC. The R-FSC algorithm 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.
3) For CO-PFSC, we design a rounding-based algorithm for this problem, and the approximation factor of the proposed algorithm is also analyzed.
4) To study the control link cost for flow statistics collection, we extend the above switch cost optimization problems, and define the link cost-optimized flow statistics collection (LCO-FSC) problem, and present two approximation algorithms for the LCO-FSC problem.
5) We implement the proposed flow statistics collection methods on an SDN platform using Open vSwitch [21]. The testing and simulation results show that our algorithms help to reduce the bandwidth overhead by over 39% 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.

II. Preliminaries

In this section, we first introduce the network and flow models in an SDN. Then, we define the cost-optimized flow statistics collection (CO-FSC) and cost-optimized partial flow statistics collection (CO-PFSC) problems, respectively.

A. Network and Flow Models

An SDN typically consists of a logically-centralized controller and a set of switches, $V = \{v_1, \ldots, v_n\}$, with $n = |V|$.

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

Fig. 1. Illustration of a standard flow entry in a flow table.

These switches comprise the data plane of an SDN. Thus, the network topology from a view of the date 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 [22], [23], which help to balance the overhead among these controllers. Since the metric we first evaluate is per-switch bandwidt/delay cost, we assume that there is only one controller for simplicity of presentation. It should be noted that we will extend the per-switch cost optimization to the link cost optimization in Section IV.

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 rules). A standard flow entry, specified by OpenFlow 1.3 [8], 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 lease 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 requests, it is feasible to collect statistics of a subset of flows from a switch. Thus, the wildcard solution is able to distribute the statistics collection overhead of all flows among all switches, which helps to reduce the switch cost compared with the per-switch method. *Second*, 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.

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 $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$, 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. For simplicity, we assume that each switch is directly connected to a controller. We will extend this assumption to a more general scenario, in which some switches may not be directly connected with the controller, in Section IV.

Assume that there is a set of wildcards, denoted by $\mathcal{R} = \{r_1, r_2, \ldots r_q\}$, 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, *i.e.*, $\Gamma_i^{j_1} \cap \Gamma_i^{j_2} = \Phi,\ j_1 \neq j_2, \forall v_i \in V$.

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 [8], the bandwidth cost for statistics collection of a flow set Γ_i^j consists of two parts: (1) the request packet, whose length is 114 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 packet header, and l_f is the length for each flow entry, respectively. According to [8], l_h and l_f are 74 bytes and 96 bytes, respectively. More specifically, the length of packet header includes 16 bytes (multipart request header) and 58 bytes (Ethernet+IP+TCP headers). The length for each flow entry includes 40 bytes (match fields for each entry) and 56 bytes (statistics information for each entry). These parameters have been validated through our open virtual switch (OVS) platform. Thus, the bandwidth cost is modeled as $c(\Gamma_i^j) = 96 \cdot |\Gamma_i^j| + 188$ with unit byte.
- We consider the cost $c(\Gamma_i^j)$ as the delay for statistics collection of a flow set Γ_i^j. By testing on the HP switch [13], 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. It should be noted that, the values of two constant parameters for delay cost may vary with switch traffic load [13], which will be discussed in Section III-C.

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. The controller may send several collection 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, $\min \max\{c(v_i), 1 \le i \le n\}$, so that no performance bottleneck will happen in SDNs.

Accordingly we formalize the CO-FSC problem as follows:

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

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 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.

The proof of theorem 1 has been relegated to the appendix VII.

D. Cost-Optimized Partial Flow Statistics Collection

The CO-FSC problem will collect statistics information of all flows in a network. In fact, the statistics information of partial flows is also helpful for some applications. For example, 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 [24]. 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 sends 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 CO-PFSC(β) as follows.

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

where z_k denotes whether the statistics information of flow γ_k is collected ($z_k = 1$) or not. 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 $\beta = 1$. By theorem 1, it follows

Theorem 2: The CO-PFSC(β) problem is NP-hard.

III. Algorithms for CO-FSC and CO-PFSC

A. Algorithm Design for 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.1*), and give performance analysis (*Section III-A.2*). Then, we present an algorithm based on primal-dual with lower time complexity (*Section III-A.3*).

1) A Rounding-Based Algorithm for CO-FSC: This section develops a rounding-based algorithm, called R-FSC, to solve the CO-FSC problem. The 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. (3).

$$\min\ \lambda \quad S.t. \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} x_i^j \ge 1, & \forall \gamma_k \in \Gamma \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j) \le \lambda, & \forall v_i \in V \\ x_i^j \ge 0, & \forall r_j \in \mathcal{R} \end{cases} \quad (3)$$

We can solve Eq. (3) in polynomial time, and obtain the fractional solution, denoted by $\widetilde{x}$. In the second step, the fractional solution $\{\widetilde{x}\}$ will be rounded to 0-1 solution $\{\widehat{x}\}$ 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.

2) 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, $|\Gamma^\gamma| \le f$, where f is the maximum number of switches visited by each flow. Since each flow γ will be covered, $\sum_{\Gamma_i^j \in \Gamma^\gamma} \widetilde{x}_i^j \ge 1$. Assume that a flow set Γ_i^j is chosen in some iteration, for $\widetilde{x}_i^j$ is maximum among all these sets containing flow γ. It follows $\widetilde{x}_i^j \ge \frac{1}{f}$ or $f \cdot \widetilde{x}_i^j \ge 1$.

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

Algorithm 1 R-FSC: Rounding-Based FSC

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

to the algorithm description, the final cost of switch v_i is:

$$c(v_i) = \sum\nolimits_{r_j \in \mathcal{R}} \widehat{x}_i^j \cdot c(\Gamma_i^j) \leq \sum\nolimits_{r_j \in \mathcal{R}} f \cdot \widetilde{x}_i^j \cdot c(\Gamma_i^j) \leq f \cdot \widetilde{\lambda} \tag{4}$$

Thus, we conclude that

Theorem 3: The R-FSC algorithm can achieve the f-approximation for the CO-FSC problem.

3) A Lower-Complexity Algorithm Using Primal-Dual: When a flow statistics collection event is triggered, we expect that the controller can immediately determine the solution for FSC. it is an important step to solve the linear program Eq. (3) in the R-FSC algorithm. As the number of variables in Eq. (3) 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 presents a lower-complexity algorithm using primal-dual for CO-FSC. The primal-dual version of Eq. (3) is given in Eq (5), in which two sets of variables χ and μ denote the first and second sets of constraints in Eq. (3).

$$\max \ \sum\nolimits_{k=1}^{m} \chi_k$$
$$S.t. \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} \chi_k - \mu_i \cdot c(\Gamma_i^j) \leq 0, & \forall i, j \\ \sum_{i=1}^{n} \mu_i \leq 1, & \\ 0 \leq \mu_i \leq 1, & \forall i \end{cases} \tag{5}$$

We design the FSC-PD algorithm for the CO-FSC problem. According to Eq. (5), we should increase the values of variables χ with constraints so as to maximize the objective function. For each flow γ_k, there is a corresponding variable χ_k. At the beginning, the algorithm initializes a variable, Γ^u, which denotes the uncovered flow set. For the ease of algorithm design, we expect that the variable χ_k for each uncovered flow will almost grow in line to a value, denoted by δ, which will be updated in algorithm execution. By the first set of inequalities of Eq. (5), we have

$$\sum\nolimits_{\gamma_k \in \Gamma_i} \chi_k = \sum\nolimits_{r_j \in \mathcal{R}} \sum\nolimits_{\gamma_k \in \Gamma_i^j} \chi_k \leq \sum\nolimits_{r_j \in \mathcal{R}} \mu_i \cdot c(\Gamma_i^j) \tag{6}$$

For simplicity, let $tc(v_i)$ be $\sum_{r_j \in \mathcal{R}} c(\Gamma_i^j)$. As we consider the critical case, it follows that $\delta \approx \frac{\mu_i \cdot tc(v_i)}{n_i}$, where n_i is the number of flows through switch v_i. Combining with the second inequality of Eq. (5), we set each variable μ_i as

$$\mu_i = \frac{\frac{n_i}{tc(v_i)}}{\sum_{v_i \in V} \frac{n_i}{tc(v_i)}} \tag{7}$$

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 a value, denoted by δ_i^j, so that $\sum_{\gamma_k \in \Gamma_i^j} \chi_k = \mu_i \cdot c(\Gamma_i^j)$. Under this situation, the variable $\chi_{k'}$ for the covered flow $\gamma_{k'}$ remains unchanged. We then choose a flow set Γ_i^j with the minimum value δ_i^j among all the uncovered flow sets. According to the Primal-Dual's property, the corresponding variable x_i^j in Eq. (1) is set to 1, 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 uncovered 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.

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

$\Gamma^u = \Gamma$
for each switch $v_i \in V$ **do**
  Compute μ_i by Eq. (7)
while $|\Gamma^u| > 0$ **do**
  for each uncovered flow set Γ_i^j **do**
    Increase χ_k for each uncovered flow γ_k to value δ_i^j, so that $\sum_{\gamma_k \in \Gamma_i^j} \chi_k = \mu_i \cdot c(\Gamma_i^j)$
  Choose an uncovered flow set Γ_i^j with the minimum value δ_i^j among all the uncovered flow sets
  for each uncovered flow $\gamma_k \in \Gamma_i^j - \Gamma^u$ **do**
    $\chi_k = \delta_i^j$
  $\Gamma^u = \Gamma^u - \Gamma_i^j$

Theorem 4: 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, respectively.

Proof: The FSC-PD 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 $O(mnqf)$. □

B. Algorithm Design for CO-PFSC

This section studies a more general case, called partial flow statistics collection. We design an approximation algorithm using the rounding method for the CO-PFSC problem (*Section III-B.1*), and give approximation performance analysis (*Section III-B.2*).

1) 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.

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

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 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 $z \ge \beta$ **do**
7: Choose an uncovered flow γ_k, with maximum $\widetilde{z}$
8: $z = \widetilde{z}_k$
9: **if** $\widetilde{z}_k \ge \beta$ **then**
10: Choose a set Γ_i^j, whose $\widetilde{x}_i^j$ is maximum among all these sets containing the flow γ_k, and set $\widehat{x}_i^j = 1$
11: $\Gamma^c = \Gamma^c + \Gamma_i^j$

2) Performance Analysis: We first give a famous lemma for performance analysis.

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

Number of Covered Flows Constraints: 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 $\widetilde{z}_k \ge \beta$, should be covered. By the second constraint in Eq. (8), $\sum_{\gamma_k \in \Gamma} \widetilde{z}_k \ge \beta \cdot m$, 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-C. 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 $\theta_k = Pr(z_k \ge \beta|\varphi)$. We compute $\mathbb{E}(z_k|\varphi)$ as follows:

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

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

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

According to the Chernoff bound in Lemma 5, we have

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

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

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

As a result, we obtain

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

We have the following lemma

Theorem 6: The R-PFSC algorithm will cover $(1-\rho)\beta m$ flows at least for statistics collection, with $\rho \ge 2\sqrt{\log n/\beta m}$ 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 $\overline{\lambda}$ 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 $\widetilde{z}_k \ge \beta$. If one flow set Γ_i^j is chosen, we know that $\widetilde{x}_i^j \ge \frac{\beta}{f}$. The cost of switch v_i is:

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

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.

C. Discussion

In this section, we give some discussions, including the impact of switch CPU utilization on the collection cost, the case study of the FSC, and the applicability of our algorithms.

- **The impact of switch CPU utilization**. As specified in the OpenFlow standard, each SDN switch has a software implemented OpenFlow agent (OFA) that communicates with the controller over a secure TCP connection. After an FSC request is sent from the controller to the switch, the OFA encapsulates the counter field combined with other fields (*e.g.*, match fields) of all matched flow entries into a reply packet, and sends back the reply packet to the controller. The OFA performance mainly depends on the switch CPU capacity. Due to the limited CPU capacity on the switch, the processing delay for flow statistics collection will be increased with CPU utilization [13]. That is, the delay cost of FSC depends on the current switch CPU state. To deal with this case, each switch v_i is assigned with a weight ω_i, which denotes the cost growth coefficient with CPU utilization. Let $p(v_i)$ denote the current CPU utilization. By applying the Little's law [22], the weight can be approximately set as $\omega_i = \frac{1}{1-p(v_i)}$. Different from CO-FSC, the objective of the weighted CO-FSC problem is to minimize the maximum weighted cost of all switches, that is, $\min \ \max\{\omega_i \cdot c(v_i), \forall v_i \in V\}$. Similar to Eq. (1), we can formalize this problem as follows:

$$\min \ \lambda \quad S.t. \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} x_i^j \geq 1, & \forall \gamma_k \in \Gamma \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j \omega_i \cdot c(\Gamma_i^j) \leq \lambda, & \forall v_i \in V \\ x_i^j \in \{0,1\}, & \forall r_j \in \mathcal{R} \end{cases} \tag{15}$$

  From Eq. (15), this is the weighted version of the CO-FSC problem. We can modify the R-FSC and FSC-PD algorithms a little for the weighted version. It is similar to consider the impact of the switch CPU utilization on the switch cost for the CO-PFSC problem.
- **The case study of the FSC.** Many works [13], have mentioned the fact that a small percentage of large (or elephant) flows typically accounts for a large percentage of total traffic, and plays an important role for route performance. After collecting the flow statistics information from switches, the controller determines a set of elephant flows, denoted by Γ^e, and ranks all these elephant flows in the decreasing order of traffic intensity. For each flow $\gamma \in \Gamma^e$, the controller finds the least congested path between the flow's endpoints as its new route. After determining new routes for all the elephant flows in Γ^e, we execute the route reconfiguration [20], [25], which can reduce the route reconfiguration delay compared with that for all flows in a network. Note that our proposed flow statistics collection method can also be combined with other routing methods in an SDN.
- **The applicability of our algorithms.** This paper assumes that all the flows in the network are known. We note that our proposed algorithms can also be applied in different situations. 1) For the proactive manner, the controller pre-installs rules for flows so as to improve the network scalability. One may say that though the rules for some flows are pre-installed, there may be no traffic for these flows. That is, the controller may not exactly know all the flows running through the network. To deal with this case, we try to collect the statistics information of all flows with installed rules for the following two reasons. On one hand, when the controller installs a rule with proactive manner, the controller can not judge whether this flow is running or not. To guarantee the completeness of FSC, we regard that all the flows may be running. On the other hand, as described in B4 [26], the controller will periodically update the flow tables according to the traffic amount in the last period. Thus, it is reasonable to regard that most flows with pre-installed rules are active in this period. 2) In some environments, the wildcard rules, which match IP prefixes or arbitrary header ranges, may be installed in a flow table for high scalability. When a wildcard rule is installed on a flow table, the statistics of all flows matching this wildcard rule will be aggregated, and we can not distinguish the statistics information of each individual flow. In this situation, all flows matching a wildcard rule can be regarded as a "flow", also called macroflow [27]. In fact, our flow statistics collection is designed based on installed rules in the flow tables. That is, we collect the values of the counter field in flow entries. Thus, when the wildcard rules are applied, we can not collect the statistics information of individual flows, but the aggregated flows specified by the wildcard rules.
- **The applicability for multiple controllers.** In a large-scale SDN with more switches, it is an efficient way to deploy multiple controllers for the single-controller congestion avoidance. However, the distributed processing on different controllers may lead to inconsistency of the flow statistics information [28]. Though some applications, such as traffic engineering, may not require highly consistent statistics, but most of FCAPS management applications [29] require consistency of managed information. In the future, we will study the efficient mechanism for consistency maintenance of flow statistics information. For wildcard-based flow statistics collection, when the controller sends a wildcard rule command, the switch will compare with the flow entries with this wildcard rule, encapsulate the statistics information of all matched flow entries into a reply packet, and send back to the controller. Compared with the per-flow and per-switch FSC method, the wildcard-based method needs to match all the entries in the whole flow table.

IV. Link Cost-Optimized FSC

There are two kinds of connection schemes between the controller and switches, out-band and in-band, respectively. The above sections focus on the switch cost optimization for

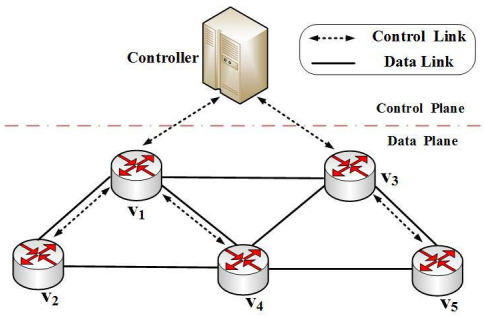


Fig. 2. Illustration of different switch-controller connection schemes. For example, switch v_1 is directly connected with the controller. But, switch v_2 is connected with the controller through a control path $v_2 - v_1 -$ controller.

flow statistics collection in an SDN, *i.e.*, for the out-band scheme. In this section, we consider the in-band connection scheme, which has also been applied in many applications. For example, to provide dynamic controller provisioning in an SDN, many works [22] adopt the in-band controller connection, in which the control messages are transferred through a data plane path. Under this connection scheme, since flow statistics information will be forwarded to the controller through some links, this section will extend the CO-FSC problem to achieve the (control) link cost optimization.

A. Link Cost-Optimized Flow Statistics Collection (LCO-FSC) Problem

In this section, we define the link cost-optimized flow statistics collection (LCO-FSC) problem, which is also the extended version of the CO-FSC problem. In an SDN, we assume that there is a cluster of controllers in the control plane. To better control/manage all the switches, each switch will connect/associate with a controller through TCP long-term connection. Physically, there are two cases for the switch-controller connection. One is that a switch is directly connected with a controller. In this case, the LCO-FSC problem is equivalent to the CO-FSC problem, which has been considered in Section III-A. The other is that the physical control connection may pass through one or several switches. As shown in Fig. 2, switch v_1 is directly connected with the controller. But, switch v_2 is connected with the controller through a control path $v_2 - v_1 -$ controller. For simplicity, we assume that each switch v has built a path $p(v)$ to the associated controller.

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 switches will report the corresponding statistics information to the controller and all the flows can be covered. As described in Section II-C, the bandwidth cost on switch v_i is denoted by $c(v_i)$. For each control link e, its traffic load is the total amount of all statistics information through this link, which is described as $l(e) = \sum_{v \in V,\ e \in p(v)} c(v_i)$. In Fig. 2, both the data traffic and control traffic will path through some links connecting switches, *e.g.*, v_1v_2 and v_1v_4. Logically, we divide such a link into two sub-links, data sub-link and control sub-link, respectively. Without confusion, these sub-links are also called as links. To better guarantee the successful forwarding of FSC and control commands, each control link will be allocated a bandwidth (also called its capacity), denoted by $c(e)$. In the scenario of flow statistics collection, we expect that *each* control link should remain some bandwidth for control commands. Otherwise, due to congestion on some control links, the control commands can not be successfully sent to some switches. Thus, our objective is to achieve the load balancing among all the control links, that is, $\min \max\{\frac{l(e)}{c(e)}, e \in E^c\}$, where E^c is a set of control links. Accordingly we formalize the LCO-FSC problem as follows:

$$\min\ \eta$$
$$S.t.\ \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} x_i^j \geq 1, & \forall \gamma_k \in \Gamma \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j), & \forall v_i \in V \\ l(e) = \sum_{v \in V,\ e \in p(v)} c(v_i) \leq \eta \cdot c(e), & \forall e \in E^c \\ x_i^j \in \{0,1\}, & \forall r_j \in \mathcal{R} \end{cases} \quad (16)$$

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 denotes the total bandwidth cost on each switch. The third set of inequalities means that the total traffic load on each control link, where η is the control link load factor. The objective is to achieve the load balancing among all the control links, that is, $\min\ \eta$.

Theorem 8: The LCO-FSC problem is NP-hard.

Proof: Consider a special case of the LCO-FSC problem, in which each switch will directly connect with a controller. This becomes the CO-FSC problem. Combining with Theorem 1, it follows that the LCO-FSC problem is NP-hard. □

B. An Approximation Algorithm for LCO-FSC

In this section, we will modify the R-FSC algorithm a little so as to solve the LCO-FSC problem. We call this algorithm as MR-FSC. The proposed MR-FSC algorithm consists of two main steps. In the first step, we relax the integer program formulation in Eq. (16) to a linear program as Eq. (17).

$$\min\ \eta$$
$$S.t.\ \begin{cases} \sum_{\gamma_k \in \Gamma_i^j} x_i^j \geq 1, & \forall \gamma_k \in \Gamma \\ c(v_i) = \sum_{r_j \in \mathcal{R}} x_i^j c(\Gamma_i^j), & \forall v_i \in V \\ l(e) = \sum_{v \in V,\ e \in p(v)} c(v_i) \leq \eta \cdot c(e), & \forall e \in E^c \\ x_i^j \geq 0, & \forall r_j \in \mathcal{R} \end{cases} \quad (17)$$

We can solve Eq. (17) in polynomial time, and obtain the fractional solution, denoted by $\widetilde{x}$. In the second step, the fractional solution will be rounded to 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 MR-FSC algorithm is described in Alg. 4.

Algorithm 4 MR-FSC: Modified Rounding-Based FSC

1: **Step 1: Solving the Relaxed LCO-FSC Problem**
2: Construct the relaxed formulation as Eq. (17)
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 γ
8: Choose a flow set Γ_i^j, whose $\widetilde{x}_i^j$ is maximum among all these sets containing flow γ, and set $\widehat{x}_i^j = 1$
9: $\Gamma^u = \Gamma^u - \Gamma_i^j$

C. 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 MR-FSC algorithm, we derive a fractional solution $\widetilde{x}$ and an optimal result $\widetilde{\eta}$ for the relaxed LCO-FSC problem. According to the algorithm description, the final cost of each control link e is:

$$\begin{aligned}\widehat{l}(e) &= \sum\nolimits_{v\in V,\ e\in p(v)} \sum\nolimits_{r_j\in\mathcal{R}} \widehat{x}_i^j c(\Gamma_i^j) \\ &\leq \sum\nolimits_{v\in V,\ e\in p(v)} \sum\nolimits_{r_j\in\mathcal{R}} f\cdot \widetilde{x}_i^j c(\Gamma_i^j) \\ &\leq f\cdot\widetilde{\eta}\cdot c(e) \end{aligned} \tag{18}$$

Thus, we can conclude that

Theorem 9: The MR-FSC algorithm can achieve the f-approximation for the LCO-FSC problem.

D. A Greedy Algorithm for LCO-FSC

To determine the FSC solution immediately, this section designs a greedy algorithm, called G-FSC, with lower time complexity for LCO-FSC. In each iteration, we randomly choose an uncovered flow, denoted by γ. Let Γ^γ denote all the sets that contain this flow. Then, we will choose a flow set from Γ^γ, so that the load factor of all control links is minimized. Then, the algorithm will update the uncovered flow set, and will be terminated until all flows are covered. The greedy algorithm is described in Alg. 5.

Theorem 10: The time complexity of G-FSC is $O(mfh)$, where m, f and h are the number of flows, the maximum number of switches visited by each flow, and the maximum hop number of all control links in a network, respectively.

Proof: Since there are m flows in a network, the algorithm consists of $O(m)$ iterations at most. In each iteration, we will randomly choose an uncovered flow. Then, we determine the switch and its wildcard with the less control traffic load factor, which takes a time complexity of $O(fh)$. Then, the total time complexity of the G-FSC algorithm is $O(mfh)$. □

Algorithm 5 G-FSC: Greedy Method for FSC

1: $\Gamma^u = \Gamma$
2: **while** $|\Gamma^u| > 0$ **do**
3: Randomly choose an uncovered flow γ
4: Determine the set Γ^γ
5: Choose a flow set from Γ^γ so that the load factor of all control links is minimized
6: Update the uncovered flow set Γ^u

We should note that our proposed G-FSC algorithm can be modified to solve the partial flow statistics collection problem. For PFSC, we just modify the iteration condition as $\Gamma^u > (1-\beta)\cdot m$ in Line 2. After the algorithm terminates, there are at least $\beta \cdot m$ covered flows in a network.

V. Performance Evaluation

This section first introduces the metrics and benchmarks for performance comparison (*Section V-A*). Then, we implement our algorithms on the SDN platform, and give the testing results (*Section V-B*). Finally, we evaluate our proposed algorithms by comparing with the previous methods through simulations (*Section V-C*). Our simulations are run on Mininet [30], which is a widely-used simulator for SDN.

A. Performance Metrics and Benchmarks

This paper mainly cares for switch (bandwidth/delay) cost and link cost of FSC for different problems, including CO-FSC, CO-PFSC and LCO-FSC. When a switch is with a heavy FSC load, it may heavily interfere with its basic functions, such as data forwarding and updates. Thus, we expect to minimize the maximum bandwidth cost on all switches. Moreover, we expect to finish the flow statistics collection in a fast manner. Thus, it is required to minimize the maximum delay on all switches. When a control link is with a heavy traffic control load, it may be congested, which may increase the delay and loss ratio of control commands. Thus, we expect to minimize the maximum bandwidth cost on all control links. We use the following performance metrics in our numerical evaluation.

1) The maximum bandwidth cost on any switch aduring a run of testing/simulation. As described in Section II-C, for a flow set Γ_i^j, its bandwidth cost is defined $c(\Gamma_i^j) = 96\cdot|\Gamma_i^j| + 188$ with unit byte.
2) The maximum delay cost on any switch during a run of simulation. 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 total number of requests per FSC. We compare the total number of requests necessary to collect statistics information of all flows.
4) 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.

5) The ratio of covered flows. Given a flow 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.
6) The control link load factor. We measure the traffic amount of flow statistics collection on each control link, and compte the load factor of all control links in a network.
7) 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 data link is the traffic load divided by the data link capacity. The load balancing factor is the maximum load factor among all data links.

We compare the proposed FSC algorithms with the most-related, state-of-the-art work of OpenTM [19] and Cemon [17] by both testing and simulations. OpenTM and Cemon are typical studies for per-flow and per-switch statistics collection mechanisms, respectively. OpenTM is a flow-based FSC method, in which the controller will collect the statistics information of each flow from a switch along the route path randomly, 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 algorithm chooses the most cost-effective switches and removes the covered flows, until all flows are covered.

B. Test-Bed Evaluation

1) Implementation on the Platform: There are two different ways for building the SDN platform. One is based on the physical switches, such as H3C S5120-28SC-HI switches. The current version of this physical 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 some 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 H3C SDN switches are developed based on the traditional switching framework, they do not support the wildcard-based collection currently. In the future version, provided that the hardware can support the wildcard-based flow statistics collection, we can easily implement our proposed algorithms using RESTful API interfaces by adapting our per-flow and per-switch implementations. The other is based on the virtual switches. Since these virtual switches are implemented by the software, they usually support all three schemes of flow statistics collection.

We implement the per-flow, per-switch and wildcard-based FSC algorithms on a real OVS-based test-bed. Our SDN platform is mainly composed of three parts: a server installed with the controller's software, a set of virtual switches, and some virtual machines (also called terminals). Specifically, we choose Ryu [31], which is an open source project, as the controller's software. The Ryu controller is running on a server with a core i5-3470 processor and 4GB of RAM. The virtual switch is implemented using the OVS 2.0.2 [21]. The topology of our SDN platform is illustrated in Fig. 3. The forwarding plane of an SDN comprises of 6 virtual switches, which support the OpenFlow v1.3 standard. In the system testing, each flow is identified by three elements, source IP, destination IP and TCP port, so that each terminal is able to generate different numbers of flows to other terminals.

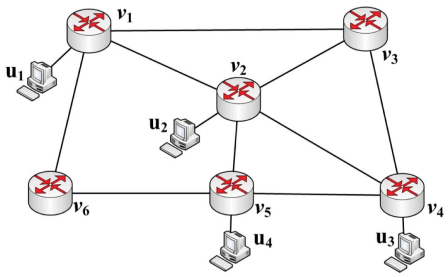


Fig. 3. Topology of the SDN platform. Our platform is composed of three parts: a controller, six OpenFlow enabled virtual switches $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ and four virtual machines $\{u_1, u_2, u_3, u_4\}$. The controller is directly connected with each switch. For simplicity, we omit the controller in this figure.

To implement the wildcard-based flow statistics collection on the virtual switch, the controller sends the request interface, which is defined as the "OFPFlowStatsRequest" class, to the target switch, and the match field in the "OFPFlowStatsRequest" class is described by the "OFPMatch" class. For example, when we try to collect the statistics information of flows whose destination is "192.168.3.5", we create a "OFPFlowStatsRequest" object, in which we set two parameters in the "OFPMatch" object as: eth_type=0x800, and ipv4_dst = ('192.168.3.5','255.255.255.255'), and send it to the target switch.

2) Testing Results: We generate 600 and 1200 flows in the network, respectively, and observe the maximum bandwidth cost of all switches by different FSC algorithms. There are four terminals and accordingly 12 terminal pairs in the system. In the testing, each terminal pair will generate the same number of flows, and the controller will choose a path randomly for each flow. Moreover, we allocate the size for each flow according to the classical 2-8 rule [13]. The left plot of Fig. 4 shows that our proposed R-FSC and FSC-PD algorithms can perform better than OpenTM and Cemon, respectively. More specifically, the maximum bandwidth cost of OpenTM is about 0.342Mb and 0.689Mb for 600 flows and 1200 flows, respectively. Given 1200 flows, the maximum bandwidth costs of the Cemon, FSC-PD and R-FSC algorithms are 0.665Mb, 0.434Mb, and 0.274Mb, respectively. In other words, our R-FSC and FSC-PD algorithms can reduce the maximum bandwidth cost about 60.2% and 37.0%, respectively. The right plot of Fig. 4 shows the maximum bandwidth cost by changing the value of flow recall ratio β. Our testing shows that, when the flow recall ratio is smaller, the controller needs to collect statistics information of a smaller number of flows, which results in a smaller bandwidth cost. For the case of 1200 flows,

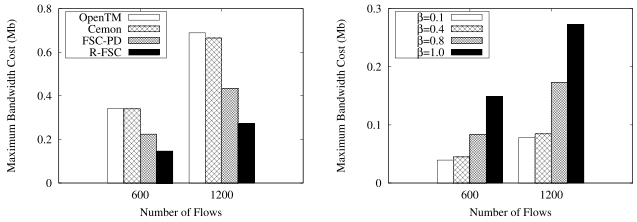


Fig. 4. Maximum bandwidth cost. *Left plot*: Different algorithms; *right plot*: Different flow recall ratios β.

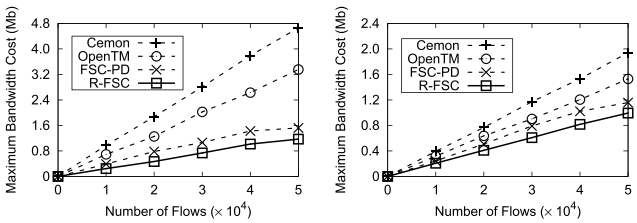


Fig. 5. Maximum bandwidth cost vs. number of flows for CO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

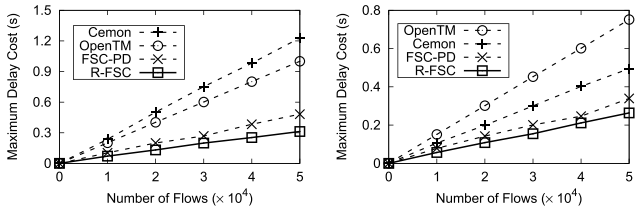


Fig. 6. Maximum delay cost vs. number of flows for CO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

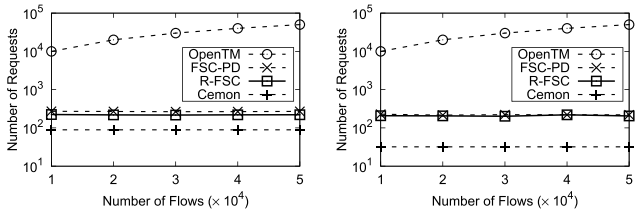


Fig. 7. Number of total requests vs. number of flows for CO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

the maximum bandwidth cost is 0.079Mb, 0.084Mb, 0.173Mb, and 0.274Mb, respectively. That is, the partial flow statistics collection mechanism (*e.g.*, $\beta = 0.4$) can reduce the maximum bandwidth cost 69.2% compared with the flow statistics collection mechanism (*i.e.*, $\beta = 1.0$). These testing results show higher efficiency of our proposed wildcard-based flow statistics collection methods compared with per-flow and per-switch FSC methods.

C. 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 [32]. The second one is a fat-tree topology [33], which has been widely used in many datacenter networks. The fat-tree topology has totally 80 switches (including 16 core switches, 32 aggregation switches, and 32 edge switches) and 192 servers. Due to capacity constraint of our simulation platform, the capacity of each physical 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 [13] 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 four groups of simulations to check the effectiveness of our proposed FSC algorithms. The first set of simulations observes how the number of flows affects the switch cost performance, including maximum bandwidth cost and maximum delay cost, of different algorithms on two topologies. From Fig. 5, 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 switch and the controller. For example, when there are 1000 flows per server on average, our FSC-PD algorithm can reduce the maximum bandwidth cost from 19Mb by Cemon to only 6Mb. Fig. 5 shows that FSC-PD reduces the maximum bandwidth cost by about 39% and 53% compared with the OpenTM and Cemon methods, respectively. From Fig. 6, our proposed algorithms can significantly reduce the maximum delay cost compared with both per-switch and per-flow collection methods. For the per-flow method, frequent collection requests lead to serious collection delay. Fig. 6 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. 5 and 6, we find that our rounding-based R-FSC algorithm performs better than the FSC-PD algorithm based on primal-dual. Fig. 7 shows the total number of requests for per FSC event. Obviously, since Cemon is a per-switch FSC method, it needs the least number of requests for FSC among four algorithms. Our proposed R-FSC and FSC-PD algorithms require almost the similar number of requests with varied number of flows, and only need $1/30$-$1/50$ FSC requests as OpenTM. Moreover, FSC-PD needs almost the same number of requests as R-FSC in topology (b) due to its structured topology. However, Fig. 8 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 $1/4$-$1/10$ as that of R-FSC, and the increasing ratio of running time of FSC-PD is less than that of number of flows in the network. Though both Cemon and OpenTM algorithms need less running time than FSC-PD, running time of all three algorithms is acceptable. These results show that our proposed FSC-PD algorithm is much scalable and fit for the large-scale networks.

The second set of six simulations observes the different performance metrics of our R-PFSC algorithm by changing the flow recall ratio β on two topologies. In these simulations, there are 40K flows (*i.e.*, about 200 flows per server) by default

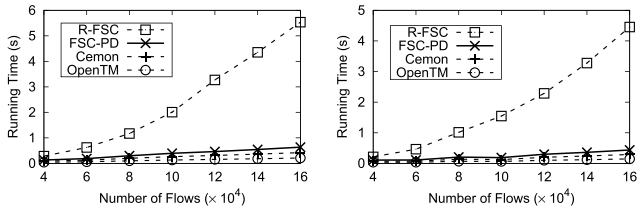


Fig. 8. Comparison of running time. *Left plot*: Topology (a); *right plot*: Topology (b).

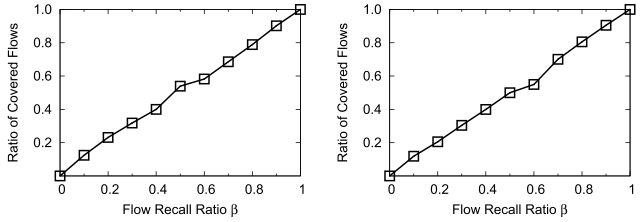


Fig. 9. Ratio of covered flows vs. β by the R-PFSC algorithm. *Left plot*: Topology (a); *right plot*: Topology (b).

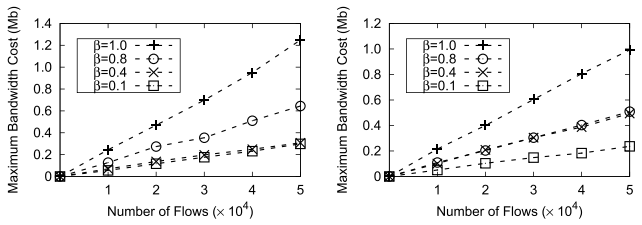


Fig. 10. Maximum bandwidth cost vs. number of flows for CO-PFSC. *Left plot*: Topology (a); *right plot*: Topology (b).

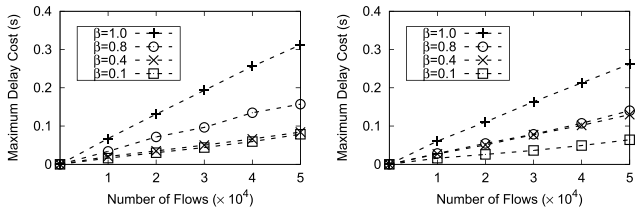


Fig. 11. Maximum delay cost vs. number of flows for CO-PFSC. *Left plot*: Topology (a); *right plot*: Topology (b).

in the network. From Fig. 9, 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. 10 and 11, 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 $\beta = 1.0$.

The third set of simulations observes how the number of flows affects the link cost performance, including control link load factor and maximum bandwidth cost on links, of different

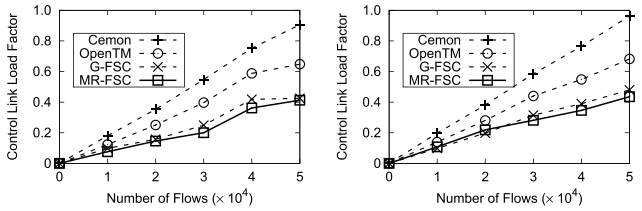


Fig. 12. Control link load factor vs. number of flows for LCO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

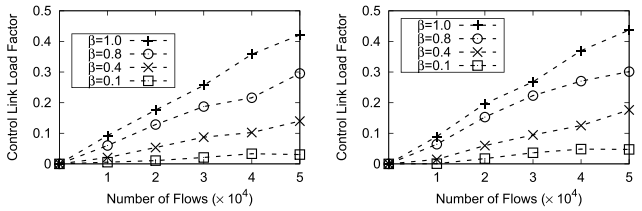


Fig. 13. Control link load factor vs. number of flows by varying β for LCO-FSC. *Left plot*: Topology (a); *right plot*: Topology (b).

algorithms. For both two topologies, we assume that a third of all switches are directly connected with the controller. Then, we will construct a width-first tree for other switches so that each switch can build connection with the controller. We set the capacity of each (logical) control link as 6Mbps for topologies (a) and (b), respectively. Specifically, according to the simulation setting, the capacity of each link connecting two switches is 100Mbps. We divide this link into two logical links. One is the data link with 94Mbps, and the other is the control link with 6Mbps. From Fig. 12, our proposed algorithms, both MR-FSC and G-FSC, can significantly reduce the control link load factor compared with both per-switch and per-flow collection methods, especially the per-switch method. That's because, Cemon collects the statistics information of all flows just from some switches, which leads to higher cost on some control links between these switch and the controller. For example, when there are totally 50K flows in a network, and the collection period is $1s$, our MR-FSC and G-FSC algorithms reduce the control link load factor from 0.79 by Cemon to only 0.22. Fig. 14 shows that G-FSC reduces the control link load factor by about 52% and 65% compared with the OpenTM and Cemon methods, respectively. From Fig. 13, the control link load factor of FSC increases when the flow recall ratio β increases, which is similar to Fig. 10. Fig. 13 shows that the G-FSC algorithm with $\beta = 0.4$ can decrease the maximum bandwidth/delay cost by about 48% compared with that with $\beta = 1.0$. Figs. 14 and 15 plot the maximum bandwidth cost of all control links for different algorithms. There figures have the similar curves as those in Figs. 5 and 10, respectively. We can obtain two conclusions from these figures. 1) For FSC, both MR-FSC and G-FSC algorithms significantly reduce the maximum bandwidth cost compared with both Cemon and OpenTM, respectively. 2) With increasing of the parameter β, the maximum bandwidth cost is increasing accordingly.

As an application example, after (partial) flow statistics collection, we can re-route flows using the routing method, as described in Section III-C, for better network performance,

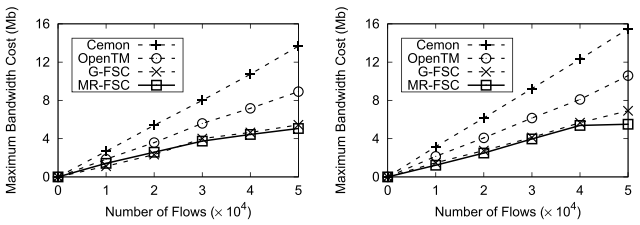


Fig. 14. Maximum bandwidth cost of control links vs. number of flows. *Left plot*: Topology (a); *right plot*: Topology (b).

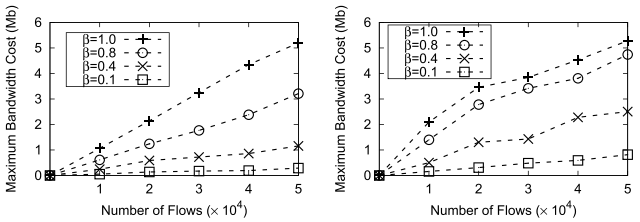


Fig. 15. Maximum bandwidth cost of control links vs. number of flows. *Left plot*: Topology (a); *right plot*: Topology (b).

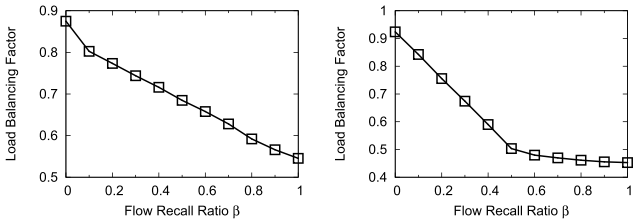


Fig. 16. Load-balancing factor vs. β by flow statistics collection and re-routing. *Left plot*: Topology (a); *right plot*: Topology (b).

such as load balancing. In the simulation, there are 40K flows in a network. The fourth group of simulations observe the route performance by changing the parameter β. Fig. 16 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 [2] has become an emerging technology for future networks. Google designed and implemented B4, which took a software defined networking architecture to connect their data centers across the planet [26]. The authors of [3] presented SWAN, which boosted the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sent. Most previous works [3], 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 [34]. The previous traffic measurement solutions are mostly implemented through the sampling technique. OpenSample [10] leveraged sFlow packets [35] to provide near-real-time measurements of both network load and individual flows. Yu *et al.* [9] 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 [36]. The authors of [12] allocated resources for sketch-based measurement tasks to ensure a user-specified minimum accuracy. Some works studied the rule placement and traffic measurement for an SDN. iSTAMP [37] used (de)aggregation measurement mechanism, which dynamically partitioned the flow entries to allow fine-grained or course measurement tasks of incoming flows. The authors of [24] combined the direct measurement (*e.g.*, flow statistics collection) and inference techniques based on network tomography to derive a hybrid network monitoring scheme, which could strike a balance between measurement overhead and accuracy. 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 [8] specified two different approaches, push-based and pull-based, for flow statistics collection.

The first one is the push-based collection. FlowSense [38] 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 [13] 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 [35]). 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 [19] 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 [16] 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 [39] focused on the tradeoff between accuracy and network overhead. It provided a flexible RESTful API for flow statistics collection at different aggregation levels. The most related works with ours were FlowCover [15] and CeMon [17], 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 [13], 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 testing and extensive simulation results show high efficiency of our proposed algorithms. Since the delay on the switch may depend on its traffic load, and not be fully linear with the traffic amount of FSC, in the future, we will study more practical delay model for flow statistics collection.

Appendix
Proof of Theorem 1

Proof: We prove the NP-hardness by showing that the unrelated processor scheduling (UPS) problem [40] is a special case of CO-FSC. Consider a special case, in which one wildcard can only match one flow on each switch. We regard the switches and flows in the CO-FSC problem as the processors and tasks in the UPS problem. Assume that a flow γ passes through a set of switches, denoted by $V^\gamma = \{v_1, \ldots, v_s\}$. It means that the statistics information of flow γ will be collected from one of switches in V^γ. It can be seen as that a task γ will only be scheduling on one of the processors $v_1, \ldots, v_s$. Thus, the cost of task γ on switch v is $c_1 + c_2$, if $v \in V^\gamma$; otherwise, its cost is ∞. As a result, the cost of each switch becomes the cost of each processor. Then, CO-FSC is equivalent to the following problem: how to schedule these tasks, so that the makespan of all the processors is minimized. Thus, this is a typical UPS problem, which is NP-Hard [40]. Since UPS is a special case of the CO-FSC problem, CO-FSC is an NP-Hard problem too. □

References

[1] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. IEEE INFOCOM*, 2017.

[2] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[3] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.

[4] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," in *Proc. IEEE SDN Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–7.

[5] H. Xu, X. Li, L. Huang, J. Wang, and B. Leng, "High-throughput anycast routing and congestion-free reconfiguration for SDNs," in *Proc. IEEE/ACM 24th Int. Symp. Quality Service (IWQoS)*, Jun. 2016, pp. 1–6.

[6] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," *Comput. Netw.*, vol. 81, pp. 308–319, Apr. 2015.

[7] 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 Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 602–622, 1st Quart., 2016.

[8] B. Pfaff *et al.*, "OpenFlow switch specification v1.3.0," Tech. Rep., 2012.

[9] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 29–42.

[10] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. 34th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun./Jul. 2014, pp. 228–237.

[11] Y. Afek, A. Bremler-Barr, S. L. Feibish, and L. Schiff, "Sampling and large flow detection in SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 345–346, 2015.

[12] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Scream: Sketch resource allocation for software-defined measurement," in *Proc. CoNEXT*, Heidelberg, Germany, 2015, pp. 1–14.

[13] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[14] M. Aslan and A. Matrawy, "On the impact of network state collection on the performance of SDN applications," *IEEE Commun. Lett.*, vol. 20, no. 1, pp. 5–8, Jan. 2016.

[15] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1956–1961.

[16] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow software-defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp.*, May 2014, pp. 1–8.

[17] S. Zhiyang, T. Wang, Y. Xia, and M. Hamdi, "CeMon: A cost-effective flow monitoring system in software defined networks," *Comput. Netw.*, vol. 92, pp. 101–115, Dec. 2015.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, 2010, p. 19.

[19] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Proc. Int. Conf. Passive Active Netw. Meas.*, 2010, pp. 201–210.

[20] H. Xu *et al.*, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. IEEE ICNP*, Nov. 2016, pp. 1–10.

[21] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *Proc. USENIX NSDI*, 2015, pp. 117–130.

[22] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. INFOCOM*, Apr. 2016, pp. 1–9.

[23] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 203–216.

[24] Z. Hu and J. Luo, "Cracking network monitoring in DCNs with SDN," in *Proc. IEEE INFOCOM*, Apr./May 2015, pp. 199–207.

[25] X. Jin *et al.*, "Dynamic scheduling of network updates," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 539–550.

[26] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 3–14.

[27] R. Narayanan *et al.*, "Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane," in *Proc. Eur. Workshop Softw. Defined Netw. (EWSDN)*, Oct. 2012, pp. 79–84.

[28] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 1–6.

[29] A. Clemm, *Network Management Fundamentals*. Indianapolis, IN, USA: Cisco Press, 2006.

[30] *The Mininet Platform*. Accessed: Feb. 20, 2016. [Online]. Available: http://mininet.org/

[31] S. Ryu, "Framework community: Ryu SDN controller," Tech. Rep., 2016.

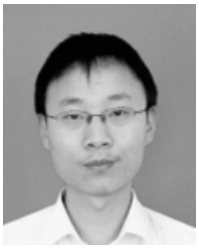
[32] *The Network Topology From The Monash University*. Accessed: Feb. 20, 2016. [Online]. Available: http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies

[33] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[34] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrum. Meas. Mag.*, vol. 18, no. 2, pp. 42–50, Apr. 2015.

[35] P. Phaal and M. Lavine. (Jul. 2004). *SFlow Version 5*. [Online]. Available: http://www.sflow.org/sflow_version_5.txt

[36] T. Wellem, Y.-K. Lai, and W.-Y. Chung, "A software defined sketch system for traffic monitoring," in *Proc. 11th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, May 2015, pp. 197–198.
[37] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, "Intelligent SDN based traffic (de)aggregation and measurement paradigm (iSTAMP)," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 934–942.
[38] C. Yu *et al.*, "FlowSense: Monitoring network utilization with zero measurement cost," in *Passive and Active Measurement*. Berlin, Germany: Springer, 2013, pp. 31–41.
[39] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.
[40] J. K. Lenstra, D. B. Shmoys, and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Math. Program.*, vol. 46, nos. 1–3, pp. 259–271, 1990.

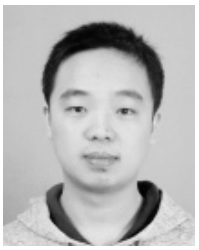


Hongli Xu (M'08) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2002 and 2007, respectively. He is currently an Associate Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored over 70 papers, and held about 30 patents. His main research interest is software-defined networks, cooperative communication, and vehicular ad hoc network.

Zhuolong Yu is currently pursuing the M.S. degree in computer science with the University of Science and Technology of China. His research interests include software-defined networks and mobile computing.

Chen Qian (M'08) received the B.S. degree from Nanjing University in 2006, the M.Phil. degree from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from The University of Texas at Austin in 2013, all in computer science. He is currently an Assistant Professor with the Department of Computer Engineering, University of California at Santa Cruz. He has authored or coauthored over 60 research papers in highly competitive conferences and journals. His research interests include computer networking, network security, and Internet of Things. He is a member of the ACM.

Xiang-Yang Li (F'15) received the bachelor's degrees from the Department of Computer Science and the Department of Business Management, Tsinghua University, in 1995, and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana–Champaign, in 2000 and 2001, respectively. He was a Professor with the Illinois Institute of Technology. He held an EMC-Endowed Visiting Chair Professorship with Tsinghua University. He is currently a Professor and the Dean with the School of Computer Science and Technology, University of Science and Technology of China. He has authored a monograph *Wireless Ad Hoc and Sensor Networks: Theory and Applications*. His research interests include wireless networking, mobile computing, security and privacy, cyber physical systems, social networking, and algorithms. He and his students received several best paper awards and a best demo award. He has been an ACM Distinguished Scientist since 2014. He was a recipient of China NSF Outstanding Overseas Young Researcher (B).



Zichun Liu is currently pursuing the Ph.D. degree in computer science with the University of Science and Technology of China. His research interests include software-defined networks and data security.

Liusheng Huang received the M.S. degree in computer science from the University of Science and Technology of China in 1988. He is currently a Senior Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored six books and over 300 journal/conference papers. His research interests are in the areas of Internet of Things, vehicular ad hoc network, information security, and distributed computing.