Cooperative Flow Statistics Collection with Per-Switch Cost Constraint in SDNs

Xuwei Yang, Hongli Xu, *Member, IEEE,* Xiwen Yu, Chen Qian, *Member, IEEE,* Gongming Zhao, *Member, IEEE,* He Huang, *Member, IEEE,*

***Abstract*—In a software defined network, the controller needs to obtain/collect traffic measurement information (*i.e.*, flow statistics) from switches for different applications, such as traffic engineering. Existing solutions seldom consider the per-switch cost, which may lead to heavy statistics collection cost (*e.g.*, high CPU overhead) on some switches. Due to limited computing power on most commodity switches, heavy statistics collection cost on those switches may seriously interfere with the basic rule operations, especially when some switches need to deal with many new-arrival flows or update routes of existing flows. To address this challenge, we design and implement efficient flow statistics collection (FSC) with limited interference on the basic rule operations. We formally propose a cooperative flow statistics collection with per-switch cost constraint (CP-FSC) problem. We prove that the CP-FSC problem is NP-hard and present an efficient algorithm with approximation ratio $1/2$, based on dynamic programming. To reduce the time complexity, a greedy-based algorithm with approximation ratio $1/3$ is also presented. We implement the proposed FSC algorithms on our SDN platform. The experimental results and the extensive simulation results show 36%-59% performance improvement compared with the existing solutions.**

***Index Terms*—Flow Statistics Collection, Cost, Delay, Wildcard, Approximation.**

I. Introduction

To explore the full advantages of SDN, accurate traffic measurement information in the data plane is essential to various applications, such as traffic engineering [1], security protection and attack detection [2] [3]. For example, many data centers collect traffic statistics for dynamical flow scheduling [4]. Accurate statistics information of flow traffic helps to improve the routing QoS, such as low latency and low packet loss ratio. As another example, traffic statistics information has been widely used to detect attacks and protect the network. Some security attacks, *e.g.* DDoS [2] [3], are often detected by analyzing the changes or entropy of flow traffic. Thus, *it is of vital importance to collect accurate flow statistics from switches*.

X. Yang, H. Xu, X. Yu and G. Zhao are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: issacyxw@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, venn@mail.ustc.edu.cn, gmzhao@ustc.edu.cn.

C. Qian is with the Department of Computer Engineering, University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064. E-mail:cqian12@ucsc.edu.

H. Huang is with the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu, China, 215006. E-mail:huangh@suda.edu.cn.

Most commodity SDN switches are able to measure different types of traffic statistics, including packets, bytes or durations, through specifications in flow entries. To obtain traffic statistics information, the *flow statistics collection* (FSC) problem studied here is fully different from the traditional traffic measurement problem in SDNs [5] [6] [7] [8] [9], which studies how switches derive flow statistics. FSC focuses on **how the controller collects the derived flow statistics from the switches**.

OpenFlow [10] specifies two different approaches for FSC from switches. One is the *push-based* mechanism. The controller learns active flows and derives their statistics by passively receiving reports from switches. The push-based FSC results in low communication overhead between switches and the controller [11]. However, several factors limit its application in some practical scenarios. First, current commodity switches often do not inform the controller about the behavior of a flow until the entry times out. Accordingly, *the push-based FSC approach can not be useful for many real-time applications*, *e.g.*, dynamic flow scheduling [12] [13] and event-triggered statistics collection. Second, to support push-based FSC with smart policy, it poses some additional requirements on both hardware and/or software, such as counters and comparators [12]. In fact, many commodity switches do not equip with these hardwares to support the flexible push-based FSC. Moreover, due to traffic dynamics, it is also a challenging issue to set smart policies for statistics pushing. Third, when the traffic varies dynamically, the FSC events will be frequently triggered and a massive number of measurement reports will be sent to the controller, causing large cost of the switch's CPU resource [14] [13]. The other is the *pull-based* solution: the controller just sends a `Read-State` message (also called FSC request) to retrieve the flow statistics from a switch. Since this mechanism is triggered by the controller, it does not pose additional requirements of both hardware and software on switches, and has been widely used in various SDN applications [13] [15] [16]. In this work, we focus on the pull-based FSC method.

There are three strategies of pull-based flow statistics collection, *per-flow* [15], *per-switch* [13] [16] [17] and wildcard-based [8]. The difference among these schemes is the granularity of statistics collection. Specifically, the per-flow (or per-switch) method will collect the statistics information of one flow (or all flows through this switch) for each FSC request. While for the wildcard-based method, the controller collects the statistics information of a set of flows matching with the wildcard rule in the request. In a recent work [8], the authors have reported that the wildcard-based FSC method

usually could achieve lower processing latency on the switches and cost of the control channel bandwidth than both the per-flow and per-switch methods by combining flow statistics and distributing the FSC requests among all switches. Thus, this paper also focuses on the wildcard-based FSC approach.

In this paper, we study the problem of *efficient flow statistics collection, while less interfering with flow entry operations on switches*. With the development of information technology, the network burdens with more and more flows. To give an example, in a moderate-size data center network [18], the volume of flows arriving at a switch will reach 75K-100K flows/min for a rack composed of 40 servers. It would be in the order of 1,300K if each server hosts around 15 virtual machines. On one hand, the massive number of flows make network links very busy. By the empirical study of the network traffic conducted among 10 data centers, the authors in [19] observed that about 20% of the core links were hot-spots at least 50% of time intervals. Thus, these links are apt to be congested without efficient route control. To provide highly efficient route control, traffic statistics information is necessary and instrumental. On the other hand, a great quantity of flows make a switch's CPU on high loads for dealing with rule operations, such as rule setup for new-arrival flows, and rule modification for flow rerouting. In fact, each commodity SDN switch is usually equipped with a low-end CPU with limited processing capacity [20] [21]. Under this situation, most of the CPU capacity is expected to deal with the switch's basic rule operations. However, DevoFlow [12] has shown that the flow statistics collection overhead will significantly interfere with the switch's basic operations. For example, the FSC for 4500 counters/rules per second will reduce the number of installed rules from 275 to 150 on HP5460zl switches, which may result in long-delay route update or blocking new-arrival flows. Thus, it is important and challenging to achieve FSC with less interference on basic rule operations.

To efficiently obtain the statistics information from switches, the existing methods [8] [13] [16] usually target on reducing the cost for statistics collection of all (or most) flows in the network, without considering the per-switch cost constraint. However, it may lead to massive cost on some switches, which seriously interferes with basic rule operations on these switches, thus decreasing the user experience.

Therefore, *it is important to perform efficient flow statistics collection with per-switch cost constraint*, so that basic rule operations on each switch will be less impeded. When an FSC event is triggered (*e.g.*, the timer is fired or some links are congested), the controller sends FSC requests, each of which contains one wildcard rule, to switches, while satisfying cost constraint on each switch. Our objective is to collect the statistics information of more flows from distributed switches so as to draw a more accurate traffic view in the data plane. The main contributions of this paper are as follows:

1) In order to avoid flow statistics collection interfere with the basic rule operations of switch, we formulate the problem of how to determine which set of wildcard request (or rules) to use for flow statistics collection at each switch such that one can maximize the number of flows whose statistics are collected, while ensuring the FSC cost at each switch does not exceed its cost constraint. The complexity of this problem is also analyzed.
2) We propose the cooperative flow statistics collection (CP-FSC) with cost constraint problem, and prove its NP-hardness. To clarify this problem, we also discuss the difference of proposed CP-FSC from previous problems.
3) We then propose an efficient algorithm with approximation ratio $1/2$ based on dynamic programming. Moreover, a greedy-based algorithm with approximation ratio $1/3$ is also presented. We analyze the time complexity of both algorithms.
4) We implement the proposed FSC algorithms on our SDN platform. The extensive experiment and simulation results show that our algorithms can collect traffic statistics of 36%-59% more flows compared with the existing per-flow and wildcard-based solutions.

The rest of this paper is organized as follows. We introduce the cost of FSC in Section II, formulate the CP-FSC problem, and give the NP-hardness proof. We propose two algorithms for CP-FSC in Section III. The experimental and simulation results are reported in Section IV. We review related works in Section V and conclude the paper in Section VI.

II. Preliminaries

In this section, we first produce the network model, and the specific process for flow statistics collection. We then analyze the FSC cost based on the practical testing, and accordingly give the definition of the cooperative flow statistics collection (CP-FSC) problem.

A. Network Model

An SDN consists of a logically-centralized controller and a set of switches, $V = \{v_1, ..., v_n\}$, $n = |V|$. The data plane of an SDN network is composed of these switches. Therefore, we model the network topology of the data plane as $G = (V, E)$, where E denotes the set of links connecting switches. Note that in a large-scale network, the control plane usually consists of multiple controllers [21], which is helpful to achieve load balancing among individual controllers and enhance the network robustness. Since we care for per-switch FSC cost, the number of controllers will not significantly impact the per-switch performance for FSC. Therefore, for the sake of simplicity, we assume that the control plane consists of only one controller.

As specified by OpenFlow 1.3 [10], a flow table consists of a finite number of flow entries, which is also called rules, and each flow entry consists of several fields. In the flow table, a unique entry is together identified by the match fields and priority. The OpenFlow switch implements traffic measurement through the counter field. When a packet reaches a switch, if there is one or several flow entries that match the packet, this switch will choose the one which has the highest priority, carry out the action specified by the instruction field of this entry, and increase the value of the counter field according to different traffic measurements, such as packets or bytes. Otherwise, the switch reports the packet header to the controller. Then, the controller computes an appropriate route path for this flow, and setups a sequence of entries to the switches on this path.

In fact, *we collect the statistics information of the counter field in the flow entries*. In an SDN, each flow entry may match one or several flows. For simplicity, we first assume that the controller setups exact-match entries (*i.e.*, one entry just matches one flow) in the flow tables using the reactive model. To be more practical, the controller may install some wildcard entries (*i.e.*, one entry may match more than one flow) using the proactive model, which will be discussed in Section III-C.

B. Specific Process for Flow Statistics Collection

We introduce the interaction between switches and the controller. The OpenFlow standard specifies that, each SDN switch communicates with the controller through OpenFlow agent (OFA) implemented by software over a secure TCP connection [20]. A TCP connection between OFA and controller encrypted by TLS will be established. The OFA allows the controller to interact with the switch to control its behavior through the TCP connection. When a packet reaches a switch and there is no matched flow entry, the switch will inform the controller through this connection, which then configures the switch's flow table for packet forwarding.

We then introduce the specific process for FSC from switches and the FSC traffic amount. An FSC event will trigger the following actions. *First*, the controller sends an FSC request, whose length is 144bytes [10], to a switch. *Second*, the counter field and other fields (*e.g.*, match fields) of all matched flow entries are encapsulated into an FSC reply packet by the OFA. *Third*, the controller receives the FSC reply packet sent by the OFA. For simplicity, the set of matched flows (or entries) with the FSC request is denoted as Π'. The length of the reply packet is determined by the number of flows in Π', and is expressed as $l_h + l_e \cdot |\Pi'|$, where l_h stands for the length of the packet header, and l_e denotes the length for each flow entry. As specified in [10], l_h and l_e are 74 bytes and 96 bytes, respectively. We have conducted an experiment to validate these values through our OVS platform. Therefore, the length of a reply packet is $96 \cdot |\Pi'| + 74$ bytes.

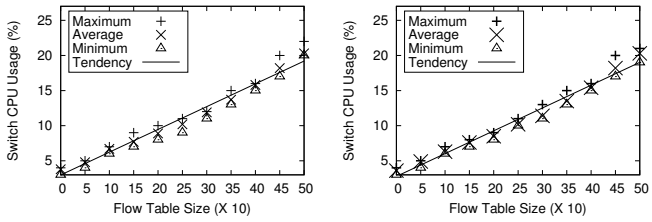


Fig. 1: Switch's CPU Utilization vs. Number of Covered Flows (or entries) per FSC request. *Left plot*: Idle State without Traffic; *right plot*: With Traffic Load 1Gbps.

C. Cost of Flow Statistics Collection

For ease of expression, we assume that all entries are exact-matched rules. In Section III-C, we will extend our solution to the wildcard rules. When an FSC request is sent to switch v, we assume that the statistics information of a flow set Π' will be collected, and denote the cost on switch v as $c(\Pi')$. Intuitively, we expect that the flow statistics collection should less interfere with the basic rule operations, or the CPU consumption for FSC should be constrained. More specifically, if we expect that α (*e.g.*, 80%) CPU capacity will be reserved to deal with the basic rule operations, the switch's CPU utilization for FSC should not exceed $1-\alpha$ (*e.g.*, 20%). Note that, the value of parameter α depends on the user's QoS requirement. The larger the value of parameter α becomes, the more rule operations per second the switch can support. In practice, the OpenFlow standard [10] does not provide the interface to acquire the real-time switch's CPU utilization. Thus, it is infeasible to measure the CPU utilization directly. We should consider an alternative way to reflect the CPU utilization.

Observing the process of FSC as described in Section II-B, each switch's CPU is responsible for parsing the FSC request and encapsulating the flow statistics information. Intuitively, encapsulating more statistics information consumes more CPU resources, especially for low-end CPU on most commodity switches. To validate the intuition, we test on the H3C S5120 switch with different traffic loads. Note that, although OpenFlow does not provide the interface for acquiring the real-time CPU utilization, we log in the switch and use the "display cpu-usage" command to directly acquire the switch's CPU utilization. The left plot of Fig. 1 indicates that the CPU utilization increases almost linearly with the number of collected flows under the idle state. The fitting function is $y = 0.03244x + 2.9818$, where y and x denote the average CPU utilization and the number of collected flows. When we increase the traffic load to 1Gbps on the switch, a similar performance is shown in the right plot of Fig. 1. That's because the SDN switch has special hardware for traffic forwarding, which scarcely affects the CPU utilization for FSC. Accordingly, the fitting function can be described as $y = 0.03249x + 2.7773$.

This figure shows that the CPU utilization for FSC linearly depends on the number of collected flows. As described above, though the controller collects the statistics information of the same number of flows, different FSC schemes will lead to various traffic amount. Thus, to be more precise, we alternatively use the FSC traffic amount, including FSC request and encapsulated statistics information, $c(\Pi') = 96 \cdot |\Pi'| + 218$ with unit byte as the FSC cost. *We believe that this cost metric is sufficient to reflect the CPU utilization for each FSC request*. In [8], the authors give the formal delay cost for each FSC request based on practical testing. However, this cost metric may rely on the performance of physical switches. On the other hand, the cost of FSC traffic amount is independent of different physical switches.

D. Definition of Cooperative Flow Statistics Collection (CP-SFC)

In a typical SDN, when a new-arrival flow reaches a switch, the switch reports the header packet of this flow (*e.g.*, using Packet-in messages [22]) to the controller. Therefore, it is appropriate to suppose that the controller knows the existing flows in a network, denoted as $\Pi = \{\gamma_1, ..., \gamma_h\}$, with $h = |\Pi|$. To be general, the controller may not exactly know the existing flow set in the proactive model. We will discuss how to deal

with this case in Section III-C. In practice, the number of flows through a switch may be dynamic. Since FSC is expected to be finished usually in a fast manner (*e.g.*, 200ms), it is reasonable to suppose that the impact of flow dynamics on FSC can be ignored. Since the route paths for flows are determined by the controller with a centralized manner, we also know the flow set, represented as Π_i, through each switch v_i. For a flow, the controller knows its actual number of packets (or traffic intensity) once its traffic statistic is collected. We say that *this flow is covered*. In the following case, the controller will remove the corresponding entries on switches, and update the flow set Π. The traffic statistic of some flow does not vary, or this flow has finished.

In this paper, we adopt the wildcard-based FSC method. Assuming that there is a set of wildcards, denoted as $\mathbb{R} = \{r_1, r_2, ..., r_m\}$, with $m = |\mathbb{R}|$. To give an example, we describe the general way to derive wildcards as follows [8]: each wildcard r_j only identifies the termination v_j, and does not care where the flow comes from. That is, it can match every source in the network. Each wildcard can also identify other fields in the packet header, such as protocol or source or both. When the controller delivers an FSC request with wildcard r_j to switch v_i, the switch assembles all of the flow entries matching with this wildcard r_j into a reply packet, which will be sent to the controller. In this situation, let the covered flow set be Π_i^j. As the example described above, the wildcard rules for FSC usually meet the following two characteristics: (1) Completeness, *i.e.*, $\bigcup_{r_j \in \mathbb{R}} \Pi_i^j = \Pi_i$, $\forall v_i \in V$. (2) Disjointness, *i.e.*, $\Pi_i^{j_1} \cap \Pi_i^{j_2} = \emptyset$, $\forall r_{j_1} \neq r_{j_2}$, $\forall v_i \in V$. It is noted that because there are different flows whose terminal is v_j on different switches, the controller can deliver FSC requests with the same wildcards r_j to different switch simultaneously.

The controller will deliver Read-State requests to different switches when it expects to collect the statistics information. Each Read-State request is composed of a wildcard. It should be noted that, for an FSC event, a switch may receive more than one request from a controller. Consequently, the total cost (*i.e.*, the total FSC traffic amount) on switch v_i is denoted as $c(v_i)$. To avoid interfering with the basic functions, such as rule setup, the FSC cost on each switch should not surpass the threshold $\mathbb{B}_i$ (*e.g.*, 1Mb), which depends on different application requirements, and will be discussed in Section IV-B. We aim to maximize the number of covered flows in the network, which benefits for various applications, such as flow re-routing or traffic engineering.

One may think that the flow statistics information can be collected only from edge switches. The authors [23] have shown high efficiency of network-wide measurement instead of that only on edge switches. In fact, for the statistics collection, cooperative FSC from all switches also helps to reduce the CPU overhead of all switches, which can support more basic rule operations in an SDN.

Accordingly, we formulate CP-FSC as follows:

$$\max \ \sum\nolimits_{\gamma \in \Pi} z_\gamma$$

$$S.t. \begin{cases} z_\gamma \leq \sum_{\gamma \in \Pi_i^j} x_i^j, & \forall \gamma \in \Pi \\ c(v_i) = \sum_{r_j \in \mathbb{R}} x_i^j c(\Pi_i^j) \leq \mathbb{B}_i, & \forall v_i \in V \\ x_i^j \in \{0,1\}, & \forall v_i, r_j \\ z_\gamma \in \{0,1\}, & \forall \gamma \in \Pi \end{cases} \quad (1)$$

where z_γ indicates whether the statistic of flow γ will be collected or not. x_i^j indicates whether a Read-State request with wildcard r_j will be sent from the controller to switch v_i or not. The first set of constraints decides whether the statistics information of flow γ will be collected or not. The second set of constraints tells that each switch v_i costs at most $\mathbb{B}_i$ to respond to FSC requests. We aim to maximize the number of covered flows, that is, $\max \ \sum_{\gamma \in \Pi} z_\gamma$.

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

Proof: We prove the NP-hardness by showing that the single knapsack (SKP) problem [24] is a special case of CP-FSC. We consider a network situation, in which switch v_i is in the idle state and all others are in the saturated state. As a result, we can collect flow statistics information only from this idle switch. For each wildcard rule r_j, we regard the flow set Π_i^j as an item. Moreover, its weight and value are defined as $c(\Pi_i^j)$ and $|\Pi_i^j|$, respectively. Then, our CP-FSC problem turns to find a set of items to maximize the total values with total weight constraint B_i. Thus, this is a typical SKP problem, which is NP-hard [24]. Since SKP is a special case of our problem, CP-FSC is an NP-hard problem too. ■

E. Differences to Existing Problems

One may think that the CP-FSC problem is similar to the existing problems, such as the budgeted maximum coverage problem [25] or the maximum coverage problem with group budget constraints [26]. Before discussing the differences between these two problems, the definition of the budgeted maximum coverage problem will be given first.

Definition 1: Budgeted Maximum Coverage (BMC) Problem [25]: Given a ground set X, a collection of sets $\mathbb{S} = \{S_1, S_2, ..., S_p\}$, with each set S_j associated with a cost $c(S_j)$ defined over a domain of weighted elements, and a cost threshold $\mathbb{C}$, the objective is to find a subset of $\mathbb{S}$, denoted as $\mathbb{S}'$, which maximizes the total weights of elements covered by $\mathbb{S}'$ and the total cost of $\mathbb{S}'$ should not exceed $\mathbb{C}$.

Differences with the BMC Problem: The BMC problem regards that all the element sets $\mathbb{S}$ are put in a group, and there is only one total cost constraint $\mathbb{C}$ on this group. However, for the CP-FSC problem, each switch corresponds to a group. Thus, there are n groups, in which each group for each switch v_i is associated with a cost constraint $\mathbb{B}_i$.

Definition 2: Maximum Coverage with Group Budget Constraints (MCG) Problem [26] Given a ground set X and subsets $\{S_1, S_2, ..., S_p\}$, each set S_j is combined with a cost/budget $c(S_j)$. Given sets $G_1, G_2, ..., G_l$, each G_i, called a group, is a subset of $\{S_1, S_2, ...S_p\}$. Further, there are given an overall cost $\mathbb{C}$ and a cost $\mathbb{C}_i$ for each group G_i, $1 \leq i \leq l$. The objective is to find a subset $H \subseteq \{S_1, S_2, ..., S_p\}$, which satisfies that the total cost of the set in H is at most $\mathbb{C}$, and the union of sets in H contains as many elements as possible. At the same time, ensure that, for any group G_i, the total cost of the sets in $H \cap G_i$ will not exceed $\mathbb{C}_i$.

Differences with the MCG problem: In this case, each group G_i for switch v_i can be regarded as $\{\Pi_i^1, ..., \Pi_i^m\}$. There are two main differences between CP-FSC and MCG. One is that CP-FSC has no total cost/budget constraint on all switches, or we can say that the total cost constraint is infinite for CP-FSC. The other is the mutual exclusion feature of wildcard rules for CP-FSC, that is, $\Pi_i^{j_1} \cap \Pi_i^{j_2} = \Phi, \forall r_{j_1} \in \mathbb{R}, r_{j_2} \in \mathbb{R}, 1 \leq i \leq n$. Thus, we can say that CP-FSC is a special case of MCG.

The authors in [26] have designed a greedy algorithm for the MCG problem with approximation ratio $\frac{\mu}{6(\mu+1)}$, where μ is the approximation ratio for a polynomial-time oracle, with $0 < \mu \leq 1$. Though we can directly apply the designed algorithms for MCG [26] to find an approximation solution for our CP-FSC problem, the approximation performance may not be satisfied. In this paper, according to the special features of the CP-FSC problem, we will design two efficient algorithms with better approximation factors in Section III.

III. Algorithm Design of Proportional Fairness

Since the CP-FSC problem is NP-hard, we first propose an approximation algorithm using dynamic programming and analyze the performance of this algorithm (*Section III-A*). Then we derive a greedy-based approximation algorithm which has lower time complexity (*Section III-B*) compared with the DP-based algorithm (*Section III-B*). We also give some discussion to enhance our proposed algorithms (*Section III-C*).

A. A $1/2$ Approximation Algorithm via Dynamic Programming

1) Algorithm Description: This section introduces an approximation algorithm, called D-FSC, based on dynamic programming (DP) to solve the CP-FSC problem. Before the algorithm description, we consider a special case in which there is only one switch (*e.g.*, v_i) in the network. Obviously, this special case belongs to the 0-1 knapsack problem [24]. Specifically, the size of the knapsack (or switch v_i) is its cost constraint, *i.e.*, $\mathbb{B}_i$. For each wildcard rule r_j, Π_i^j can be regarded as an individual object, whose cost is $c(\Pi_i^j)$. The profit of each set Π_i^j, denoted by $p(\Pi_i^j)$, is the number of uncovered flows in set Π_i^j. It can be solved by the previous knapsack algorithms, *e.g.*, [27].

The D-FSC algorithm is composed of a group of iterations. In each iteration, there are two main steps. In the first step, we use dynamic programming for the 0-1 knapsack problem to determine the maximum number of incrementally covered flows for each switch (lines 6-11 in Alg.1), which will be described in Section III-A2. We then choose the switch, denoted by v_i, with the maximum profit among all the switches. The DP method also determines a set of wildcard rules applied on switch v_i. In the second step, the algorithm updates the profit of each flow set Π_i^j. For simplicity, let $\overline{\Pi}$ be the set of covered flows. The profit of a flow set Π_i^j is updated as $p(\Pi_i^j) = |\Pi_i^j - \overline{\Pi}|$. The algorithm will not terminate until all switches have been checked. The D-FSC algorithm is formally described in Alg. 1.

Algorithm 1 D-FSC: DP-based FSC

1: The switch set is denoted by V.
2: **while** $|V| > 0$ **do**
3: **Step 1: Choosing a switch with the maximum profit**
4: **for** each switch $v_i \in V$ **do**
5: Apply the **dynamic programming method** to compute the maximum profit $p(v_i)$ for switch v_i with cost constraint $\mathbb{B}_i$:
6: Set $A(1, p(\Pi_i^1)) = c(\Pi_i^1), \forall v_i$. Set $A(1,k) = \infty, \forall k$ when $k \neq p(\Pi_i^1)$.
7: Set $p(v_i) = p(\Pi_i^1)$ and $\mathbb{R}_i = \{\Pi_i^1\}$
8: **for** j in 2 to $|V|$ **do**
9: **for** k in 1 to $\lfloor \mathbb{B}_i / c_1 \rfloor$ **do**
10: Compute $A(j,k)$ according to Eq. (2) and record related set $S_{j,k}$.
11: **if** $k > p(v_i)$ and $A(j,k) \leq \mathbb{B}_i$ **then**
12: Set $p(v_i) = k$ and $\mathbb{R}_i = S_{j,k}$
13: Select switch v_q with the maximum profit $p(v_q)$, and the chosen rule set is denoted by $\mathbb{R}' = \mathbb{R}_q$
14: **for** Each wildcard rule $r_j \in \mathbb{R}'$ **do**
15: $\overline{\Pi} = \overline{\Pi} + \Pi_q^j$
16: $V = V - \{v_q\}$
17: **Step 2: Updating the profit of each flow set**
18: **for** each switch $v_i \in V$ **do**
19: **for** each wildcard rule $r_j \in \mathbb{R}$ **do**
20: $p(\Pi_i^j) = |\Pi_i^j - \overline{\Pi}|$

2) Solving 0-1 knapsack using DP: This section will describe a DP-based algorithm to compute the maximum profit $p(v_i)$ for each switch v_i with cost constraint $\mathbb{B}_i$. Besides, z denotes the largest profit-cost ratio among these flow sets. It means that the achievable profit under the cost constraint $\mathbb{B}_i$ is no more than $z \cdot \mathbb{B}_i$. The profit-cost ratio of each flow set Π_i^j is $\frac{p(\Pi_i^j)}{c(\Pi_i^j)} \leq \frac{|\Pi_i^j|}{(c_1|\Pi_i^j|+c_2)/(1-q_i)} \leq \frac{1-q_i}{c_1} \leq \frac{1}{c_1}$. So we have $z \leq \frac{1}{c_1}$, and $z \cdot \mathbb{B}_i \leq \frac{\mathbb{B}_i}{c_1}$. For each wildcard $r_j \in \mathbb{R}$ and $k \in \{1, ..., \lfloor z \cdot \mathbb{B}_i \rfloor\}$, $S_{j,k}$ denotes a subset of $\{\Pi_i^1, ..., \Pi_i^j\}$, whose total cost is minimum among all subsets with total profit of exactly k. $A(j,k)$ denotes the cost of set $S_{j,k}$ ($A(j,k) = \infty$ if no such set exists). Obviously, $A(1,k)$ is known for every $k \in \{1, ..., \lfloor \mathbb{B}_i / c_1 \rfloor\}$. The following DP expression helps to compute all values $A(j,k)$.

$$A(j+1,k) = \begin{cases} \min\{A(j,k), c(\Pi_i^{j+1}) \\ +A(j, k - p(\Pi_i^{j+1}))\}, \text{if } p(\Pi_i^{j+1}) < k \\ A(j,k), \text{otherwise} \end{cases} \tag{2}$$

The maximum profit under the cost constraint $\mathbb{B}_i$ can be expressed by $\max\{k | A(m,k) \leq \mathbb{B}_i\}$, which is the flow sets with the highest profit such that the total cost of them is less than or equal to the constraint. We thus get a polynomial-time algorithm for knapsack. The time complexity of the dynamic programming method for 0-1 knapsack is as follows [27].

Theorem 2: The running time of the DP method is $O(m \cdot \mathbb{B}_i / c_1)$ or $O(m \cdot \mathbb{B}_i)$, where the unit of $\mathbb{B}_i$ is byte.

One may think that the time complexity of the DP method for 0-1 knapsack may be pseudo-polynomial due to the large size of the knapsack [27]. We consider two cases of cost

constraint $\mathbb{B}_i$. As $\mathbb{B}_i$ exceeds the value of $h \cdot (c_1 + c_2)$, where c_1 and c_2 are two constants, and h is the number of flows in the network, all the flow sets through switch v_i can be collected. Then, the DP method just cares for the case of $\mathbb{B}_i \leq h \cdot (c_1 + c_2)$, *i.e.*, $\mathbb{B}_i = O(h)$. Thus, the time complexity of DP is not pseudo-polynomial, but polynomial.

3) Performance Analysis: In the following, Q_G denotes the set of covered flows by the D-FSC algorithm. In the l^{th} iteration of D-FSC, the covered flow set is G'_l, and the incremental profit is denoted by X'_l. Obviously $X'_l = \omega(G'_l \backslash \bigcup_{i=1}^{l-1} G'_i)$.

Lemma 3: The D-FSC algorithm can achieve the approximation ratio $1/2$ for the CP-FSC problem.

Proof: Let β be the approximation ratio of the greedy algorithm for 0-1 knapsack. Consider an instant that the D-FSC algorithm has executed l-1 iterations. In the l^{th} iteration, the algorithm chooses the switch $v_{l'}$. Assume that the optimal solution will select a flow set, denoted by O_l, from switch $v_{l'}$. If we choose O_l instead of G'_l in this iteration, the incremental profit becomes $\omega(O_l \backslash \bigcup_{i=1}^{l-1} G'_i)$, denoted by X''_l. Obviously, we have $X'_l \geq \beta \cdot X''_l = \beta \cdot \omega(O_l \backslash \bigcup_{i=1}^{l-1} G'_i) \geq \beta \cdot \omega(O_l \backslash Q_G)$. It follows

$$\begin{aligned} \omega(Q_G) &= \sum_{l=1}^{m} X'_l \geq \sum_{l=1}^{m} \beta \cdot \omega(O_l \backslash Q_G) \\ &= \beta \cdot \sum_{l=1}^{m} \omega(O_l \backslash Q_G) \geq \beta \cdot \omega(\bigcup_{l=1}^{m} O_l \backslash Q_G) \\ &= \beta \cdot \omega(OPT \backslash Q_G) \geq \beta \cdot [\omega(OPT) - \omega(Q_G)] \quad (3) \end{aligned}$$

Thus, we have

$$\omega(Q_G) \geq \frac{\beta}{1+\beta} \cdot \omega(OPT) \quad (4)$$

Since the dynamic program method achieves the optimal result for 0-1 knapsack [28], by Eq. (4), the D-FSC algorithm can achieve the approximation ratio $1/2$ for CP-FSC. ■

Assume that the maximum cost constraint of all switches is denoted by $\mathbb{B}$, *i.e.*, $\mathbb{B} = \max\{\mathbb{B}_i,\ 1 \leq i \leq n\}$. f denotes the maximum number of switched visited by each flow.

Theorem 4: The time complexity of the D-FSC algorithm is $O(n^2 \cdot m \cdot \frac{\mathbb{B}}{c_1} + n \cdot f \cdot h)$.

Proof: Suppose that there are n switches in this network. In each iteration, we regard every switch v_i as a knapsack. According to Theorem 2, for switch v_i, the time complexity for Lines 5-16 is $O(m \cdot \frac{\mathbb{B}}{c_1})$, where m is the number of wildcard rules in a network. Since there are at most n unchecked switches, the time complexity for the first step is $O(n \cdot m \cdot \frac{\mathbb{B}}{c_1})$. In the second step, we update the profit of each flow set, it takes $O(f \cdot h)$ time, for each flow will appear at most f flow sets, and h is the number of flows in the network. Since the algorithm consists of at most n iterations, the time complexity of D-FSC is $O(n^2 \cdot m \cdot \frac{\mathbb{B}}{c_1} + n \cdot f \cdot h)$. ■

B. A $1/3$ Approximation Algorithm for CP-FSC

Though the approximation ratio of the D-FSC algorithm is very close to the best ratio for the CP-FSC problem, it may not be always feasible for some real-time applications due to its high time complexity. Thus, this section presents an efficient algorithm with low time complexity for the CP-FSC problem. The proposed algorithm is called G-FSC.

1) Algorithm Description: By Theorem 4, the time complexity of D-FSC mainly stems from the DP method for solving 0-1 knapsack in the first step. The authors in [28] have designed a greedy method with low time complexity for the 0-1 knapsack problem. Basically, for each switch, the greedy method iteratively chooses the flow set (corresponding to the wildcard rule) with the largest profit-cost ratio with a cost constraint. Here, we omit the description of the greedy method for 0-1 knapsack, and the readers can refer [28] for details. We choose a switch, denoted by v_i, with maximum profit. Then, we determine the set of rules $\mathbb{R}'$, which will be sent to switch v_i to collect the flow statistics, and update the covered flow set and the profit for each flow set (Line 13). The G-FSC algorithm is described as follows.

Algorithm 2 G-FSC: Greedy FSC

```
1:  V = all switches set.
2:  while |V| > 0 do
3:      Step 1: Choosing a switch with maximum profit
4:      Regard every switch v_i as a package and compute the
        profit p(v_i) using greedy 0-1 knapsack
5:      Select switch v_i with the maximum profit
6:      V = V − {v_i}
7:      The chosen rule set is denoted by R'
8:      for Each wildcard rule r_j ∈ R' do
9:          Π̄ = Π̄ + Π_i^j
10:     Step 2: Updating the profit of each flow set
11:     for each switch v_i ∈ V do
12:         for each wildcard rule r_j ∈ R do
13:             p(Π_i^j) = |Π_i^j − Π̄|
```

2) Performance Analysis: In the following, Q_G denotes the set of covered flows by the G-FSC algorithm. In the l^{th} iteration of G-FSC, the covered flow set is G'_l, and the incremental profit is denoted by X'_l. Obviously $X'_l = \omega(G'_l \backslash \bigcup_{i=1}^{l-1} G'_i)$.

Lemma 5: The G-FSC algorithm can achieve the approximation ratio $1/3$ for the CP-FSC problem.

Proof: Since the greedy method achieves the approximation ratio $1/2$ for 0-1 knapsack [28], by Eq. (4), the G-FSC algorithm can achieve the approximation ratio $1/3$ for the CP-FSC problem. ■

Lemma 6: The time complexity of G-FSC is $O(n^2 \cdot m \log m + n \cdot f \cdot h)$.

Proof: Similar to the D-FSC algorithm, there are at most n iterations in the G-FSC algorithm, and each iteration consists of two main steps. In the first step, the time complexity of the greedy algorithm is $O(m \cdot \log m)$ [28]. In the second step, we update the profit of each flow set, which takes $O(f \cdot h)$ time. As a result, the total time complexity of the G-FSC algorithm is $O(n^2 \cdot m \cdot \log m + n \cdot f \cdot h)$. ■

C. Discussion

We discuss some practical issues to make the proposed algorithms more applicable.

First, the objective of the problem definition Eq. (1) is to maximize the number of collected flow statistics. For applications such as traffic engineering or security analytic, flow statistics from specific network locations or specific flows (*e.g.* identified by different protocol features) might be more important than others. So we introduce an "importance label" θ_γ for flow/rule γ and maximize the collection of rules under

consideration of their importance. The statistics of rules with larger importance labels will be more likely to be collected by our proposed solution. The importance of rules depends on the application's requirement. For example, elephant flows are more important than mouse flows in flow rerouting, while traffic from a specific IP may be more important than other traffic in some attack detection applications. The importance label can be set according to the applications' needs. The definition of the extended problem can be rewritten as follows.

$$\max \sum\nolimits_{\gamma\in\Pi} \theta_\gamma z_\gamma$$
$$S.t. \begin{cases} z_\gamma \le \sum_{\gamma\in\Pi_i^j} x_i^j, & \forall\gamma\in\Pi \\ c(v_i)=\sum_{r_j\in\mathbb{R}} x_i^j c(\Pi_i^j)\le \mathbb{B}_i, & \forall v_i\in V \\ x_i^j\in\{0,1\}, & \forall v_i, r_j \\ z_\gamma\in\{0,1\}, & \forall\gamma\in\Pi \end{cases} \quad (5)$$

We should note that the D-FSC/G-FSC algorithms can solve Eq. (5) through some modifications. We need to redefine the meaning of profit $p(\Pi_i^j)$ of each set Π_i^j as the summation of important label of uncovered flows in this set rather than the number of uncovered flows in this set in Alg. 1. We also need to adjust the update value of $p(\Pi_i^j)$ in the second step of D-FSC/G-FSC algorithms for Eq. (5). Specifically, let $\overline{\Pi}$ be the set of covered flows. The profit of a flow set Π_i^j is updated to $p(\Pi_i^j)=\sum_{\gamma\in\Pi_i^j\setminus\overline{\Pi}}\theta_\gamma$ rather than $p(\Pi_i^j)=|\Pi_i^j-\overline{\Pi}|$ in Line 20 of Alg. 1 and Line 13 of Alg. 2.

In fact, after the above modification, the proposed D-FSC/G-FSC algorithms can guarantee the same approximation performance for Eq. (5) with the importance label θ. Specifically, the proposed D-FSC/G-FSC algorithms can be divided into several rounds, and each round of both algorithms needs to solve the 0-1 knapsack problem for the remaining switches. The goal of 0-1 knapsack problem is to maximize the total profits of selected items. In the original algorithms, the profit of each item (flows with the same wildcard) is the number of uncovered flows in this item when we solve the 0-1 knapsack problem. After modification, the profit of each item is the summation of importance label values of uncovered flows. According to the performance analysis in Section III-A-3) and Section III-B-2), if the approximation ratio of algorithm for the 0-1 knapsack problem is β, the approximation ratio of both D-FSC/G-FSC algorithms is $1/(1+\beta)$. We use dynamic programming and greedy algorithm to solve the 0-1 knapsack problem in D-FSC and G-FSC algorithms, respectively. In fact, no matter the importance label of each flow is 1 or other value, the approximation ratio of dynamic programming and greedy algorithm for the 0-1 knapsack problem is 1 and 1 / 2, respectively. So after the above modification, the proposed algorithms can guarantee the same approximation performance (*i.e.*, 1/2 for D-FSC and 1/3 for G-FSC) for Eq. (5).

Second, if only exact-match rules are installed at all switches in the network, our proposed solution will collect the statistics of some rules, and all the statistics collected are from exact-match rules. Our proposed solution can also be applied in the network where the wildcard rules are deployed, and the statistical collected may be from the wildcard rule. In practice, in order to schedule more flows in the network, the wildcard rules/entries that can match arbitrary header ranges are allowed to be installed at the switches. The statistics of all flows matching this wildcard entry will be aggregated into one value. As a result, the statistics information of each individual flow can not be distinguished. On the premise that the original-destination pairs are known, denoted as $\overline{\Gamma}$ in an SDN. To deal with this, we regard all flows matching a wildcard entry as one "flow", which is also called macro-flow [29]. For example, if one entry e_1 matches three flows $\{\gamma_1,\gamma_2,\gamma_3\}$, the combination of these three flows can be regarded as one macro-flow. Our proposed algorithms can be adopted for this case after some adjustments. However, the proposed solution has some limitations when dealing with wildcard rules, such as the partial overlap between "macro-flows" (*i.e.*, covering the same flows) and reduced monitoring granularity (and hence reduced accuracy of per-flow statistics) when collecting macro-flow statistics. When the wildcard rule is used, what the proposed D-FSC/G-FSC algorithms solve in each round is not a 0-1 knapsack problem, but a BMC problem [25]. We can replace the dynamic programming or greedy algorithm with an appropriate algorithm [25]. Moreover, the decline of monitoring accuracy is inevitable, which is not addressed by the proposed algorithms.

After FSC, we obtain the statistics of aggregated flow e_1. We can also get the statistics of individual flow γ_1 if there is another entry for γ_1. Collecting individual flow statistics can improve the accuracy of FSC but increase the cost, while collecting aggregated flow statistics reduces the cost of FSC but decreases the accuracy. We can adjust the algorithm's preference for accuracy by carefully setting the importance label for each individual flow rule and wildcard rule. Recall that the importance of rules depends on the application's requirement. For example, if the application just wants to cover flows as more as possible, we can set the importance label (or profit) of the per-flow rule to 1, and set the importance label of the wildcard rule to the number of flows covered by this wildcard rule. If the application has higher requirements for the accuracy of statistical information, the importance label of the wildcard rule can be appropriately reduced. Elephant flows may be more important than mouse flows in some traffic engineering applications, so we can set a larger importance label for elephant flows than mouse flows (*e.g.*, 5 for elephant flows and 1 for mouse flows). Traffic from a specific IP may be more important than other traffic in some attack detection applications, so we can list these sensitive IP and set a larger importance label for the matched flows (*e.g.*, 10 for flows matched sensitive IP and 1 otherwise). The importance label can be set according to the applications' needs.

Third, we discuss **how to determine the cost threshold $\mathbb{B}_i$ for each switch** v_i in the practical applications. A simple way is to estimate parameter $\mathbb{B}_i$ through the required CPU capacity for basic rule operations, as illustrated in Section II-C. More precisely, provided that the switch can support the wildcard-based FSC, it is feasible to test the impact of the FSC on the quantity of installed rules per second [12]. According to the requirement on the number of installed rules per second on switches, it is feasible to determine the required CPU utilization for rule operations. Then, we can estimate the rough value of parameter $\mathbb{B}_i$ by Fig. 1.

Fourth, we discuss **delay-constraint FSC in an SDN**. In

some practical applications, it is also required that the flow statistics collection should be completed within a given delay. For example, for many flow schedulers such as Hedera [30], the control loop needs less than 500ms for better network performance. Thus, the delay for FSC should be less than 200ms, so that there remains enough time for flow scheduling and re-routing [12]. However, even the FSC delay increases almost linearly with the number of covered flows in the ideal state [8], the delay will dynamically change with CPU load. Since it is difficult to predict the real-time CPU utilization, the FSC delay will be dynamically changed. To deal with this, after we run the D-FSC or G-FSC algorithms, the controller will determine a set of wildcard rules, denoted by $\mathbb{R}_i$, for switch v_i. We randomly choose a wildcard, denoted by r_j, and send a request with wildcard r_j to the switch. The controller repeats these operations until the delay threshold expires. In this situation, the number of collected flows may be reduced under FSC delay constraint, which will be validated in Section IV-B.

Fifth, the FSC request can also match multiple fields. We discuss **how matching multiple fields that may affect the performance of the system**. An FSC request identified multiple wildcard fields will collect fewer flow statistics compared with single wildcard field. So the controller will send more FSC requests. Specifically, assuming that each FSC request matches the source and terminal simultaneously. The controller will send $O(|V|^2)$ FSC requests to each switch, which is about $|V|$ times of the number of FSC requests ($O(|V|)$) using wildcard based on the terminal. Moreover, the switch needs to process more requests rapidly, which will cause more control/switch overhead. It is clear that the control/switch overhead increases as the number of fields a wildcard matches increases. So a wildcard should not match too many fields in our work.

Sixth, we discuss **when to collect flow statistics**. Flow statistics are needed for applications such as traffic engineering or security analytic, which runs periodically (*e.g.* 5 minutes in [31]). So we can collect flow statistics at the minimum operating frequency of these applications to support their needed flow statistics.

IV. Performance Evaluation

In this section, we mainly give the simulation results and the testing results to show the high efficiency of our proposed algorithms.

A. Performance Metrics and Benchmarks

Given a per-switch (FSC traffic amount) cost constraint, we expect to determine an FSC solution to cover more flows, which benefit different applications, *e.g.*, traffic engineering. Thus, in our numerical evaluation, we use the following metrics.

1) The number of covered flows. After determining the FSC solution, the controller will send FSC requests to different switches, and we can compute the number of covered flows in a network.
2) Switchs CPU Utilization. When the switch receive the FSC requests, it will reply the flow statistics. We will test the CPU utilization of the switch at this time.

Compared with the state-of-the-art solutions and most-related solutions, we validate the performance of the proposed D-FSC and G-FSC algorithms by both simulations and prototype experiments. The first one, called per-flow, is adapted from OpenTM [17], which is a per-flow FSC method. In OpenTM, the statistic information of each flow is collected by the controller from a switch along the route path randomly. In our evaluation, for a flow, if the cost of the selected switch exceeds its constraint, the statistic information of this flow will not be collected. The second one is a random wildcard-based method. Specifically, the controller will randomly choose wildcard rules for FSC with the cost constraint, and send them to this switch.

B. Simulation Evaluation

1) Simulation Setting: As running examples, two practical and typical topologies are adopted in the simulation. one is the campus network, and the other is the data center network. We denote the first topology as (a), which is composed of 100 switches, 200 servers and 397 links from [32]. The second one is the fat-tree topology [33], which has in total 80 switches (*i.e.*, 32 edge switches, 16 core switches, and 32 aggregation switches) and 192 servers. Since the authors of [12] have presented that, for the flow size, more than 80% of the top-ranked flows may host less than 20% of the total traffic. Therefore, the size for each flow is allocated according to this guideline. The average active time of flows is 10 seconds and the average flow rate is 1Mbps in the test. To measure the FSC cost in the simulations, we adopt the same parameter values as in Section II-C. That is, $c_1 = 96$ and $c_2 = 218$ with unit byte. Each simulation is conducted 100 times, and we take the average of the numerical results.

2) Simulation Results: To validate the effectiveness of the presented FSC algorithms, two groups of simulations are conducted. By default, we generate 120K flows in the network. We study the different constraints, including FSC traffic amount constraint and FSC delay constraint, on the efficiency of FSC. Since the practical connection bandwidth is tested to be less than 10Mbps [34] [35], the FSC traffic amount constraint is by default set as 1Mbits (or 1Mb) for each switch so that there is enough CPU resource for flow scheduling and re-routing.

The first set of four simulations observes the percentage of covered flows by altering the values of different parameters, like the number of flows and the cost constraint threshold. Fig. 2 shows the number of covered flows by changing the number of flows from 20K to 160K. We observe that, for a given cost constraint (*e.g.*, 1Mb), when there are more flows in the network, the number of covered flows increases for all these algorithms. Specifically, given 100K flows in the network, of 40.6%, 63.7%, 87.1% and 100% traffic statistics information are collected by four algorithms, respectively, by the left plot of Fig. 2. It means that, compared with the random method and the per-flow FSC method, our proposed D-FSC algorithm can improve the percentage of covered flows by at least 36.3% and 59.4%, respectively. When the FSC traffic amount constraint on each switch obeys the Gaussian distribution with various expected values, Fig. 3 shows that the number of covered

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCOMM.2021.3078296, IEEE Transactions on Communications

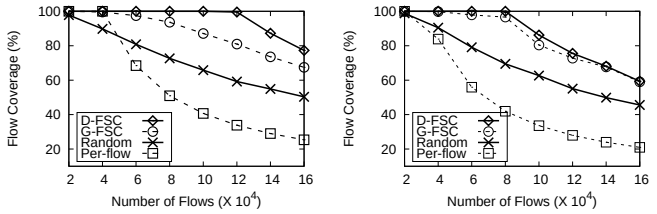


Fig. 2: Flows Coverage (%) vs. Number of Flows with FSC Traffic Cost Constraint under Uniform Distribution. *Left plot*: Topology (a); *right plot*: Topology (b).

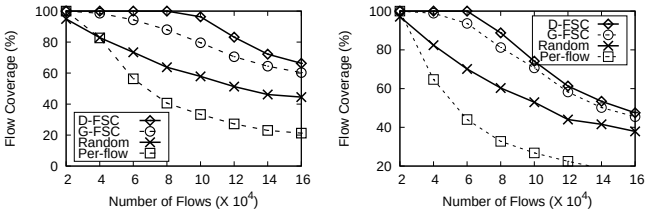


Fig. 3: Flows Coverage (%) vs. Number of Flows with FSC Traffic Cost Constraint under Gaussian Distribution. *Left plot*: Topology (a); *right plot*: Topology (b).

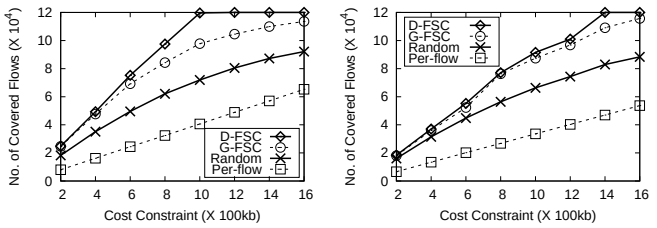


Fig. 4: Number of Covered Flows vs. FSC Traffic Cost Constraint under Uniform Distribution. *Left plot*: Topology (a); *right plot*: Topology (b).

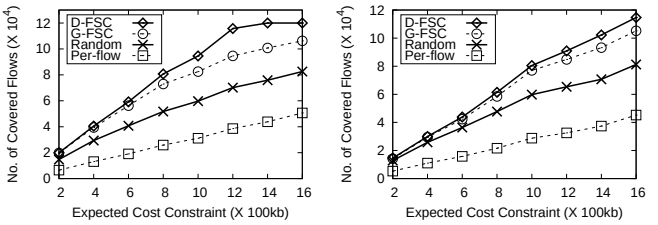


Fig. 5: Number of Covered Flows vs. FSC Traffic Cost Constraint under Gaussian Distribution. *Left plot*: Topology (a); *right plot*: Topology (b).

flows also increases with more flows in the network. Given 100K flows in topology (a), 33.2K, 57.8K, 79.5K and 96.3K traffic statistics information are collected by four algorithms, respectively, by the left plot of Fig. 3. In other words, the D-FSC algorithm can improve the number of covered flows by about 39.9% and 65.5% compared with the random method and the per-flow FSC method, respectively. Figs. 2 and 3 also show that the increasing ratio for covered flows is much slower with more flows in both topologies. Fig. 4 shows that the number of covered flows increases almost linearly with the traffic amount constraint for all algorithms while the constraint is not large, *e.g.*, less than 1Mb on topology (a). When it exceeds 1Mb on topology (a), the number of covered flows by the D-FSC algorithm remains almost unchanged (*i.e.*, number of all flows in the network). When the traffic constraint of

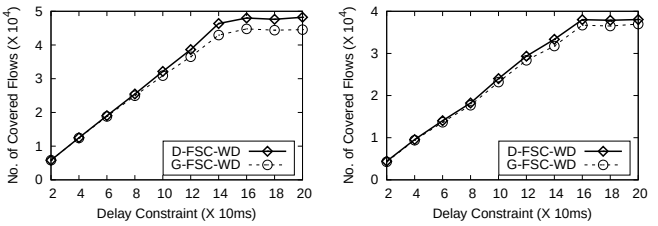


Fig. 6: Number of Covered Flows vs. Delay Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

each switch obeys the Gaussian distribution, Fig. 5 shows the similar performance as that in Fig. 4. Specifically, given a cost constraint 1Mb on topology (a), four algorithms can collect the statistics information of 31.1K, 59.7K, 82.4K and 94.3K flows, respectively.

To respond to the fourth issue in Section III-C, the second set of simulations shows that the number of covered flows with both FSC traffic amount and delay cost constraints for two proposed algorithms. For the delay of each FSC request, we adopt the delay cost model as follows [8]: $c'(\Pi') = 0.19 \cdot |\Pi'| + 1.21$ with unit ms, where Π' is the covered flow set by this FSC request. In practice, the delay for each FSC request may be dynamic. To express the struggle case, we update the delay cost model as: $c(\Pi') = \delta(0.19 \cdot |\Pi'| + 1.21)$, where δ is a random constant from 1 to 2. We generate 80K flows in the network and the FSC traffic amount constraint for each switch is set as 500K bits. For clear description, D-FSC-WD and G-FSC-WD denote the D-FSC and G-FSC algorithms with delay constraint. The simulation results in Fig. 6 shows that the number of covered flows by changing the delay constraint from 20ms to 200ms. The number of covered flows increases almost linearly with the delay constraint for both algorithms while the delay constraint is not large, *e.g.*, less than 160ms on topology (a). When it exceeds 160ms on topology (a), the number of covered flows by both D-FSC-WD and G-FSC-WD remains almost unchanged.

From the simulation results in Figs. 2-6, three conclusions can be drawn as follows. First, these simulation results show that three wildcard-based FSC algorithms (*i.e.*, D-FSC, G-FSC and random) can achieve better FSC performance than the per-flow method, which validates the advantage of the wildcard-based scheme for FSC. Second, both the D-FSC and G-FSC algorithms can cover more flows in comparison with the random algorithm and the per-flow FSC method from Figs. 2-5. Specifically, the D-FSC algorithm can increase the number of covered flows by about 36.3% and 59.4% in comparison with the random and per-flow FSC methods, respectively. Third, though the approximation performance of G-FSC is worse than that of the D-FSC algorithm, our simulation results show that the performance of G-FSC is very close to that of D-FSC. Especially on topology (b), two algorithms achieve almost the similar performance on a structured topology, *e.g.*, topology (b). Thus, we can conclude that both D-FSC/G-FSC can achieve a satisfactory FSC performance.

C. Test-bed Evaluation

1) Implementation On the Platform: There are two options to build our SDN platform. The first one is to use the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCOMM.2021.3078296, IEEE Transactions on Communications

physical switches, *e.g.*, H3C S5120-28SC-HI switches in our lab. It should be noted that the present design of these H3C switches only allow us to perform the flow statistics collection using the per-flow and per-switch schemes. These two schemes have been implemented through the RESTful APIs. The Ryu controller has specified these APIs. We have also tested the statistics collection using the per-flow and per-switch schemes on our H3C S5120 switches. Unfortunately, the wildcard-based flow statistics collection is not supported by H3C S5120 switches presently, which are designed based on the conventional switching fabric. The second is based on the virtual switches. Compared with hardware switches, the virtual switches, which are implemented by the software, have all the three functions of FSC to collect flow statistics. Thus, our experiments are conducted on the virtual switches. Fortunately, many hardware vendors produce switches that support OpenFlow wildcard-based statistics collection, such as H3C h5560X-EI, Huawei S12700, and Pica8 AS4610-54T. Therefore, our scheme can be used not only in the cloud network composed of OVS, but also in the network composed of hardware switches from different vendors.

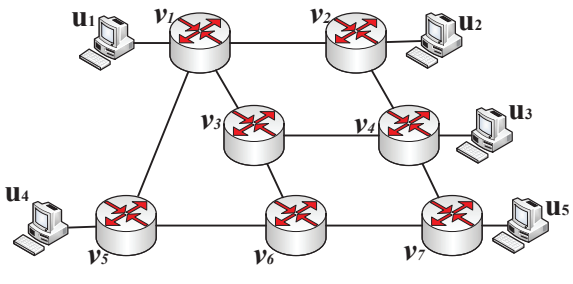


Fig. 7: Topology of the SDN Platform. Our platform is mainly composed of three parts: a controller, seven OpenFlow enabled virtual switches $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, and five terminals $\{u_1, u_2, u_3, u_4, u_5\}$.

In our platform, there are two main categories of devices. One is the SDN controller. We use Ryu (version 4.13 [36]) as the controller software, and choose a server equipped with a Core i7-6700 processor and 8GB of RAM to run this controller software. The other is the virtual switch, which is implemented using the OVS 2.7.2 [37], running on a VMware with 1GB of RAM. Fig. 7 shows the topology of our SDN platform. Seven virtual switches, which are implemented by the OpenFlow v1.3 standard, compose the data plane. Additionally, five terminals are implemented on virtual machines. Three elements, which are source IP, source port and destination IP, identify a flow together. With the identified flows, each terminal can produce different quantities of flows to others.

2) Testing results: We first measure the CPU utilization to collect statistics from the OVS as we change the quantity of covered flows. In this experiment, we generate 60,000 flows through this switch. To acquire measurement results in the idle state, which means no other load on each switch, we configure the switch with the intention that its flow entries will not be expired. After all terminals finish forwarding packets, the controller waits for 10s to make sure that each switch is idle. Then the controller pulls the traffic statistics from the

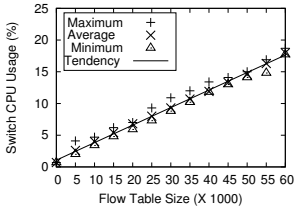


Fig. 8: Switch's CPU Utilization vs. Number of Covered Flows in the Idle Status

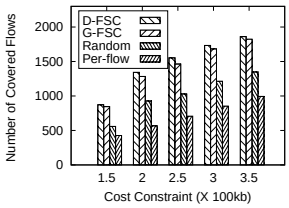


Fig. 9: Number of Covered Flows vs. FSC Traffic Cost Constraint

switch using the different wildcard rules. From the results in Fig. 8, we observe that the CPU utilization is almost linearly increasing with the number of covered flows (or entries) on the switch. Then, we fit these testing data to a straight line as $y = 0.0002762 \cdot x + 1.0692$ using the linear least square method, where y and x denote the CPU utilization and the number of covered flows, respectively. Specifically, when we collect the statistics information of 30,000 and 60,000 flows from the OVS, the CPU utilization is about 9.3% and 18.0%, respectively. Using this fitting function, the CPU utilization for 30,000 and 60,000 flows are 9.4% and 17.6%, respectively. The difference between the estimated utilization and the real utilization is not more than 3%. We should note that two constant parameters c_1 and c_2 will change with the hardware configuration for OVS.

In the second experiment, when 2000 flows are generated in the network, we observe the number of covered flows by changing the FSC traffic amount constraints for different FSC algorithms. In the experiment, each source-destination pair generates the identical number of flows (*i.e.*, 100 flows), and the controller chooses a path randomly for each flow. Since the switch's CPU utilization is difficult to be controlled, it may lead to inaccurate testing results under busy status. Thus, our experiments are for the switch's idle state. Fig. 9 shows that all algorithms can collect statistics information of more flows with the increase of FSC traffic amount constraints. Moreover, D-FSC and G-FSC perform better than the per-flow and random approaches. For example, given an FSC traffic constraint of 200kb, four algorithms can cover 567, 927, 1283, and 1343 flows, respectively. That is to say, D-FSC and G-FSC can improve the number of covered flows by about 44.9% and 38.4% compared with the random method, respectively.

V. Related Works

In an SDN, using the counter field of the flow entry, the switches can count the traffic of each flow. Then, the controller should know the results of the traffic measurement [38]. There are two different ways, specified in OpenFlow [10], to perform flow statistics collection. One is push-based, the other is pull-based.

We first introduce the push-based flow statistics collection. When a new flow arrives at a switch and the flow entry is expired, the switch will send the Packet_in and FlowRemoved messages to the controller, respectively. Using these two kinds of messages, FlowSense [11] implemented the push-based collection. Devoflow [12] implemented a new push-based statistics collection through extending OpenFlow. It can

identify the elephant flows, and then forward these flows through a new route path. It should be noted that the push-based collection needed additional hardware on switches, or modified the packet head (such as sFlow [39]) to support this function. In fact, since most commodity switches do not completely support these requirements, it limits the application of push-based collection.

Then we introduce the pull-based FSC. It is uncomplicated and many SDN applications support it. There are three different schemes for FSC. The first one is the per-flow FSC. Through simple logic, OpenTM [17] queried the flow table counters using traffic matrix estimation. The authors [15] devised an adaptive fetching scheme. In this scheme, the data was pulled from switches to the controllers, and the rate of queries changed according to the flow rates. PayLess [40] traded off the accuracy and network overhead for FSC. To collect flow statistics, it designed a flexible RESTful API, which could be used at different aggregation levels. The second one is the per-switch FSC. Both FlowCover [13] and CeMon [16] presented per-switch monitoring schemes with low cost. Various network management tasks were supported in these per-switch monitoring schemes. The statistics information of all the flows were collected by the controller, when FSC was triggered. The authors [8] have shown that both the per-flow and per-switch FSC schemes caused the serious cost of each switch, and prevented the packet forwarding. Unfortunately, some applications, like flow re-routing [12], demanded that the FSC should be performed frequently enough when using the pull-based scheme. The third one is the wildcard-based FSC. Xu *et al.* [8] proposed a cost-optimized FSC mechanism, which supported wildcard-based requests [41], and presented a rounding-based algorithm for this problem. However, this work does not take the switch's computing resource constraint into accounts, and may result in higher switch cost, which will seriously interfere with the switch's basic functions.

VI. Conclusion

In this paper, we studied how to perform FSC with less interference with the switch's basic functions. We proposed the CP-FSC problem. Then, to solve this problem, two approximation algorithms were designed, and are implemented on our SDN platform. The extensive simulation results show the high efficiency of our proposed algorithms. In the future, we will study how to determine the feasible value of cost constraint for more design choices of efficient FSC.

Acknowledgment

Hongli Xu and Gongming Zhao are corresponding authors. This paper is supported by the NSFC under Grant No. 61472383, U1301256, and 61472385, and the Natural Science Foundation of Jiangsu Province in China under No. BK20161257, and the Fundamental Research Funds for the Central Universities under No. WK5290000001. The work of Qian is supported by the NSF under grant CNS-1701681.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *34th ICDCS*. IEEE, 2014, pp. 228–237.

[8] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of sdn using wildcard requests," in *IEEE INFOCOM*, 2017, pp. 1–9.

[9] 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.

[10] B. Pfaff *et al.*, "Openflow switch specification v1.3.0," 2012.

[11] 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.

[12] 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*, vol. 41, no. 4, 2011, pp. 254–265.

[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] 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.

[15] 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.

[16] 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.

[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] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *International Conference on Distributed Computing and Networking*. Springer, 2013, pp. 439–444.

[19] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[20] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *ACM CoNEXT*. ACM, 2014, pp. 403–414.

[21] 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.

[22] "The openflow switch," openflowswitch.org.

[23] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "csamp: A system for network-wide flow monitoring." in *NSDI*, vol. 8, 2008, pp. 233–246.

[24] G. P. Ingargiola and J. F. Korsh, "Reduction algorithm for zero-one single knapsack problems," *Management science*, vol. 20, no. 4-part-i, pp. 460–463, 1973.

[25] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.

[26] C. Chekuri and A. Kumar, "Maximum coverage problem with group budget constraints and applications," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2004, pp. 72–83.

[27] K. Lai and M. Goemans, "The knapsack problem and fully polynomial time approximation schemes (fptas)," *Retrieved November*, vol. 3, p. 2012, 2006.

[28] A. Gupta, "Approximations algorithms," 2005.

[29] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam, "Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane," in *Software Defined Networking (EWSDN), 2012 European Workshop on*. IEEE, 2012, pp. 79–84.

[30] 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. 281–296.

[31] J. Bogle, N. Bhatia, M. Ghobadi, I. Menache, N. Bjørner, A. Valadarsky, and M. Schapira, "Teavar: striking the right utilization-availability balance in wan traffic engineering," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2019, pp. 29–43.

[32] "The network topology from the monash university," 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," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[34] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE, 2015, pp. 239–250.

[35] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "Flooddefender: protecting data and control plane resources under sdn-aimed dos attacks," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[36] S. Ryu, "Framework community: Ryu sdn controller," 2016.

[37] "Open vswitch," http://openvswitch.org/.

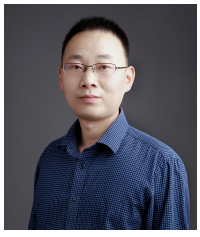
[38] S. Bera, S. Misra, and A. Jamalipour, "Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530–539, 2019.

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

[40] 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.

[41] S. Shirali-Shahreza and Y. Ganjali, "Rewiflow: Restricted wildcard openflow rules," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 29–35, 2015.

Xuwei Yang received B.S. degree in network engineering from the Changan University in 2016. He is currently a doctor-candidate student in Computer Science at the University of Science and Technology of China. He will receive the doctor degree in 2021. His main research interest is software defined networks, network function virtualization and data center network.



Hongli Xu received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph.D. degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC), China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, INFOCOM and ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing and Internet of Thing.

Xiwen Yu received the B.S. degree in information security from the University of Science and Technology of China in 2015, and the M.S. degree in computer science from the University of Science and Technology of China in 2018. His main research interest is software-defined networks.

Chen Qian 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. His research interests include computer networking, network security, and the Internet of Things. He has authored more than 60 research articles in highly competitive conferences and journals. He is a member of the ACM.

Gongming Zhao received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is now an associate professor in University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.

He Huang is an associate professor in the School of Computer Science and Technology at Soochow University, P.R. China. He received his Ph.D. degree in Department of Computer Science and Technology from University of Science and Technology of China (USTC), in 2011. His current research interests include traffic measurement, spectrum auction, privacy preserving in auction, and algorithmic game theory. He is a Member of both IEEE and ACM.