Lightweight Coordinated Sampling for Dynamic Flows under Budget Constraints

Mingming Chen, Thomas La Porta
Computer Science and Engineering Department
Penn State University, University Park
University Park, PA, 16802 emails: {mzc796,tfl12}@psu.edu

Trent Jaeger, Srikanth Krishnamurthy
Computer Science and Engineering Department
University of California, Riverside
Riverside, CA, 92521 emails:trentj@ucr.edu,krish@cs.ucr.edu

***Abstract*—As cyber-attacks on networks become more stealthy, monitoring techniques relying on low-rate packet sampling may prove insufficient to detect attacks. While various sampling methods have been proposed to address capacity limitations and enhance detection rates, achieving sampling at line speed at a single point remains challenging due to limited CPU or bandwidth capacity at sampling points. In this paper, we propose harnessing coordinating sampling across switches to create a unified system that can dynamically activate sampling points to meet sampling rate needs. We introduce and implement a coordinated sampling algorithm on multiple P4-programmable switches and show that the algorithm ensures coordination among multiple sampling points for each flow, preventing duplicate samples, with negligible network overhead and real-time configurability. We formulate sampling point placement as budgeted maximum multi-coverage problems, solving them optimally in pseudo-polynomial time. We show our system far outperforms those based on greedy algorithms along many key dimensions.**

***Index Terms*—P4-programmable switch, coordinated sampling, budgeted maximum multi-coverage, balanced matrix**

I. Introduction

Traffic sampling is an important technique used for network monitoring for both network management and security purposes. Due to limitations in processing capacity and bandwidth, switches cannot typically forward every packet to a monitor. As a result, sampling is implemented, where specific packets are selected and forwarded to a monitor for analysis. People strive to maximize the monitoring gain within limited sampling capacity [15], [30].

Regardless of the techniques used to sample at switches, sampling flows at line rate under heavy traffic load may not be achievable with a single sampling point. When either entire packets or portions of packets are sent from the sampling point to the monitor, port capacity on switches may be violated [11] even if programmable ASICS are used to sample the packets.

However, lowering sampling rates to accommodate switch limitations introduces the risk of allowing low and slow attacks to go undetected. For example, a low-and-slow attack initiated by Slowloris running on a single VM can cause a 24% increase on the Web server's memory with less than a 40kbps increase on the network traffic [29]. The unpredictable dynamics of network flows require the sampling points, sampling rate, and sampling strategy to change with it. Our system provides coordinated sampling, smart sampling point placement, and dynamic sampling point configuration to meet this need.

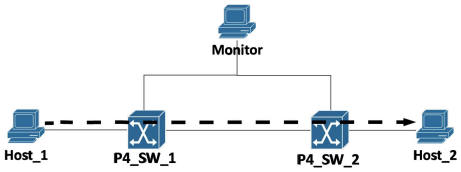

Fig. 1: Testbed Topology

Unlike existing works which focus on single point sampling optimization [15], [30], we develop and implement an efficient algorithm to achieve high-rate line-speed flow-based sampling using multiple P4-programmable switches along the paths that are *coordinated* to extract samples. P4 is a domain-specific language for programming packet processing pipelines in network devices [9]. We chose P4-programmable switches because the sampling can be programmed on the P4-programmable ASIC without CPU constraints and the programs can be installed in real-time. Consequently, the functionality of sampling and forwarding samples can be tuned in real-time.

Given the substantial cost difference between P4-programmable switches and OpenFlow switches (a factor of up to 10 [1]), an objective of our work is to strategically place a budgeted number of programmable switches as potential sampling points in a network. The goal is to maximize the average number of flows that can be sampled at a sufficient rate, perhaps by activating multiple sampling points to share the load. To address the placement problem, we formulate budgeted maximum (multi-) coverage problems. We show these NP-complete Integer Linear Programming (ILP) problems are optimally solvable in pseudo-polynomial time by leveraging the balanced matrix property.

A. System Overview

A high-level view of our system is shown in Fig. 1. Hosts are connected through a network containing at least some P4-programmable switches, which can sample packets on a per-flow basis and send them to one of many monitors for analysis.

This research was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory of the U.S. government. The U.S. government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation here on.

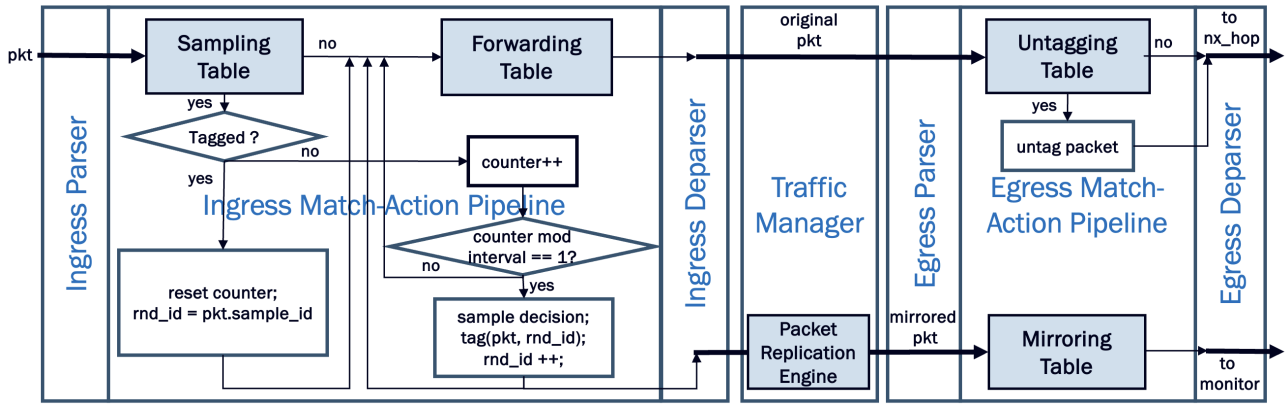

Fig. 2: Coordinated Sampling Workflow on TNA Pipeline

If the SDN controller detects that a sampling point becomes overloaded and can only sample some flows with a fraction of the desired rate, it reallocates sampling responsibilities, perhaps splitting sampling among multiple sampling points. Our coordinated sampling algorithm executes on the sampling points and guarantees there are no duplicate samples.

We aim to strategically deploy a budgeted number of P4-programmable switches to maximize the number of flows that can be sampled at the required rate considering that some flows may be sampled at multiple points. This requires solving three problems: (1) coordinating per-flow sampling when multiple points are sampling the same flow, (2) placing the budgeted number of programmable switches to maximize the number of flows that have sufficient coverage to handle sampling load, and (3) dynamically assigning flows to sampling points.

We operate under the assumption that we have knowledge of potential flow paths due to traffic history, and paths can be enumerated using standard routing policies and realistic network topologies. This assumption holds true for enterprise networks and SD-WANs [20], especially when employing popular routing protocols that result in path stability [10].

In this paper, we make the following contributions:

- We introduce and implement "coord_sampling" to efficiently coordinate flow-based sampling at multiple network points without communication overhead. We show it incurs negligible performance overhead and can be activated within 0.05 seconds.
- We formulate budgeted maximum (multi-)coverage problems for sampling point placement, and show that the optimal solutions for this NP-complete problem can be achieved in pseudo-polynomial time in our scenario using balanced matrix properties on real topologies. The optimal solution outperforms the greedy method by up to 90% with a comparable runtime of 10^{-1} to 10^{-2} seconds on realistic topologies.
- We show that the set of sampling points placed in our evaluated networks using our budgeted optimal algorithm provides the capacity to sufficiently sample flows under heavy loads by coordinating sampling whereas the sampling points placed by the greedy algorithm under the same budget do not.

II. Design of the Coordinated Sampling

The goal of the coordinated sampling algorithm is to distribute the sampling load for individual flows across multiple sampling points, so sampling can be done at the required rate. The coordinated sampling system aims to (1) prevent duplicated samples from the same flow across different sampling points and (2) ensure that samples from the same flow across different sampling points can be collected and sorted by the monitor. We introduce a method for achieving coordinated sampling among sampling points with minimal overhead. We begin with an overview of our coordinated sampling algorithm and the underlying switch architecture. Subsequently, we delve into the algorithm's details and implementation.

A. Coordinated Sampling Overview

Our coordinated sampling algorithm operates on the Tofino Native Architecture (TNA) shown in Fig. 2. TNA consists of two programmable sections: the Ingress and Egress Pipelines, each comprising a parser, match-action pipeline, and deparser. The parser stage dissects packets into distinct fields. In the match-action pipeline, packets are matched and actions are executed as per the algorithm. This pipeline houses the primary components of the coordinated sampling algorithm. Parsed packets are reconstructed at the deparser stage. In the Ingress deparser, packets proceed to the Egress pipeline for further handling. In the Egress deparser, packets are either forwarded to the next hop or discarded based on configured actions.

Our design incorporates four Match-Action tables: the **Sampling Table** determines whether a packet should be sampled. The **Forwarding Table** forwards flows' packets normally. The **Mirroring Table** determines the samples' output port to the monitor. The **Untagging Table** removes the tag from sampled packets at the network edge.

B. Coordinated Sampling Algorithm

The coord_sampling algorithm (Algorithm 1) is implemented on the TNA architecture in Fig. 2. The P4 code is available on GitHub [2]. Upon entering the pipeline, the Sampling Table $table_s$ evaluates the packet first. If there is no match in $table_s$, it proceeds to the Forwarding Table $table_f$ for standard forwarding processing. Conversely, if a match occurs, the packet is directed to the coord_sampling algorithm.

Algorithm 1 Coord_Sampling on P4-programmable Switches

Input: pkt_i: Recent packet of flow f_i;
$table_s$: Ingress flow table for sampling some flows;
$counter_i$: Stateful counter of flow f_i, initial value is 0;
$interval_i$: Sampling interval for flow f_i;
id_i^{rd}: Stateful round id number, initial value is 0;
m_i: Mirroring session id of flow f_i.
Output: is_out, id_i^{rd}.
Function coord_sampling:
($id_i^{sa}, pkt_i^{parsed}$) = parse($pkt_i$);
($is\_hit, interval_i, counter_i, m_i$) = read($pkt_i^{parsed}, table_s$);
if is_hit **then**
  if id_i^{sa} *is NULL* **then**
    $counter_i = counter_i + 1$;
    if $counter_i \mod interval_i$==*1* **then**
      tag_sampled(pkt_i, id_{rd});
      $id_i^{rd} = id_i^{rd} + 1$;
      $pkt_i^{mirrored}$=mirror(pkt_i, m_i);
      output($pkt_i^{mirrored}, port_{monitor}$);
    end
  else
    $counter_i = 0$;
    $id_i^{rd} = id_i^{sa}$;
  end
end

To achieve Goal (1), our algorithm relies on two parameters: $counter_i$ tracks packets of flow f_i, while $interval_i$ stores the dynamically configured sampling interval value (the reciprocal of the sampling fraction) of flow f_i. In a traditional flow-based single-point sampling setup, a switch samples a packet when condition (A) ($counter_i \mod interval_i == 1$) is met. However, this condition alone is insufficient for coordinating sampling across multiple points to avoid duplication, especially considering packet dropping or swapping. To address this, we tag sampled packets to inform downstream switches not to sample them. Thus, a P4-programmable switch samples a packet only when conditions (A) and (B) (the packet is not tagged) are both met. If condition (B) is not met, we reset $counter_i$ to prevent potential overlapped sampling.

To achieve Goal (2), switches track the number of rounds of sampling, id_i^{sa}, and report that to the monitors by putting the id_i^{sa} value in sampled packet headers, where a round is completed when a sampling switch has seen an interval number of packets on the flow. As shown in Algorithm 1, each switch tracks the number of rounds performed in the variable id_i^{rd} for flow f_i. When a switch finds that it should sample a packet pkt, it samples pkt and tags pkt's "Options" header field with $id_i^{sa} = id_i^{rd}$. id_i^{rd} is then incremented on switch to prepare for its next round. Monitors that know topology (i.e., the order of sampling switches for a flow) can order packets from sampling switches in each round for each flow.

To maintain the correct round order across switches when packet dropping/swapping happens between sampling points, we adjust $counter_i$ and id_i^{rd} whenever a sampled packet pkt^{sa} is seen. Specifically, the switch's $counter_i$ is reset, and flow f_i's round id id_i^{rd} is synchronized with upstream switch by setting $id_i^{rd} = id_i^{sa}$ where id_i^{sa} is the sampled id of pkt^{sa}.

The monitor arranges sampled packets of f_i from multiple switches by gathering packets in ascending order with the same id_i^{sa} tag along the path of f_i, moving from upstream to downstream points. The detailed steps of Algorithm 1 focus on switch sw's sampling decisions and tagging. When packet pkt_i hits a match in $table_s$, we verify the existence of id_i^{sa}:

- If it does not exist, sw increases $counter_i$ and checks the indicator value of $counter_i \mod interval_i$:
  - If equals 1, sw tags pkt_i's id_i^{sa} with its current id_i^{rd}, increases its id_i^{rd}, mirrors pkt_i, sends $pkt_i^{mirrored}$ to the monitor, and forwards pkt_i normally;
  - Otherwise, sw forwards pkt_i normally.
- Otherwise (pkt_i has been sampled with a tag id_i^{sa}), sw resets $counter_i$ and synchronizes its round id id_i^{rd} with the upstream switch's round id by setting its value as pkt_i's value id_i^{sa}, and forwards pkt_i normally.

Our coord_sampling algorithm is computationally efficient with a complexity of $O(k)$ (where k is a constant) within an if-else pipeline. Moreover, the mirroring process incurs no CPU cost on the programmable ASIC but only entails memory costs for configuration. The algorithm is also memory-efficient for the following reasons: Each flow f_i requires variables $counter_i$, $interval_i$, id_i^{rd}, and m_i to configure its sampling task, akin to a flow entry. The bit length of m_i increases sublinearly with the number of sampled flows. 1-byte id_i^{rd} is enough to avoid round id confusion caused by id_i^{rd} overflow [1]. The bit length of $interval_i$ is inherently short to facilitate frequent sampling. Notably, the value of $counter_i$ does not continually increase to track the absolute number of packets; instead, it is reset whenever $counter == interval$ or meets a tagged packet, keeping its bit length as short as $interval_i$'s.

Our design is robust, with minimal communication and time costs. It ensures sampling coordination and prevents misorder propagation even with packet dropping or sequence swapping during transmission. Importantly, it eliminates the need for extra communications among P4-programmable switches or between the SDN controller and switches. Additionally, our design allows real-time, configurable, and manageable flow-based sampling. Experimental results in Section VI reveal low overhead in network performance and swift activation/deactivation of sampling points.

Coordination among sampling points for the same flow is crucial. Equally significant is strategically placing a budgeted number of P4-programmable switches to maximize the number of dynamic flows sampled at their required rates. We further explore problem formulations for sampling point placement.

III. Problem Formulation

The budgeted sampling point placement problems are framed as Integer Linear Programming (ILP) problems considering multi-coverage. When a set of flows requires a

[1]No same round id in < 72 seconds even under an extreme case assuming bandwidth is 10 Gbps, packet size is 84 bytes, and the sampling fraction is 1

higher sampling rate than a single point can provide, multiple sampling points are necessary. With a budgeted number of P4-programmable switches supporting coordinated sampling, the ultimate objective is to maximize the number of flows in a topology that can be sampled at their required rate possibly using multiple sampling points. We define a flow as **k-covered** if at least k points are chosen as sampling points on its path.

The **Budgeted Maximum k-Coverage (BMkC)** problem aims to maximize the number of k-covered flows while ensuring all flows are at least $(k-1)$-covered, given a budgeted number of sampling points. When $k = 1$, this becomes the **Budgeted Maximum Coverage (BMC)** problem [19].

Universal problem setting: Given a set of flows $F = \{f_1, ..., f_m\}$ on topology $G(V, E)$, V is the set of switch positions and E is the set of links. A flow $f_i \in F$ is denoted by the switch positions it traverses as $f_i : v_k \rightarrow ... \rightarrow v_l$. For every $v_j \in V$, if v_j is part of f_i, $f_i \in x_j$ where x_j is a set of flows such that they all traverse v_j. We call x_j the **position set** and v_j is its position. X is the collection of all position sets x_j, i.e. $X = \bigcup_{j=1}^{n} x_j$. We define **path matrix**: Let A_P denote the path matrix $A_P = [a_{i,j}]$, where $a_{i,j} \in \{0, 1\}$. If flow $f_i \in x_j$, $a_{i,j} = 1$; otherwise, $a_{i,j} = 0$. The formulation notations are summarized in Table I.

A. Budgeted Maximum Coverage (BMC)

Given the universal problem setting, find a sub-collection of X such that the number of 1-covered flows is maximized within a budgeted number of P4-programmable switches B.

BMC Formulation:

$$\arg\max \sum_{i=1}^{m} g_i \tag{1}$$

Subject to :

$$\mathbf{g} - A_P\mathbf{x} \leq \mathbf{0} \tag{2}$$

$$\sum_{j=1}^{n} x_j \leq B \tag{3}$$

$$x_j \in \{0, 1\} \tag{4}$$

$$g_i \in \{0, 1\} \tag{5}$$

The BMC problem is NP-hard [19]. The objective (1) is to maximize the number of 1-covered flows g_i such that $g_i = 1$ if there is a sampling point on flow f_i. This is achieved by (2) combined with (5): For each flow f_i, if $A_{Pi}\mathbf{x} = 0$ which means the chosen sampling points cannot cover it, then $g_i \leq 0$ and (5) results in $g_i = 0$. If $A_{Pi}\mathbf{x} > 0$ which means f_i can be covered, we have $g_i = 0$ or $g_i = 1$. Because (1) maximizes the sum of g_i, $g_i = 1$ will be enforced. As a result, these ensure that if a flow f_i is covered, $g_i = 1$. Otherwise, $g_i = 0$. (3) is the constraint on the budgeted number of sampling points. (4) are the binary decision variables.

Although existing works adopt a polynomial time greedy algorithm to solve this problem with a $(1-\frac{1}{e})$-approximate result [19], we achieve the optimal solution in pseudo-polynomial time by using the MILP solver with proof in Section IV and experiments in Section VI.

TABLE I: Parameters in the Formulations

Parameter	Definition
A_P	$m \times n$ path matrix
A_{P_i}	ith row of $m \times n$ path matrix, $i \in \{1, 2, ..., m\}$
x_j	Decision variable for switch position $j, j \in \{1, 2, ..., n\}$
$\mathbf{x}$	$n \times 1$ binary vector of each switch position, $\mathbf{x} = \{x_j\}^T$
k	Times of sufficient coverage of flow f_i, $i \in \{1, 2, ..., m\}$
g_i	Gain of flow $i, i \in \{1, 2, ..., m\}$.
$\mathbf{g}$	$m \times 1$ 0-1 gain vector of $g_i, \mathbf{g} = \{g_1, ..., g_i, ..., g_m\}^T$
B	Budgeted number of P4-programmable switches

B. Budgeted Maximum k-Coverage (BMkC)

Given the universal problem setting, find a sub-collection of X such that the number of k-covered flows is maximized while all flows are $(k-1)$-covered within a budgeted number of P4-programmable switches B.

The BMkC problem is the general version of BMC problem.

BMkC Formulation:

$$\arg\max \sum_{i=1}^{m} g_i \tag{6}$$

Subject to:

$$\forall i \in \{1, ..., m\}, \quad g_i - \frac{1}{k} A_{P_i}\mathbf{x} \leq 0 \tag{7}$$

$$\forall i \in \{1, ..., m\}, \quad -A_{P_i}\mathbf{x} \leq -(k-1) \tag{8}$$

$$\sum_{j=1}^{n} x_j \leq B \tag{9}$$

$$x_j \in \{0, 1\} \tag{10}$$

$$g_i \in \{0, 1\} \tag{11}$$

Constraints (6)(9)(10)(11) map to (1)(3)(4)(5). The difference between BMC and BMkC arises from (7) and (8). Because each flow f_i needs to be covered at least k times to achieve a gain $g_i = 1$, the $\frac{1}{k}$ coefficient is added to the i-th row of $A_P\mathbf{x}$ based on (2). The role of (8) is to guarantee all flows are (k-1)-covered. The g_i will be 0 if the f_i cannot be k-covered within the budget. In BMC, $g_i = 1$ when f_i can be covered at least once within the budget. In BMkC, $g_i = 1$ when f_i can be k-covered within the budget.

IV. The Sampling Point Placement Problems Hardness Walk-Through

Even though the budgeted sampling point placement problems are NP-complete, it is possible to obtain optimal solutions in polynomial time for useful realistic cases. We develop the discussion of budgeted sampling point placement problems from the minimum cost sampling point placement problems (i.e., set multi-covering problems). In detail, we look at the structure of the path matrices and show under what structures problems can be solved optimally in polynomial time.

In section IV-A, we show that when a path matrix is balanced, the set multi-covering problem can be solved optimally using an LP. From this, we show that the left-hand-side matrix of the BMC constraints forms a $(0, \pm 1)$ balanced matrix if the path matrix of its corresponding minimum cost sampling point placement problem is a $(0, 1)$ balanced matrix. Under the

conditions that the left-hand-side matrix is $(0, \pm 1)$ balanced, we prove a theorem guaranteeing BMC an optimal solution by LP. Finally, we discuss the runtime of the unbalanced cases in which a pseudo-polynomial optimal solution is achieved by LP and branch and bound.

A. Linear Programming Optimum-Balanced Matrix

1) Minimum Cost Sampling Point Placement: Intuitively, the **Set k-Covering** problem minimizes the number of sampling points to cover all flows k times. The formulation is: $min\{x|x \in \{0,1\}; A_P x \geq k\}$.

Theorem 1. *If A is a $(0,1)$ balanced matrix, b and c are integral vectors and one of them is an all-one vector, then $min\{cx|x \geq 0; Ax \geq b\}$ and $max\{cx|x \geq 0; Ax \leq b\}$ have integral optimum solutions (if the optima are finite) [25].*

By Theorem 1, we conclude that if the path matrix A_P is balanced, the polynomial running time LP produces an optimal integral solution for the set k-covering problem.

Because a (0,1) matrix is balanced if and only if it does not contain a submatrix that is an incidence matrix of any odd cycle [25], the structure composed by all paths is sufficient to show the balanced property of its path matrix. Roughly speaking, the path matrix is balanced if the structure composed of all paths does not have an odd-cycle. Thus, it is the structure composed of all the target paths that results in the hardness of the min-cost objective sampling point placement problem.

2) Budgeted Sampling Point Placement: Here we prove that when the path matrix A_P is balanced, the corresponding BMC problem also achieves optimal integral solutions by LP. We show that the left-hand-side matrix of the constraints of the BMC formulation forms a $(0, \pm 1)$ balanced matrix if its path matrix A_P is a $(0,1)$ balanced matrix.

We organize the constraints of BMC in the matrix form as in Theorem 1. First, the variable vector is extended to be $\{g_1, ..., g_m, x_1, ..., x_n\}^T$. Then, the matrix form of BMC constraints (2) and (3) is: $\begin{bmatrix} \mathbf{I} & -A_P \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \mathbf{x} \end{bmatrix} \leq \begin{bmatrix} \mathbf{0} \\ B \end{bmatrix}$. We call $A_E = \begin{bmatrix} I_{m\times m} & -A_{P_{m\times n}} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$ **extended path matrix.**

Theorem 2. *If the path matrix A_P is a $(0,1)$ balanced matrix, its corresponding extended path matrix A_E is a $(0, \pm 1)$ balanced matrix.*

Proof. A $(0, \pm 1)$ matrix is balanced if no submatrix of it is an odd hole matrix [8]. A hole matrix is a $(0, \pm 1)$ matrix that contains two nonzero entries per row and per column, and no proper submatrix of it has this property [8].

(1) Because A_P is $(0,1)$ balanced, it only has even-cycle incidence matrices which are hole submatrices. The sum of the entries in the hole submatrices of $-A_{P_{m\times n}}$ can only be $-2c, c = 2, 4, 6, ...$ which is always a multiple of 4. So $-A_{P_{m\times n}}$ does not voilate $(0, \pm 1)$ balanced property for A_E.

(2) $\begin{bmatrix} -A_{P_{m\times n}} \\ \mathbf{1} \end{bmatrix}$ is also balanced because the only case such that $\mathbf{1}$ is involved into a hole matrix is $\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$ and the sum of all entries is 0 which is a multiple of 4.

(3) As a result, $\begin{bmatrix} I_{m\times m} & -A_{P_{m\times n}} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$ is balanced because involving $\begin{bmatrix} I \\ \mathbf{0} \end{bmatrix}$ does not make any other hole matrix. □

Next, we prove an extended version of Theorem 1 to support that when A_E is $(0, \pm 1)$ balanced, the LP produces integral/optimal solutions for BMC also.

Theorem 3. *If A is a $(0, \pm 1)$ balanced matrix, then $min\{x|x \geq 0; Ax \geq b - n(A)\}$ and $max\{x|x \geq 0; Ax \leq b - n(A)\}$ has integral optimum solutions, where b is an integral vector, $n(A)$ is the column vector whose ith components $n_i(A)$ is the number of '-1's in the ith row of matrix A.*

Proof. The strategy follows [12]. We transform $(0, \pm 1)$ balanced matrix $A_{m\times n}$ into a $(0,1)$ balanced matrix $B_{m\times 2n}$. Given A is $(0, \pm 1)$ balanced, we have B is $(0,1)$ balanced because (1) The corresponding elements in B transformed from a hole matrix of A, which is an even hole matrix, either still compose an even hole matrix, or cannot be a hole matrix; (2) The other elements transformed from a non-hole matrix of A cannot compose any hole matrix. So, no sub-matrix of B is an incidence matrix of an odd cycle.

A vector x satisfies $max\{x|x \geq 0; Ax \leq b - n(A)\}$ if and only if there is a vector $y = [y^p, y^N]^T = [x, \mathbf{1} - x]^T$ that satisfies $max\{y|y \geq 0; By \leq b\}$ where $B = [B^P, B^N]$ and $y = \{y_1^P, ..., y_n^P, y_1^N, ..., y_n^N\}^T$, because: $By = [B^P, B^N][x, \mathbf{1} - x]^T = B^P x - B^N x + B^N = Ax + n(A) \leq b$, which implies $Ax \leq b - n(A)$. Based on Theorem 1, $max\{y|y \geq 0; By \leq b\}$ has integral optimum solutions where B is $(0,1)$ balanced and b is an integral vector. This transformation maps integral vectors y into integral vectors x. The proof of $min\{x|x \geq 0; Ax \geq b - n(A)\}$ case is the same. □

By Theorem 2 and 3, we directly conclude that if the path matrix is balanced, the BMC problem can be solved in polynomial time by LP because (1) A_E is $(0, \pm 1)$ balanced; (2) $\begin{bmatrix} \mathbf{0} \\ B \end{bmatrix}$ is an integral vector (It is worth noting that "$b - n(A)$" is no different than "b" when the requirement of "b" is just to be any integral vector); (3) the solution of objective $max \sum_{i=1}^{m} g_i$ is the same as the solution of objective $max(\sum_{i=1}^{m} g_i + \sum_{j=1}^{n} x_j)$.

However, these are not enough to deal with BMkC case because its left-hand-side matrix is not a $(0, \pm 1)$ matrix. We next show how the branch and bound technique achieves pseudo-polynomial optimal solutions in realistic network scenarios.

B. Branch and Bound Enabled Pseudo-Polynomial Optimum

While the path matrix may not always be balanced, ILP solvers using LP-based branch and bound algorithms [6] quickly find optimal solutions for both BMC and BMkC. Because the practical network topology tends to be planar and the odd cycle length is constant, we show that the branch and bound technique returns the optimal solution in pseudo-polynomial time in such cases. We consistently achieve optimal solutions for budgeted sampling point placement problems

Algorithm 2 Greedily Allocating Flows to Sampling Points

Input: $\{p\}$: Set of sampling points; $\{f\}$: Dynamic flows; $\{c\}$: Sampling capacity on points $\{p\}$; $\{f_s\}$: Sampled flows; l_j: Sampling load on point p_j;
Output: $\{f^u\}$: Unsampled flows.

```
Function greedy_alloc:
  {f^u} = {f}; {f^s} = ∅
  while {f^u}! = ∅ do
    {f} = {f} − {f^s}
    {p} = {p} − {p_j | c_j = 0}
    (p_j, l_j, {f_j}) = select_max({f}, {p})
    if l_j > c_j then
      {f_j} = sort({f_j})
      {f^u} = update_unsampled({f_j}, l_j, c_j)
      {f^s} = sample({f_j} − {f^u})
      c_j = 0
    else
      {f^s} = sample({f_j})
      c_j = c_j − l_j
    end
  end
```

using the Gurobi solver [3] in reasonable runtimes across 20,000 experiments on real topologies in Section VI-C.

The branch and bound technique systematically explores all possible integral solutions for optimal selection [13]. Despite its exhaustive nature, the running time is not always exponential since only non-integral decision variables need consideration. Some variables remain integral (0 or 1) even under LP due to certain properties (e.g., balanced). The conditions in Theorem 1 and Theorem 3 are sufficient, though not necessary, for this conclusion. As the LP is solvable in polynomial time L [18], the running time of BMC is mainly influenced by the number K of branch and bound nodes for all odd-cycles. The running time $O(LK)$ is pseudo-polynomial, treating K as a constant (independent of input size) [16], and can be further reduced using parallel branch and bound [5].

Fortunately, our problem is constrained, as practical topologies typically have a limited number of nodes, forming planar graphs. A planar graph with a constant length of odd cycles has a polynomial number of odd cycles [17, Theorem 4], aligning with our real topology cases. Additionally, flows on a topology do not create more odd cycles than those present in the topology. Thus, the pseudo-polynomial optimum case represents the general scenario of the BMC problem, reflecting its weak NP-complete property [16].

V. Sampling Tasks Allocation

We provide a concise overview of the algorithm (Algorithm 2) which allocates flows to sampling points to maximize the number of flows sampled at their required rate. This allocation continuously runs on the controller unless all flows are sampled at a desired rate or all points are fully utilized.

Each sampling point p_j has a traffic load l_j which it attempts to sample at the required rate, and a residual sampling capacity c_j which is the capacity it has left to sample more flows. Sampling points that are not currently overloaded from a sampling perspective are in the *serving pool*. The controller greedily selects the sampling point p_j from the serving pool with the maximum current traffic load l_j. If $l_j > c_j$, indicating the switch is susceptible to sampling overload, some flows or fractions of flows may not be sufficiently sampled at p_j.

To decide which (fractional) flows are sampled at p_j and which must be removed from sampling at p_j and placed in a *sampling request pool*, we prioritize sampling tasks based on two criteria: (1) flows uniquely covered at p_j take precedence over multi-covered flows, and (2) mice flows take precedence over elephant flows. Consequently, an elephant flow with multiple available sampling points is more likely to have its sampling rate reduced at p_j and potentially be sampled at a second sampling point. The controller removes flows from being sampled at this point until the sampling point is not overloaded and places them in the sampling request pool. The controller assigns flows in the request pool to a sampling point on their path if one exists that has a sampling capacity. This process recurs until all flows are sampled or all sampling points are fully loaded.

The controller continuously monitors dynamic flows. When new flows arrive, it routes them and allocates sampling tasks based on the sampling points they traverse and their residual capacity. When flows depart, sampling capacity is regained.

VI. Evaluation

In this section, we quantify the overhead and performance of the coord_sampling algorithm on actual P4-programmable switches. We then evaluate the effectiveness of the placement algorithms in terms of how many flows they can cover k-times within a given budget, their computation time, and their effectiveness at providing sufficient sampling capacity to meet sampling demand using real network flows.

A. Coordinated Sampling Overhead Evaluation

We assess the coordinated sampling algorithm's impact by comparing round-trip time (RTT) and throughput across multiple coordinated sampling settings including port-based sampling using sFlow [23]. sFlow is an industry-standard technology for sampling packets at layer 2.

The testbed topology aligns with Fig. 1, utilizing Arista 7170-32CD switches for $P4\_SW\_1$ and $P4\_SW\_2$. $Host\_1$, $Host\_2$, and $Monitor$ are Intel NUC 10 mini PCs. We present RTT and throughput evaluation results for 6 sampling fraction (sf) settings below on $P4\_SW\_1$ and $P4\_SW\_2$.

- s1:$sf_1 = 0$, $sf_2 = 0$
- s2:$sf_1 = 0$, $sf_2 = 0.25$
- s3:$sf_1 = 0$, $sf_2 = 0.5$
- s4:$sf_1 = 0$, $sf_2 = 1$
- s5:$sf_1 = 0.5$, $sf_2 = 0.5$
- s6: $sFlow.sf_2 = 1$

To eliminate factors like switch queuing, we conduct RTT and throughput tests without introducing background traffic. The results are presented in Fig. 3. RTT evaluation involves generating 10,000 pings between $Host\_1$ and $Host\_2$ under the six different sampling settings. Notably, the coord_sampling program has negligible influence on the RTT compared to both no sampling and sFlow.

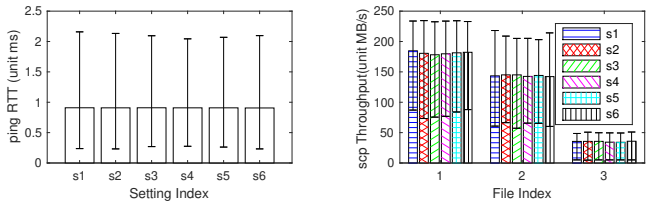

Fig. 3: Algorithm Performance Evaluation on Real Testbed

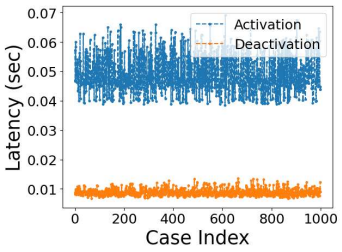

Fig. 4: (De)activation Latency on Arista 7170-32CD

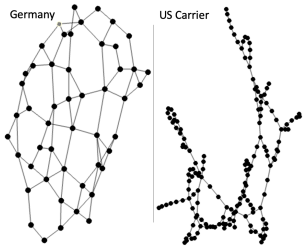

Fig. 5: Real World Topologies

The throughput evaluation utilizes the secure copy protocol (scp) to transfer three different-sized files under the six sampling settings. Each file is transferred 1,000 times, with sizes of 7.5MB, 4.2MB, and 214KB for File 1, File 2, and File 3, respectively. Across all six settings, throughput remains nearly identical to no sampling, exhibiting minimal fluctuations.

B. Coordinated Sampling (De-)activating Latency Evaluation

We evaluate the latency of activating and deactivating coord_sampling algorithm on the real P4-programmable switches in Fig. 1. We use "tcpreplay" to send 100,000 sequenced ICMP packets from $Host\_1$ to $Host\_2$ at a rate of 1,000 packets per second to test the activation and deactivation time 1000 times.

The procedure is as follows: We activate the coord_sampling of $P4\_SW\_1$ from the $Monitor$. The activation time t_a is therefore captured on $Monitor$. At the same time, $Monitor$ keeps listening for any incoming ICMP packets, and the time t_f of the first arrival packet is recorded. We use the value of $t_f - t_a$ as the activation time. We then deactivate the coord_sampling of $P4\_SW\_1$ on $Monitor$ remotely. The deactivation time t_d, the time t_l of the last captured ICMP packet, and the last interval time k between two received ICMP packets are captured on $Monitor$. We use the value of $t_d + k - t_l$ as the deactivation time. Notably, the actual (de)activation time is less than the experimental result because there is a round trip time from the $Monitor$ to the $P4\_SW\_1$.

The activation and deactivation latencies are around 0.05 and 0.01 seconds shown in Fig. 4. Combined with the throughput and RTT results, this indicates our implementation is suitable for scaleable dynamic coordinated sampling. Next, we show the coverage and runtime of the optimal budgeted placement algorithm and its advantage to sample sufficiently in realistic settings.

C. Budgeted Placement Calculation Evaluation

We evaluate the sampling point placement solutions of BMK and BMkC produced by: (1) the ILP solver Gurobi [3] denoted as **optimal algorithm**; (2) a **greedy algorithm** that

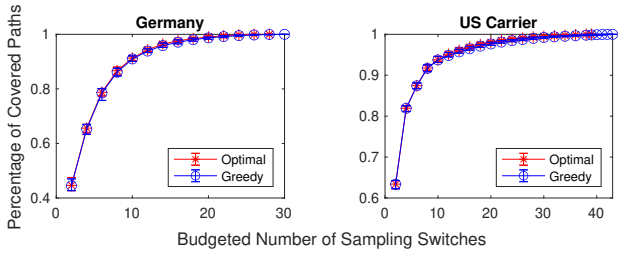

Fig. 6: Coverage Percentage Comparision of BMC

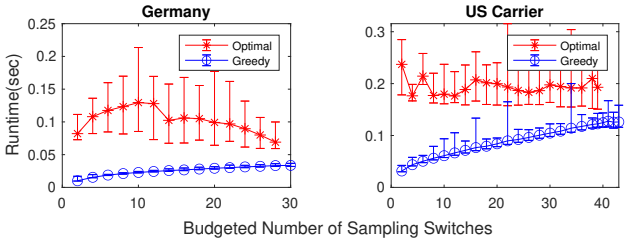

Fig. 7: Runtime Comparision of BMC

greedily selects the placement points using the maximum number of uncovered flows in each iteration until the number of chosen points reaches the budget. We also evaluate the runtime of the two algorithms. We run our experiments based on two real-world topologies [4], [22] shown in Fig 5.

On the US carrier topology, we select nodes with degrees of 2 to construct a pool from which we randomly select the sources and destinations. On the Germany topology, all nodes are eligible to be selected as a source or destination. Based on the Germany and US Carrier topologies, we randomly select 1,000 and 1,500 different pairs of sources and destinations 100 times. We show the max, min, and average of the result based on the 100 experiments for each setting on each case.

1) BMC: Given a budgeted number of sampling points, the goal is to determine the sampling point placement to maximize the number of 1-covered flows.

In 4800 experiments on two real topologies, the result of the optimal algorithm is a little better than the greedy algorithm, as shown in Figs. 6 and 7. Over 90% of the flows are covered with only half the number of the sampling points required to cover all flows which reflects the necessity to find a balance between the path coverage and cost. The runtime of the optimal algorithm is comparable with the greedy algorithm in $10^{-2} \sim 10^{-1}$ seconds.

2) BMkC: Given $k = 2$ and a budgeted number of sampling points, the goal is to determine the sampling point locations that maximize the number of 2-covered flows while ensuring all flows are 1-covered.

In 4,500 experiments, shown in Fig. 8 and 9, the optimal algorithm consistently outperforms the greedy algorithm and its runtime is comparable with the greedy algorithm in $10^{-2} \sim 10^{-1}$ second level. Notably, when the budgeted number of sampling points is at or slightly above the minimum required to cover all flows once, the optimal algorithm achieves a significantly higher number of 2-covered flows than the greedy algorithm. For example, with a budget of 26 for the Germany topology, the optimal algorithm achieves

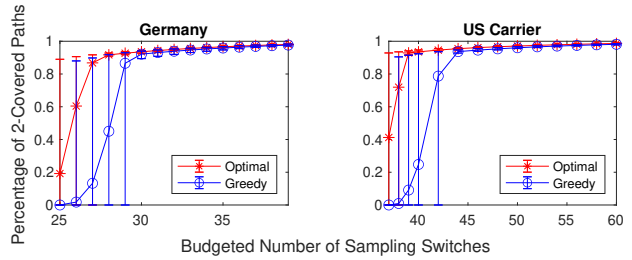

Fig. 8: Coverage Percentage Comparision of BMkC

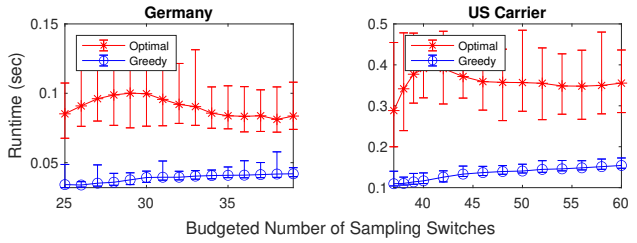

Fig. 9: Runtime Comparision of BMkC

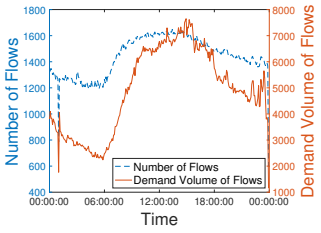

Fig. 10: Dynamic Flows on Germany Topo

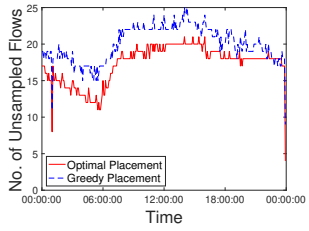

Fig. 11: Placement Effectiveness Comparision

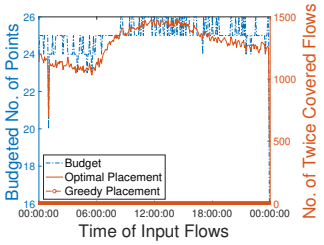

Fig. 12: No. of 2-coverage by Optimal and Greedy with Different Budget and Input Flows

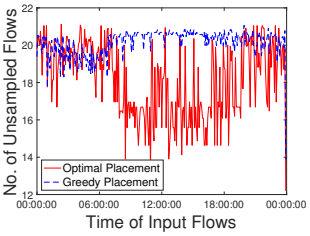

Fig. 13: Placement Effectiveness Comparison with Different Input Flows

around 90% 2-covered flows in 68 out of 100 randomized flow set instances (60.43% 2-coverage on average), while the greedy algorithm fails to achieve any 2-covering in 98 out of 100 instances (1.76% 2-coverage on average). This highlights the potential risk of substantial sampling loss when using the greedy algorithm in such scenarios, which becomes critical when required sampling rates are high and multiple sampling points are needed to sample flows at sufficient rates, as evaluated in the next subsection.

D. Budgeted Placement Effectiveness on Dynamic Flows

To evaluate the budgeted placement effectiveness of the optimal and greedy methods for the coordinated sampling in practice, we use the Germany topology with a real-world dataset [22] of flows on that topology.

In detail, we use both methods to produce the budgeted sampling point placement of BMkC with $k = 2$ on the Germany topology, feed 24-hour dynamic flows to the network, and allocate dynamic flow sampling tasks to the placed sampling points for best-effort sufficient sampling introduced in Section V. We capture the number of unsampled/insufficientlysampled flows for both placement results to quantify the budgeted placement effectiveness.

The dynamic flows on the Germany Topology, illustrated in Fig. 10, are recorded every 5 minutes, providing information on (src, dst) pairs and corresponding demand values, l, which we use as the flow sampling load. Each sampling point p_i is set with a maximum demand capacity of $c = 50$. When a sampling point p_i is selected, and the total demand value of flows on p_i exceeds 50, only a subset or fractional flows are sampled. Consequently, unsampled or fractional sampled flows await the next opportunity for sufficient sampling.

1) Optimal vs Greedy Placement with Input of All Flows: Here we consider all 1225 possible paths on the Germany topology as input paths for budgeted placement calculation. To maximize cost savings, we set the budget to 28 sampling points, which under the optimal algorithm is enough to cover all flows at least once. The optimal placement achieves 1117 2-covered flows, while the greedy placement does not yield any 2-covered flows. In this case, if a sampling point becomes overloaded, with the greedy placement, there is no alternative sampling point and thus flows are un- or under-sampled.

We greedily allocate 24-hour flows on the sampling points calculated by optimal and greedy placement algorithms. The results are shown in Fig. 11. The optimal placement takes advantage of the multiple sampling points on many of the paths and enables on average 5 more flows to be sufficiently sampled per minute than with greedy placement, showcasing its robustness and superiority for coordinated sampling.

2) Optimal vs Greedy Placement with Input of Different Flows: To illustrate the impact of input flows on optimal and greedy placement strategies, we execute the placement algorithms using flows from different times (x-axis) of the day as input, with a budget that allows for the lowest cost solution to cover at least 90% of the flows twice using optimal placement, the required budgets range mostly between 24-26 switches shown in Fig. 12. Note with these budgets, the greedy algorithm is incapable of achieving any 2-covered flows.

Within a budget range of approximately 25, the optimal algorithm achieves 1,000-1,500 2-covered flows, while the greedy algorithm does not achieve 2-coverage but only 1-coverage. Employing these sampling point placement settings, we introduce 24-hour flows to the network and employ the greedy allocation algorithm for sufficient sampling. The results are shown in Fig.13. The optimal placement surpasses the greedy placement when the input flows are from 6:00:00 to 18:00:00 and is especially stable from 8:00:00 to 11:00:00.

As shown in Fig. 10, the number of flows is high from 8:00:00 to 18:00:00. We conclude that an increase in the number of input paths for optimal placement calculation is associated with a decrease in the average number of unsampled flows in dynamic scenarios. However, this phenomenon is

not obvious for the greedy placement algorithm because the greedy algorithm tends to select nodes with high betweenness centrality regardless of the total number of flows.

VII. Related work

Recent works on network monitoring adopt external monitors building on an SDN architecture to avoid degrading switch performance, as do we. Those based on OpenFlow [7] forward entire packets to the monitor, wasting network bandwidth if only portions of a packet are required. Among P4-enabled monitor systems, Ding, et al. [14] focus on a greedy incremental P4-programmable switch deployment strategy, while Jonatas, et al. [21] select locations for in-band telemetry (INT) to append statistics without degrading performance. These are different from our problem setting and purpose.

Among the works that consider sampling scalability, Sogand, et al [24] separate sampling decisions on short and long flows to enhance sampling efficiency. Du, et al [15] adapts sampling rate for different flows with non-duplicate sampling [28]. Reuven, et al [11] aim at a best-effort single point per-flow sampling allocation regardless of sampling rate demand. Conversely, we study and coordinate flow-based sampling at multiple points. Another coordinated sampling work [26] uses a hash-based method to guarantee each flow is only sampled at one router to avoid duplicated sampled packets. Therefore, it does not support sampling flows at multiple points, as we do.

Furthermore, existing sampling point placement methods [27], [30] typically use heuristic greedy algorithms without examining the characteristics of practical network flows on the ability to obtain optimal solutions, as we do.

VIII. Conclusion

Our coordinated sampling solution enhances the scalability of flow sampling across networks. The placement algorithm maximizes the number of multi-covered flows within a budget, enabling an increased count of sufficiently sampled flows. These lightweight and fast algorithms are well-suited for dynamic sampling activation.

References

[1] Arista price list 2024. https://itprice.com/arista-price-list/7170.html. Accessed on 2024-03-11.

[2] Coordinated sampling on p4-programmable switches. https://github.com/mzc796/coord_sampling. Accessed on 2023-11-10.

[3] Gurobi mixed-integer programming (mip) – a primer on the basics. https://www.gurobi.com/resource/mip-basics/. Accessed on 2023-11-10.

[4] Topology zoo. http://www.topology-zoo.org/dataset.html, 2012. Accessed on 2023-10-31.

[5] David A Bader, William E Hart, and Cynthia A Phillips. Parallel algorithm design for branch and bound. *Tutorials on Emerging Methodologies and Applications in Operations Research: Presented at INFORMS 2004, Denver, CO*, pages 5–1, 2005.

[6] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[7] Samaresh Bera, Sudip Misra, and Abbas Jamalipour. Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn. *IEEE Journal on Selected Areas in Communications*, 37(3):530–539, 2019.

[8] Claude Berge. Balanced matrices. *Mathematical Programming*, 2:19–31, 1972.

[9] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[10] Kevin Butler, Patrick McDaniel, and William Aiello. Optimizing bgp security by exploiting path stability. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 298–310, 2006.

[11] Reuven Cohen and Evgeny Moroshko. Sampling-on-demand in sdn. *IEEE/ACM Transactions on Networking*, 26(6):2612–2622, 2018.

[12] Michele Conforti, Gérard Cornuéjols, and Kristina Vušković. Balanced matrices. *Discrete Mathematics*, 306(19-20):2411–2437, 2006.

[13] Koen MJ De Bontridder, BJ Lageweg, Jan K Lenstra, James B Orlin, and Leen Stougie. Branch-and-bound algorithms for the test cover problem. In *Algorithms—ESA 2002: 10th Annual European Symposium Rome, Italy, September 17–21, 2002 Proceedings 10*, pages 223–233. Springer, 2002.

[14] Damu Ding, Marco Savi, Gianni Antichi, and Domenico Siracusa. An incrementally-deployable p4-enabled architecture for network-wide heavy-hitter detection. *IEEE Transactions on Network and Service Management*, 17(1):75–88, 2020.

[15] Yang Du, He Huang, Yu-E Sun, Shigang Chen, and Guoju Gao. Self-adaptive sampling for network traffic measurement. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[16] Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the*, 1979.

[17] Seifollah Louis Hakimi and Edward F Schmeichel. On the number of cycles of length k in a maximal planar graph. *Journal of Graph Theory*, 3(1):69–86, 1979.

[18] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[19] Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

[20] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[21] Jonatas Adilson Marques, Marcelo Caggiani Luizelli, Roberto Irajá Tavares da Costa Filho, and Luciano Paschoal Gaspary. An optimization-based approach for efficient network monitoring using in-band network telemetry. *Journal of Internet Services and Applications*, 10:1–20, 2019.

[22] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[23] Peter Phaal. sflow version 5. https://sflow.org/sflow_version_5.txt. Accessed on 2023-11-10.

[24] Sogand Sadrhaghighi, Mahdi Dolati, Majid Ghaderi, and Ahmad Khonsari. Flowshark: Sampling for high flow visibility in sdns. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 160–169. IEEE, 2022.

[25] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[26] Vyas Sekar, Michael K Reiter, Walter Willinger, Hui Zhang, Ramana Rao Kompella, and David G Andersen. csamp: A system for network-wide flow monitoring. 2008.

[27] Kyoungwon Suh, Yang Guo, Jim Kurose, and Don Towsley. Locating network monitors: complexity, heuristics, and coverage. *Computer Communications*, 29(10):1564–1577, 2006.

[28] Yu-E Sun, He Huang, Chaoyi Ma, Shigang Chen, Yang Du, and Qingjun Xiao. Online spread estimation with non-duplicate sampling. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2440–2448. IEEE, 2020.

[29] Oksana Yevsieieva and Seyed Milad Helalat. Analysis of the impact of the slow http dos and ddos attacks on the cloud environment. In *2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, pages 519–523. IEEE, 2017.

[30] Seunghyun Yoon, Taejin Ha, Sunghwan Kim, and Hyuk Lim. Scalable traffic sampling using centrality measure on software-defined networks. *IEEE Communications Magazine*, 55(7):43–49, 2017.