Coordinated Sampling in SDNs with Dynamic Flow Rates

Soroosh Esmaeilian*, Mahdi Dolati†, Sogand Sadrhaghighi*, and Majid Ghaderi*

*Dept. of Computer Science, University of Calgary, Calgary, Canada, †School of Computer Science, IPM, Tehran, Iran.

Abstract—Traffic sampling has become an indispensable tool in network management. While there exists a plethora of sampling systems, they generally assume flow rates are stable and predictable over a sampling period. Consequently, when deployed in networks with dynamic flow rates, some flows may be missed or under-sampled, while others are over-sampled. This paper presents the design and evaluation of dSamp, a network-wide sampling system capable of handling dynamic flow rates in Software-Defined Networks (SDNs). The key idea in dSamp is to consider flow rate fluctuations when deciding on which network switches and at what rate to sample each flow. To this end, we develop a general model for sampling allocation with dynamic flow rates, and then design an efficient approximate integer linear program called APX that can be used to compute sampling allocations even in large-scale networks. To show the efficacy of dSamp for network monitoring, we have implemented APX and several existing solutions in ns-3 and conducted extensive experiments using model-driven simulations. Our results indicate that, by considering dynamic flow rates, APX outperforms the existing solutions by up to 10% in sampling more flows at a given sampling rate.

I. INTRODUCTION

Motivation. Network traffic monitoring is crucial for many network management tasks such as traffic engineering [1], anomaly detection [2], traffic matrix estimation [3] and customer accounting [4]. While aggregate traffic statistics (*e.g.*, average traffic rate) are sufficient for some network management tasks (*e.g.*, traffic engineering), there is a growing need for having detailed flow-level measurements in other tasks, such as traffic classification [5]. To address this need, many network operators employ monitoring solutions based on packet sampling, where network switches sample only a subset of the packets that pass through them [6], [7]. However, sampling is a resource-intensive operation for network switches, given their restricted processing capabilities. A typical switch can only sample a tiny fraction of the packets it encounters before its performance starts to degrade [8].

Sampling can be performed on a per-port or per-flow basis. In per-port sampling solutions, such as NetFlow [9] and sFlow [10], a sampling rate is specified for each switch input port. As a result, flows with higher packet rates are more likely to be sampled compared to flows with lower rates. Consequently, these solutions are inadequate for network management tasks that require a minimum per-flow sampling rate, *e.g.*, anomaly detection [11]. Per-flow sampling addresses this bias toward high-rate flows by specifying a target sampling rate for each *flow*. Several works such as [12] and [13] have proposed extensions to the standard OpenFlow to enable per-flow sampling on commodity OpenFlow switches. Sampling solutions based on per-flow sampling are proposed in [13]

and [14]. However, in these solutions, switches sample flows independently, resulting in redundant flow samples and inefficient use of switch resources.

To prevent redundant sampling and enhance the flow monitoring capabilities of the network, coordinated sampling is considered [6], [15]–[17], where a centralized controller coordinates monitoring responsibilities across different switches in the network. To determine per-flow sampling rates on each switch, these works generally assume that the flow rates are deterministic in the sense that they are 1) fixed over time, and 2) known a priori. In reality, flow rates are variable and fluctuate over time. As a result, these solutions may lead to under-sampling or over-sampling when the actual traffic rate of a flow deviates from the one assumed by the solution. This weakness renders such solutions unsuitable for many network management tasks that require a consistent minimum flow sampling rate to ensure effective and accurate performance.

As a motivating example, consider the small network depicted in Fig. 1. The network consists of two switches denoted by S_1 and S_2, each having a sampling capacity of 3 packets per second (pps). There are four traffic flows denoted by f_1, f_2, f_3 and f_4 that are required to be sampled at the sampling rate of 0.1. The rates of f_1 and f_2 follow a Normal distribution with a mean of 5 pps and a standard deviation of 10 pps, while the rates of f_3 and f_4 follow a Normal distribution with a mean of 14 pps and standard deviation of 1 pps. A deterministic sampling solution that ignores rate fluctuations assigns flows f_1 and f_3 to switch S_1, and flows f_2 and f_4 to switch S_2 for sampling. This assignment ensures equal sampling load distribution (with respect to the mean flow rates) between the switches, which is a common objective in coordinated sampling solutions [15], [17]. Let δ_i denote the probability that the capacity of switch i is violated (*i.e.*, its sampling capacity is not sufficient to sample its flows at the specified target sampling rate). We have:

$$\delta_1 = \delta_2 = \mathbb{P}\left\{0.1 \times \mathcal{N}(14+5, 1+100) > 3\right\} = 0.14\,.$$

Next, consider a sampling solution that considers variability in flow rates, such as the one proposed in this paper. It may decide on the following assignment, where flows f_1 and f_2 are assigned to switch S_1 and f_3 and f_4 are assigned to switch S_2. In this case, we obtain that:

$$\delta_1 = \mathbb{P}\left\{0.1 \times \mathcal{N}(5+5, 100+100) > 3\right\} = 0.08,$$
$$\delta_2 = \mathbb{P}\left\{0.1 \times \mathcal{N}(14+14, 1+1) > 3\right\} = 0.08,$$

which represents almost 43% reduction in the switch capacity violation probability by simply considering fluctuations in
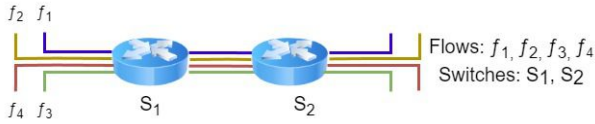
Fig. 1: A small network with two switches and four flows.

flow rates. Any reduction in the switch capacity violation probability translates to more precise flow sampling and, consequently, enhanced visibility of network flows especially for management tasks that require a minimum sampling rate.

Our work. We introduce (dSamp), a network-wide sampling solution capable of handling dynamic flow rates. dSamp is designed to operate with commodity OpenFlow switches and SDN network controllers. The key idea behind dSamp design is to incorporate statistical information about flow rates into the process of making sampling decisions. This is in contrast to existing works that assume knowledge about the *exact rate* of flows. Specifically, dSamp only requires information about the mean and variance of flow rates. As discussed earlier, it is unrealistic to assume that flow rates are fixed and known exactly in advance. However, not only it is straightforward to obtain statistical information about flows, for example, based on historical measurements or service level agreements, but also this information is more robust (*i.e.*, does not fluctuate significantly) compared to the instantaneous flow rates.

Our main contributions in this paper are:

- We present the design and evaluation of dSamp, a network-wide sampling system, capable of handling flow rate fluctuations in SDNs.
- We formulate flow sampling with statistical flow rate information as an Integer Second Order Cone Program (ISOCP). Given the complexity of solving the ISOCP formulation for both small and large-scale networks, we present two reformulations to obtain solutions with practical runtimes. The first reformulation is a transformation and more suitable for small size networks, while the second reformulation is approximate but suitable for larger networks.
- We provide model-driven simulations using ns-3. Our results show that dSamp outperforms existing sampling solutions based on deterministic flow rates.

Organization. Related works are reviewed in Section V. The high-level design of dSamp is presented in Section II. In Section III, we present our algorithms and their analysis. Evaluation results are presented in Section IV. The concluding remarks are presented in Section VI.

II. System Design

The high-level architecture of dSamp is depicted in Fig. 2. The main component of dSamp is the orchestrator, which is implemented on top of the SDN controller. The design of dSamp also includes an optimizer module that works over sampling epochs. The sampling queries that are sent to the sampling service during the current sampling epoch are batched
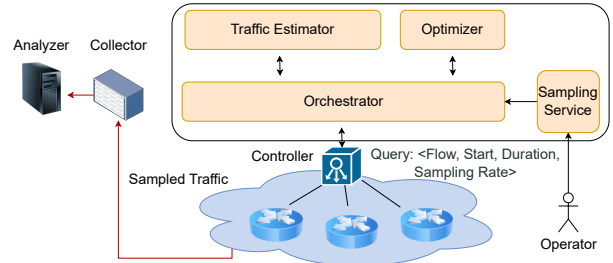

Fig. 2: High-level architecture of dSamp.

together and then queried by the orchestrator. Afterwards, the orchestrator processes them in a single operation at the beginning of the following sampling epoch. The epoch-based structure enables dSamp to adapt its sampling decisions to the dynamic set of flows at the start of each epoch. In our design, a flow is specified by a source-destination IP address pair. Thus, a flow may represent the aggregated traffic between an Origin-Destination (OD) pair in an Internet Service Provider (ISP) network or a pair of Virtual Machines (VMs) in a Data Center (DC) network. The length of the sampling epoch is a design parameter that can be tuned to achieve different trade-offs. In particular, a shorter epoch allows the system to adapt more quickly to changes in network traffic, while a longer epoch helps reduce the control overhead. In the following, we provide a high-level overview of the system's operational workflow, followed by a detailed examination of the individual components within the architecture.

System Workflow. There are two phases in dSamp operation, namely the setup phase and the sampling phase. In the setup phase, the network operator adjusts various system parameters such as the sampling epoch length. Once the setup phase is completed, the sampling phase starts. As stated before, dSamp works over sampling epochs. At the beginning of each epoch, the orchestrator queries the controller to collect statistical data about flows and their respective paths. Additionally, it queries the sampling service to identify the flows to be sampled. It then invokes traffic estimator and optimizer modules to compute a sampling schedule for the selected flows. A *sampling schedule* determines which switch in the network will sample each flow. Finally, the orchestrator communicates with the controller to install the newly computed sampling schedule on the relevant switches. The destination of all sampled traffic is the collector server (essentially a load balancer in front of storage servers), which is linked to analysis applications that carry out the required management task.

Sampling Service. The sampling service receives sampling queries from the users (*e.g.*, network operators and tenants). At the end of each epoch, the orchestrator collects all the sampling queries from the sampling service. Queries are defined by a tuple $< \text{flow}, \text{start}, \text{duration}, \text{sampling rate} >$. The filed "start" denotes the start time of sampling, while "duration" indicates the duration of sampling for the specified flow. The filed "sampling rate" determines the desired sampling rate.

Orchestrator. The orchestrator is the main component of dSamp and includes two modules, namely traffic estimator and optimizer. All communications between dSamp and network switches go through the orchestrator. At the start of each epoch, the orchestrator collects all sampling queries from the sampling service. Additionally, it obtains flow statistics and their paths from the controller for the recently ended epoch. The orchestrator passes this information to the traffic estimator and, in turn, receives the mean and variance for each flow's rate. Using the mean and variance of each flow's rate, as well as their paths, the orchestrator leverages the optimizer module to create a sampling schedule for each flow. This schedule is then installed on network switches via the controller.

Traffic Estimator. To calculate an optimal sampling schedule for flows with non-deterministic rates, dSamp requires the mean and variance estimates of the flow rate. To this extent, various approaches can be utilized, *e.g.*, [18] employs an autoregressive model, while [19] uses machine learning to estimate flow rates. In our design, we applied a simple estimator based on past traffic statistics. Specifically, the traffic estimator queries the controller to collect statistics about flow rates over the past epochs and then estimates the mean and variance of flow rates for the upcoming epoch based on the collected statistics.

Optimizer. The optimizer is responsible for computing a sampling schedule while considering flow rate fluctuations. The goal is to optimize the use of switch sampling resources in order to maximize the number of fully sampled flows, *i.e.*, flows that are sampled at their specified target rate. To achieve this goal, dSamp limits sampling each flow to a single switch along its path. Hence, depending on the available switch capacity, a flow either could be sampled on one switch along its path or will not be sampled at all. This will satisfy two goals: 1) avoiding duplicate sampled packets, and 2) improving flow visibility, as dSamp only samples flows for which there is a high confidence of fully sampling them. This way, dSamp gives higher priority to fully sampling flows (for improved flow visibility) rather than sampling many flows but only partially. This design is advantageous for tasks requiring a specific minimum sampling rate for each individual flow. The primary focus of the following section is the design of efficient algorithms for the optimizer.

Discussion. While the design of dSamp primarily targets ISPs, it could also be altered to be used in DCs. In the DC setting, the sampling query specifies a list of VM pairs belonging to a tenant. Similar to the case of ISP deployment, the system reconfigures switches only at the beginning of each sampling epoch. This strategy is consistent with the original design, minimizing control overhead associated with collecting flow statistics and reconfiguring switches.

III. SAMPLING OPTIMIZATION

Our goal in this section is to design an *efficient* algorithm for computing the network-wide sampling allocations with dynamic flow rates. The algorithm must be efficient in terms of

TABLE I: Table of Notations.

Notation	Description
r_f	Sampling load of flow f
μ_f	Mean sampling load for flow f
σ_f^2	Variance of sampling load for flow f
$x_{f,s}$	Sampling decision for flow f on switch s
B_s	Sampling capacity of switch s
$\mathcal{F}$	Set of flows to be sampled
$\mathcal{S}$	Set of switches in the network
$\mathcal{F}_s$	Set of flows that pass through switch s
$\mathcal{S}_f$	Set of switches on the path of flow f
δ	Probability of violating a switch sampling capacity
α	Pre-specific target sampling rate

monitoring performance as well as computational complexity so it can be applied to large network scenarios. To this end, we show that dynamic flow rates can be incorporated in the problem formulation, but they lead to second-order constraints that are computationally intensive to solve, even for small network scenarios. Consequently, we focus on developing an approximate formulation that can be utilized in large-scale networks, while achieving performance that is close to that of the exact formulation. The important notions used in this section are shown in Table I.

A. Problem Formulation

Let $\mathcal{F}$ denote the set of *flows* to be sampled in the network. We define $\mathcal{S}$ to be the set of all switches in the network and $\mathcal{S}_f$ to be the set of switches on the path of flow $f \in \mathcal{F}$. For notation convenience, we also define $\mathcal{F}_s$ to be a subset of flows in $\mathcal{F}$ that pass through switch $s \in \mathcal{S}$. The network orchestrator requires each flow in $\mathcal{F}$ to be sampled at a target sampling rate (to achieve a network monitoring objective). Consequently, the sampling load of each flow, defined as the rate of sampled traffic transmitted to the collector, is computed from its rate and its target sampling rate. Each switch has a limited capacity to send sampled packets to the collector. We use B_s to denote the transmission capacity of switch $s \in \mathcal{S}$ for sending the sampled packets. We consider a scenario where the flow rates are dynamic and fluctuate over time. Thus, we use the random variable r_f to show the sampling load of flow f. This random variable models the fluctuation of the flow's rate, and any realization of r_f is valid.

To formulate the sampling allocation problem, we define binary decision variable $x_{f,s}$, which is set to one if flow f is sampled on switch s, and to zero otherwise. To avoid duplications, where a packet is sampled redundantly on multiple switches, we require that each flow be sampled either on one switch or nowhere at all if the available sampling capacity is not sufficient to handle the resulting sampling load. Consequently, we introduce the following constraint to ensure that each flow is sampled by at most one switch in the network:

$$\sum_{s \in \mathcal{S}_f} x_{f,s} \leq 1, \quad \forall f \in \mathcal{F}. \tag{1}$$

Given the stochastic nature of the flow rates, we use the following probabilistic constraints to capture the limited sampling

capacity for each switch:

$$\mathbb{P}\left\{\sum_{f\in\mathcal{F}_s} r_f \cdot x_{f,s} \leq B_s\right\} \geq 1 - \delta, \quad s \in \mathcal{S}, \qquad (2)$$

where, $0 < \delta < 1$ is a small probability with which the network orchestrator accepts the violation of the sampling capacity of a switch. Note that the violation of the sampling capacity results in the loss of a subset of sampled packets, which translates into a lower realized sampling rate for the affected flows on that switch. Although a lower value for δ reduces the capacity violation probability, it also leads to rejecting more flows (*i.e.*, setting more $x_{f,s}$ to zero) to ensure that the available capacity is sufficient to absorb any fluctuation in the rate of those flows that are accepted for sampling (*i.e.*, flows whose corresponding $x_{f,s}$ is set to one).

Incorporating (2) directly into an optimization program results in a stochastic optimization, which is generally non-trivial to solve efficiently unless the exact distribution of variables r_f is known, and can be expressed via simple closed-form expressions, none of which are true in real-world settings. Instead, a common approach in the literature [6], [17], [20] is to consider a deterministic version of (2), as follows:

$$\sum_{f\in\mathcal{F}_s} r_f \cdot x_{f,s} \leq B_s, \quad s \in \mathcal{S}, \qquad (3)$$

which is a linear constraint, but it entirely ignores the flow rate dynamics. However, a closer inspection of constraints (2) reveals that we are only interested in computing *the tail probability of the sum of random variables*. There are many concentration bounds, such as Chernoff bounds [21], that can be used to estimate such tail probabilities with different degrees of accuracy and complexity. If the target sampling rates are small, which is indeed the case in network-wide sampling, then each switch will have sufficient capacity to sample multiple flows. In such a setting, we can apply the central limit theorem to provide a tight estimation for the tail probability in (2), which yields the following constraints:

$$\sum_{f\in\mathcal{F}_s} \mu_f \cdot x_{f,s} + z_\delta \sqrt{\sum_{f\in\mathcal{F}_s} \sigma_f^2 \cdot x_{f,s}} \leq B_s, \qquad (4)$$

where, z_δ denotes the $(1-\delta)$ quantile of the standard Normal distribution. Putting this all together, the network-wide sampling allocation problem with dynamic flow rates can be defined as the following integer second order cone program:

$$(\text{ISOCP}): \quad \max. \sum_{f\in\mathcal{F}}\sum_{s\in\mathcal{S}} x_{f,s}, \qquad (5)$$
$$\text{s.t. } (1), (4).$$

B. Linearization

Given the conic constraints (4) in the ISOCP formulation, it is generally infeasible to solve with off-the-shelf solvers in a reasonable timeframe for sampling in realistic-sized networks. However, it is possible to transform these constraints into linear ones by introducing additional auxiliary integer variables in the formulation. Our numerical experiments indicated that this transformation significantly improves the computation time when employing off-the-shelf solvers such as Gurobi [22]. To

this end, we start by rewriting constraints (4) as follows where both sides of the inequality are squared:

$$z_\delta^2 \sum_{f\in\mathcal{F}_s} \sigma_f^2 \cdot x_{f,s} \leq \left(B_s - \sum_{f\in\mathcal{F}_s} \mu_f \cdot x_{f,s}\right)^2. \qquad (6)$$

To ensure the original constraints are enforced after this transformation, it is necessary that both sides of the inequality are non-negative. Clearly, the left-hand side is always positive, while the right-hand side is unrestricted. To resolve this, we also include the following constraints in the formulation:

$$B_s - \sum_{f\in\mathcal{F}_s} \mu_f \cdot x_{f,s} \geq 0. \qquad (7)$$

Then, we expand the right-hand side of (6) to obtain:

$$B_s^2 - 2B_s \sum_{f\in\mathcal{F}_s} \mu_f \cdot x_{f,s} + \left(\sum_{f\in\mathcal{F}_s} \mu_f \cdot x_{f,s}\right)^2, \qquad (8)$$

where, the last term can be further expanded to yield the following expression:

$$\sum_{f\in\mathcal{F}_s} \mu_f^2 \cdot x_{f,s}^2 + 2\sum_{f,f'\in\mathcal{F}_s} \mu_f \cdot x_{f,s} \cdot \mu_{f'} \cdot x_{f',s}. \qquad (9)$$

Now, we introduce auxiliary variables $w_{f,f',s}$ to model the multiplication of binary variables $x_{f,s}$ and $x_{f',s}$. The following constrains ensure that $w_{f,f',s}$ is only equal to one when both $x_{f,s}$ and $x_{f',s}$ are equal to one:

$$w_{f,f',s} \leq x_{f,s}, \qquad (10)$$
$$w_{f,f',s} \leq x_{f',s}, \qquad (11)$$
$$w_{f,f',s} \geq x_{f',s} + x_{f,s} - 1. \qquad (12)$$

Using $w_{f,f',s}$, we can re-write (6) as follows:

$$z_\delta^2 \sum_{f\in\mathcal{F}_s} \sigma_f^2 \cdot x_{f,s} \leq B_s^2 - 2B_s \sum_{f\in\mathcal{F}_s} \mu_f \cdot x_{f,s}$$
$$+ \sum_{f\in\mathcal{F}_s} \mu_f^2 \cdot x_{f,s} + 2\sum_{f,f'\in\mathcal{F}_s} \mu_f \mu_{f'} \cdot w_{f,f',s}, \qquad (13)$$

which has a linear structure. Therefore, the original ISOCP can be transformed into the following linear program:

$$(\text{ILP}): \quad \max. \sum_{f\in\mathcal{F}}\sum_{s\in\mathcal{S}} x_{f,s}, \qquad (14)$$
$$\text{s.t. } (1), (7), (10), (11), (12), (13).$$

Theorem 1. *ILP requires at most $O(|\mathcal{F}|^2\mathcal{S})$ more decision variables compared with ISOCP.*

Proof. ILP requires decisions variables $x_{f,s}$ for constraints (1) and decision variables $w_{f,f',s}$ for the linearization process. The number of former variables is equal to the number of variables in ISOCP. However, the latter variables are new and there are at most $O(|\mathcal{F}|^2\mathcal{S})$ of them. □

C. Approximation

While experimenting with ILP, we found that even this formulation is computationally too expensive, and can be solved only for small-sized problem instances using Gurobi. Thus, we turned our attention to developing an approximation formulation that can be applied to large-scale problem

instances. To this end, notice that the following inequality is always satisfied:

$$\sqrt{\sum_{f \in \mathcal{F}_s} \sigma_f^2 \cdot x_{f,s}} \le \sum_{f \in \mathcal{F}_s} \sigma_f \cdot x_{f,s} \,. \qquad (15)$$

Using this inequality, we can rewrite (4) as follows:

$$\sum_{f \in \mathcal{F}_s} \mu_f \cdot x_{f,s} + z_\delta \sum_{f \in \mathcal{F}_s} \sigma_f \cdot x_{f,s} \le B_s, \qquad (16)$$

which is linear. By substituting the above linear constraints in the original ISOCP, we obtain the following formulation:

$$(\text{APX}): \qquad \max. \sum_{f \in \mathcal{F}} \sum_{s \in \mathcal{S}} x_{f,s}, \qquad (17)$$
$$\text{s.t. } (1), (16),$$

which is an integer linear program with the *same number of decision variables* as in the original formulation.

IV. EVALUATIONS

In this section, we evaluate the performance of dSamp through extensive ns-3 simulations using synthetic traffic traces. In our simulations, we implemented a controller that configures sampling allocations and routing tables on the switches at the start of each epoch. When a packet arrives at a switch, it checks the sampling allocations provided by the controller, which map each flow's source and destination IP to a sampling rate. The switch identifies the packet's flow using its source and destination IP addresses and samples it with a probability corresponding to the assigned sampling rate. The routing tables are installed on the switches based on Dijkstra's algorithm, assuming static paths.

A. Methodology

We present *model-driven* simulations to compare APX with existing sampling solutions presented in [15] and [6]. Model-driven simulations provide us with the flexibility to simulate a range of network conditions without being restricted to a specific setup as in trace-driven simulations. Specifically, we use model-driven experiments to fine-tune the baseline algorithms that are used for comparison, and use these fine-tuned versions in our trace-driven experiments.

Trace-driven simulations and a set of *micro benchmarks* for APX are provided in the following technical report: [arXiv:2409.05966].

Simulation Settings. The model-driven simulations presented in this section were conducted in ns-3. The simulations were conducted on a standard desktop machine, equipped with an Intel(R) Core(TM) i9-12900 CPU @ 2.40 GHz and 16 GB of RAM. The *default* settings are as follows:

- *Network:* The small-scale simulations utilize the Abilene topology with 11 nodes and 14 links. The link capacities are set to 10 Gbps. The duration of each sampling epoch is set to 5 seconds by default. The results are computed as the average of 5 independent simulation runs.
- *Traffic:* The ns-3 on-off application was used to generate traffic based on the input traffic model. The packet size is set to 1 KB in all simulations. For each flow, the source and

destination are chosen at random. To describe the burstiness of traffic, we use the Coefficient of Variation (CoV), defined as $\text{CoV} = \mu/\sigma$. A high CoV indicates high variability in flow rates, whereas a low CoV suggests low variability.

- *Sampling:* By default, the target sampling rate for each flow is set to $\alpha = 0.1$, while the switch sampling capacity violation probability is set to $\delta = 0.20$.

Implemented Algorithms. In addition to APX, we have implemented the following algorithms for comparison:

- *Deterministic Sampler (DS):* This algorithm makes sampling decisions solely based on the mean flow rates.
- *Deterministic with Headroom (DS+2σ):* This algorithm is similar to DS except that it adds a 2σ headroom to flow rates in order to allow the sampling algorithm to absorb flow rate fluctuations at runtime. It is the approach proposed in [6] to deal with dynamic traffic rates. Notice that, for a normal distribution, 95% of samples reside within 2σ of the mean.
- *cSamp+ϵ (cS+ϵ):* This algorithm is presented in [15]. It assumes that the fluctuation in traffic rates around their means is bounded by some $\epsilon > 0$. It then scales the solution computed by the DS algorithm to account for the rate fluctuations that can happen in the worst case.

At a high level, APX also reserves some headroom on each switch to deal with rate fluctuations. However, in APX, the amount of headroom depends on the actual variability (variance) of flow rates, while in DS+2σ, it is a fixed value. Also, while the headroom in cSamp depends on the actual flow rates, it is reserved based on the worst case fluctuations of rates leading to a consistently more conservative behavior than APX, as we will show later in this section.

Performance Metrics. The main performance metric, referred to as *"Fully Sampled Flows"*, is the number of flows that are fully sampled by the sampling algorithm, *i.e.*, whose target sampling rate is satisfied by the sampling algorithm. This metric concisely captures the sampling accuracy achieved by an algorithm. However, to provide more insights about the performance of each sampling algorithm, we also report the following metrics:

- *Admitted Flows:* The number of flows that are chosen by the algorithm for sampling. The sampled flows may not be sampled at their desired target sampling rate.
- *Sampling Rate:* The actual sampling rates measured from the simulations. We would like the measured Sampling Rate to be close to the target sampling rate.

B. Model-Driven Simulations

Traffic Generation. Since the Abilene topology is used, a total of 550 flows (110×5) are available for sampling over 5 epochs. Out of these, 446 flows (about 80%) were chosen across all 5 epochs. We assign flow rates by randomly selecting their mean from the set $\{200, 300, 500\}$ KBps. Additionally, for each flow, with the probability of 30% and 70%, its CoV is set to 0.2 (low variability) or 2 (high variability), respectively. The flow rates are then sampled from a truncated normal

(a) Total number of admitted flows.   (b) Total number of fully sampled flows.   (c) Measured sampling rates (excluding zero rates).
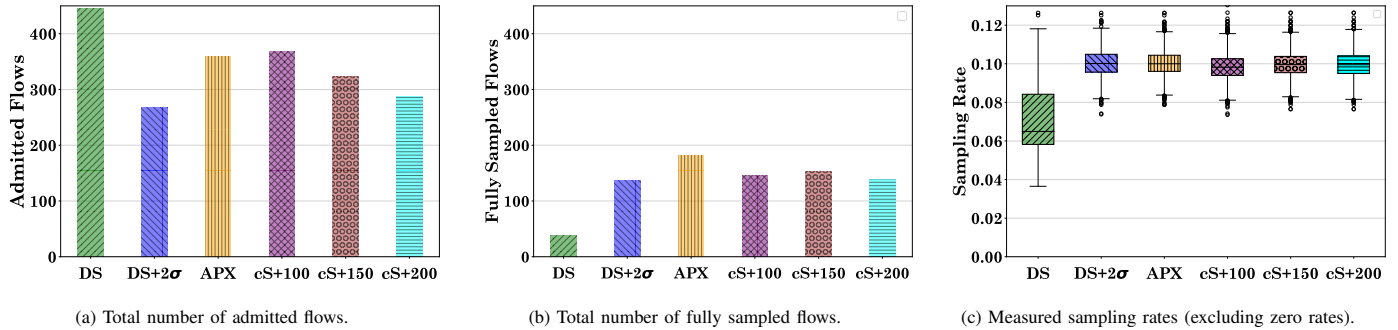
Fig. 3: Model-driven ns-3 simulations results.

distribution based on the assigned mean and variance. During each experiment, the flow rates change every 0.1 seconds. The switch sampling capacities are set to 400 pps.

Results and Discussion. The results are depicted in Fig. 3. Specifically, Fig. 3(a) shows Admitted Flows for each algorithm (from 446 flows). As expected, DS is the most aggressive algorithm due to its lack of consideration for flow rate fluctuations. Fig. 3(b) presents the results for Fully Sampled Flows with different algorithms. We see that while DS was the most aggressive in admitting flows, it achieves the lowest Fully Sampled Flows among all the algorithms. For cSamp, we had to decide about the parameter ϵ. We experimented with a range of values, but have only plotted cS+100, cS+150, and cS+200, as we found that other values make the algorithm too aggressive or conservative leading to lower performance. Nevertheless, we observe that APX can fully sample more flows that even the best version of cSamp, namely cS+150. Not surprisingly, we also find that DS+2σ falls somewhere in the middle of the pack, but clearly less efficient than APX. Finally, Fig. 3(c) presents the Sampling Rate of each algorithm, where it can be seen that except DS and cS+100, all other algorithms can actually achieve sampling rates that are very close to the target sampling rate. In fact, while there are some variations in measured sampling rates for these algorithms, the median sampling rates closely match the 0.10 target rate.

V. RELATED WORKS

Traffic sampling is a well-studied area. In the following, we briefly review some representative works with a focus on per-flow sampling solutions.

Per-Port Sampling. NetFlow [9] and sFlow [10] are widely used for per-port sampling. In per-port sampling, a switch samples a fraction of packets that pass through one or more of its ports regardless of which flow these packets belong to. Several works try to improve the performance of NetFlow and sFlow by more intelligently distributing sampling load on network switches. For instance, the works [25] and [26] propose sampling solutions that aim at minimizing the sampling load on network switches by coordinating sampling across switches. However, since per-port sampling solutions are based on a fixed sampling rate for each switch port, they are biased toward flows that send at a higher rate, which results in low

accuracy for many management tasks that require a minimum per-flow sampling rate [27]–[29].

Uncoordinated Per-Flow Sampling. The works in this category consider per-flow sampling, but the sampling decisions are made on a per-switch basis without any network-wide coordination. For instance, the works [30] and [31] propose algorithms for adjusting the sampling rate on each switch in response to variations in traffic rate. Other works in this category focus on providing better traffic estimates from the sampled packets [4], [32] or reducing the communication overhead [33]. These works, however, do not consider sampling coordination among network switches, and as such, are less efficient compared to coordinated sampling solutions. The reason is that when switches sample flows without coordination, it is possible that some packets are sampled redundantly by multiple switches. Given the limited sampling capacity of switches, the result is a lower effective sampling rate for each flow, which translates into lower flow visibility in the network.

Coordinated Per-Flow Sampling. The works in this category leverage a centralized controller to coordinate sampling responsibilities across network switches. The works [6], [15], [16], [34] distribute sampling load among different switches. These works, however, assume that the sampling controller knows the set of flows and their sending rates in advance, which is not realistic. The work [17] considers coordinated sampling in an online setting, where the controller does not have any information about the flows that will arrive in the system for sampling in the future. They also separate sampling decisions for short and long flows, by having a fixed sampling rate for short flows and only performing coordinated sampling for long flows. However, all the aforementioned solutions assume fixed flow rates during the sampling period, which may lead to under-sampling, resulting in lower accuracy in some network management tasks such as anomaly detection. Several works consider flow rate fluctuations in sampling to increase the accuracy of per-flow sampling. For instance, OpenMeasure [35] predicts the size of flows based on previously collected flow samples. While their method considers variations in flow sizes across different measurement intervals, it does not account for the instantaneous changes in flow rates within the same measurement interval. Perhaps the work closest to ours is cSamp [15], which considers coordinated network-wide

sampling. While the cSamp sampling algorithm is designed assuming fixed flow rates, it has a post-processing mechanism to make the solution robust against estimation errors in flow rates. Specifically, the sampling solution computed for fixed flow rates is *scaled* to account for the maximum deviation in flow rates based on the assumed estimation error. As we show in Section IV, this approach is very conservative by essentially considering the worst-case flow rates, which means that it may fail to fully utilize the available network sampling capacity. A somewhat similar approach is considered in [6], where the authors scale the switch sampling capacities by a fixed factor to leave some headroom for potential flow rate fluctuations. However, a fixed headroom is not very efficient since it does not account for dynamic changes in network traffic, as demonstrated in our evaluations.

In contrast to the above works, our approach is to explicitly include flow rate fluctuations in the design of the sampling controller. Rather than assuming the worst-case flow rates, we consider a probabilistic approach in which we directly bound the probability that any switch is overloaded (its sampling capacity is violated) due to fluctuation in flow rates.

VI. Conclusion

In this paper, we presented the design and evaluation of a sampling system called dSamp, capable of handling dynamic flow rates. We formulated the problem of network-wide co-ordinated sampling with dynamic flow rates as an ISOCP, which proved infeasible for even small-scale networks. To address the computational complexity of larger networks, we created two alternative reformulations: one transformation and one approximation. We found that the transformation works well for small-scale networks, whereas the approximation is more effective for larger-scale networks. Through model-driven ns-3 simulations, we showed that our proposed system outperforms existing sampling systems in achieving a higher percentage of fully sampled flows. A potential future direction is to design an online algorithm to solve the problem, where sampling requests are processed instantaneously instead of being batched over sampling epochs.

References

[1] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, no. 4, 2018.

[2] T. Li, S. Chen, W. Luo, and M. Zhang, "Scan detection in high-speed networks based on optimal dynamic bit sharing," in *Proc. IEEE INFOCOM*, 2011.

[3] Y. Tian, W. Chen, and C.-T. Lea, "An SDN-based traffic matrix estimation framework," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, 2018.

[4] N. Duffield, C. Lund, and M. Thorup, "Charging from sampled network usage," in *Proc. ACM SIGCOMM, Workshop on Internet Measurement*, 2001.

[5] J. Zhang, Y. Xiang, W. Zhou, and Y. Wang, "Unsupervised traffic classification using flow statistical properties and IP packet payload," *Journal of Computer and System Sciences*, vol. 79, no. 5, 2013.

[6] R. Cohen and E. Moroshko, "Sampling-on-demand in SDN," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, 2018.

[7] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *Proc. USENIX NSDI*, 2016.

[8] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. IEEE ICDCS*, 2014.

[9] B. Claise, "Cisco systems NetFlow services export version 9," Tech. Rep., 2004.

[10] sFlow "Making the network visible". [Online]. Available: https://sflow.org/

[11] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran, "Reformulating the monitor placement problem: Optimal network-wide sampling," in *Proc. ACM CoNEXT*, 2006.

[12] J. Suárez-Varela and P. Barlet-Ros, "Towards a NetFlow implementation for OpenFlow software-defined networks," in *Proc. IEEE ITC*, 2017.

[13] S. Shirali-Shahreza and Y. Ganjali, "Flexam: Flexible sampling extension for monitoring and security applications in openflow," in *Proc. ACM SIGCOMM, Workshop on Hot Topics in SDN*, 2013.

[14] F. Raspall, "Efficient packet sampling for accurate traffic measurements," *Computer Networks*, vol. 56, no. 6, 2012.

[15] 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 *Proc. USENIX NSDI*, 2008.

[16] V. Sekar, M. K. Reiter, and H. Zhang, "Revisiting the case for a minimalist approach for network flow monitoring," in *Proc. ACM SIGCOMM*, 2010.

[17] S. Sadrhaghighi, M. Dolati, M. Ghaderi, and A. Khonsari, "FlowShark: Sampling for high flow visibility in SDNs," in *Proc. IEEE INFOCOM*, 2022.

[18] B. Zhou, D. He, and Z. Sun, "Traffic predictability based on ARIMA/GARCH model," in *Proc. NGI*, 2006.

[19] D. Andreoletti, S. Troia, F. Musumeci, S. Giordano, G. Maier, and M. Tornatore, "Network traffic prediction based on diffusion convolutional recurrent neural networks," in *Proc. IEEE INFOCOM, Workshop on Network Intelligence*, 2019.

[20] V. Sekar, A. Gupta, M. K. Reiter, and H. Zhang, "Coordinated sampling sans origin-destination identifiers: algorithms and analysis," in *Proc. COMSNETS*, 2010.

[21] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, pp. 493–507, 1952.

[22] Gurobi Optimization. [Online]. Available: http://www.gurobi.com/

[23] MAWI "MAWI Working Group Traffic Archive". [Online]. Available: http://mawi.wide.ad.jp/mawi/

[24] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "On the scalability of BGP: The role of topology growth," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 8, 2010.

[25] H. Xu, S. Chen, Q. Ma, and L. Huang, "Lightweight flow distribution for collaborative traffic measurement in software defined networks," in *Proc. IEEE INFOCOM*, 2019.

[26] S. Sadrhaghighi, M. Dolati, M. Ghaderi, and A. Khonsari, "SoftTap: A software-defined TAP via switch-based traffic mirroring," in *Proc. IEEE NetSoft*, 2021.

[27] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proc. ACM SIGCOMM*, 2006.

[28] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta, "Analysis of the impact of sampling on NetFlow traffic classification," *Computer Networks*, vol. 55, no. 5, 2011.

[29] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," *Computer Networks*, vol. 109, 2016.

[30] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better NetFlow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, 2004.

[31] R. R. Kompella and C. Estan, "The power of slicing in Internet flow measurement," in *Proc. ACM SIGCOMM*, 2005.

[32] N. Hohn and D. Veitch, "Inverting sampled traffic," in *Proc. ACM SIGCOMM*, 2003.

[33] Y. Du, H. Huang, Y.-E. Sun, S. Chen, G. Gao, and X. Wu, "Self-adaptive sampling based per-flow traffic measurement," *IEEE/ACM Trans. Netw.*, 2022.

[34] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah, "Leisure: A framework for load-balanced network-wide traffic measurement," in *Proc. ACM/IEEE ANCS*, 2011.

[35] C. Liu, A. Malboubi, and C.-N. Chuah, "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN," in *Proc. IEEE INFOCOM WKSHPS*, 2016.