Efficient Lightweight Coordinated Sampling for Dynamic Flows: Theory and Implementation

Mingming Chen, *Member, IEEE*, Thomas F. La Porta, *Fellow, IEEE*, Trent Jaeger, *Fellow, IEEE*, and Srikanth V. Krishnamurthy, *Fellow, IEEE*

***Abstract*—As cyber-attacks on networks become stealthier, monitoring techniques relying on low-rate packet sampling may prove insufficient to detect attacks. While various methods, such as truncating packets, flow-based sampling, and adaptive sampling rates, have been proposed to enhance detection rates and ease capability limitations, it remains challenging to perform sufficient sampling at line speed and high rates at a single sampling point due to limited CPU or bandwidth capacity and fluctuating network traffic. To address these challenges, we propose CoordSamp, a system that distributes the sampling workload across multiple sampling points and coordinates their actions to avoid duplicate sampling of the same packet. This design enables scalable, resource-aware monitoring—particularly suited for dynamic, agentless cloud-based environments—relying solely on network-level deployment that can be dynamically assigned and adjusted by the provider. We develop 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. At its core, CoordSamp separates *offline placement*—the budgeted selection of sampling points—from *online allocation*—the capacity-aware assignment of sampling tasks—allowing practical deployment in hybrid networks that combine programmable and legacy switches. We formulate sampling point placement as budgeted maximum multi-coverage problems, solving them optimally in pseudo-polynomial time. Our system far outperforms those based on greedy placement along many key dimensions.**

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

I. Introduction

NETWORK traffic monitoring is an important technique used for both network management and security purposes. Due to limitations in processing capacity and bandwidth, switches cannot typically forward every packet to a monitor. To address this, two main approaches are adopted: traffic summarization (e.g., sketch [1]) and sampling [2]. While summarization is effective for flow-level metrics, it lacks the fidelity required for detecting fine-grained, packet-level attacks. In contrast, packet sampling outputs raw packets, providing the detailed context necessary for payload-aware forensics and intrusion detection, which is essential in operational security practice [3]. Packet sampling selectively forwards individual packets to external monitors, preserving the rich per-packet context essential for identifying anomalies and threats—though it is constrained by limited bandwidth. As a result, researchers strive to maximize monitoring effectiveness within limited sampling capacity [4], [5].

Regardless of the techniques used to sample at switches, sampling flows with a sufficient sampling rate at line rate under heavy traffic load may not be achievable at 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 [6] even if programmable ASICS are used. For example, experiments on physical switches with 10GB/s ports showed that copying and forwarding every raw packet from 2GB traffic to an external monitor caused 70% throughput overhead in average [7]. Such performance degradation significantly impacts network efficiency.

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 [8]. The unpredictable dynamics of traffic require the sampling points, sampling rate, and sampling strategy to change with them. Our system provides multi-point coordinated sampling, smart sampling point placement, and dynamic sampling point configuration to meet this need.

Unlike prior sampling works that focus primarily on single-point sampling optimization [4], [5], we propose CoordSamp, an efficient multi-point sampling scheme to achieve high-rate, line-speed *coordinated* sampling by placing multiple sampling points along the paths. This work extends our previous study [9], which introduced lightweight coordinated sampling for dynamic flows under budget constraints. In this extended version, we include a broader comparison with related distributed monitoring techniques (§ II-A), a proof-of-concept demonstration of the algorithm's robustness (§ II-D), an expanded theoretical treatment (§ IV), expanded sampling points placement experiments on two real datacenter topologies(§ VI-C), expanded related work (§ VII), and new discussions of security applications (§ VIII).

Several prior works have explored distributed network monitoring through coordination-free estimation [10], [11], [12], distributed sketches [13], [14], and distributed sampling

Received 2 November 2024; revised 13 June 2025, 11 November 2025, and 2 December 2025; accepted 8 December 2025; approved by IEEE Transactions on Networking Editor S. Alouf. Date of publication 15 December 2025; date of current version 12 January 2026. This work was supported by the U.S. Army Combat Capabilities Development Command Army Research Laboratory Cyber Security Collaborative Research Alliance (ARL Cyber Security CRA) under Grant W911NF-13-2-0045. *(Corresponding author: Mingming Chen.)*

Mingming Chen is with the Department of Computer Science, Kansas State University, Manhattan, KS 66503 USA (e-mail: mingc@ksu.edu).

Thomas F. La Porta is with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802 USA (e-mail: tfl12@psu.edu).

Trent Jaeger and Srikanth V. Krishnamurthy are with the Department of Computer Science and Engineering, University of California, Riverside, Riverside, CA 92521 USA (e-mail: trentj@ucr.edu; krish@cs.ucr.edu).

Digital Object Identifier 10.1109/TON.2025.3644238

[15], [16]. While distributed sketching and summarization techniques have achieved impressive scalability, they address a fundamentally different goal—approximating flow-level statistics rather than exporting raw packets. Our work instead targets the complementary packet-sampling modality, where coordination must prevent redundant packet exports. We focus on the complementary problem of efficiently coordinating packet sampling across distributed switches, ensuring fidelity without redundancy. Our framework enables fine-grained, packet-level coordination across multiple sampling points, providing packet-level visibility—an essential capability for identifying stealthy or low-rate threats. Our design achieves this coordination using lightweight, in-network logic on P4-programmable switches,[1] while intentionally offloading complex traffic analysis to external, lower-cost monitors, instead of relying on the expensive P4-programmable switches for that task. This analysis component is part of the broader system architecture but lies outside the scope of this paper.

CoordSamp serves as a packet broker that unifies two essential functions in large-scale monitoring: scalable distributed sampling and packet deduplication. Open-source monitors Suricata [18], Zeek [19], and its commercial version Corelight [20] rely on mirrored packets for analysis yet suffer from CPU and bandwidth waste when duplicates are sent from multiple points. CoordSamp naturally fits as a packet broker, ensuring non-duplicated, budget-aware packet export to these monitors.

Additionally, our framework supports flexible sampling granularity, from coarse port-based sampling (e.g., match on ingress port) to fine-grained per-flow sampling (e.g., match on src-dst pair), enabling hybrid static/dynamic flow assignment. This design allows operators to balance the tradeoff between the sampling granularity and the real-time coordination overhead on the controller, adapting to different deployment constraints and monitoring goals. In this paper, we focus on the case of fine-grained, per-flow sampling to demonstrate the capabilities and challenges of CoordSamp.

Multi-point sampling offers advantages in sampling sufficiency by distributing the packet-mirroring workload (i.e., the volume of sampled packets exported to monitors) across multiple switches. This approach reduces pressure on individual switches, mitigates resource bottlenecks, and improves scalability. The load-sharing design enables more efficient utilization of constrained programmable hardware. Our study focuses on multi-point sampling for load-balancing purposes. Other use cases may also benefit from deploying multiple sampling points, but these are for different purposes. For example, sampling a flow before and after a firewall enables verification of middlebox behavior and detection of stealthy rule bypassing [21]. The formulation of such scenarios differs from our load-balancing setting, as it adds extra constraints on the sampling points placement (e.g., before and after firewalls). We discuss how our framework can be extended to support security- and resilience-oriented scenarios in Section VIII.

A key challenge in realizing the benefits of multi-point sampling lies in coordinating the sampling actions to avoid duplicate samples and to preserve the ability to reconstruct the original sample sequence, even in the presence of packet drops, swamping, or bursts. To address this, we propose a P4 program installed on the P4-programmable switches as the sampling points to automatically coordinate the sampling across multiple sampling points to avoid duplicated sampling. We choose P4-programmable switches as the sampling points because their sampling functionality can be programmed and tuned in real-time on the ASIC without CPU constraints.

[1]P4 is a domain-specific language for programming packet processing pipelines in network devices [17].

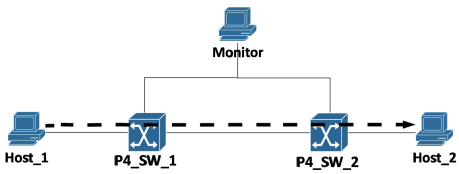


Fig. 1. Testbed topology.

Given the substantial cost difference between P4-programmable and OpenFlow switches (up to a factor of 10 [22]), we propose to perform *offline placement* of a budgeted number of programmable switches as potential sampling points to maximize the average number of flows that can be sampled at sufficient rates, and to enable *online activation* of multiple sampling points to share the load dynamically. To address the sampling point placement problem, we formulate the placement problem as a budgeted maximum (multi-)coverage problem, which is NP-complete in general. However, we show that under the structure of our networking scenario, the problem exhibits weak NP-completeness. This is established by leveraging the balanced matrix property and analyzing the problem using a branch-and-bound technique.

A. System Overview

Fig. 1 shows the system architecture. Hosts connect via a network with P4-programmable switches that sample packets at configurable granularity and send them to monitors for analysis. When the SDN controller detects an overloaded sampling point, it redistributes the sampling load, splitting it among multiple points if needed. Our CoordSamp algorithm at each sampling point ensures no duplicate samples—even if packets are dropped or swapped during transmission.

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) performing *offline placement* of the budgeted number of programmable switches to maximize the number of flows that have sufficient coverage to handle sampling load, and (3) performing *online assignment* of flows to sampling points dynamically at runtime.

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 [11], especially when employing popular routing protocols that result in path stability [23].

In this paper, we make the following contributions:

- We introduce CoordSamp to efficiently coordinate sampling at multiple points using lightweight in-band signaling via packet tags. Experiments on P4-programmable switches show negligible runtime performance overhead, with sampling rule activation latency of approximately 0.05 seconds and deactivation latency of 0.01 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.
- We evaluate our optimal placement with dynamic traffic in real-life topology, and it shows that the sampling points placed using our budgeted optimal algorithm provide the capacity to sufficiently sample more flows than those placed by the greedy algorithm under the same budget.

II. System Design

Design overview. CoordSamp is a packet-sampling system with a two-phase architecture: (1) an *offline placement* phase that selects sampling points under a budget to maximize coverage, and (2) an *online allocation* phase that assigns sampling tasks in a capacity-aware manner at runtime, enforcing packet-level coordination via in-band tags. This separation allows the controller to adapt to traffic dynamics while respecting budget constraints in hybrid network settings.

In CoordSamp, the sampling load is distributed across multiple points in the network with coordination to efficiently utilize switch sampling capacity. CoordSamp also supports flexible sampling granularity, configurable via the "match" field, enabling either coarse grouping (e.g., match on ingress port) with no real-time coordination cost on the controller or fine-grained per-flow sampling (e.g., match on source-destination pairs) with real-time sampling tasks assignment by the controller. This flexibility helps balance the tradeoff between sampling granularity and real-time coordination overhead on the controller. In this paper, we use per-flow sampling to demonstrate CoordSamp's challenges and capabilities.

Because a single flow may traverse multiple switches, the sampling algorithm must avoid redundant packet samples while keeping the data analyzable. To maintain the usefulness of these samples, the controller must reconstruct ordered packet sequences from different sampling points for timing analysis, anomaly detection, and retransmission identification. Therefore, our CoordSamp system is designed to meet:

Goal (1): Avoid duplicate sampling of the same packet at different sampling points.
Goal (2): Ensure the controller correctly orders sampled packets, even when captured at different locations.

We introduce a method for achieving coordinated sampling among sampling points with minimal overhead. We compare the state-of-the-art distributed network monitoring systems to motivate our CoordSamp, an overview of the CoordSamp algorithm, and the underlying switch architecture, followed by the algorithm's details and implementation.

A. Motivation and Taxonomy

The distributed network monitoring approaches share a common goal: enabling scalable, resource-aware monitoring in distributed settings. However, prior systems do not fully resolve the tension between measurement fidelity, overhead, and flexibility. None of them considers the budgeted sampling point placement. Many are tailored to specific assumptions—fixed topologies, static traffic, or single-point measurement per flow—and lack support for adapting coordination granularity to evolving network conditions or monitoring objectives. This motivates our work, which seeks to develop a more general framework for analyzing and designing coordination mechanisms that are both efficient and tunable.

Distributed network monitoring systems face an inherent tradeoff between accuracy and scalability. Monitoring at multiple switches improves visibility, but also introduces challenges: without careful design, duplicated measurements can inflate resource usage, complicate analysis, or distort statistics. One class of solutions embraces duplication tolerance, using compact encodings and post-hoc merging to manage redundancy—at the cost of decoding complexity, delayed or compromised accuracy, and reduced control over measurement granularity. Others aim to prevent duplication through coordination. However, coordination itself introduces control-plane overhead and/or increases switch logic complexity. A central design question is therefore: how should coordination be applied/avoided to balance fidelity, efficiency, and flexibility?

To clarify these tradeoffs, we classify the distributed networking monitoring studies based on coordination granularity and mechanisms in Table I with three representative categories:

- **Coordination-free Estimation** eliminates coordination. Each switch independently samples or encodes packets (typically via hash-based mechanisms), and a centralized collector merges the results. These schemes, such as *Routing-Oblivious* [10], *UnivMon* [11], and *FlowRadar* [12], are scalable to deploy, but tolerate measurement duplication and rely on statistical reconstruction to estimate traffic metrics.
- **Flow-level Coordination** is managed on a per-flow basis. Realizations include: *(i) Probabilistic anti-correlation* (e.g., [24]), which biases local selections to reduce overlap in expectation (duplicates may still occur); *(ii) Flow-level exclusive coordination*, where the workload is partitioned by flow, and each flow's workload is assigned to exactly one switch—this appears both in *distributed sampling* [15], [16]) and in *cooperative sketching* [14]; Both distributed sampling works [15], [16] implement *flow-level exclusive* sampling (one sampler per flow); CoordSamp instead coordinates at the *packet level* with in-band tags, which strictly generalizes exclusive assignment under budgets.
- **State-sharded Coordination** (e.g., *Dist. Sketch* [13]) allows each flow to traverse multiple switches, where each switch updates a disjoint shard of a global sketch along the path, and the shards are later merged at the controller. Because packet sampling outputs raw packets rather than aggregated flow statistics as sketching does, the coordination mechanisms in these two domains serve fundamentally different purposes and are not directly comparable. We next introduce the fine-grained coordination mechanism tailored to packet sampling.
- **Packet-level Coordination** provides fine-grained control by allowing a single flow to be **sampled** at multiple switches while ensuring duplication-free monitoring. It enables more effective utilization of sampling capacity across the network than flow-level coordination. Our CoordSamp supports packet-level coordination through packet **tagging**: When a switch samples a packet, it adds a small tag or modifies a designated field in the packet header. This tag informs downstream switches that the packet has already been sampled, allowing them to

TABLE I
COMPARISON OF COORDINATION MECHANISMS ON DISTRIBUTED NETWORK MONITORING ARCHITECTURES

Coordination Granularity	Coordination Method	Coord. Overhead on Switch	Coord. Overhead on Controller	Coordination Method Strengths	Coordination Method Limitations
Coordination-Free Estimation: Duplication-tolerant estimation without explicit coordination [10]–[12].	**Hash-based Distribution:** Switch hashes packet or flow ID independently to sample or encode traffic, without coordination across switches. Duplication is tolerated or leveraged through centralized post-processing.	Parse each packet and compute multiple hash values of it. **Prog. pipeline Cost (Hash Computation):** Num. of stages: $1 \rightarrow 3$ Max. atoms/stage: $3 \rightarrow 12$ (complex arithmetic logic unit + register logic)	May install shared measurement logic **during setup**; **no real-time coordination overhead**.	▲ Duplication-tolerant flow statistics estimation. ▲ No real-time overhead on controller.	▲ Heavy hash computation overhead. ▲ Coarse flow statistics estimation. ▲ Complex decoding-based Monitor-side processing to merge sampled data.
Flow-level Coord.: Control duplicated monitoring at flow granularity: ∗ ***Probabilistic:*** Local hash+TTL grading; each switch monitors only a small set of flows; overlap reduced in expectation (non-exclusive) [24]. ∗ ***Exclusive:*** One flow is monitored at only one switch [14] [16] [15].	**TTL-based Anti - Correlation:** Independent per-switch grading biases selections across hops; reduces overlap, not exclusive [24].	One-time params per switch; no real-time coordination overhead; per-switch epoch telemetry.	▲ Limits duplication in expectation. ▲ No real-time overhead on the controller.	▲Coarse flow-level coordination. ▲ Heavy hash computation on switches. ▲ Overlap persists.
	Preconfigured Exclusive Coord.: CountMax assigns each flow to a single ingress or egress switch using a deterministic (odd/even) hash partition [14].	Compute and distribute hash rules or thresholds **during setup; no real-time overhead**.	▲ Duplication-free. ▲ **Preconfigured:** No real-time overhead on the controller. ▲ **Dynamic:** Adaptive sampling configuration based on live traffic.	▲ Coarse flow-level coordination. ▲ Heavy hash computation overhead on switches. ▲ **Preconfigured:** Limited adaptability to dynamic traffic due to static rule design. ▲ **Dynamic:** Real-time coordination overhead on the controller.
	Dynamic Exclusive Coord.: Controller installs per-flow mapping based on hashed flow IDs so each flow maps to exactly one switch [15], [16].	**Real-time per-flow assignment** and rule installation based on a global view.		
State-sharded Coord.: One flow is measured at multi-switches; each switch updates a disjoint shard of the sketch state [13].	Logical sketch per path jointly built by switches on that path; each switch updates a disjoint shard shared across paths, later merged by the controller [13].	Offline setup and periodic state-sharded sketch aggregation; **no real-time (per-packet) coordination**.	▲ Zero redundant increments. ▲ Path-wide load sharing.	▲ Sensitive to path changes. ▲ Monitor-side merging.
Packet-level Coordination: Avoid duplicated sampling while allowing one flow to be monitored at multiple switches [9].	**Tag-based Coordination:** Switch tags the sampled packet before sending it out; downstream switches check the tag and skip previously sampled packets to guarantee duplication-free.	Parse each packet and read/write tag. **Prog. pipeline cost (Read/Write tag)** Num. of stages: 1 Max. atoms/stage: $1 \rightarrow 3$	Sampling parameters are configured **during setup**; **optional real-time** updates may occur in dynamic mode to adjust goals or granularity.	▲ Duplication-free. ▲ Fine-grained packet-level coordination. ▲ Lightweight tagging overhead on switch. ▲ Flexible static and/or dynamic assignment.	▲ High export volume to the monitor. ▲ Monitor-side merging and analyzing data. (Acceptable tradeoff in low-cost commodity monitors).

skip redundant sampling. This ensures that each packet is counted only once, even when traversing multiple measurement points. COORDSAMP is lightweight and data-plane friendly, and supports a flexible trade-off between sampling granularity and control-plane coordination overhead. Despite higher export and merging overhead, this acceptable trade-off shifts complexity to commodity monitors instead of programmable switches.

While our taxonomy summarizes coordination mechanisms, it also highlights a fundamental distinction between two major monitoring paradigms: *sketching* and *packet sampling*. **Sketching-based approaches** [13], [14] are designed for *aggregated flow statistics*—such as estimating flow sizes, heavy hitters, or traffic changes. They summarize traffic using compact in-switch data structures that approximate per-flow statistics, achieving scalability and low export cost but necessarily forgoing per-packet visibility. In contrast, **sampling-based approaches** preserve *raw packets* for tasks such as intrusion detection, policy verification, and forensic analysis. These methods provide fine-grained visibility but can generate high export traffic. When distributing the sampling workloads for scalability, coordination is required to prevent redundant packet sampling to prevent bandwidth waste.

Although monitoring task assignment for sketching [13] and sampling [9] share structural similarity—both seek to avoid redundant monitoring on multiple points—their coordination considerations and resource constraints are fundamentally different. Sketching coordinates state updates under switch memory limits, whereas sampling coordinates packet selection under export bandwidth constraints. These distinct bottlenecks arise from differing objectives: sketching aggregates flow-level statistics, while sampling preserves packet-level observability.

Existing **distributed monitoring systems**—including sketch-based [13] and sampling-based [14], [15] designs — typically assume a homogeneous SDN environment in which all switches can be monitoring points (either OpenFlow or P4-programmable). When only a budgeted subset of switches can monitor, our *budgeted maximum multi-coverage placement* strategy can serve as a complementary component to these frameworks, helping determine where to deploy programmable monitoring functions (e.g., sketches or sampling) and where to rely on legacy forwarding. This enables coordinated monitoring optimization across hybrid networks that combine programmable and legacy switches.

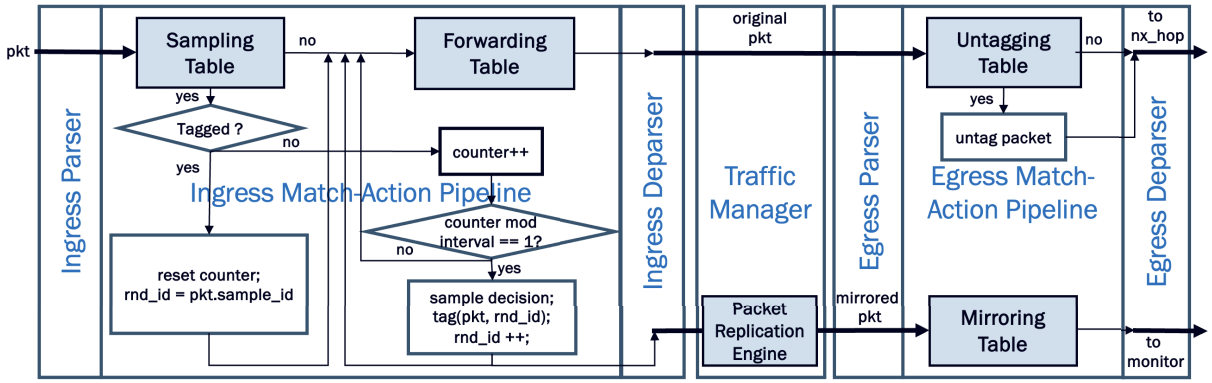


Fig. 2. CoordSamp workflow on TNA pipeline.

In comparison, CoordSamp offers several key advantages:

- Fine-grained, duplication-free packet-level coordinated sampling that minimize resources usage by avoiding real-time hashing (whereas existing distributed sampling works only offer flow-level coordination); tagging introduces minimal pipeline overhead.
- Flexibility tradeoff between sampling granularity and real-time coordination overhead: supports both static coarse port-based sampling and dynamic per-flow sampling depending on scenario and condition.
- Budget-awareness: our system enables optimal placement of sampling tasks to minimize cost and maximize visibility—an area largely overlooked by prior works.

These advantages make CoordSamp well-suited for hybrid network that demand packet-level analysis—such as for detecting stealthy or low-rate attacks—while operating under tight monitoring budgets or hardware constraints.

B. P4/TNA Realization

Our CoordSamp 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 CoordSamp 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. The core logic of CoordSamp resides primarily in the ingress pipeline.

C. Packet-Level Coordination Algorithm

The key idea behind preventing duplicated sampling—while allowing multiple switches to sample the same flow—is to determine whether a packet has already been sampled before making a new sampling decision. To achieve this, CoordSamp instructs the sampling switch to put a tag in packet header immediately after sampling it, before forwarding. Downstream switches can then inspect the tag to determine if the packet has already been sampled and skip redundant sampling. While several header fields (e.g., VLAN [25]) can be used to carry the tag, CoordSamp utilizes the IP "Options" field to remain standard. The tag is removed at the final sampling point to minimize interference with normal forwarding.

The coord_sampling algorithm (Algorithm 1) is implemented on the TNA architecture in Fig. 2. The P4 code is available on GitHub [26]. 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.

To achieve Goal (1)—preventing duplicate sampling of the same flow across multiple sampling points—our algorithm maintains two parameters for each flow f_i:

- $\texttt{counter}_i$: counter of packets for f_i in a round;
- $\texttt{interval}_i$: dynamically configured sampling interval (i.e., the inverse of the sampling fraction/frequency).

In a traditional single-point flow-based sampling system, a packet is sampled when the following condition is met:

Condition (A): $\texttt{counter}_i \bmod \texttt{interval}_i == 1$

However, Condition (A) is insufficient in multi-point setups, where issues of packet drops or reordering may lead to duplicate samples. Therefore, we introduce a second requirement:

Condition (B): the packet is *not tagged*.

Therefore, a switch samples a packet only if **both Condition (A) and Condition (B)** are satisfied. If Condition (B) is not met, the switch resets $\texttt{counter}_i$ to prevent overlapping sampling decisions in subsequent packets.

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.,

Algorithm 1 CoordSamp on P4-Prog. Switches

Data: 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.
Result: is_out, id_i^{rd}.

```
(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
```

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}.

Multiple packets can belong to the same round, but they are sampled from different points along the flow path. A round is defined as a group of sampling opportunities, with its size equal to the number of sampling points on that path. Since flows are continuous, a single round cannot fully capture the flow. However, packet metadata—such as sampling location and flow direction—can work with sampling rounds, assisting in reconstructing the correct order when needed. 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 $O(1)$, performing one table lookup and a constant number of conditional actions. 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.[2] 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.

Finally, CoordSamp is **table-space-efficient**, does not materially increase TCAM or flow-table usage. Exact/Longest Prefix Matching (LPM) matches reside in SRAM/LPM (no ternary required); if TCAM is used, sampling can be attached as an action to existing forwarding entries (forward+sample), avoiding new TCAM entries. So, memory usage grows proportionally to the number of flows rather than expanding the table size. When table space is tight, operators can adopt coarser, SRAM-friendly matches (e.g., matching ingress-port); packet tags still prevent duplicates. Finally, the number of flow entries roughly tracks sampling capacity, which our runtime allocator already enforces: when a point is overloaded, the controller shifts sampling to other points within the budget.

D. Prototype Demonstration

To avoid duplicated sampling amidst drops and reordering,[3] we enable sampled packets to be tagged upstream, so downstream switches skip already-sampled packets—preserving synchronization across all cases. Fig. 3 demonstrates Algorithm 1 in the scenarios of normal transmission, packet dropping, and packet swapping between points. The algorithm promptly restores synchronization after anomalies by resetting counters and round identifiers when marked packets are processed—requiring no special inter-switch messaging.

Coordination among sampling points to avoid duplicated sampling 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. Our solution comprises two stages: **sampling point placement** and **sampling task allocation**, where placement is a physical, pre-deployment decision and allocation occurs at runtime. The placement algorithm solves a budgeted maximum k-coverage optimization problem to maximize the number of sufficiently covered paths in dynamic runtime. It avoids relying on static assumptions about traffic volumes or switch utilization, which cannot be expected to hold in dynamic network environments. The allocation phase then uses observed traffic conditions to assign sampling tasks while enforcing per-switch capacity constraints, balancing sampling load distribution across switches. We next explore formulations for sampling point placement.

[2]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.

[3]Coordination only happens between sampling points. Packet loss or swapping before the first or after the last point does not affect coordination.

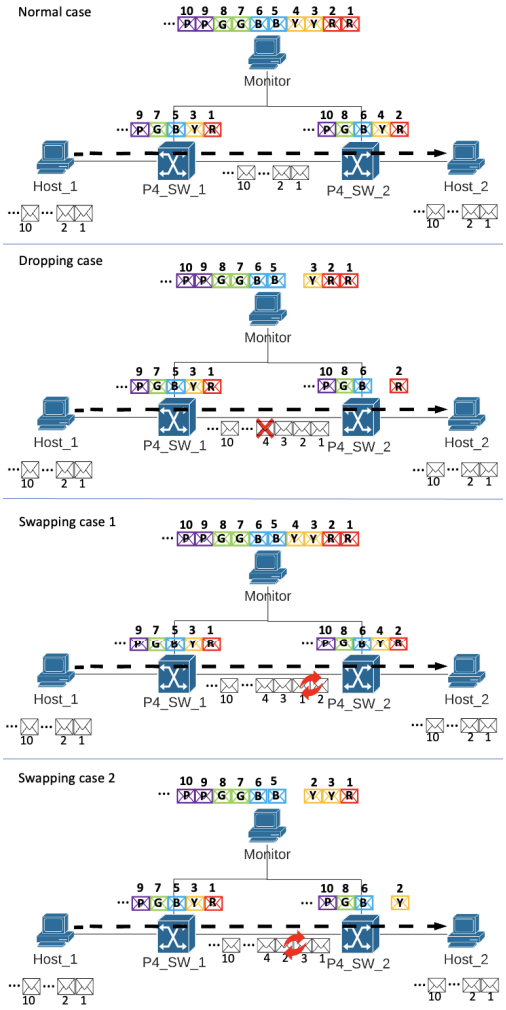


Fig. 3. Demonstration of COORDSAMP algorithm in different situations.

III. PLACEMENT PROBLEM FORMULATION

While the coordination algorithm is lightweight in terms of memory and processing overhead, the sampling load refers to the volume of traffic each switch must handle at line rate. In practice, this load can exceed what a single switch can sustain, especially under high-throughput or resource-constrained conditions. Therefore, we use multiple sampling points to distribute the load and avoid overloading any switch.

Our formulation aims to strategically placing sampling points under a global budget to achieve sufficient flow coverage. The budgeted sampling point placement problems are framed as Integer Linear Programming (ILP) problems considering multi-coverage. When a set of flows requires a higher sampling rate than a single point can provide, multiple sampling points are necessary. With a budgeted number of P4-programmable switches supporting COORDSAMP, 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 [27].

TABLE II
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

Universal setting: Given a set of flows $F = \{f_1, \ldots, f_m\}$ 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 $f_i : v_k \rightarrow \ldots \rightarrow v_l$. For every $v_j \in V$, if v_j is part of f_i, $f_i \in s_j$ where s_j is a set of flows such that they all traverse v_j. S is the collection of all position sets s_j, i.e., $S = \bigcup_{j=1}^{n} s_j$. 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 $f_i \in s_j$, $a_{i,j} = 1$; otherwise, $a_{i,j} = 0$. Given B budgeted number of P4-programmable switches, when a flow f_i is covered at least k times, the gain $g_i = 1$. Otherwise, $g_i = 0$. Table II summarizes the parameters used in the formulations, excluding those from the universal problem setting to avoid redundancy.

We assume homogeneous capacity across candidate placement positions rather than modeling heterogeneous capacities, because the latter requires predicting traffic load and sampling requirements, which are highly dynamic and whose transient estimates cannot guarantee optimality in a changing network. This assumption keeps the placement problem focused on the structural task of selecting positions that ensure sufficient multi-coverage of flows, while any capacity-aware adaptation can be handled dynamically by the runtime allocator. Its resulting $(0, 1)$-valued path matrix provides the foundation for our near-balanced analysis in networking scenarios, enabling the pseudo-polynomial optimality guarantees proved in Section IV. If operators have sufficiently reliable capacity estimates for certain positions, extensions to heterogeneous-capacity settings are discussed in Section VIII.

A. Budgeted Maximum Coverage (BMC)

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

BMC Formulation:

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

$$\text{s.t.} : \mathbf{g} - A_P\mathbf{x} \leq \mathbf{0} \tag{2}$$

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

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

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

The 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) enforces the budget on the number of sampling points. (4) are the binary decision variables.

The BMC problem is NP-hard [27]. Although existing works adopt a polynomial time greedy algorithm to solve this problem with a $\left(1 - \frac{1}{e}\right)$-approximate result [27], in our scenario, we achieve the optimal solution in pseudo-polynomial time by using the MILP solver with proof in Section IV and experiments in Section VI.

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:

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

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

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

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

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

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

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. Hardness of Budgeted Placement

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 k-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: $min\{x | x \in \{0,1\}; A_P x \geq k\}$. A theorem related to this formulation is given below:

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) [28].

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 [28], 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. Specifically, the flows on a spine-leaf architecture, which is essentially a tree topology, will always compose a balanced path matrix because a tree has no cycle, not to say an odd-cycle. So LP can produce an optimal solution on such a topology. 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. We discuss odd-cycles and how they influence the runtime of ILP solvers in Section IV-B.

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 $\{g_1, \ldots, g_m, x_1, \ldots, x_n\}^T$. 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 [29]. 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 [29]. A hole matrix is odd if the sum of its entries is congruent to 2 mod 4, and it is even if the sum of its entries is congruent to 0 mod 4 [29]. In other words, a $(0, \pm 1)$ matrix is balanced if: In every submatrix with two nonzero entries per row and column, the sum of the entries is a multiple of 4 [29].

(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 $-2c$ (where $c = 2, 4, 6, \ldots$), which is always a multiple of 4. So, $-A_{P_{m\times n}}$ is $(0, \pm 1)$ balanced.

(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.

Otherwise, any other subvector of $\mathbf{1}$ has less than or more than 2 nonzero entries and that makes it impossible to construct another hole matrix with any submatrix from $-A_P$.

(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.

Therefore, we have proven A_E is $(0, \pm 1)$ balanced if A_P is $(0, 1)$ balanced. □

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

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 [30]. We transform $(0, \pm 1)$ balanced matrix $A_{m\times n}$ into a $(0, 1)$ balanced matrix $B_{m\times 2n}$. Each column a_j of A associates with two columns b_j^P and b_j^N of B. For each element b_{ij}^P and b_{ij}^N: if $a_{ij} = 1$, $b_{ij}^P = 1$. Otherwise, $b_{ij}^P = 0$; if $a_{ij} = -1$, $b_{ij}^N = 1$. Otherwise, $b_{ij}^N = 0$.

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 $y = \{y_1^P, \ldots, y_n^P, y_1^N, \ldots, y_n^N\}^T$, 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.

Thus, we proved the case of $\max\{x|x \geq 0; Ax \leq b - n(A)\}$. The $min\{x|x \geq 0; Ax \geq b - n(A)\}$ case is analogous. □

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 $\arg\max \sum_{i=1}^{m} g_i$ is the same as the solution of objective $\arg\max \left(\sum_{i=1}^{m} g_i + \sum_{j=1}^{n} x_j\right)$.

However, these theorems are insufficient to address the BMkC case because its left-hand-side matrix is not a $(0, \pm 1)$ matrix while maintaining the "b" integral. Next, we demonstrate how the branch and bound technique achieves pseudo-polynomial optimal solutions in realistic network scenarios.

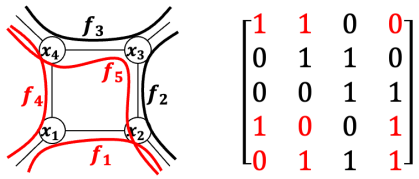


Fig. 4. Odd-Cycle example.

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 [31] 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 using the Gurobi solver [32] in reasonable runtimes across 20,000 experiments on real topologies shown in Section VI-C.

Branch and bound resolves non-integral solutions in the relaxed ILP by recursively assigning 0 or 1 to fractional variables and pruning substates that cannot contain the optimal integral solution [33]. 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). Moreover, the conditions (A is balanced, b is an integral vector) of Theorem 1 and Theorem 3 are sufficient but not necessary to achieve integral solutions under LP. The LP will still produce an integral/optimal solution when all of the odd-cycles are dominated by some even-cycles in A, even though A is not balanced under such a condition. An example based on Theorem 1 to illustrate this statement is given below.

Without loss of generality, we give an example in Fig. 4. The goal is to cover flows $F = \{f_1, f_2, f_3, f_4, f_5\}$ with minimum number of nodes. The path matrix of F is not balanced due to f_5. Removing f_5 makes the path matrix balanced. In detail, the submatrix composed of the red elements of the path matrix is an odd-cycle (3-cycle) incidence matrix. Even though the path matrix of F is not balanced, the LP returns an integral solution to cover all flows ($\{x_2 = x_4 = 1, x_1 = x_3 = 0\}$ or $\{x_1 = x_3 = 1, x_2 = x_4 = 0\}$) because the vertices on the 3-cycle(f_1, f_4, f_5) are dominated by a 4-cycle(f_1, f_2, f_3, f_4). In other words, the optimal integral solution of covering $F - \{f_5\}$ is also good enough to cover F optimally.

When we only cover flows $F - \{f_2, f_3\}$ with the minimum number of nodes, an unbalanced path matrix is produced due to the odd-cycle, the solution returned by LP is $\{x_1 = x_2 = x_4 = 0.5, x_3 = 0\}$ which is non-integral. Thus, the branch and bound mechanism is triggered on x_1, x_2, x_4. There are at most $O(2^3)$ combinations to exhaust 0 and 1 cases for each of x_1, x_2, x_4. The branch and bound technique selects one of the combinations producing the best objective. The $O(2^3)$ is a constant factor that does not influence the polynomial running time property. To be clear, the process of testing the odd-cycle vertices with 1 and 0 will not make the other originally integral variables non-integral because removing odd-cycle paths will not make the originally balanced sub-matrix unbalanced.

As the LP is solvable in polynomial time L [34], the running time of BMC is mainly influenced by the number

Algorithm 2 Greedy Allocation to Sampling Points

Data: $\{p\}$: Set of sampling points;
$\{f\}$: Dynamic flows;
$\{c\}$: Sampling capacity on points (c_j for p_j);
$\{f^s\}$: Set of sampled flows;
$\{f_j\}$: Set of flows on point p_j;
l_j: Sampling load on point p_j.
Result: $\{f^s\}$: Set of sampled flows.

1 $\{f^s\} \leftarrow \emptyset$;
2 **while** $\{f\} - \{f^s\} \neq \emptyset \wedge \{p\} - \{p_j \mid c_j = 0\} \neq \emptyset$ **do**
3 $(p_j, l_j, \{f_j\}) \leftarrow$ SELECT_MAX$(\{f\} - \{f^s\}, \{p\} - \{p_j \mid c_j = 0\})$;
4 **if** $l_j \leq c_j$ **then**
5 SAMPLE$(\{f_j\})$;
6 $\{f^s\} \leftarrow \{f^s\} \cup \{f_j\}$;
7 $c_j \leftarrow c_j - l_j$;
8 **else**
9 $\{f_j\} \leftarrow$ SORT$(\{f_j\})$;
10 $\{f^u\} \leftarrow$ UNSAMP$(\{f_j\}, l_j, c_j)$;
11 SAMPLE$(\{f_j\} - \{f^u\})$;
12 $\{f^s\} \leftarrow \{f^s\} \cup (\{f_j\} - \{f^u\})$;
13 $c_j \leftarrow 0$;
14 **end**
15 **end**

K of branch and bound nodes for all odd-cycles. The running time $O(LK)$ is pseudo-polynomial, if K can be treated as a constant (independent of input size) [35], and can be further reduced using parallel branch and bound [36].

Fortunately, our problem is constrained in that K is independent of input size, as practical topologies 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 [37, 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-completeness [35].

V. Sampling Task Allocation

After placing the budgeted number of P4-programmable switches with CoordSamp algorithms installed as sampling points, we dynamically allocate flows to the sampling point to maximize the number of sufficiently sampled flows in a dynamic network scenario. We provide a concise overview of the algorithm (Algorithm 2), which allocates flows to (multiple) sampling points to maximize the number of flows sampled at their required rate. Let l_j denote the traffic load at switch j, which is the aggregate sampling rate of flows at switch j (measured in bits/sec), while the sampling capacity c_j reflects the traffic load the switch can sample based on its available resources, which is essentially the bandwidth limitation of the outgoing port to the monitor. 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*. When there are any unsampled flows and the serving pool is not empty (line 2), the controller greedily selects the sampling point p_j from the serving pool with the maximum current traffic load l_j (line 3). If $l_j \leq c_j$, all the flows at p_j are sampled, the controller updates the set of sampled flows and the sampling point capacity (lines 4-7). 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 (line 8).

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 (line 9). 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, places them in the sampling request pool (line 10), and samples the fit amount of flows (line 11). When a flow is fractionally sampled, the remaining fractional sampling load is reserved for later allocation. Finally, the controller updates the set of sampled flows and the sampling point capacity (lines 12-13). 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.

In overload events ($l_j > c_j$), Algorithm 2 may split a flow across on-path points using tags; flow-exclusive baselines forbid splitting. The Proposition 1 formalizes that our packet-level coordination dominates flow-exclusive coordination.

Proposition 1: Under identical budgets and per-point capacities, the allocation set achievable by Algorithm 2 (packet-level coordination with tag-based splitting when $l_j > c_j$) strictly contains the set of all flow-exclusive assignments (one sampling point per flow). Consequently, for any nondecreasing coverage objective G:

$$\max_{\text{packet-exclusive}} G \geq \max_{\text{flow-exclusive}} G.$$

Proof: Any flow-exclusive assignment can be emulated by Algorithm 2 by never splitting flows (skipping lines 8-13), preserving budgets, capacities, and coverage. Strictness holds when no single on-path point can satisfy a flow's requirement but the sum of residual capacities across on-path points can; packet-level splitting pools those residuals without duplicated sampling, achieving coverage unattainable by any flow-exclusive scheme. □

VI. Evaluation

We evaluate CoordSamp on P4-programmable switches with real traffic. We adopt a placement-centric protocol: the online allocator is fixed to the packet-level coordinator (Alg. 2), and we compare placement strategies under matched budget and capacity constraints. We begin with microbenchmarks of the device and export overheads, using an industry sFlow configuration as a reference for per-packet processing and export cost. We then assess placement effectiveness—BMC/BMkC coverage (flows covered k times within a budget), ability to meet sampling demand given

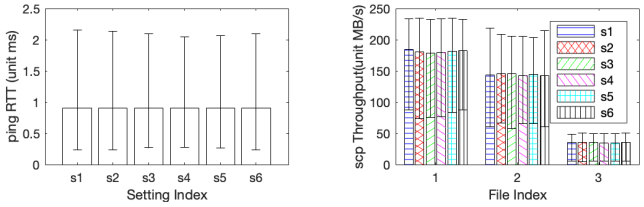


Fig. 5. Algorithm performance evaluation on real testbed.

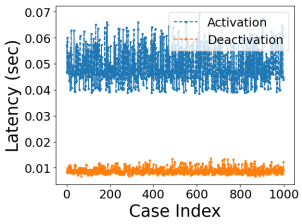


Fig. 6. Activation and deactivation latency on arista 7170-32CD.

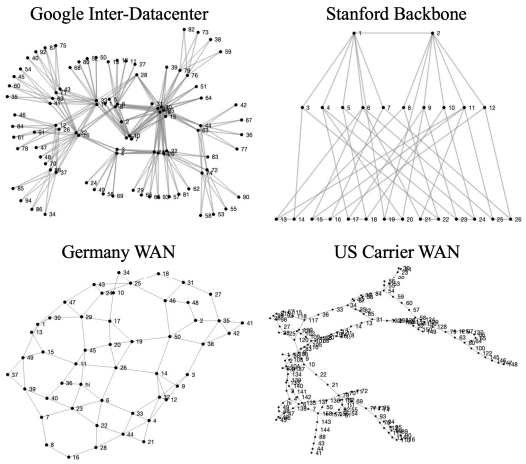


Fig. 7. Real-world topologies.

per-point capacities—across multiple topologies and dynamic traffic.

A. CoordSamp Overhead Evaluation

We assess the COORDSAMP algorithm's impact by comparing round-trip time (RTT) and throughput across multiple coordinated sampling settings, including port-based sampling using sFlow [2]. 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[4]:$sFlow.sf_2 = 1$

To eliminate factors like switch queuing, we conduct RTT and throughput tests without introducing background traffic. The results are presented in Fig. 5. 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.

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. CoordSamp (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 latency. 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 $t_d + k - t_l$ to approximate the deactivation latency. Notably, the actual activation and deactivation latency 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 (Fig. 6). Combined with the throughput and RTT results, this indicates our implementation is suitable for scalable, dynamic coordinated sampling. Next, we show the coverage and runtime of the optimal budgeted placement algorithm and its advantage in realistic settings.

C. Budgeted Placement Calculation Evaluation

We evaluate the sampling point placement solutions for BMC and BMkC using two approaches:

Optimal algorithm: Implemented via the Gurobi ILP solver utilizing relaxed LP and branch and bound technique [32];
Greedy algorithm: Iteratively selects the sampling point covering the maximum number of currently uncovered flows, until the budget is exhausted or all flows are sufficiently covered. For the k**-covering** scenario, the greedy algorithm first ensures $(k-1)$-coverage flows, then continues greedily selecting sampling points based on the remaining insufficient under-covered flows, stopping when the budget is used up or all flows are k-covered.

We also compare the runtime performance of both algorithms. Experiments are conducted on four real-world network topologies [38], [39], [40], illustrated in Fig. 7.

To simulate flow paths, we shuffle the source and destination of flows from *edge/host-facing* endpoints. Because the

[4]Using sFlow only on $P4\_SW\_2$.

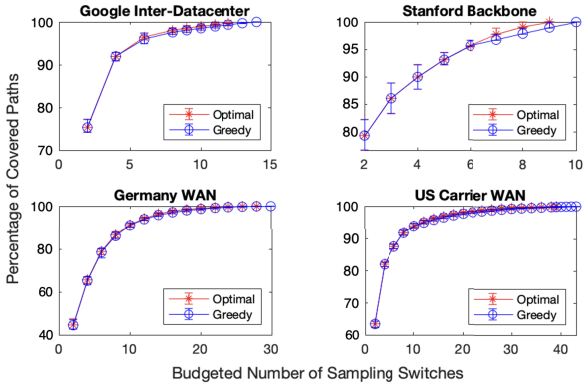


Fig. 8. Coverage percentage comparison of BMC.

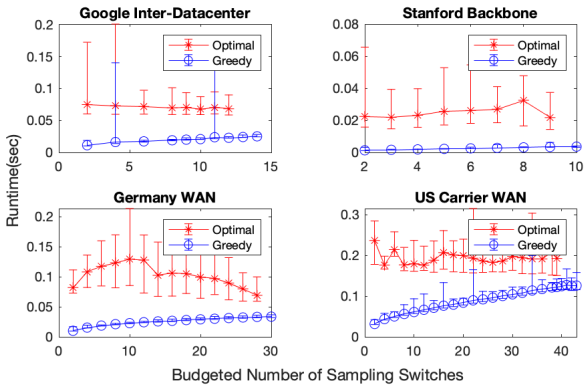


Fig. 9. Runtime comparison of BMC.

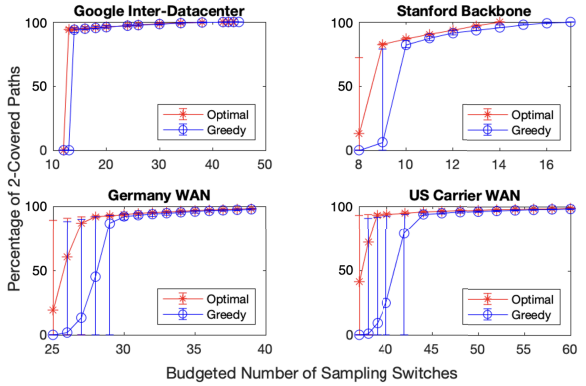


Fig. 10. Coverage percentage comparison of BMkC.

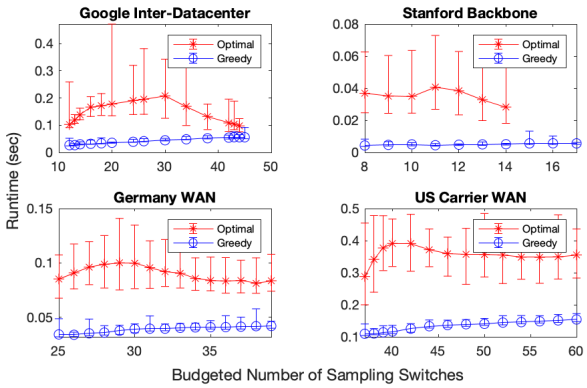


Fig. 11. Runtime comparison of BMkC.

topologies differ in structure, we use topology-aware degree heuristics to select source/destination nodes: (1) in Google Inter-DC and US carrier, degree-2 nodes most closely represent access/edge positions, so we sample source/destination nodes from that pool; (2) in Stanford Backbone, "leaf" nodes are the natural edge, so we randomly select the leaf nodes as a source or destination nodes; (3) in Germany topology, the graph does not separate edge vs. core, so all nodes are eligible as source and destination nodes.

Based on the Google Inter-Dataceneter, Stanford Backbone, Germany WAN, and US Carrier WAN topologies, we randomly select 1,000, 90, 1,000 and 1,500 different pairs of sources and destinations 100 times and compute the shortest paths. We show max, min, and average of the result based on the 100 experiments for each setting in 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 5,800 experiments conducted on four real-world topologies, the optimal algorithm consistently outperforms the greedy algorithm by achieving higher path coverage with the same budget and comparable runtime of 10^{-2} to 10^{-1} seconds, as illustrated in Figures 8 and 9. The slight fluctuation—often within 0.1 seconds—is due to runtime jitter and reflects system-level noise rather than a meaningful performance difference. Specifically, the optimal algorithm requires 1–4 fewer sampling points than the greedy algorithm to achieve complete path coverage. Moreover, over 90% of flows are covered using fewer than half the sampling points necessary for full coverage, underscoring the importance of balancing path coverage and cost.

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 5,400 experiments shown in Fig. 10 and 11, the optimal algorithm consistently outperforms the greedy algorithm and its runtime is comparable to the greedy algorithm in $10^{-2} \sim 10^{-1}$ seconds level. In multi-rooted spine-leaf architectures (e.g., Google Inter-Datacenter), greedy performs closer to optimal due to the presence of high-degree spine nodes. However, it offers no optimality guarantees and can still yield suboptimal results. Our evaluation shows that the optimal algorithm consistently outperforms greedy with modest runtime overhead. Crucially, we evaluate both spine-leaf (e.g., Stanford Backbone) and general WAN topologies, where efficient sampling is equally important, highlighting the broader applicability of our approach beyond structured settings.

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 13 for the Google Inter-Datacenter topology, the optimal method covers 941 out of 1000 paths on average of 100 times randomization, while the greedy algorithm covers 0. Similarly, with a budget of 26 for the Germany WAN topology, the optimal algorithm achieves around 90% 2-covered flows in 68 out of 100 randomized

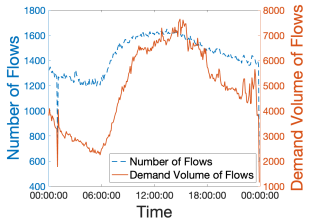


Fig. 12. Dynamic flows on germany topology.

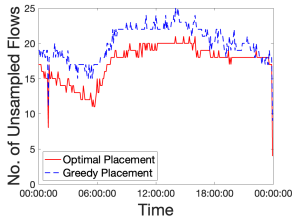


Fig. 13. Placement effectiveness comparision.

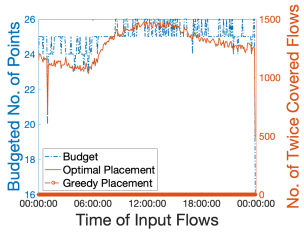


Fig. 14. 2-Coverage: Optimal vs. greedy across budgets and flows.

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

CoordSamp has two phases: (1) *budgeted placement* (offline) and (2) *dynamic sampling task allocation* (online; Alg. 2). Here we isolate the contribution of *placement*: we compute placements using the ILP (optimal) and a greedy heuristic, then fix the online sampling task allocation method for both. We replay a 24-hour dynamic-flow dataset on the Germany topology [40] and compare placements under matched budgets/capacities. This single 24-hour case is sufficient for our purpose: we fix the allocator to isolate placement, the trace spans a wide range of dynamic conditions, and it complements our static multi-topology results (Sec. VI-C).

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/insufficiently sampled flows for both placement results to quantify the budgeted placement effectiveness.

The dynamic flows on the Germany Topology, illustrated in Fig. 12, are recorded every 5 minutes, providing information on (src, dst) pairs and corresponding demand values, l, which is proportional to traffic volume with a unit of MBit/s. We use l as the flow sampling load. We set each sampling point p_i with a maximum capacity $c = 50$, matching the unit of flow demand (MBit/s), to simulate limited switch sampling conditions based on the dataset. This ensures most points operate near capacity and highlights the benefit of our optimal multi-placement algorithm over greedy under such constraints. 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 fractionally sampled flows await the next opportunity for sufficient sampling.

1) Optimal Versus Greedy Placement With Input of All Flows: Here we use all 1,225 paths from the 24-hour dynamic-flow dataset on the Germany topology as the input paths for the budgeted placement computation. 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 placement** and **greedy placement** algorithms. As shown in Fig. 13, the optimal placement—computed in advance—results in an average of **5 more sufficiently sampled flows per 5-minute interval** than the greedy placement, using the same number of sampling points. Since backbone and datacenter flows are often short-lived or vary in volume and path [41], [42], many of the additional flows sampled in each interval are likely *different identities* from those in the prior interval. With 288 such intervals in a day, this modest per-interval improvement translates into significantly broader flow coverage with dynamic traffic, underscoring the value of optimal placement. This demonstrates that even a small per-interval improvement can lead to substantial overall flow coverage in dynamic environments, highlighting the value of the optimal placement.

Ideally, the total aggregated traffic volume (in bit/sec) of sampling under greedy and optimal multi-cover placements should be similar, as both approaches are tested with the same number of sampling points deployed with identical capacities. However, the greedy method often lacks redundant coverage, which may leave some sampling points underutilized, while some are overloaded. Specifically, if a sampling point selected by the greedy algorithm is not on the path of any insufficiently sampled flow, it cannot help improve coverage. In contrast, the optimal placement ensures better overlap and resource utilization. Since our objective is to maximize the number of sufficiently sampled flows rather than less meaningful partially sampled ones, we evaluate effectiveness based on the number of flows meeting their sampling targets.

2) Optimal Versus 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. 14. Note with these budgets, the greedy algorithm is incapable of achieving any 2-covered flows.

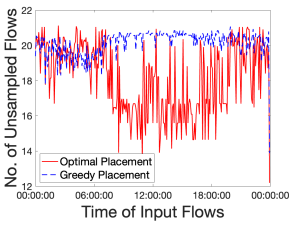


Fig. 15. Placement effectiveness across input flow counts.

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. 15. 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. 12, 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

Network traffic measurement has been addressed through a variety of architectures. Recent SDN-based network monitoring approaches [43], [44] adopt external monitors to avoid degrading switch performance, but either waste bandwidth by forwarding full packets or lack flexibility. In contrast, CoordSamp has the flexibility of inserting metadata in the packets and truncating mirrored packets to save bandwidth with P4-programmable switches.

Among the works that consider sampling scalability, Giotis, et al [45] imply an assumption about limited sampling workload, Shirali-Shahreza and Ganjali [46] exclude elephant flows to mitigate the sampling workloads, and FlowShark [47] separate sampling decisions on short and long flows to enhance sampling efficiency. Du, et al [4] propose a method to achieve an adaptive sampling rate for different flows. Others [5], [6] aim at a best-effort single point per-flow sampling allocation regardless of the high sample rate demand of the intrusion detection system. Another non-duplicate sampling work [48] prevents duplicated sampling on the traffic element (e.g. destinations), not flows. The work considers the sampling scalability issue and allows several monitor points along the path of flows. However, it requires the SDN controller to drop duplicate samples, which wastes network resources and adds overhead at the controller [49]. Conversely, we study flow-based sampling at multiple points with coordinated sampling. Another coordinated sampling work [16] 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.

Recent efforts have aimed to distribute the network monitoring load across multiple points to alleviate pressure on individual switches. These approaches can be broadly categorized by their coordination granularity: coordination-free, flow-level coordination, and packet-level coordination.

Coordination-free monitoring methods utilize multiple switches to distribute traffic measurement load and avoid inter-switch coordination by tolerating sampling redundancy. FlowRadar [12] encodes flow identifiers and counters in a compact Bloom filter-like structure to enable per-packet monitoring without coordination. Ben-Basat et al. [10] employ a duplication-tolerant hash-based method to estimate network-wide statistics. UnivMon [11] offers a universal sketch framework that supports multiple flow-level queries such as heavy hitter detection and entropy estimation using a single sketch structure. While powerful in scope, it focuses on coarse-grained estimations and lacks mechanisms for sampling granularity control or duplication prevention, which wastes resources on duplicated measurements. They also lack packet-level visibility and do not consider budget-aware placement. In contrast, CoordSamp achieves duplication-free, packet-level monitoring while maintaining low overhead through lightweight tagging and selective coordination.

Flow-level coordination aims to ensure that each flow is measured by only one switch, avoiding duplication by construction. Systems such as Distributed Sketch [13] and CountMax [14] support state-shared and flow-level coordination by partitioning flows or splitting sketches across switches to measure flows efficiently. Flow Distribution [15] assigns each flow to a single monitor using centralized control, while Cooperative Flow Selection [24] uses TTL-based probabilistic decisions to achieve flow-level coordination to distribute flows across switches. CSAMP [16] also enforces flow-level exclusivity via routing-aware flow assignments. While these methods avoid duplicated measuring, they cannot leverage multiple measure points per flow, and most lack packet-level visibility. Additionally, none of them consider monitor placement under budget constraints. CoordSamp addresses these gaps by supporting multi-point, packet-level coordinated sampling via optimization for placing sampling-capable switches.

Packet-level coordination aims to prevent duplicated sampling while allowing a flow to be monitored at multiple points. Afek et al. [25] only conceptually mentions tagging sampled packets to prevent duplication, but lacks deployment details or practical implementation. CoordSamp systematically designs a packet-level coordinated sampling and implements the system using P4-programmable switches. It uses in-band packet tagging to prevent redundant sampling and supports robust sample sequence reconstruction at the monitor. Unlike prior work, CoordSamp explicitly incorporates a budget-aware placement algorithm that maximizes sampling coverage within a monitoring budget. Though exporting sampled packets incurs more overhead than flow-level statistics, it offloads data processing to inexpensive external monitors to support packet-level detections (e.g., deep packet inspection), making the trade-off practical and efficient.

Furthermore, existing monitor placement methods [5], [50], [51] typically use heuristic greedy algorithms to solve set covering problem without considering scenario characteristics on the ability to obtain optimal solutions, as we do.

VIII. Discussion

Trade-Offs. The coordination logic in CoordSamp is deliberately lightweight, enabling flexible sampling and efficient switch resource use. Although exporting raw packets

increases bandwidth, CoordSamp offloads processing to external commodity monitors rather than costly programmable switches, reflecting a conscious trade-off between in-switch complexity and export overhead.

Capacity Heterogeneity. In deployments where operators have prior knowledge that certain positions should host higher-capacity switches—e.g., due to centralized topology roles or historical traffic patterns—this heterogeneity can be modeled by assigning proportionally scaled coverage weights in a generalized path matrix $A_P = [a_{i,j}]$, where $a_{i,j} \geq 0$ reflects the effective sampling contribution of heterogeneous switches. Switch heterogeneity also leads to weighted budget constraints $\sum_j w_j x_j \leq B$. However, introducing such coverage and cost weights converts the left-hand-side matrix of the formulation into a general integer matrix, causing the balanced-matrix conditions to no longer hold and eliminating the pseudo-polynomial solvability of the unweighted formulation. A full analysis of the weighted case remains an open question.

Scalability Considerations. The runtime of the placement solver depends on the structure of the path matrix, not on the raw number of nodes or edges. Adding nodes or links affects runtime only when it creates additional, diverse paths that expand the matrix or introduce new odd-cycle structures. Since topology size and flow count are not independent, the meaningful scalability factor is the number of flows, which directly determines the number of ILP constraints. To assess this, we also varied the number of flows on the fixed US Carrier topology under the minimum-cost placement formulation (Section IV-A); runtime increased only mildly (0.015–0.05 seconds) as flows grew from 100 to 1500. After placement, the online allocator operates only over the budget-limited sampling points and scales linearly with the number of flows, making its overhead insensitive to topology size.

Scenarios and Potential Security Extensions. CoordSamp is well-suited for scenarios where fine-grained packet visibility is needed beyond what flow-level summaries can provide, such as detecting stealthy attacks, policy violations, or subtle traffic anomalies. It is also effective in deployments with limited switch-side monitoring capacity, where distributing sampling load to commodity servers improves scalability. Dynamic or multi-path networks likewise benefit from multi-point sampling to enhance monitoring resilience and validate middlebox behavior or traffic policies. Beyond current functionality, CoordSamp can support more advanced attack detection. Tagging packets and sampling at strategic points—such as before and after a middlebox—enables operators to identify packet loss, catch inconsistent transformations, or detect stealthy rule bypassing. The same mechanism facilitates zone-based monitoring, where different network regions (e.g., data center, demilitarized zone) apply tailored sampling logic that respects policy or privacy constraints.

IX. Conclusion

We have designed and implemented a dynamic coordinated sampling system using P4-programmable switches. Our system is lightweight, flexible, and generalizable to any P4-programmable switch, with negligible performance overhead, and it can achieve coordination automatically through dynamic configuration. We formulated budgeted sampling point placement problems ranging from single-coverage to multi-coverage. Although these problems are NP-complete in general, we have theoretically and experimentally proven that their optimal solutions can be achieved efficiently in real-world topology cases, indicating weak NP-completeness. Our CoordSamp system is well-suited for dynamic sampling activation, and we have validated the sampling superiority of optimal placement over greedy placement using real traffic.

Acknowledgment

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.

References

[1] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. USENIX Symp. Networked Syst. Design Implement.*, 2013, pp. 29–42.

[2] P. Phaal. (2023). *Sflow Version 5*. [Online]. Available: https://sflow.org/sflowversion5.txt

[3] (Mar. 2023). *24x7x365 Security Operations Monitoring*. [Online]. Available: https://it.vt.edu/content/dam/itvtedu/dcss/2023/spring/DCSS-SOC-monitoring-2023331.pdf

[4] Y. Du, H. Huang, Y.-E. Sun, S. Chen, and G. Gao, "Self-adaptive sampling for network traffic measurement," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[5] S. Yoon, T. Ha, S. Kim, and H. Lim, "Scalable traffic sampling using centrality measure on software-defined networks," *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 43–49, Jul. 2017.

[6] R. Cohen and E. Moroshko, "Sampling-on-demand in SDN," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2612–2622, Dec. 2018.

[7] A. Aqil et al., "Jaal: Towards network intrusion detection at ISP scale," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2017, pp. 134–146.

[8] O. Yevsieieva and S. M. Helalat, "Analysis of the impact of the slow HTTP DOS and DDOS attacks on the cloud environment," in *Proc. 4th Int. Sci.-Practical Conf. Problems Infocommun. Sci. Technol. (PIC S&T)*, Oct. 2017, pp. 519–523.

[9] M. Chen, T. La Porta, T. Jaeger, and S. Krishnamurthy, "Lightweight coordinated sampling for dynamic flows under budget constraints," in *Proc. 33rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2024, pp. 1–9.

[10] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz, "Routing-oblivious network-wide measurements," *IEEE/ACM Trans. Netw.*, vol. 29, no. 6, pp. 2386–2398, Dec. 2021.

[11] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 101–114.

[12] Y. Li, R. Miao, C. Kim, and M. Yu, "*FlowRadar*: A better *NetFlow* for data centers," in *Proc. 13th USENIX Symp. Networked Syst. Design Implement.*, 2016, pp. 99–112.

[13] L. Gu, Y. Tian, W. Chen, Z. Wei, C. Wang, and X. Zhang, "Per-flow network measurement with distributed sketch," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 411–426, Feb. 2024.

[14] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "CountMax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2774–2786, Dec. 2018.

[15] H. Xu, S. Chen, Q. Ma, and L. Huang, "Lightweight flow distribution for collaborative traffic measurement in software defined networks," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2019, pp. 1108–1116.

[16] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "CSAMP: A system for network-wide flow monitoring," Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-07-139, 2008.

[17] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

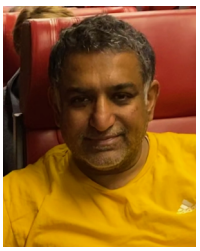
[18] (2025). *Suricata: Open Source Network Threat Detection Engine*. [Online]. Available: https://suricata.io/

[19] (2025). *Zeek Network Security Monitor*. [Online]. Available: https://zeek.org/
[20] Corelight, Inc. (2025). *Corelight*. [Online]. Available: https://corelight.com/
[21] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack–how firewall middleboxes reduce security," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 347–361.
[22] (2024). *Arista Price List*. [Online]. Available: https://itprice.com/arista-price-list/7170.html
[23] K. Butler, P. McDaniel, and W. Aiello, "Optimizing BGP security by exploiting path stability," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, Oct. 2006, pp. 298–310.
[24] R. B. Basat, G. Einziger, and B. Tayh, "Cooperative network-wide flow selection," in *Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2020, pp. 1–11.
[25] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff, "Detecting heavy flows in the SDN match and action model," *Comput. Netw.*, vol. 136, pp. 1–12, May 2018.
[26] (2023). *Coordinated Sampling on P4-Programmable Switches*. [Online]. Available: https://github.com/mzc796/coordsampling
[27] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, no. 1, pp. 39–45, Apr. 1999.
[28] A. Schrijver, *Theory of Linear and Integer Programming*. Hoboken, NJ, USA: Wiley, 1998.
[29] C. Berge, "Balanced matrices," *Math. Program.*, vol. 2, no. 1, pp. 19–31, Feb. 1972.
[30] M. Conforti, G. Cornuéjols, and K. Vušković, "Balanced matrices," *Discrete Math.*, vol. 306, nos. 19–20, pp. 2411–2437, 2006.
[31] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, "Experiments in mixed-integer linear programming," *Math. Program.*, vol. 1, no. 1, pp. 76–94, Dec. 1971.
[32] (2023). *Gurobi Mixed-Integer Programming (MIP)—A Primer on the Basics*. [Online]. Available: https://www.gurobi.com/resource/mip-basics/
[33] K. M. J. D. Bontridder, B. J. Lageweg, J. K. Lenstra, J. B. Orlin, and L. Stougie, "Branch-and-bound algorithms for the test cover problem," in *Proc. 10th Annu. Eur. Symp.*, Sep. 2002, pp. 223–233.
[34] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. 16th Annu. ACM Symp. Theory Comput.*, 1984, pp. 302–311.
[35] M. R. Garey and D. S. Johnson, *Computers and Intractability*, vol. 29. New York, NY, USA: Freeman, 2002.
[36] D. A. Bader, W. E. Hart, and C. A. Phillips, "Parallel algorithm design for branch and bound," in *Tutorials on Emerging Methodologies and Applications in Operations Research*, 2006.
[37] S. L. Hakimi and E. F. Schmeichel, "On the number of cycles of length k in a maximal planar graph," *J. Graph Theory*, vol. 3, no. 1, pp. 69–86, Mar. 1979.
[38] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 99–111.
[39] (2012). *Topology Zoo*. [Online]. Available: http://www.topology-zoo.org/dataset.html
[40] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Networks, Int. J.*, vol. 55, no. 3, pp. 276–286, May 2010.
[41] F. Carpio, A. Engelmann, and A. Jukan, "DiffFlow: Differentiating short and long flows for load balancing in data center networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.
[42] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," in *Proc. Conf. Appl., Technol., architectures, protocols Comput. Commun.*, Aug. 2003, pp. 325–336.
[43] S. Bera, S. Misra, and A. Jamalipour, "FlowStat: Adaptive flow-rule placement for per-flow statistics in SDN," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 530–539, Mar. 2019.
[44] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 1, pp. 75–88, Mar. 2020.
[45] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014.
[46] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in OpenFlow controller with FleXam," in *Proc. IEEE 21st Annu. Symp. High-Performance Interconnects*, Aug. 2013, pp. 49–54.
[47] S. Sadrhaghighi, M. Dolati, M. Ghaderi, and A. Khonsari, "FlowShark: Sampling for high flow visibility in SDNs," in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 160–169.
[48] Y.-E. Sun, H. Huang, C. Ma, S. Chen, Y. Du, and Q. Xiao, "Online spread estimation with non-duplicate sampling," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 2440–2448.
[49] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, Aug. 2014, pp. 85–90.
[50] K. Suh, Y. Guo, J. Kurose, and D. Towsley, "Locating network monitors: Complexity, heuristics, and coverage," *Comput. Commun.*, vol. 29, no. 10, pp. 1564–1577, Jun. 2006.
[51] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 1956–1961.

Mingming Chen (Member, IEEE) received the Ph.D. degree from The Penn State University in 2025. She is currently an Assistant Professor with the Computer Science Department, Kansas State University. Her research focuses on network security, with interests in software-defined networking (SDN) security, optimization for defense placement, P4 programming, and machine learning-assisted security. Her work has appeared in ACM CCS and USENIX Security and has led to the disclosure of multiple CVEs affecting widely used SDN controllers. She contributed to OpenDaylight earlier in her career. She serves on program committees and artifact evaluation boards in the security community.

Thomas F. La Porta (Fellow, IEEE) was the Founding Director of the Institute of Networking and Security Research at The Penn State University (Penn State). He is currently the Director of the School of Electrical Engineering and Computer Science and Penn State. He is an Evan Pugh Professor. Prior to joining Penn State, he was the Director of the Mobile Networking Research Department, Bell Laboratories, Lucent Technologies, where he led various projects in wireless and mobile networking. He is a Bell Labs Fellow. He received the Bell Labs Distinguished Technical Staff Award and the Eta Kappa Nu Outstanding Young Electrical Engineer Award.

Trent Jaeger (Fellow, IEEE) received the Ph.D. degree from the University of Michigan. He is currently a Professor with the Computer Science and Engineering Department, University of California, Riverside. He is the Consortium Lead of the Army Research Lab's Cybersecurity Collaborative Research Alliance (CSec CRA). His research interests include computer and network security. He serves as the Associate Editor-in-Chief for IEEE SECURITY AND PRIVACY and on the Editorial Board for the Communications of ACM. He has previously served as the Chair for the ACM Special Interest Group on Security, Audit, and Control (ACM SIGSAC).



Srikanth V. Krishnamurthy (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of California San Diego in 1997. From 1998 to 2000, he was a Research Staff Scientist at the Information Sciences Laboratory, HRL Laboratories, LLC, Malibu, CA, USA. Currently, he is a Professor of computer science at the University of California, Riverside. His research interests include networks, systems, machine learning, and security. He was a recipient of the NSF CAREER Award from ANI in 2003. He has served as the Associate Editor-in-Chief for IEEE TRANSACTIONS ON MOBILE COMPUTING.