Lightweight Flow Distribution for Collaborative Traffic Measurement in Software Defined Networks

Hongli Xu[1,2] Shigang Chen[3] Qianpiao Ma[1,2] Liusheng Huang[1,2]
Email: xuhongli@ustc.edu.cn, sgchen@cise.ufl.edu, maqiu@mail.ustc.edu.cn, lshuang@ustc.edu.cn
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Suzhou Institute for Advanced Study, University of Science and Technology of China, China
[3]Department of Computer & Information of Science & Engineering, University of Florida, USA

***Abstract*—Many important functions in software defined networks can benefit from fine-grained traffic measurement at flow level. Because TCAM-based flow entries only provide aggregate traffic statistics, prior research has suggested to perform flow-level measurement in SRAM and balance the measurement load across the network through collaborative traffic measurement. The key problem of collaborative measurement is to provide a mechanism to distribute flows to switches such that each switch can identify its subset of flows to measure. We observe that the prior work has focused on optimizing flow distribution among switches, but overlooked their high space and per-packet processing overhead introduced to the data plane, which becomes a serious issue in large SDN systems. In this paper, we propose a new lightweight solution to the flow distribution problem. It follows the design principle of alleviating complexity of the data plane by minimizing the data-plane space and processing overhead. At the control plane, we formulate flow distribution as optimization problems under two scenarios that implement collaborative measurement by edge switches only and by edge/core switches together, respectively. Our extensive simulations demonstrate that, comparing with the best existing work, the proposed lightweight solution achieves a comparable performance in terms of load balancing, while drastically reducing both space overhead and per-packet processing overhead, making it more practical in real-world systems that are sensitive to the additional overhead introduced by flow distribution.**

***Index Terms*—Flow Distribution, Collaborative Traffic Measurement, Software Defined Networks, Overhead.**

I. Introduction

Software defined networking (SDN) makes network management flexible and improves network resource utilization compared to traditional non-SDN networks [1] [2]. Many essential network functions rely on fine-grained traffic measurement at flow level to take full advantage what SDN can potentially offer in load balancing [3], attack/anomoly detection [4] [5], and traffic engineering [6]. For a few examples, data centers implement dynamic flow scheduling using flow-level traffic statistics [7]; SDN path selection uses traffic measurement to find large flows for re-routing [8] [9]; flow-level traffic statistics can help in detecting network intrusions and identifying the attackers [10]; DDoS detection can benefit from analyzing changes or entropy in flow traffic [5]; for stealthy attacks, fluctuations in flow distribution including the small flows are also important clues [11].

Although the TCAM-based flow table in OpenFlow [12] contains statistic counters for traffic measurement, the table size is very limited, *e.g.*, 1,500 entries on HP 5406zl switches [7] and 4,000 entries on Broadcom Trident switches [13], due to TCAM's cost and power consumption. To accommodate a large number of flows (*e.g.*, 10^6 flows in a moderate-sized data center [14]), wildcard rules are routinely used in the flow table, each of them matching many individual flows. Consequently, the counters only provide aggregate statistics of all matching flows, instead of individual flows. To circumvent this problem, researchers propose to perform flow-level traffic measurement in SRAM [15].

However, SRAM is also limited on commodity SDN switches. Even for the high-end Trinder2 switch with a link rate of 960GB per second, its SRAM size is only 16MB, which is shared by all online network functions for routing, management, performance and security purposes [16]. For example, a forwarding information database (FIB) alone takes 10MB in practical use [17]. Hence, the SRAM memory available for traffic measurement will be small. Moreover, to keep up with the line rate, the per-packet processing overhead for traffic measurement must also be minimized in order to avoid introducing a throughput bottleneck.

One promising approach to relieve the space and processing overhead constraints is called *collaborative traffic measurement*, with the observation that each flow may traverse multiple switches on its routing path and its measurement can be performed by any one of them. In fact, we do not have to require each switch to measure all its flows because otherwise flows with multiple hops will be unnecessarily measured multiple times. By assigning each flow to a single switch for measurement, every switch only measures a subset of passing flows, which reduces not only memory requirement but also processing overhead. The key problem for collaborative traffic measurement is called *flow distribution* that is to distribute all flows among

the switches so that each flow is measured only once and a certain global performance criteria is optimized, *e.g.*, a possible load balance goal is to minimize the maximum number of flows measured by any switch.

Any solution to the problem of flow distribution consists of a control-plane component and a data-plane component. The former runs by the SDN controller to decide how to distribute flows to switches, while the latter runs by each individual switch to decide, on a per-packet basis, whether packets should be recorded locally. The switch may record the selected packets in a compact data structure called sketches as does in [7] [9]. There is a vast literature on sketches for traffic measurement, which is able to provide per-flow statistics in tight memory averaging 1 bit per flow or less [18] [19] [20] [21]. We stress that traffic measurement itself is not the subject of this paper. *Our focus is on the problem of flow distribution*.

The prior work [22] on flow distribution emphasizes on balancing the number of flows measured by different switches across the network, while lacking adequate consideration on *the additional space overhead and per-packet processing overhead introduced by the function of flow distribution to the data plane*, which we believe is critical to the viability of collaborative traffic measurement. cSamp [22] requires every switch v to maintain an auxiliary table to support flow distribution, where there is one entry for each pair of ingress/egress routers (or edge switches) as long as v sits on a routing path between the two. In a data center with a Fat-tree topology, a top-level core switch sits between most edge-switch pairs. Hence, its table size will be $O(n^2)$ where n is the number of edge switches. That means millions of table entries if n is in thousands. In addition, the lookup of this table causes significant extra per-packet processing overhead. A more recent work called DCM [23] requires each switch to store two Bloom filters (produced by the controller), with the first one encoding the set of flows to be measured locally and the second one helping remove false positives from the first filter. The memory overhead is significant because it takes a Bloom filter 10 bits per flow on average to ensure a false positive ratio less than 1%, considering that sketch-based traffic measurement itself may take 1 bit per flow on average [18] [19]. More importantly, the lookup of each Bloom filter takes k hash operations and k memory accesses, where k is 7 for an optimal Bloom filter with 1% false positive ratio. This is again much larger than the overhead of sketch-based traffic measurement [20] [21], for which the function of flow distribution serves as a pre-processing step (determining which packets to record). Therefore, lightweight flow distribution that minimizes its own space/processing overhead remains an open problem in collaborative traffic measurement.

Following the general principle of alleviating complexity of the data plane in SDN design, we explore a new approach for lightweight flow distribution that minimizes the memory/space overhead on each switch to a single sampling probability value p. The processing overhead is at most one hash operation per packet to implement sampling if the packet is not recorded by one of the preceding switches on the routing path — if the packet is recorded earlier, the hash operation will not be performed. The controller will decide the optimal sampling probabilities of all switches in two implementation scenarios. In the first scenario, we push the task of traffic measurement away from core switches to edge switches. By distributing flows among all ingress/egress switches, we optimally balance the load of traffic measurement among them. In the second scenario, we consider collaborative traffic measurement among all (core and edge) switches as the prior art [22] [23] does. While a single sampling probability per switch is a limiting factor in our flow distribution formulation, we propose an iterative optimization approach to overcome the constraints imposed by the traffic statistics available from the switches. We compare our lightweight solution to the existing works [22] [23] and demonstrate that it achieves a slightly worse but comparable performance in terms of load balancing, while drastically reducing space overhead and per-packet processing overhead, which makes our new solution more practical in real-world systems that are sensitive to the additional space/processing overhead introduced by flow distribution to the data plane.

The rest of this paper is organized as follows. Section II introduces the network/flow models and problem statement. Section III presents the lightweight flow distribution framework. We design efficient probability assignment for edge switches in Section IV. Probability assignment among edge/core switches is studied in Section V. Simulation results are reported in Section VI. We conclude the paper in Section VII.

II. Problem Statement

A typical SDN consists of a logically-centralized controller and a set of switches, $\mathcal{V} = \{v_1, ..., v_n\}$, with $n = |\mathcal{V}|$. The controller is responsible for managing the whole network, including route selection/update and flow distribution for collaborative traffic measurement. The switches, comprising the data plane, are responsible for packet forwarding and actual traffic measurement. For convenience, $\mathcal{V}^i$, $\mathcal{V}^e$, and $\mathcal{V}^d$ denote the sets of ingress switches, egress switches, and edge switches, respectively.

Network traffic is modeled as flows. Each flow is composed of packets that share a common flow identifier, consisting of several selected fields from the packet header. A typical flow identifier is the 5-element tuple, containing the fields of source address, source port, protocol, destination address and destination port.

The problem of flow distribution is for the controller to decide how to distribute flows to switches for measurement and for the switches to perform the traffic measurement, such that each flow is measured by one and only one switch and the measurement load is balanced among switches. In particular, we want to minimize the maximum number of

flows that are measured by any switch. A key desirable property for a flow distribution solution is that it introduces minimum space and processing overhead to the switches.

The controller makes its decision on flow distribution and download such a decision (in a form that differs in each prior solution [22] [23] and ours) to switches periodically. The controller has full knowledge about the routing structure and how flows will be forwarded in the network [3] [24]. We also assume that the following information about recent network traffic is available to the controller [25] [26] [27]: the number $\mathcal{S}_j^i$ of flows from any ingress switch v_i to any egress switch v_j, which can be measured at the ingress switches by counting the first packets (*e.g.*, SYN packets) of its flows. Note that any edge switch in a data center serves both as an ingress switch and as an egress switch.

III. Lightweight Flow Distribution for Collaborative Traffic Measurement

As a typical solution of flow distribution, cSamp [22] maintains a hash range table for ingress-egress switch pairs, thus requiring a massive SRAM resource overhead. Moreover, DCM needs to maintain two bloom filters, whose memory sizes depend on the number of flows passing through this switch. To be more scalable, each switch v_i will be assigned a sampling probability, denoted as p_i, in our framework. To support the efficient flow distribution, adapting the traditional packet processing requires some extra lightweight operations.

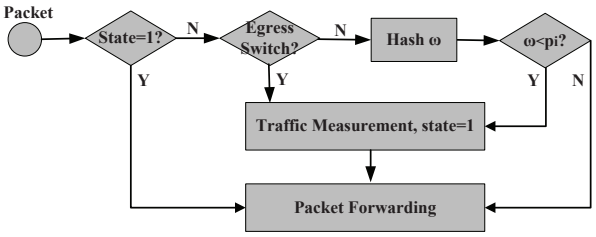

Fig. 1: Illustration of real-time packet processing for flow distribution

We maintain a state bit on each packet head, initially as 0, which indicates that this packet has not been measured. The real-time packet processing framework is illustrated in Fig. 1. Specifically, on arriving at switch v_i, there are two cases:

1) If this packet has not been measured, or its state bit is 0, there are two sub-cases.
   a) If switch v_i is the egress switch of this packet, it will measure this packet.
   b) Otherwise, the switch will generate a hash value ω, with $0 \leq \omega < 1$, using its 5-element tuple. If ω is less than its assigned probability p_i, switch v_i will measure this packet and modify its state to 1.
2) If the state bit is 1, it is no need to measure this packet.

Under our flow distribution framework, each flow will be distributed to a switch along its routing path. Note that it will not significantly increase the overhead by adding some state information to the packet header, which has been widely used in various applications, such as middlebox routing [28] and segment routing [29].

By Fig. 1, sampling probability plays an important role for measurement load. In the following, we will assign the optimal sampling probabilities for all switches in two different scenarios. First, traffic measurement is performed only on ingress/egress switches (*Section IV*). Second, we consider collaborative traffic measurement among all (core and edge) switches (*Section V*).

IV. Traffic Measurement on Ingress/Egress Switches

In many practical topologies, *e.g.*, VL2 [30] or HyperX [31], (almost) all switches will act as ingress and/or egress switches. Thus, this section considers how to efficiently distribute flows only to ingress and egress switches through sampling probability assignment.

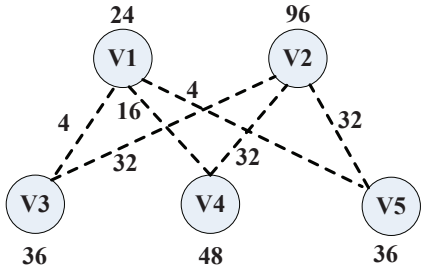

Fig. 2: A Network Example of Flow Distribution

TABLE I: Comparison of Measurement Load on Switches

Solutions	v_1	v_2	v_3	v_4	v_5	Max.
IO	24	96	0	0	0	96
EO	0	0	36	48	36	48
I-0.5	12	48	18	24	18	48
Ours	24	24	24	24	24	24

A. Three Baseline Solutions and Observations

To motivate our design, we first introduce three baseline solutions, and illustrate them using a network example. As shown in Fig. 2, an SDN network consists of a switch set, including two ingress switches $\{v_1, v_2\}$ and three egress switches $\{v_3, v_4, v_5\}$. The value attached with each switch denotes the number of flows through this switch. The number of flows passing from an ingress switch to an egress switch is associated with each dotted line. We use the number of measured flows as its measurement load on a switch. The comparison of measurement load on switches by different solutions is given in Table I. When traffic is measured on ingress or/and egress switches, according to our proposed flow distribution framework in Fig. 1, we only need to assign sampling probabilities for those ingress switches.

The first solution, called Ingress-Only or IO, is that all traffic (or flows) will be measured by ingress switches. Under this scenario, $p_1 = p_2 = 1$. As a result, the measurement loads of v_1 and v_2 are 24 and 96, respectively.

The second solution, called Egress-Only or EO, means that all traffic will be measured by egress switches. Under this scenario, $p_1 = p_2 = 0$. Then, the measurement loads on three egress switches v_3, v_4, and v_5 are 36, 48, and 36, respectively.

Different from the above two solutions, the following two solutions belong to collaborative traffic measurement, which distributes flows to both ingress and egress switches. For a natural way, each ingress switch is assigned with a sampling probability of 0.5, *i.e.*, $p_1 = p_2 = 0.5$. For convenience, this solution is denoted as I-0.5. Specifically, ingress switch v_2 measures 48 flows, while other 48 flows will be measured by three egress switches. Switch v_4 will measure 24 flows (including 8 flows from v_1 and 16 flows from v_2). Consequently, the maximum measurement load is 48 (on switch v_2).

The fourth solution is that we assign sampling probabilities $p_1 = 1$ and $p_2 = 0.25$ for switches v_1 and v_2, respectively. As a result, the measurement load on each switch is 24. Specifically, all flows arriving at v_1 will be measured by this ingress switch. So, each egress switch just measures the flows from v_2. Then, the measurement load of each egress switch is $32 \cdot (1 - 0.25) = 24$.

From this example, we can make one important conclusion. Efficient flow distribution helps to balance the measurement load among all switches, provided with appropriate sampling probability assignment for switches (*e.g.*, the fourth solution). On the contrary, the maximum measurement load may not be improved without proper probability assignment (*e.g.*, the third solution I-0.5 for this example). Thus, it is of great importance to design efficient sampling probability assignment for switches so as to minimize the maximum measurement load among all switches.

B. Probability Assignment for Ingress Switches (PAIS)

To efficiently distribute flows among ingress and egress switches, we will assign a sampling probability, denoted as p_i, for an ingress switch $v_i \in \mathcal{V}^i$. Let $\mathcal{S}_j^i$ be the number of flows from ingress switch v_i to egress switch v_j from measured traffic matrix [26] [27], which is also known to the controller. We consider the measurement load of edge switch $v_i \in \mathcal{V}^d$ with two cases:

1) On arriving at an ingress switch v_i, each flow will be measured with probability p_i. The number of flows, whose ingress switch is v_i, is denoted as $\mathcal{I}_i$, which can be derived as $\mathcal{I}_i = \sum_{v_j \in \mathcal{V} \setminus \{v_i\}} \mathcal{S}_j^i$. Thus, the number of measured flows on v_i, as an ingress switch, is $p_i \cdot \mathcal{I}_i$.
2) On arriving at an egress switch v_i, this packet/flow will be measured by v_i, if it has not been measured by one of the preceding switches on the routing path. From the view of each ingress switch v_j, the measurement load on an egress switch v_i is $(1-p_j) \cdot \mathcal{S}_i^j$. As a result, the expected total number of measured flows on v_i, as an egress switch, is $\sum_{v_j \in \mathcal{V} \setminus \{v_i\}} (1 - p_j) \cdot \mathcal{S}_i^j$.

Hence, the measurement load on an edge switch $v_i \in \mathcal{V}^d$ is $l_i = p_i \cdot \mathcal{I}_i + \sum_{v_j \in \mathcal{V} \setminus \{v_i\}} (1 - p_j) \cdot \mathcal{S}_i^j$. We formulate the PAIS problem as follows:

$$\min \ \max\{l_i, v_i \in \mathcal{V}^d\}$$
$$S.t. \begin{cases} \mathcal{I}_i = \sum_{v_j \in \mathcal{V} \setminus \{v_i\}} \mathcal{S}_j^i, & \forall v_i \in \mathcal{V}^i \\ l_i = p_i \cdot \mathcal{I}_i + \sum_{v_j \in \mathcal{V} \setminus \{v_i\}} (1 - p_j) \cdot \mathcal{S}_i^j, & \forall v_i \in \mathcal{V}^d \\ 0 \le p_i \le 1, & \forall v_i \in \mathcal{V}^i \end{cases} \tag{1}$$

The first set of equations computes the number of measured flows whose ingress switch is v_i. The second set of equations respects the expected measurement load on each switch. Our objective is to minimize the maximum number of measured flows among all edge switches, that is, $\min \ \max\{l_i, v_i \in \mathcal{V}^d\}$.

Since Eq. (1) is a linear program, we can optimally solve it in polynomial time using the linear program solver, *e.g.*, pulp [32]. The solution for Eq. (1) determines a sampling probability for each ingress switch. To cope with traffic dynamics, we divide the time into fixed-size periods (*e.g.*, 5 min). At the start of each period, the controller updates the sampling probabilities of ingress switches. Since each switch only maintains one probability, its update (or control) overhead is much smaller compared with that for both cSamp [22] and DCM [23].

V. Network-wide Traffic Measurement

In some structured networks, *e.g.*, Fat-tree [33], some core switches act neither ingress switches nor egress switches. Thus, to further balance the measurement load among switches, this section considers a more general version, in which traffic will be measured by any switch along its routing path. We will define the network-wide switch probability assignment (NSPA) problem, and present the algorithm for NSPA.

A. Network-wide Switch Probability Assignment (NSPA)

Similar to PAIS, we consider the measurement load of switch $v_i \in \mathcal{V}$, consisting of two components:

1) Switch v_i is the non-egress switch of this packet. According to Fig. 1, if this packet has been measured (or the state is 1), it will not be measured by switch v_i. Let $\overline{\mathcal{Q}}_j^i$ denote the number of unmeasured flows, whose egress switch is v_j, through switch v_i. The measurement load on switch v_i, as a non-egress switch, is $\sum_{v_j \in \mathcal{V} \setminus \{v_i\}} p_i \cdot \overline{\mathcal{Q}}_j^i$.
2) Otherwise, on arriving at the egress switch v_i, if this packet has not been measured, it will be measured by v_i. The number of all flows, whose egress switch is v_i, is denoted as $\mathcal{E}_i$, which can be derived as $\mathcal{E}_i = \sum_{v_j \in \mathcal{V} \setminus \{v_i\}} \mathcal{S}_i^j$. Let $\overline{\mathcal{Q}}_i^j$ denote the number of unmeasured flows, whose egress switch is v_i,

through switch v_j. As a result, the total number of measured flows on switch v_i, as an egress switch, is $\mathcal{E}_i - \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} p_j \cdot \overline{\mathcal{Q}}_i^j$.

Hence, the number of measured flows (or the measurement load) on switch v_i is expressed as $l_i = \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} p_i \cdot \overline{\mathcal{Q}}_j^i + \mathcal{E}_i - \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} p_j \cdot \overline{\mathcal{Q}}_i^j$. How to derive the value of variable $\overline{\mathcal{Q}}_j^i$ will be discussed in Section V-C. We formulate the NSPA problem as follows:

$$\min \ \max\{l_i, v_i \in \mathcal{V}\}$$

$$S.t. \begin{cases} \mathcal{E}_i = \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} \mathcal{S}_i^j, & \forall v_i \in \mathcal{V} \\ l_i = \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} p_i \cdot \overline{\mathcal{Q}}_j^i + \\ \qquad \mathcal{E}_i - \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} p_j \cdot \overline{\mathcal{Q}}_i^j, & \forall v_i \in \mathcal{V} \\ \mathcal{E}_i - \sum_{v_j \in \mathcal{V} \backslash \{v_i\}} p_j \cdot \overline{\mathcal{Q}}_i^j \geq 0, & \forall v_i \in \mathcal{V} \\ 0 \leq p_i \leq 1, & \forall v_i \in \mathcal{V} \end{cases} \quad (2)$$

The first set of equations computes the number of flows whose egress switch is v_i. The second set of equations respects the expected measurement load on each switch. Since we only estimate the value of variable $\overline{\mathcal{Q}}_i^j$ (or $\overline{\mathcal{Q}}_j^i$), the third set of inequalities guarantees that the number of measured flows on v_i, as an egress switch, should not be negative. The objective of NSPA is to balance the measurement load, or minimize the maximum number of measured flows, among all switches, that is, $\min \ \max\{l_i, v_i \in \mathcal{V}\}$.

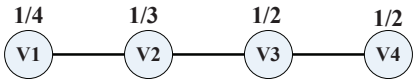

Fig. 3: Illustration of the NSPA problem. The sampling probability of each switch is attached. Assume that 800 flows will be forwarded from the ingress switch v_1 to the egress switch v_4. Under this sampling probability assignment scenario, the measurement load on each switch is 200.

B. An Example of NSPA

We give an example to illustrate the NSPA problem. In Fig. 3, assume that 800 flows will be forwarded from the ingress switch v_1 to the egress switch v_4, and the sampling probability of each switch is attached. 800 flows will arrive at switch v_1. As its sampling probability p_1 is $\frac{1}{4}$, v_1 will measure $800 \times \frac{1}{4} = 200$ flows. Consequently, there are remaining 600 flow unmeasured, arriving at switch v_2. As the sampling probability of switch v_2 is $\frac{1}{3}$, it will measure $600 \times \frac{1}{3} = 200$ flows too. Similarly, 400 unmeasured flows will arrive at switch v_3, which also measures 200 flows, for its sampling probability is $\frac{1}{2}$. All the remaining 200 unmeasured flows will be measured by switch v_4. As a result, the maximum measurement load among these switches is 200.

C. Algorithm Description for NSPA

Since each switch may sit on different routing paths $\mathcal{P}$ of many flows, and the measurement load of a switch is relative with the probabilities of the preceding switches on each path in $\mathcal{P}$, it becomes difficult to explicitly give the closed form of variable $\overline{\mathcal{Q}}_i^j$. Though Eq. (2) is a linear program, we can not optimally solve it. In the following, we propose an iterative optimization approach to assign probabilities for all switches using the traffic statistics available from the switches.

Algorithm 1 Network-Wide Probability Assignment

1: **Step 1: Algorithm Initialization**
2: **for** $v_i \in \mathcal{V}$ **do**
3: **for** $v_j \in \mathcal{V} \backslash \{v_i\}$ **do**
4: $\mathcal{Q}_i^j = \sum_{v_{i'} \in \mathcal{V} \backslash \{v_i\}} \alpha_{i,i'}^j \cdot \mathcal{S}_i^{i'}$
5: $\overline{\mathcal{Q}}_i^j = \mathcal{Q}_i^j$
6: **Step 2: Assigning Sampling Probability**
7: $t = 1$
8: **while** $t \leq m$ **do**
9: Solve Eq. (2), and the solution for sampling probability of switch v_i is denoted as p_i
10: **for** each ingress switch v_k and egress switch v_i **do**
11: Compute the number of flows unmeasured on switch v_j, as $\mathcal{T}_{i,k}^j$, by visiting all paths from v_k to v_i
12: **for** $v_i \in \mathcal{V}$ **do**
13: **for** $v_j \in \mathcal{V} \backslash \{v_i\}$ **do**
14: Update variable $\overline{\mathcal{Q}}_i^j = \sum_{v_k \in \mathcal{V} \backslash \{v_i\}} \mathcal{T}_{i,k}^j$
15: $t = t + 1$

The algorithm consists of two main steps. In the first step, we give an approximate value for each variable $\overline{\mathcal{Q}}_i^j$. For simplicity, we use $\mathcal{Q}_i^j$ to estimate $\overline{\mathcal{Q}}_i^j$, where $\mathcal{Q}_i^j$ denotes the number of flows, with egress switch v_i, passing through v_j. Note that we can derive $\mathcal{Q}_i^j$ by the applied routing strategy, which is known to the controller. For example, if all the flows follow the ECMP (or WCMP [34]) paths between switches, the controller knows the fraction of flows on switch v_j from an ingress switch $v_{i'}$ to an egress switch v_i, denoted as $\alpha_{i,i'}^j$. As a result, $\mathcal{Q}_i^j = \sum_{v_{i'} \in \mathcal{V} \backslash \{v_i\}} \alpha_{i,i'}^j \cdot \mathcal{S}_i^{i'}$.

The second step consists of an iterative procedure. After obtaining the estimation value of each variable $\overline{\mathcal{Q}}_i^j$, we can solve Eq. (2) in polynomial time. The solution for Eq. (2) provides a sampling probability p_i for each switch v_i in an SDN. Then, we update the estimation of each variable $\overline{\mathcal{Q}}_i^j$ using the derived switches' probabilities. Specifically, given the number of passing flows through a path and the sampling probabilities for switches on this path, we are able to obtain the number of flows unmeasured on each switch along this path. For each pair of ingress switch v_k and egress switch v_i, we can determine the number of flows unmeasured on each switch v_j, denoted as $\mathcal{T}_{i,k}^j$, by visiting multi-paths from switches v_k to v_i (Line 11). Based on this, we also obtain the number of unmeasured flows, whose egress switch is v_i, through v_j using the current probability assignment, as the new estimation of variable $\overline{\mathcal{Q}}_i^j$ (Line 14). The algorithm will terminate after a certain number (*e.g.*,

m) of iterations. Our simulation results show that 3 iterations are enough to assign efficient sampling probabilities for switches. The algorithm is formally described in Alg. 1.

D. Discussion

In this paper, flow distribution is studied under the scenario of collaborative traffic measurement to balance the measurement load among all switches. In fact, flow distribution can also be applied in other applications. For example, in the middlebox-based SDN network [28] [35], if a middlebox processes all passing flows, it may be overloaded and increase the processing delay. Thus, the controller usually expects to (1) handle all flows by the middleboxes; and (2) balance the processing load among these middleboxes. We can efficiently distribute each flow to only one middlebox for processing using our proposed flow distribution solutions, so that the processing load balancing among these middleboxes can be achieved.

VI. Performance Evaluation

In this section, we first introduce the metrics and benchmarks for performance comparison (*Section VI-A*). We then evaluate our proposed algorithms by comparing with some benchmarks through extensive simulations (*Section VI-B*).

A. Performance Metrics and Benchmarks

In this paper, we design lightweight flow distribution solutions in an SDN so as to balance the measurement load among different switches with minimum space/processing overhead. Thus, we use the following four metrics in our numerical evaluations: (1) the maximum measurement load among all switches; (2) the maximum space overhead for auxiliary information among all switches to support flow distribution; (3) the average processing overhead per-packet in flow distribution; (4) the running time. After sampling probability assignment, we can determine the number of flows measured by each switch, and use their maximum one as the first metric. For the second metric, we measure the space overhead for auxiliary information to distribute flows by different algorithms. For example, our proposed NSPA and PAIS solutions only maintain a sampling probability for each switch. Different solutions may perform different operations, *e.g.*, hash operation, memory access, packet-head reading/writing, *etc.*, for flow distribution. Note that memory access refers to operations (*e.g.*, lookup and update) on a memory list, *e.g.*, a table of hash ranges and a bloom filter. It is trivial to consider all operations together, for different operations lead to various processing overheads. By testing on our platform with CPU 3.7GHz and OVS version 2.5.3 [36], we find that average processing overheads (*e.g.*, about 10-30 CPU cycles) of hash and memory access are usually more than that (*e.g.*, about 1-5 CPU cycles) of packet-head reading/writing. Thus, we analyze the average number of hash operations and memory accesses per-packet in flow distribution. Since our solutions only require a sampling probability on each switch, the controller just updates a probability for each switch periodically. Its control overhead is extremely low, and we do not compare the control overhead of different flow distribution solutions.

To evaluate the proposed flow distribution solutions, we compare them with the most-related, state-of-the-art works through extensive simulations. For traffic measurement among edge/core switches, we choose cSamp [22] and DCM [23], introduced in Section I, as two benchmarks. Specifically, we set false positive ratio as 1% in DCM. For traffic measurement among ingress and egress switches, we compare PAIS with three benchmarks, IO, EO, and I-0.5, respectively. These solutions have been explained in Section IV-A.

TABLE II: Illustration of Memory Overhead from OVS [36]

Item	Size	Item	Size
Destination IP	32b	Hash Range	64b
Switch IP	32b	-	-

B. Simulation Evaluation

1) Simulation Settings: In the simulations, as running examples, we select two typical and practical topologies for data center networks. The first topology, called VL2 [30], contains 240 switches (including 200 edge switches, 20 aggregation switches, and 20 core switches) and 1000 terminals. In this topology, most switches (or 200 out of 240 switches) are ingress/egress switches. The second one is the Fat-tree topology [33], which has been widely used in many data center networks. The Fat-tree topology has in total 320 switches (including 128 edge switches, 128 aggregation switches, and 64 core switches) and 1024 terminals. Only those 128 edge switches (or about 40% of all switches) are ingress/egress switches. By default, there contains 1M flows on both topologies. For the flow size, the authors of [7] have shown that less than 20% of the top-ranked flows may occupy more than 80% of the total traffic. Thus, we allocate the size for each flow according to this distribution. Moreover, each flow contains 200 packets on average. In the simulations, we adopt ECMP for flow routing. We execute each simulation 100 times, and take the average of the numerical results.

To make the comparison of space overhead more reasonable, we give the space/memory overhead of each item in Table II. Specifically, we adopt the memory overhead from the OVS implementation [36]. For example, the memory overhead for ingress switch IP is 32 bits, while that for a hash range (including two float numbers) is 64 bits. By Table II, the space overhead for each ingress-egress switch entry in cSamp includes ingress switch IP (32 bits), egress switch IP (32 bits) and its hash range (64 bits). Thus, the space overhead for each ingress-egress switch pair is 128 bits (or 16 Bytes). Though the space overhead

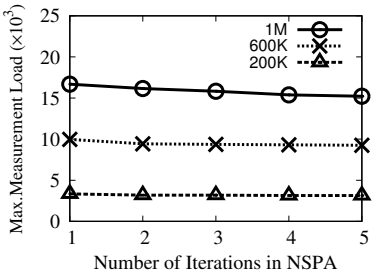

Fig. 4: Maximum Measurement Load vs. Number of Iterations on VL2

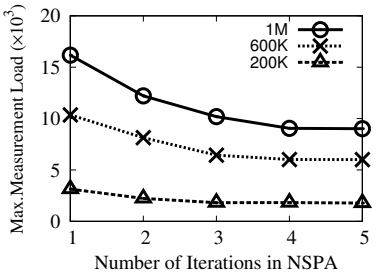

Fig. 5: Maximum Measurement Load vs. Number of Iterations on Fat-tree

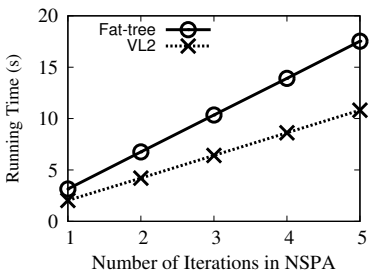

Fig. 6: Running Times vs. Number of Iterations in NSPA

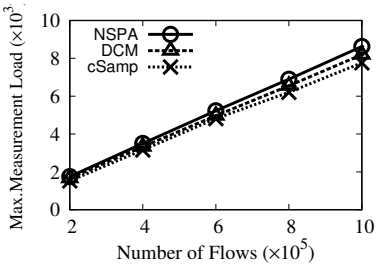

Fig. 7: Maximum Measurement Load among Edge/Core Switches on VL2

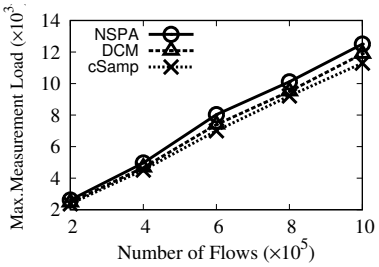

Fig. 8: Maximum Measurement Load among Edge/Core Switches on Fat-tree

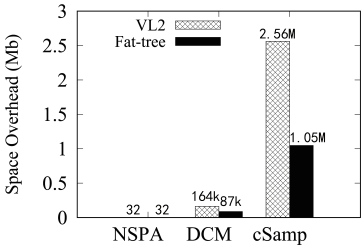

Fig. 9: Space Overhead for Auxiliary Information in Flow Distribution

can be reduced using encoding, it increases the per-packet processing overhead.

2) Simulation Results: We run four groups of simulations to check the effectiveness of the proposed flow distribution algorithms through probability assignment.

The first set of three simulations observes the performance (*e.g.*, maximum measurement load and running time) of our NSPA algorithm. Figs. 4 and 5 show the maximum measurement load by changing the number of iterations on VL2 and Fat-tree, respectively. With more iterations, the algorithm can decrease the maximum measurement load in the network through efficient sampling probability assignment. However, the decreasing ratio is much slower with more iterations in NSPA. Especially for VL2, we find that the NSPA algorithm just requires one iteration to achieve the measurement load balancing. For the Fat-tree topology, three iterations are enough to obtain a good probability assignment for measurement load balancing. On the other hand, with more iterations, the running time of the NSPA algorithm is linearly increasing, as shown in Fig. 6. The running time on Fat-tree is more than that on VL2, since the Fat-tree topology contains more switches than VL2. To achieve a better trade-off between measurement load balancing and running time, the NSPA algorithm runs three iterations for both topologies in the following simulations.

The second set of five simulations observes the performance of flow distribution among core/edge switches. Figs. 7 and 8 show that the maximum measurement load is almost linearly increasing with more flows from 200K to 1M on both two topologies. Two figures show that the gap of maximum measurement load among three algorithms (*i.e.*, cSamp, DCM and NSPA) is not more than 10%. In other words, our proposed NSPA algorithm can achieve a slightly ($< 5\%$ on average) worse performance in terms of measurement load balancing compared with cSamp and DCM. However, these two methods lead to high memory/space overhead and per-packet processing overhead for flow distribution. Fig. 9 shows that cSamp requires a massive space overhead for auxiliary information, while our NSPA algorithm only requires 32 bits to record a sampling probability on each switch. Specifically, cSamp, DCM and NSPA require the space overheads of 2.56Mb, 0.48Mb, and 32b, respectively, on VL2. NSPA reduces the space overhead almost 100% compared with other two solutions. Figs. 10 and 11 show the per-packet processing overheads of different algorithms on VL2 and Fat-tree, respectively. Per-packet processing overhead mainly includes hash operations and memory accesses. By Fig. 10, we observe that cSamp and DCM require the numbers of per-packet hash operations 2.3 and 2.8 times as NSPA on VL2. Similarly, the numbers of per-packet hash operations of cSamp and DCM are 2.2 and 2.7 times as that of NSPA on Fat-tree. Moreover, cSamp and DCM require 2.7 and 7 memory accesses per-packet while our proposed NSPA algorithm needs no memory access. By Figs. 9-11, NSPA is a lightweight flow distribution solution compared with both cSamp and DCM .

The third set of two simulations observes the maximum measurement load among ingress and egress switches for traffic measurement by changing the number of flows from 200K to 1M. Fig. 12 shows that the maximum measurement load is almost linearly increasing with more flows in the network for all four algorithms. The increasing ratio of both I-0.5 and PAIS is slower than that of IO and EO, which shows efficiency of collaborative traffic measurement on

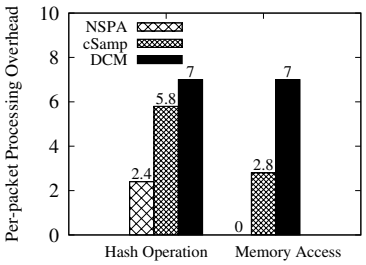

Fig. 10: Per-packet Processing Overhead on VL2

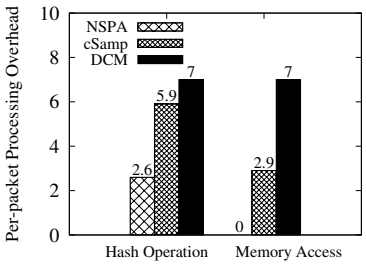

Fig. 11: Per-packet Processing Overhead on Fat-tree

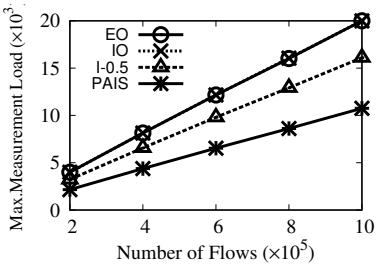

Fig. 12: Maximum Measurement Load of Edge Switches on VL2

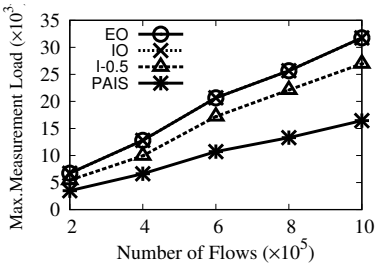

Fig. 13: Maximum Measurement Load of Edge Switches on Fat-tree

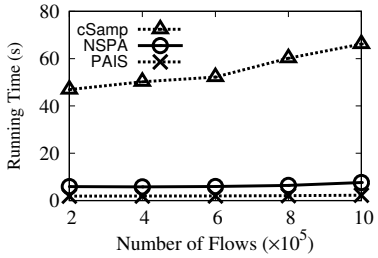

Fig. 14: Running Time vs. Number of Flows on VL2

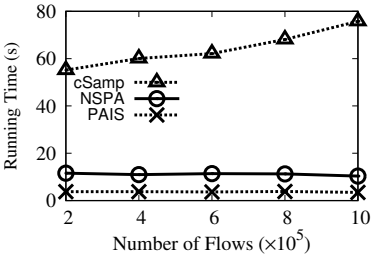

Fig. 15: Running Time vs. Number of Flows on Fat-tree

both two topologies. Meanwhile, PAIS significantly reduces the maximum measurement load compared with I-0.5, which shows high efficiency of our proposed probability assignment solution. For example, when there are 800K flows in the VL2 topology, the maximum measurement loads of IO, EO, I-0.5, and PAIS are 16.0K, 16.0K, 12.9K, and 8.6K, respectively, by Fig. 12. While in the Fat-tree topology, the maximum measurement loads of IO, EO, I-0.5, and PAIS are 25.7K, 25.7K, 22.1K, and 13.3K, respectively. In other words, PAIS reduces the maximum measurement load by 47.2%, 47.2%, and 36.6% compared with IO, EO, and I-0.5, respectively. By analyzing these algorithms, since IO (or EO) deterministically measures flows on ingress (or egress) switches, hash operation and memory access can be avoided. Moreover, both I-0.5 and PAIS require only one hash operation per-packet for flow distribution. Thus, the additional overhead of these algorithms is extremely low.

By Figs. 7-8 and 12-13, we observe that NSPA can achieve better measurement load balancing than PAIS. That is because NSPA performs traffic measurement on edge/core switches, while PAIS performs traffic measurement only on edge switches. In VL2, the gap of maximum measurement load between NSPA and PAIS is about 15-20%, for 200 out of 240 switches will measure traffic. Thus, PAIS can perform well in those topologies, in which most switches are edge switches like VL2.

The fourth set of two simulations observes the running time of probability (or range) assignment algorithms, including cSamp, PAIS and NSPA. Figs. 14 and 15 show that the running time of cSamp is much more than that of both PAIS and NSPA. That's because cSamp needs to solve a linear program with much more variables for determining the hash range of each edge switch pair, while both PAIS and NSPA solve the linear program only with n variables, where n is the number of switches in an SDN, for determining switches' probabilities. Specifically, the running times of both PAIS and NSPA are only about 1/20-1/6 times as that of cSamp, which shows high scalability of our algorithms for flow distribution.

From the simulation results in Figs. 4-15, we can make the following three conclusions. First, by Figs. 7-11, our NSPA solution can achieve a slightly worse but comparable performance ($< 5\%$ on average) in terms of measurement load balancing, while drastically reducing space overhead and per-packet processing overhead, which makes our solution more practical in applications. Second, by 12-13, efficient probability assignment for ingress switches helps to balance the measurement load among all edge switches. For example, PAIS reduces the maximum measurement load by 47.2%, 47.2%, and 36.6% compared with IO, EO, and I-0.5, respectively. Third, Figs. 14 and 15 show that both PAIS and NSPA can achieve lower time complexity compared to cSamp.

VII. Conclusion

Flow distribution is a key problem for collaborative traffic measurement to balance the measurement load. We have proposed a lightweight flow distribution framework, and designed several solutions for switch sampling probability assignment. Our evaluations have demonstrated that the proposed solutions can achieve much lower space overhead and per-packet processing overhead, while achieving almost measurement load balancing, compared with the best existing works. In the future, we will further study the impact of traffic dynamics on measurement load, and implement the

proposed solutions on the SDN platform. Moreover, some network metrics, such as end-to-end delay and throughput, can only be measured by involving multiple switches. Thus, we will also study a more general problem, in which each flow will be measured by a given number (*e.g.*, 1 or 2) of switches, for measurement load balance.

ACKNOWLEDGEMENT

This search of Xu, Ma and Huang is supported by the National Natural Science Foundation of China (NSFC) under Grant 61822210, Grant U1709217, Grant 61472383, Grant 61728207, and Grant 61472385; NSF of Jiangsu in China under No. BK20161257; by Anhui Initiative in Quantum Information Technologies under No. AHY150300; and also by Guangdong Province Key Laboratory of Popular High Performance Computers 2017B030314073. The research of Chen is supported by NSF under No. NSF CNS-1719222.

REFERENCES

[1] 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 Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 15–26.

[2] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, "Incremental deployment and throughput maximization routing for a hybrid sdn," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1861–1875, 2017.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[4] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

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

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

[7] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[8] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *Proceedings of the 10th CoNEXT*. ACM, 2014, pp. 403–414.

[9] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[10] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "Sdnips: Enabling software-defined networking based intrusion prevention system in clouds," in *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE, 2014, pp. 308–311.

[11] 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.

[12] O. S. Specification-version, "1.5.0," 2015.

[13] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 49–60.

[14] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[15] T. Li, S. Chen, and Y. Ling, "Fast and compact per-flow traffic measurement through randomized counter sharing," *Proc. of IEEE INFOCOM*, pp. 1799–1807, April 2011.

[16] "Resource monitoring usage computation overview," https://www.juniper.net/documentation/en_US/junos/topics/concept/resource-monitoring-usage-calculation.html.

[17] J. Scudder, "Routing/addressing problem solution space," in *2009-07-281. http..//www.arin.net/meetings/minutes/ARIN XX/PDF/wednesday/SolutionSpace_ Scudder. pdf*, 2007.

[18] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[19] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. ACM, 2015, pp. 417–428.

[20] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: Improving the performance of cardinality estimation of large data streams," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[21] M. Chen, S. Chen, and Z. Cai, "Counter tree: A scalable counter architecture for per-flow traffic measurement," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 2, pp. 1249–1262, 2017.

[22] 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.

[23] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proceedings of the third workshop on HotSDN*. ACM, 2014, pp. 85–90.

[24] H. Xu, Z. Yu, C. Qian, X.-Y. Li, Z. Liu, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in sdns," *IEEE/ACM ToN*, vol. 25, no. 6, pp. 3587–3601, 2017.

[25] V. Sekar, A. Gupta, M. K. Reiter, and H. Zhang, "Coordinated sampling sans origin-destination identifiers: algorithms and analysis," in *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*. IEEE, 2010, pp. 1–10.

[26] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational ip networks: Methodology and experience," *IEEE/ACM Transactions on Networking (ToN)*, vol. 9, no. 3, pp. 265–280, 2001.

[27] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale ip traffic matrices from link loads," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1. ACM, 2003, pp. 206–217.

[28] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[29] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing," in *Computer Communications (INFOCOM)*. IEEE, 2015, pp. 657–665.

[30] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[31] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 41.

[32] S. Mitchell, M. OSullivan, and I. Dunning, "Pulp: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.

[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] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "Wcmp: Weighted cost multipathing for improved fairness in data centers," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 5.

[35] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags." in *NSDI*, vol. 14, 2014, pp. 543–546.

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