DisPLOY: Target-Constrained Distributed Deployment for Network Measurement Tasks on Data Plane

Mimi Qian, Lin Cui, Xiaoquan Zhang, *Member, IEEE,* Fung Po Tso, *Senior Member, IEEE*, Yuhui Deng, Zhetao Li, *Member, IEEE*, and Weijia Jia, *Fellow, IEEE*

***Abstract*—In programmable networks, measurement tasks are placed on programmable switches to monitor network traffic at line rate. These tasks typically require substantial resources (e.g., significant SRAM), while programmable switches are constrained by limited resources due to their hardware design (e.g., Tofino ASIC), making distributed deployment essentially. Measurement tasks must monitor specific network locations or traffic flows, introducing significant complexity in deployment optimization. This target-constrained nature makes task optimization on switches (e.g., task merging) become device-dependent and order-dependent, which can lead to deployment failures or performance degradation if ignored. In this paper, we introduce *DisPLOY*, a novel target-constrained distributed deployment framework specifically designed for network measurement tasks on the data plane. *DisPLOY* enables operators to specify monitoring targets—network traffic or device/link—across multiple switches. Given the monitoring targets, *DisPLOY* effectively minimizes redundant operations and optimizes deployment to achieve both resource efficiency (e.g., minimizing stage consumption) and high-performance monitoring (e.g., high accuracy). We implement and evaluate *DisPLOY* through deployment on both P4 hardware switches (Intel Tofino ASIC) and BMv2. Experimental results show that *DisPLOY* significantly reduces stage consumption by up to 66% and improves ARE by up to 78.4% in flow size estimation while maintaining end-to-end performance.**

***Index Terms*—Network measurement, Distributed measurement deployment, Programmable data plane.**

I. Introduction

NETWORK measurement is essential for a variety of network operations, such as congestion control [1], anomaly detection [2], scheduling, and path tracing [3]. The development of programmable switches, like the RMT architecture [4], empowers network administrators to implement specific measurement functionalities directly within the data plane of programmable switches. Nevertheless, deploying multiple measurement tasks is significantly constrained by the limited resources of these switches. For instance, the total available memory on the Intel Tofino switch, encompassing both SRAM and TCAM, is restricted to just a few tens of MB [4]

Measurement tasks typically involve resource-heavy programs that necessitate the storage of numerous rules and flow statistics, thereby consuming substantial SRAM and pipeline stages resources [5]. Various approaches have been developed to implement new algorithms [6], [7] for conducting measurement tasks on a single programmable switch. However, these can overwhelm the switch's resource limits, resulting in degraded measurement accuracy or deployment failures. A more effective alternative is to adopt distributed program deployment, which utilizes the combined resources of multiple programmable network switches.

Recent studies [8], [9], [10] demonstrate that distributed program deployment can be achieved by decomposing input programs into match-action tables (MATs), merging redundant operations (e.g., common MATs) to reduce resource consumption, and distributing them across multiple programmable switches. Frameworks like SPEED and Hermes have developed techniques such as task merging and one big switch (OBS) abstraction for efficient deployment. These frameworks assume a flexible network infrastructure where programs can be installed on any network component to achieve various optimization objectives such as minimizing end-to-end latency. However, measurement tasks often have specific *target-constrained requirements*, such as the need to monitor particular *traffic flows* or *network devices/links*. These requirements are common and significantly complicate the distributed deployment of multiple concurrent measurement tasks.

Operational networks, such as ISPs and data centers, typically specify their monitoring targets, necessitating observations at various network locations to monitor specific traffic or desired network device/link information for network management [6], [11], [12]. For instance, when tenants report performance declines, operators need measurement tasks like flow cardinality [6] and DDoS detection [13] at specific locations to find the root cause. Neglecting these target constraints during deployment can impair network performance and potentially lead to monitoring failures. However, a significant limitation of existing end-to-end latency minimization frameworks is their inability to integrate target constraints when applied to measurement tasks. This deficiency leads to suboptimal placement of measurement logic (i.e., MATs) on switches. Figure 1 demonstrates how this leads to unintended flow redirection (dashed green arrow) and the failure to detect key

Mimi Qian, Lin Cui, Xiaoquan Zhang, Yuhui Deng and Zhetao Li are with the Department of Computer Science, Jinan University, Guangzhou, China. Email: qianmimi001@gmail.com, tcuilin@jnu.edu.cn, zhangxiaoquan547@gmail.com, tyhdeng@email.jnu.edu.cn, liztchina@hotmail.com.

Fung Po Tso is with the Department of Computer Science, Loughborough University, UK. Email: p.tso@lboro.ac.uk.

Weijia Jia is with BNU-UIC Institute of Artificial Intelligence and Future Networks, Beijing Normal University (BNU Zhuhai) and BNU-HKBU United International College, Zhuhai, China. Email: jiawj@uic.edu.cn.

Corresponding author: Lin Cui

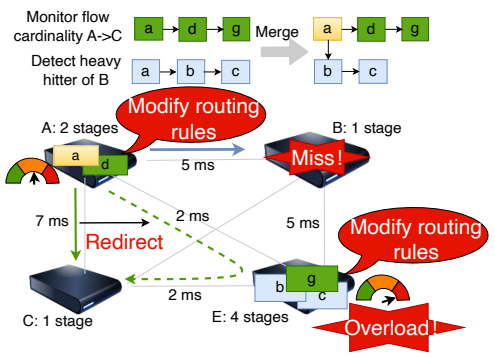


Figure 1: Motivating example of target constraints. Measurement tasks require monitoring specific flows and devices. Existing frameworks re-route flows (dashed green arrow) to minimize latency, violating target constraints and missing critical events (e.g., heavy-hitters on switch B).

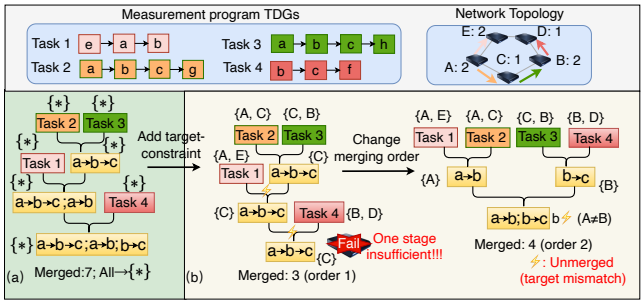


Figure 2: Compare device-independent merging (a) vs. target-constrained merging (b). Target constraints enforce MAT placement on specific switches (e.g., A), complicating merging sequences. {X} = Target switch for MAT deployment; {∗} = Any switch with sufficient resources. Yellow rectangles show cumulative merged MATs. In the network topology, numbers (e.g., "C:1") denote available stages of the switch, and colored arrows indicate monitored traffic of measurement task.

network events, such as heavy-hitter flows through switch B escaping detection. Therefore, a distributed deployment framework must consider target constraints to ensure accurate and complete monitoring without degrading network performance or requiring disruptive routing changes.

However, designing a distributed deployment framework that enables measurement tasks to specify various monitoring targets (e.g., traffic or devices/links) while ensuring resource-efficiency and high-performance monitoring is challenging for several reasons.

First, *there is currently no standardized network model for implementing the distributed deployment of measurement tasks with target constraints.* Distributed measurement tasks must not interfere with existing network routing configurations (e.g., IS-IS) or services (e.g., load balancing), as this can result in network performance degradation due to, for example, switch overloading as illustrated in Figure 1.

Second, *target constraints make measurement task merging both device-dependent and order-dependent,* complicating MAT identification and merging sequence determination for Table Dependency Graphs (TDGs). As Figure 2(a) shows, merging tasks without target constraints is straightforward: it identifies and combines all identical MATs (i.e., those with identical logic and shared resources) across input TDGs, assuming MATs can be deployed on any substrate network switch. This is a *device-independent and order-independent process.* However, unlike device-independent merging, target constraints force deployment on specific devices (Figure 2(b)). Merging MATs with conflicting targets can lead to potentially unmerged MATs (since a MAT can only be deployed on one switch) or deployment failures (e.g. $a \rightarrow b \rightarrow c$ in order 1 requires 3 stages on switch C, which only has one). Furthermore, different merging orders under target constraints yield varying results and performance (e.g., orders 1 and 2 produce 3 and 4 merged MATs, respectively).

Last but not least, *target constraints complicate distributed deployment optimization.* Differences in target constraints among measurement tasks, combined with diverse performance requirements (e.g., low measurement latency and reduced bandwidth overhead), lead to resource imbalances on switches. This poses challenges to achieving resource efficiency (e.g., minimizing stages) and high-performance monitoring (e.g., high measurement accuracy) in distributed deployment.

To address the above challenges, we propose *DisPLOY* (Distributed network measurement deployment). *DisPLOY* supports measurement tasks with monitoring targets while achieving resource-efficient and high-performance monitoring. It specializes in merging tasks with specific constraints: (1) *flow monitoring* targeting predefined traffic patterns (heavy hitters [6], DDoS detection [13]); (2) *device/link monitoring* collecting statistics (e.g., packet loss, latency) from predefined network devices or links; (3) *path tracing* implementing protocols like DeltaINT [3] for SLA compliance. These tasks require precise placement of measurement logic (e.g., MATs) on specific switches to avoid monitoring failures or network disruptions. *DisPLOY* creates extended TDGs and merges redundant MATs with target constraints, while reusing identical but *device-independent* operations to improve resource efficiency. We model the target-constrained deployment problem and prove it to be NP-hard. To solve this problem, *DisPLOY* splits the TDG segment and merges adjacent TDG segments by encoding a TDG segment directed graph (TSDG) and OBSs to minimize the number of stages occupied. *DisPLOY* utilizes Prim's algorithm to determine the shortest distance among deployed switches. *DisPLOY*'s key innovation is its ability to perform both device-dependent and order-dependent task merging and target-constraint deployment.

In summary, the contributions of this paper are as follows:

1) We propose *DisPLOY*, a target-constrained framework for distributed measurement task deployment. To our knowledge, it is the first work to explicitly address target constraints (e.g., devices, flows) in the distributed deployment context on programmable data planes.
2) With *device-dependent* & *order-dependent* merging and

Table I: Examples of network measurements.

Tasks	point	Description	Applications	Deploy position
Device-local	Heavy hitter	Gathering information within a sing node	Multi-path routing [14], Gray failure detection [6], Network provisioning [3]	Single node
	Heavy change			
	Flow size			
	Entropy			
	Cardinality			
	Flow Size Distr.			
Network-wide	INT	Obtaining multi-nodes information	Congestion control [1], Detecting abnormal jitters [6], OffsetINT [15]	Per-node or multiple nodes
	Per-hop latency			
	Packet drop			
	Routing path			
	Jitters			

reusing, *DisPLOY* reduces redundancy among measurement TDGs to save switch resources. An efficient deployment algorithm based on a TSDG and OBSs is proposed for near-optimal deployment, specifically for target-constrained scenarios.

3) We have implemented a prototype of *DisPLOY* on both P4 hardware switches (with Intel Tofino ASIC) and BMv2. Extensive evaluations show that *DisPLOY* significantly reduces stages consumption by up to 66% compared to the baseline method and improves up to 78.4% ARE in flow size estimation.

The rest of paper is organized as follows. Section II discusses the background and related works of *DisPLOY*. The overall design of *DisPLOY* is provided in Section III. Section IV evaluates *DisPLOY* on P4 hardware switches with Intel Tofino ASIC and BMv2. Section V concludes this paper.

II. Background and Related Works

A. Background

Measurement tasks can be categorized into two categories: *device-local measurement* and *network-wide measurement* (Table I). Device-local measurement refers to measuring traffic by deploying monitor on a single node (e.g., edge switch), such as heavy hitter detection. Network-wide measurement commonly focuses on gathering desired information by deploying measurement instances on multiple devices along forwarding path within the network for global analysis. Typical desired information includes INT information, routing path, per-hop latency, jitters, packet drops, and so on.

Figure 3 illustrates the typical workflow of a data plane measurement task. Upon packet arrival, the task begins by parsing the packet to extract a flowkey (e.g., 5-tuple) from its header. To manage large-scale network traffic, some tasks employ *random sampling* to selectively collect packets or flows based on predefined criteria [16]. Next, *hash computations* are performed on the flowkey to determine storage positions for statistics. The task then executes *update operations* on the storage, such as a counter array of $R \times W$ counters. Based on the hash output, a specific column in each row of the counter array is chosen for updating statistics. At the end of a measurement epoch, the collected data is transmitted to a controller for analysis using the *copy_to_cpu* function. Network administrators implement this workflow logic in data plane programs using network programming languages like P4 [17]. During program deployment, these programs are compiled, and runtime configurations are loaded onto programmable switches.

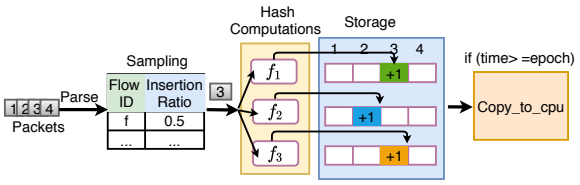


Figure 3: Measurement tasks with four common steps: *sampling*, *hash computations*, *storage*, and *copy_to_cpu*.

B. Related works

Program resource processing. Recent studies [18], [19], [20] achieve code optimization at the program level by virtualization to optimize resource usage. However, these solutions only target a single device and ignore the inherent constraints that exist for programmable switches. Some solutions [7], [21] aim to create new implementation algorithms or reuse essential hardware resources for measurement tasks on a single programmable switch. Nevertheless, these approaches either oversimplify measurement tasks to support multiple programs, which may result in decreased performance (e.g., lower accuracy), or they are designed for particular measurement tasks only, lacking generality. In contrast, the distributed deployment for measurement tasks is proposed in this paper, which extends the measurement process logic from a single switch to multiple switches within the network while offering optimal performance.

Program merging and deployment. SPEED [8] and Hermes [9] use program merging to reduce switch resource usage and distribute programs across switches. MTP [22] improves SPEED by preventing control plane overload, and Eagle [23] is designed to address the scalability in network-wide sketch deployment. These frameworks offer valuable solutions for efficient program deployment and resource optimization, while *DisPLOY* focuses on the *target constraint inherent in measurement tasks*. The target constraint remains unexplored in previous works, and ignoring it will lead to performance degradation or even failure deployment of tasks. However, target constraint significantly complicates the optimization problem, making it become both *device-dependent* and *order-dependent*. Thus, *DisPLOY* is not merely complementary but presents a novel and significantly advanced approach to distributed program deployment.

III. *DisPLOY* Design

A. Overview

Network measurement serves as an essential data collection and analysis tool for various co-located network services, including traffic engineering, scheduling, and anomaly detection [24]. Given that these services share the same infrastructure yet have different monitoring aims, any distributed deployment framework must support measurement tasks by specifying the following monitoring targets: (1) enabling any traffic to be measured, (2) gathering necessary link/device information. This allows operators to execute customized monitoring queries and adapt to network topology alterations.

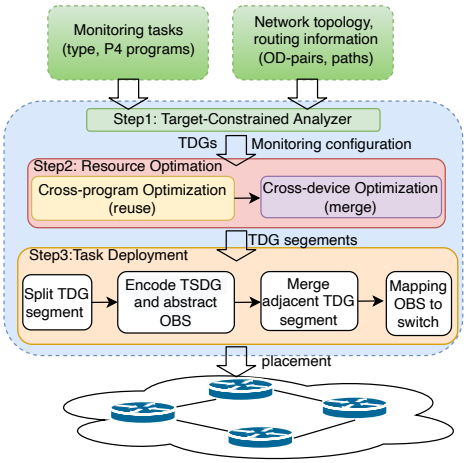


Figure 4: *DisPLOY* overview.

We present *DisPLOY*, a framework for deploying distributed measurement tasks with aforementioned monitoring targets while supporting resource-efficiency and high-performance monitoring. As depicted in Figure 4, *DisPLOY* is comprised of the following components: *target-constrained analyzer*, *resource optimization*, and *task deployment*.

Target-Constrained Analyzer. The *target-constrained analyzer* processes user-defined monitoring targets for measurement tasks and incorporates task specification into a monitoring configuration profile for *resource optimization*. It also converts input measurement programs into extended TDGs using the p4c [25] compiler tool considering target constraints. These TDGs offer an abstract representation of the packet processing logic, each comprising several MATs.

Resource Optimization. The *resource optimization* process utilizes input TDGs and monitoring configurations to perform cross-program and cross-device optimization for improving resource efficiency. During cross-program optimization, mergeable MATs are identified to merge the input TDGs. Cross-device optimization identifies device-independent operations to reuse key hardware resources across measurement instances on different devices.

Task Deployment. The *task deployment* process deploys the merged TDG into the network under the target constraints. This involves preprocessing TDG segments by splitting and then merging them based on a TDG Segment Directed Graph (TSDG) and OBSs. Eventually, the TDG segments are placed on target devices to minimize resource overhead (i.e., the number of stages occupied) and to achieve high performance by ensuring the shortest distance among deployed switches.

B. Target-Constrained Analyzer

1) Task Specification: We consider a typical measurement scenario involving the deployment of multiple measurement tasks (e.g., heavy hitter detection and cardinality estimation) in a network with multiple paths connecting various ingress and egress nodes. Monitored traffic can originate from any ingress nodes and terminate at one or more egress nodes. Beyond this any-to-any traffic, the observed devices or links can include one or multiple devices, or even all devices (per-hop) along the packet forwarding path between any network endpoints. Routing information for monitoring traffic is easily obtained and generally changes infrequently in production networks [24].

Each measurement task t is designed to monitor specific Origin-Destination pairs (OD-pairs), and each OD-pair is associated with a unique router-level path. *DisPLOY* focuses on two types of measurement tasks: device-local measurement (e.g., detecting abnormal jitters) and network-wide measurement (e.g., collecting per-hop latency). Formally, a measurement task t in *DisPLOY* is defined as:

- Measurement Type: Local (0) or Network-Wide (1);
- Physical Path: $P_t = \{d_1 \rightarrow ... \rightarrow d_k\}$ of the monitored traffic.

In multiple routing paths, such as those using equal cost multi-path routing (ECMP), *DisPLOY* treats each path as a distinct logical OD-pair.

2) Extended TDG: Existing frameworks [8], [9], [23] utilize the *p4c* compiler [25] to generate TDGs from measurement programs. A TDG is represented as $T = (V_T, E_T)$, a directed acyclic graph (DAG), where V_T is the set of nodes representing MATs, and E_T is the set of edges representing dependencies between MATs. *DisPLOY* allocates MATs to one or multiple devices along the monitored traffic path. *DisPLOY* extends this standard TDG by incorporating target constraints with $\phi(T) = \{d_1, ..., d_k\}$, where d_k represents a deployable target device for T. Each MAT $u \in V_T$ in *DisPLOY*'s TDG is augmented with a property, $\phi(u)$, initialized to $\phi(T)$.

Note that the MAT properties of $u \in V_T$, including matching fields (i.e., $M(u)$), actions (i.e., $A(u)$), modified fields (i.e., $F(u)$), matching rules (i.e., $R(u)$), capacity (i.e., $C(u)$) follow the same definitions as existing works [8], [18]. The dependency matrix D_T of T represents dependencies between MATs. $D_T(u, v)$ indicates whether MAT u has a dependency on MAT v: 0 denotes no dependency, while 1 denotes a dependency. The dependency rules (match, action, etc.) in E_T have been proven to cover all practical forwarding rules, including lpm [8]. *DisPLOY*'s extension focuses solely on augmenting TDGs with target constraints $\phi(T)$, without altering the underlying dependency model.

C. Resource optimization

In software-defined measurement (SDM) [6], [26], administrators aim to measure various traffic statistics (e.g., heavy flow size or the distinct number of flows) by simultaneously deploying multiple measurement algorithms (e.g., sketches [26]). These measurement tasks often execute the same operations in their programs, such as *random generation*, *hash computation*, and *copy_to_cpu* (Figure 3). When network-wide measurements are conducted, identical operations are performed on various devices running the same instances. For example, all devices along the packet forwarding path need to perform *hash computation* and store the flowkey (e.g., 5-tuple) to estimate per-hop per-flow latency.

These observations highlight the need for *DisPLOY* to merge redundant operations across measurement programs and reuse key hardware components (such as hash units) across devices. *DisPLOY* introduces cross-program and cross-device optimizations, which aim to improve resource efficiency.

1) Cross-program optimization: The cross-program optimization is designed to merge identical MATs with target constraint across different program TDGs to eliminate redundancy for measurement tasks. *DisPLOY* extends traditional MAT merging by incorporating target constraints into the mergeability criteria.

Device-dependent and order-dependent merging. Cross-program optimization focuses on target constraint within various measurement tasks, which may result in MATs different deployable target devices. To ensure accurate program logics, *DisPLOY* requires that mergeable MATs have at least one shared deployable device, as each MAT can only be deployed on a single device. Figure 2 demonstrates this necessity for merging MATs. The MAT b at second level in order 2 shown in Figure 2(b) can not be merged due to the absence of common device between A and B.

Thus, cross-program optimization allows two MATs in different TDGs, say u and v, can be merged if they satisfy (i.e., mergeable MATs):

$$\underbrace{\Big(M(u) = M(v) \wedge A(u) = A(v)\Big)}_{\text{Semantic Equivalence}} \wedge \underbrace{\Big(\phi(u) \cap \phi(v) \neq \emptyset\Big)}_{\text{Device Co-location}},$$

while maintaining original TDG dependencies and loop-free in merged TDG. The MAT dependencies are supposed to be preserved during TDG merging to maintain the original program logics. The loop-free requirement is necessary because programmable switches do not support loop operations.

Merging algorithm. Due to the device-dependent and order-dependent nature of merging under target constraints (as illustrated in Figure 2), *DisPLOY* employs a Huffman-based algorithm (Algorithm 1) that iteratively merges two TDGs with the maximum number of mergeable MATs until no further MATs can be merged.

This Huffman-based merging algorithm is a two-phase merging strategy that (1) greedily merges common MATs with overlapping devices (lines $1 \sim 12$), and (2) propagates constraints to ensure validity (lines $13 \sim 21$). The algorithm starts by identifying candidate TDG pairs with the largest Longest Common Subsequence (LCS) of mergeable MATs. Specifically, *DisPLOY* initializes a priority queue $\mathbb{Q}$ for the input TDGs $\mathbb{T}$, sorted by the number of mergeable MATs determined by the LCS of the TDGs (line 3). The algorithm iteratively extracts the pair of TDGs (T_i,T_j) with the maximum number of mergeable MATs (line 6). For each pair, *DisPLOY* merges the common MATs and updates their deployable target devices to the intersection of their original devices. This ensure the merged MATs are deployable on shared switches (lines $9 \sim 12$).

Non-merged MATs may have dependencies on merged ones. To ensure program logic accuracy and merging validity, *DisPLOY* propagates constraints to non-merged MATs. Specifically, let d_{first} and d_{last} denote the first and last devices of the MATs merged in LCS. *DisPLOY* updates the valid devices of a dependent MAT u as:

$$\phi(u) \leftarrow \phi(u) \cap \{d \in \phi(T_i) \mid d_{\text{first}} \leq d \leq d_{\text{last}}\},$$

where $\phi(T_i)$ (or $\phi(T_j)$) represents the initial set of target deployable devices for the input TDGs, i.e., the devices along the physical monitored traffic path. This ensures that the program logic remains accurate, which means that if MAT v depends on MAT u, MAT u is deployed on the device preceding MAT v or on the same device (lines $13 \sim 15$). After propagation, the input TDGs are split into several TDG segments, each containing MATs with the same deployable target devices. *DisPLOY* checks if there are enough resources for each TDG segment to be placed on their target devices using the CALCVALIDDEVICES() function. If yes, the merged TDG is built and reinserted into the priority queue (lines $17 \sim 19$). Otherwise, the pair is marked as attempted and will be skipped in future iterations (line 21 and lines $7 \sim 8$).

Algorithm 1 *Device- and order-dependent* TDG merging.

Require: Set of initial TDGs $\mathbb{T}$, task specification $\mathbb{P}$

1: **procedure** MERGETDGS($\mathbb{T}$, $\mathbb{P}$)
2: Initialize priority queue $\mathbb{Q}$
3: $\mathbb{Q} \leftarrow \mathbb{T}$ sorted by $\text{LCS}(T_i, T_j)$
4: Initialize set of attempted pairs $\mathbb{A} \leftarrow \emptyset$
5: **while** $\exists$mergeable pairs in $\mathbb{Q}$ **do**
6: $(T_i, T_j) \leftarrow$ BESTPAIR($\mathbb{Q}$) ▷ Extract max
7: **if** $(T_i, T_j) \in \mathbb{A}$ **then**
8: **continue** ▷ Skip already attempted pair
9: $L \leftarrow \text{LCS}(T_i, T_j)$
10: **for** $(u, v) \in L$ **do**
11: $w \leftarrow u \oplus v$ ▷ Merge MATs
12: $w.\text{dev} \leftarrow u.\text{dev} \cap v.\text{dev}$
13: $d_{\text{first}}, d_{\text{last}} \leftarrow$ the first and last device of L
14: **for** each u dependent on L in $T_i \cup T_j$ **do**
15: $\phi(u) \leftarrow \phi(u) \cap \{d \in \phi(T_i) \mid d_{\text{first}} \leq d \leq d_{\text{last}}\}$
16: $\mathbb{M} \leftarrow$ CALCVALIDDEVICES($T_i \cup T_j, L$)
17: **if** VALIDATE($\mathbb{M}$) **then**
18: $T_{new} \leftarrow$ BUILDTDG($L, \mathbb{M}$)
19: $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{T_{new}\} \setminus \{T_i, T_j\}$
20: **else**
21: $\mathbb{A} \leftarrow \mathbb{A} \cup \{(T_i, T_j)\}$ ▷ Mark as attempted
22: **return** $\mathbb{Q}$

2) Cross-devices optimization: The cross-devices optimization is used to reuse the key hardware resources across the same network-wide measurement instances on different devices.

Resource reusing. The design of cross-device optimization is based on the observation that operations such as hash computation, updating flow-related statistics (such as flowkey and flow frequency), and random generation are identical but *device-independent* across different devices in network-wide measurement. *Device-independent* means that the results of these operations are not tied to a specific device when packets arrive. For example, the packet counter for a specific flow remains consistent as it traverses various devices along the

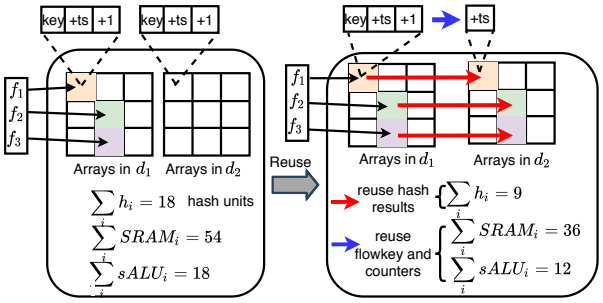


Figure 5: Reduces hash units, SALU and SRAM by reusing hash results and device-independent information[1].

forwarding path. This creates an opportunity to reuse resources across each participating device. Specifically, *DisPLOY* identifies and executes *device-independent* operations on the initial device along the packet forwarding path. It then embeds the results of these operations (e.g., hash) into the packet header before forwarding it to subsequent devices. Upon receiving the packet, the following devices can extract the metadata fields and continue performing the remaining measurement logic without redundant execution. This allows the subsequent devices to reuse the storage (e.g., SRAM and TCAM) and computation (e.g., hash) resources of the initial device.

Example. Consider a network-wide measurement task that requires deploying a sketch algorithm on k devices along the packet forwarding path. The sketch is structured as an array with r rows and each row contains w buckets. Each bucket captures m_1 device-independent and m_2 device-dependent statistics. Without optimization, the total counts of hash units, SRAM usage, and stateful arithmetic logic unit (SALU) usage are $\sum_{i=1}^{k}\sum_{j=1}^{r} w$, $\sum_{i=1}^{k}\sum_{j=1}^{r} w*(m_1+m_2)$, and $\sum_{i=1}^{k}\sum_{j=1}^{r}(m_1+m_2)$, respectively. When reusing optimization is enabled, they are reduced to $\sum_{j=1}^{r} w$, $\sum_{j=1}^{r} w*(m_1+m_2) + \sum_{i=1}^{k-1}\sum_{j=1}^{r} w*m_2$, and $\sum_{j=1}^{r}(m_1+m_2) + \sum_{i=1}^{k-1}\sum_{j=1}^{r} m_2$, accordingly. In Figure 5, the sketch measurement task is deployed on two devices. Each bucket stores two device-independent statistics, i.e., flowkey (key) and packet counter (+1), and one device-dependent statistic, i.e., packet processing latency (ts). Due to the reusing optimization, the number of hash units is decreased from 18 to 9, SRAM from 54 to 36, and SALU from 18 to 12.

D. Task Deployment under target constraints

1) Problem Statement: The deployment framework takes the merged TDG T_{t_m}, consisting of m TDG segments, and n task specifications $P_{t_1}, P_{t_2}...P_{t_n}$ as input. The output is a set of decision variables indicating which switch stage each MAT in merged TDG should be deployed to. Each switch has two properties: (1) Each switch has a finite number C_{stage} of packet processing stages. (2) Each stage of a programmable switch has a finite capacity of resources, including TCAM, SRAM, hash unit, logic tables and sALU, and C_{res} to represent the overall capacity of per-stage resources. The variables $x(u,i,a)$ represent whether the MAT $u \in V_{T_m}$ is assigned to stage i of target switch a. If MAT u is deployed, $x(u,i,a)$ is set to 1; otherwise, it is set to 0.

Objective. *DisPLOY* aims to minimize the number of stages occupied, which equals to minimizing the stages within deployable target devices taken up by the MATs in V_{T_m}:

$$\min \sum_{a\in\phi(T_1)\cup\phi(T_2)\cup\ldots,\phi(T_m)} \sum_{i\in[1,C_{\text{stage}}]} x(*,i,a),$$

where $x(*,i,a)$ represents whether stage i in switch a is taken up by MATs. $x(*,i,a)$ is set to 1 if it is occupied, otherwise 0.

Also, we aim to minimize the total distance between deployed switches for high monitoring performance. Let $l(a,b)$ represent the distance between switches a and b (e.g. the number of hops), where switch a operates on u and switch b on v for each edge $(u,v) \in E_{T_m}$. The overall distance among deployed switches is the cumulative path length of each edge traversed, given by:

$$\min \sum_{u,v\in E_{T_{tm}}} \sum_{a\in\phi(u),b\in\phi(v),a\neq b} l(a,b)\cdot x(u,*,a)\cdot x(v,*,b).$$

The binary variable $x(u,*,a)$ denotes whether the MAT u is deployed on switch a. If MAT u is deployed, $x(u,*,a)$ is assigned as 1; otherwise, it is 0.

Why Minimize Deployment Distance? Minimizing the distance between deployed switches serves two critical purposes:

- Bandwidth Efficiency: Fewer hops between switches reduce the volume of telemetry data transmitted across the network, lowering bandwidth consumption.
- Latency Reduction: Network propagation delay is proportional to path length. Closer switches inherently reduce latency for real-time measurement tasks (e.g., anomaly detection).

Unlike prior works that optimize for a single metric (e.g., SPEED for latency, Hermes for bandwidth), *DisPLOY* seeks a balanced optimization through spatial aggregation. This is critical for scenarios like data center monitoring, where both metrics impact operational efficiency.

MAT placement constraints. Each MAT $u \in V_{T_{t_m}}$ should be placed at least one stage of deployable switch for monitoring targets, i.e.,

$$\sum_{a\in\phi(u)} \sum_{i\in[1,C_{stage}]} x(u,i,a) \geq 1, \forall u \in V_{T_{t_m}}. \quad (1)$$

MAT dependency constraints. The placement solution is required to adhere to the MATs dependencies. Given two MATs u, v, if v depends on u with a match/action dependency, the start stage occupied by v must occur after the end stage occupied by u. More precisely, the task framework encodes the MAT dependency with two constraints.

(1) For each $(u,v) \in E_{T_t}$, u and v are deployed on different switches, the switch a that runs u should be the upstream of the switch b that runs v. In other words, a should deliver its

[1]This reuse optimization applies to common MATs with all possible dependencies among forwarding rules (e.g., *lpm* and *range*) if the device-independent condition is satisfied.

packets to b through the path P_t at runtime. If so, $y(a, b, P_t) = 1$; otherwise, $y(a, b, P_t) = 0$. Thus,

$$\forall (u, v) \in E_{T_t}, T_t \in \Phi, a \in \phi(u), b \in \phi(v), a \neq b,$$
$$D_{T_t}(u, v) = 1, x(u, *, a) \cdot x(v, *, b) = 1 : y(a, b, P_t) = 1. \tag{2}$$

(2) When both MATs are deployed on the same switch, MAT u must be executed before MAT v. The start and end stages for MAT u are denoted by s_u and e_u. To ensure completion of MAT u before starting MAT v, e_u must precede s_v:

$$\forall u, v \in V_{T_{t_m}}, D_{T_{t_m}}(u, v) = 1 : e_u < s_v. \tag{3}$$

Switch resource constraints. The MATs assigned to a specific stage do not exceed the stage's available resource capacity. For each resource type $z \in R = \{$*Logic Tables*, *Hash Units*, *sALU*, *SRAM*, *TCAM*$\}$, the following condition must be satisfied:

$$\sum_{u \in V_{T_{t_m}}} \sum_{z \in R} r_{s,z,u} \leq C_{res}, s \in C_{stage}, \tag{4}$$

where the boolean variable $r_{s,z,u}$ indicates the resource z consumed by the MAT u in stage s.

Theorem III.1. *The problem of task deployment is NP-hard.*

Proof. We rigorously prove the NP-hardness via a polynomial-time reduction from the bin packing problem (BPP), a well-known NP-hard problem.

Step 1: Considering an instance of the BPP with items of sizes s_1, s_2, ..., s_n and bin capacity C, we construct a corresponding instance of the task deployment problem. In this instance, each input TDG is restricted to having only one MAT, and the target deployable switch can accommodate only one. Thus, each item s_i corresponds to a MAT with resource demand s_i; each bin corresponds to a stage in the deployable switch, with stages capacity C. The objective is to minimize the total number of stages that the MATs occupy within the single switch, while adhering to the constraints specified in Equations (3) and (4).

Step 2: We establish the equivalence between solutions of the BPP and the task deployment problem:

- *BPP* $\Rightarrow$ *Task Deployment*: A valid bin packing solution (items packed into k bins) directly provides a task deployment solution with k stages.
- *Task Deployment* $\Rightarrow$ *BPP*: Conversely, a deployment solution with k stages implies a valid bin packing of items into k bins.

Step 3: Since BPP is NP-hard and the above reduction is polynomial-time, the above problem is also NP-hard. □

2) Deployment Algorithm: We incorporate a heuristic algorithm in the framework to efficiently address the deployment challenge. This algorithm prioritizes deploying measurement programs on a single device when possible and, if not, favors deploying on the nearest devices through three steps (Algorithm 2).

First, *DisPLOY* pre-processes TDG segments by recursively splitting them using `SPLIT_TDGSEGMENT`. This addresses potential non-adjacent target devices from resource optimization. The function divides TDG segment (say T) until sub-segment (T_1, T_2) associate with adjacent deployable devices ($\phi(T_1), \phi(T_2)$). It iteratively moves topologically sorted MATs from V_{T_2} to V_{T_1}, recursively splitting T_2 until complete (lines $1 \sim 12$).

Second, *DisPLOY* merges adjacent TDG segments by encoding TDG segments into a TDG Segment Directed Graph (TSDG) to optimize resource utilization. Each TSDG is a directed graph $G = (V_G, E_G)$, where nodes in V_G represent TDG segments, and edges in E_G indicate execution dependencies between segments. Each TDG segment has one or more deployable target devices for MAT deployment abstracted as OBS. *DisPLOY* creates OBS by logically aggregating resources of deployable target switches for each TDG segement. For example, k switches with 2 stages each form an OBS with $2k$ stages, enabling MAT deployment with respect to dependencies and aggregated resources (line 16).

For TDG segment u_g and v_g, if $\phi(u_g) \subseteq \phi(v_g)$, their OBS are categorized as *sub-OBS* and *par-OBS*, respectively. *DisPLOY* identifies and sorts the TDG segments with *sub-OBS* by target deployable device count and betweenness centrality (the betweenness centrality of OBS sums the centralities of devices within the OBS), then performs merging operations sequentially. For segment u_g, it merges with adjacent v_g if $\phi(u_g) \subseteq \phi(v_g)$, moving MATs from V_{v_g} to V_{u_g} topologically. If multiple adjacent segments qualify for merging, *DisPLOY* offers two choices: either first choose the higher-level segment that can fully merge with u_g (FLFM) or directly find the segment with a higher level (FSL). The level of a TDG is determined by the length of its topological sort sequence (lines

Algorithm 2 Deploying TDG segments.

Require: The merged TDG consists of several TDG segments Ω, each segment T's deployable target devices $\phi(T)$.

1: **function** SPLIT_TDGSEGMENT(T)
2: Sort V_T via topological sorting
3: Initialize $T_1 = (V_{T_1}, E_{T_1})$ and $T_2 = (V_{T_2}, E_{T_2})$
4: Initialize $V_{T_1}, V_{T_2} \leftarrow \emptyset, V_T$
5: $\phi(T_1), \phi(T_2) \leftarrow DivTargetDevices(\phi(T))$
6: **for** $u \in V_T$ **do**
7: Move u from V_{T_2} to V_{T_1}
8: **if** $|T_1| > |\phi(T_1)|$ **then**
9: break ▷ No resources to deploy MATs
10: $E_{T_1} \leftarrow (u, v) | (u, v) \in E_T, u, v \in V_{T_1}$
11: $E_{T_2} \leftarrow (u, v) | (u, v) \in E_T, u, v \in V_{T_2}$
12: SPLIT_TDGSEGMENT(T_2)
13: **procedure** DEPLOY_SEGMENTTDGS(Ω)
14: **for** $T \in \Omega$ **do**
15: SPLIT_TDGSEGMENT(T)
16: Encode TDG segments Ω into a TSDG $G = (V_G, E_G)$ and OBS for each TDG segment
17: Identify and sort TDG Segments with sub-OBS in G
18: MERGE_ADJACENT_TDGSEGMENT(G)
19: PLACETOOBS_MINIMIZESTAGE(G)
20: SELECT_SWITCHES_WITH_PRIM(G)

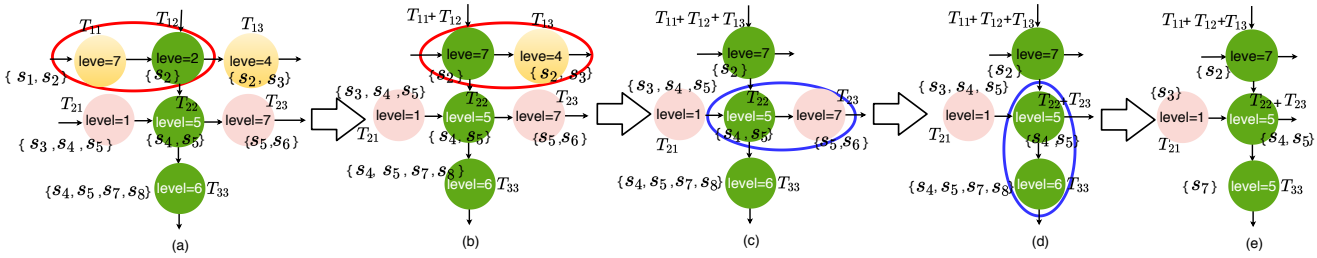


Figure 6: Example of merging TDG segments.

$17 \sim 18$). Evaluation results in Section IV demonstrate that FLFM prevents resource fragmentation, while FSL reduces stage overhead.

Finally, *DisPLOY* maps TDG segments to devices using Prim's algorithm for shortest network distance. The use of Prim's algorithm ensures TDG segments are deployed on the shortest network paths between target devices. This achieves fewer intermediate switches reduce data aggregation overhead and shorter paths minimize propagation delay. `PlaceToOBS_MinimizeStage` assigns TDG segments to OBSs to minimize stages. *DisPLOY* then calls the function `SELECT_SWITCHES_With_Prim` to select the nearest device to the current selected devices set from the deployable target devices' boundary points of the TDG segment for OBSs placement (lines $19 \sim 20$).

Example. Figure 6 shows an example of merging the TDG segments (FSL). *DisPLOY* first executes the merging operations for the TDG segment T_{12}+T_{31} with its adjacent TDG segments T_{11}, T_{13} orderly since T_{11} has higher level than T_{13}, as shown in Figures 6(a) and 6(b). Then *DisPLOY* operates merging for T_{22}+T_{32} with its adjacent TDG segments until the device of s_4, s_5 can not deployed any MATs of T_{33} as shown in Figures 6(c) and 6(d). Finally, *DisPLOY* maps the merged TDG segments to devices via the shortest paths, selecting devices $\{s_2\}$, $\{s_3\}$, $\{s_4, s_5\}$, $\{s_7\}$ over $\{s_2\}$, $\{s_3\}$, $\{s_4, s_5\}$, $\{s_8\}$ shown in Figure 6(e).

Complexity Analysis. The task deployment algorithm splits TDG segment with worst-case time complexity of $O\left((|V_{T_{t_m}}| + |E_{T_{t_m}}|) \cdot \log |V_{T_{t_m}}|\right)$. Then it merges adjacent TDG segments, leading to the complexity of $O(|V_{t_m}|^2)$. Also, it maps the OBS into the target devices by Prim algorithm that introduces the complexity of $O(|V_G|^2)$. Thus, the time complexity of the task deployment algorithm is $O\left((|V_{T_{t_m}}| + |E_{T_{t_m}}|) \cdot \log |V_{T_{t_m}}|\right) + O(|V_{T_{t_m}}|^2) + O(|V_G|^2) = O(|V_{T_{t_m}}|^2) + O(|V_G|^2)$.

IV. EVALUATION

A. Experiment Setup

We have validated *DisPLOY* through implementation on both P4 hardware switches (Intel Tofino ASIC) and BMv2. *DisPLOY* supports $P4_{14}$ and $P4_{16}$ programs.

Measurement programs and comparison. Our evaluation used fourteen P4 programs, categorized into three sets (Table II): V1 (per-hop flow-level, for resource reuse validation), V2 (typical network tasks like drop detection [29] and flow statistics [6]), and V3 (sketch algorithms, e.g., Count-Min, Bloom, Elastic Sketch). V2/V3 evaluate merging/deployment optimizations for common tasks and sketch algorithms.

Table II: Measurement P4 programs.

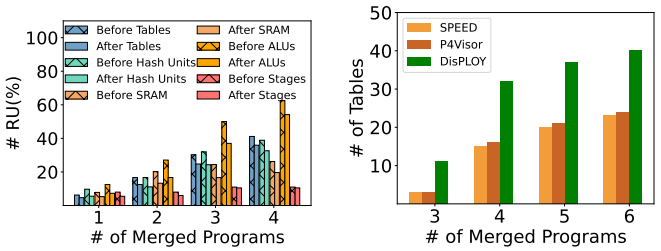
Name	# MATs	# Dependencies	Note
V1			
Lg_max [6]	29	37	Network-wide measurement (per-hop flow-level)
Lg_sum [6]	31	37	
DeltaINT [3]	6	8	
TurboFlow [27]	16	19	
V2			
LogLog [28]	4	3	Typical network measurement tasks (sketch, INT and drop detection)
FlowRadar [29]	79	93	
Elastic sketch [30]	21	29	
Spreadsketch [31]	24	26	
Lg_max [6]	29	37	
Lg_sum [6]	31	37	
V3			
CountMin [32]	16	15	Sketch-based measurement
CountSketch [33]	23	31	
HyperLogLog [34]	4	3	
PCSA [35]	6	5	
UnivMon [36]	34	46	
Sketchlearn [26]	24	26	

Table III: Topologies used in experiments of simulator.

Topology	AboveNet	AT&T	Internet2	Fat Tree (k=4)	Fat Tree (k=10)
# Nodes	22	14	13	20	125
# Edges	29	29	13	24	300

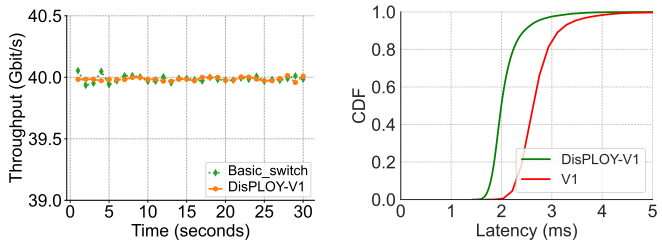
We compare *DisPLOY* with two state-of-the-art task deployment frameworks: P4Visor [18] and SPEED [8]. SPEED chooses to only merge default-only MATs for minimizing stages (using Gurobi [37]) but lacks target-constraint awareness. P4Visor is a virtualization platform that merges programs into a compound pipeline without considering target device constraints, supporting all MAT types. We further extended SPEED by incorporating target constraints, referred as "SPEED + Target Constraint". This version deploys tasks on specified switches, applying SPEED merging/deployment only when target constraints are met; otherwise, direct deployment is used. We also compare our approach with a brute-force method ("Optimal") that explores all merging orders to maximize merged MATs, disregarding deployment resource constraints. Additionally, we implement a baseline that deploys TDGs directly along packet forwarding paths without optimization.

Testbed and simulator. Our experiment setup included two Tofino switches and two servers (Intel i7-4771 3.50GHz), sequentially connected via 40Gbps fiber. Servers generated/received traffic using a 2010 data center trace (UN2) [38].



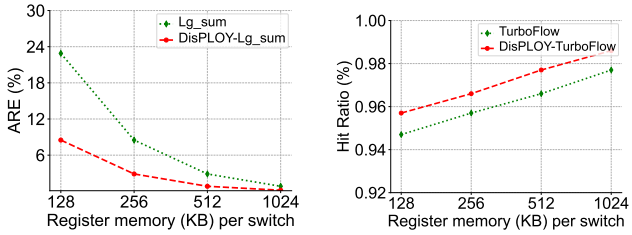
(a) Resource usages with reusing. (b) Merged MATs with reusing and target-constraint merging.

Figure 7: Resource efficiency and combined merging performance.



(a) Throughput. (b) Processing Time.

Figure 8: End-to-end performance with *DisPLOY*'s reusing.



(a) ARE. (b) Hit Ratio.

Figure 9: Measurement utility with *DisPLOY*'s reusing.

We also conducted evaluations in Mininet using P4 software switches (BMv2) and developed a Python simulator to evaluate *DisPLOY* across diverse topologies: AboveNet [39], AT&T [40], Internet2 [41], Fat Tree (k=4/k=10, Table III). For each task (OD-pairs), we randomly assigned monitoring targets (traffic flows/links), selecting random OD-pairs for monitored traffic paths.

B. Evaluation of resource reusing

Impact of resource usages. We measured hardware resource usage on Tofino switches to show resource reuse effectiveness. Deploying V1 programs concurrently (with/without reuse), we measured resource usage (tables, hash units, SRAM, meter-ALUs, stages; $1 \sim 4$ concurrent programs). As shown in Figure 7(a), *DisPLOY* reduces the usage of tables by $14\% \sim 25\%$, hash units by $15\% \sim 43\%$, SRAM by $25\% \sim 35\%$, sALU by $13\% \sim 41\%$, and stages by $4\% \sim 31\%$, compared to no-reuse deployments. This is because *DisPLOY* only stores the flowkey and performs hash functions in the initial switch. Then, *DisPLOY* applies the hash result in the subsequent switches and shares the flowkey with others in the controller. This method helps reduce the requirement for SALUs (hardware for memory accesses) and SRAM resources for measurement tasks.

Impact on end-to-end performance. To validate the *DisPLOY*'s capability in maintaining end-to-end performance with resource reusing, we conducted experiments on Tofino switches and BMv2 using three program configurations: (1) Basic_switch, a baseline program for packet forwarding, (2) *DisPLOY*-V1 (V1 programs with resource reusing optimization of *DisPLOY*), and (3) V1, bashed on the real-world traffic. Hardware experiments on Tofino (Basic_switch vs. *DisPLOY*-V1) shows equivalent throughput (Figure 8(a)), indicating no line-rate impact from reuse. BMv2 latency tests shows reduced processing time with *DisPLOY* (Figure 8(b)): average from 2.645ms to 2.123ms, 99th percentile reduction of 1.003ms. This reduction is attributed to resource reusing, which reduces match-table traversals for each packet, reducing latency overhead.

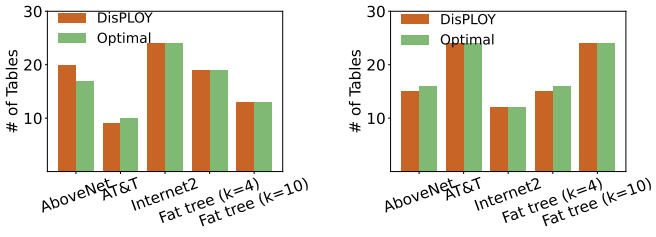
Impact on measurement utility and accuracy. We evaluate measurement utility through two metrics: 1) accuracy relative error (ARE): $\frac{|\hat{v}-v|}{v}$ where v and $\hat{v}$ denote ground truth and estimated values. 2) hit ratio: which is the ratio of packets directly processed by the register memory in the switch. Figures 9(a) and Figure 9(b) reveal that the task of lg_sum improves ARE by 62.8%-78.4% in flow size estimation and the hit ratio of TurboFlow improved from 0.947 to 0.986 compared to non-reusing deployment. This enhancement stems from resource reusing of *DisPLOY* enabling more resources to be allocated to measurement tasks without conflict.

C. Evaluation of device- and order-dependent merging

Impact on merged MATs and execution time. We evaluate the impact of device-dependent and order-dependent merging by comparing *DisPLOY* and Optimal for V2/V3 programs across topologies listed in Table II. The results show comparable merged MATs for *DisPLOY* (Figures 10). And *DisPLOY* completes the merging process in less than 0.1 seconds, even in large-scale topologies, e.g., Fat Tree (k=10) with 125 nodes, as shown in Figure 11. Note that this evaluation was conducted with a small number of programs (only six tasks). As the number of programs increases, the execution time for the optimal solution also increases (greater than 10 minutes), limiting its scalability for deploying concurrent monitoring tasks.

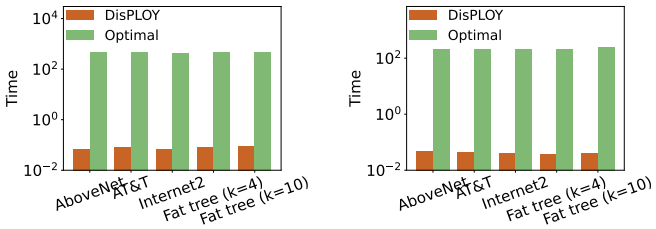
Furthermore, Figure 11 shows that the optimal strategy results in fewer merged MATs compared to *DisPLOY* when deploying the V2 program on the AboveNet topology. This is because the optimal solution overlooks potential deployment failures due to insufficient stage resources in certain merging order. In contrast, *DisPLOY* merges only if target devices have sufficient resources, ensuring deployment success and resource efficiency.

Impact on resource usages. We further assess the merging of measurement programs by varying the number of concurrent tasks ranging from 3 to 6, and measure the resource usage of



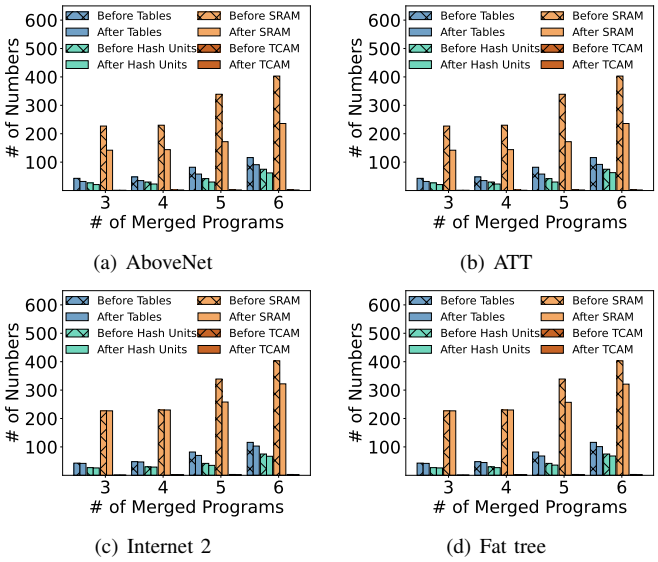
(a) V2. (b) V3.

Figure 10: Impact on merged MATs.



(a) V2. (b) V3.

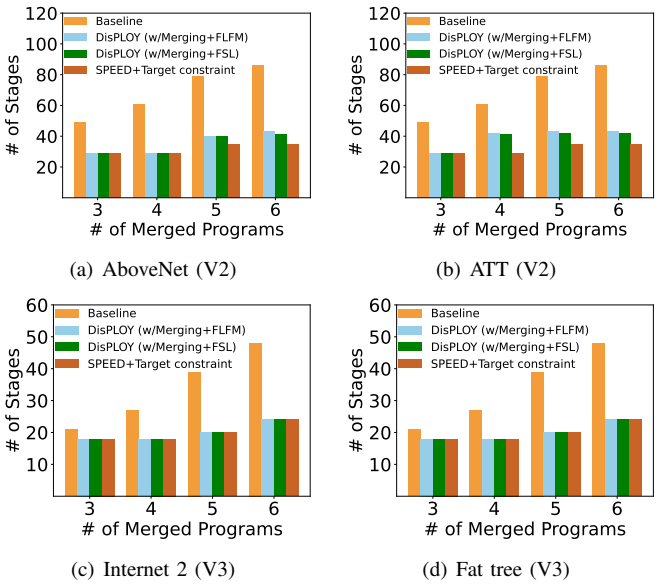
Figure 11: Execution time of merging.



(a) AboveNet (b) ATT

(c) Internet 2 (d) Fat tree

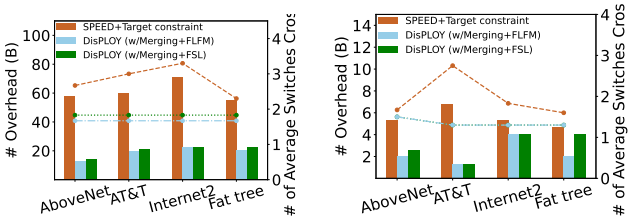
Figure 12: Impact of merging (V3) on resources.



(a) AboveNet (V2) (b) ATT (V2)

(c) Internet 2 (V3) (d) Fat tree (V3)

Figure 13: Impact on stage resources of task deployment.



(a) V2. (b) V3.

Figure 14: Impact on per-packet byte overhead and the number of hops traversed.[2]

DisPLOY with and without merging in our simulation. The results, illustrated in Figures 12, show that *DisPLOY* achieves significant reductions in resource usage, vs. no merging: up to 29% in the number of tables, 28.5% in hash unit utilization, 49% in SRAM usage, and 50% in TCAM allocation for V3 programs, highlighting merging effectiveness.

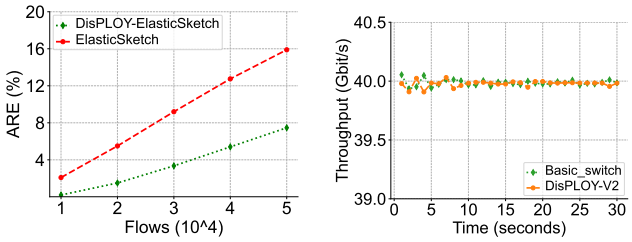
Combined merging performance. To validate the combined effect of cross-program and cross-device optimizations, we conduct a comparative study with P4Visor [18] and SPEED. The experiment adopts Fat Tree (k=4) topology based on V2 measurement tasks, but lg_max and lg_sum is set network-wide measurement task type. From the result (Figure 7(b)), *DisPLOY* achieves 66% and 73% more merged MATs than P4Visor and SPEED, respectively, demonstrating effectiveness of combined optimizations over P4Visor/SPEED.

D. Evaluation of target-constrained deployment

Impact on the occupied stages. We evaluate the impact of TDG segment placement on the utilization of occupied stages. To do this, we deploy V2 and V3 optimized by different methods in various network topologies. In *DisPLOY*, we use two placement strategies (FSL and FLFM) to place the merged TDG segments on the target switches. We compare *DisPLOY* with the baseline and "SPEED + Target constraint". From the results, *DisPLOY* reduced stages significantly (Figures 13): up to 62% (V2), 66% (V3) vs. baseline. Comparable stages to "SPEED + Target constraint".

Impact on bandwidth usages. In addition to the simulator, we also measure the overheads incurred by inter-device packet scheduling that preserves original program logics in Mininet. Compared to "SPEED + Target constraint", *DisPLOY* can

[2]Due to page limitations, we do not include experimental results for all topologies, which exhibit similar observations.



(a) ARE. (b) Throughput.

Figure 15: Flow size estimation and throughput.

significantly reduces per-packet byte overhead (up to 75% and 80% in V2 and V3, respectively) and the average number of switches crossed during measurement process, as shown in Figure 14. This can be attributed to the deployment optimization of *DisPLOY*, which aims to minimize the deployment distance. Fewer hops between switches reduce the volume of telemetry data transmitted across the network, reducing bandwidth consumption. In contrast, SPEED only considers a single metric, minimizing latency, without considering the bandwidth overhead of inter-switch coordination.

Impact on accuracy and end-to-end throughput. To further validate the practical benefits of target-constrained deployment in *DisPLOY*, we evaluate the accuracy of elastic sketch on Tofino switches using real-world traffic traces. We compared the ARE of elastic sketch with and without *DisPLOY*'s optimization under varying flow counts (10k to 50k flows). Figure 15(a) shows that the accuracy of flow size estimation on Tofino was significantly improved by *DisPLOY*: ARE 0.1983 vs. 2.0848 (conventional, 90.48% reduction), due to stage reduction enabling more hash tables for light part in elastic sketch.

V. Conclusions

We present *DisPLOY*, a target-constrained framework for distributed measurement task deployment. *DisPLOY* enables specifying monitoring traffic/devices for multiple programs. Merging and reuse reduce TDG redundancy and switch resources. A target-constrained deployment solution is proposed to minimize stages and achieve high-performance monitoring. Experiments demonstrate that *DisPLOY* achieves resource efficiency while preserving end-to-end performance.

Acknowledgments

This work is partially supported by National Natural Science Foundation of China (NSFC) under Grant 62172189; the Innovate UK grants 10098627 and 10106629.

References

[1] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, et al., HPCC: High Precision Congestion Control, in: *Proceedings of the ACM special interest group on data communication*, 2019.

[2] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi, et al., Flow event telemetry on programmable data plane, in: *Proceedings of the ACM special interest group on data communication*, 2020.

[3] S. Sheng, Q. Huang, P. P. Lee, DeltaINT: Toward general in-band network telemetry with extremely low bandwidth overhead, in: *In 2021 IEEE 29th International Conference on Network Protocols*, 2021.

[4] Barefoot network. barefoot tofino, https://www.barefootnetworks.com/technology/#tofino, Accessed on: February 17, 2025.

[5] X. Jia, F. Li, S. Chen, C. Gao, P. Wang, X. Wang, RED: Distributed Program Deployment for Resource-aware Programmable Switches, in: *IEEE Conference on Computer Communications*, 2023.

[6] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang, et al., LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets, in: *USENIX symposium on networked systems design and implementation*, 2021.

[7] H. Zheng, C. Tian, T. Yang, H. Lin, C. Liu, Z. Zhang, W. Dou, G. Chen, FlyMon: enabling on-the-fly task reconfiguration for network measurement, in: *Proceedings of the ACM special interest group on data communication*, 2022.

[8] X. Chen, H. Liu, Q. Huang, P. Wang, D. Zhang, H. Zhou, C. Wu, SPEED: Resource-efficient and high-performance deployment for data plane programs, in: *IEEE International Conference on Network Protocols*, 2021.

[9] X. Chen, H. Liu, Q. Xiao, K. Guo, T. Sun, X. Ling, X. Liu, Q. Huang, D. Zhang, H. Zhou, et al., Toward Low-Overhead Inter-Switch Coordination in Network-wide Data Plane Program Deployment, in: *IEEE International Conference on Distributed Computing Systems*, 2022.

[10] W. Xu, Z. Zhang, Y. Feng, H. Song, Z. Chen, W. Wu, G. Liu, Y. Zhang, S. Liu, Z. Tian, et al., ClickINC: In-network computing as a service in heterogeneous programmable data-center networks, in: *Proceedings of the ACM special interest group on data communication*, 2023.

[11] M. Moshref, M. Yu, R. Govindan, Software defined measurement for data centers, in: *USENIX symposium on networked systems design and implementation*, 2013.

[12] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, D. G. Andersen, cSamp: A system for network-wide flow monitoring, in: *USENIX symposium on networked systems design and implementation*, 2008.

[13] Y. Xu, Y. Liu, DDoS attack detection under SDN context, in: *IEEE Conference on Computer Communications*, 2016.

[14] V. Sekar, Effective network management via system-wide coordination and optimization, Carnegie Mellon University, 2010.

[15] M. Qian, L. Cui, F. P. Tso, Y. Deng, W. Jia, OffsetINT: Achieving high accuracy and low bandwidth for in-band network telemetry, in: *IEEE Transactions on Services Computing*, 2023.

[16] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, et al., Packet-level telemetry in large datacenter networks, in: *Proceedings of the ACM special interest group on data communication*, 2015.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., P4: Programming protocol-independent packet processors, in: *Proceedings of the ACM special interest group on data communication*, 2014.

[18] P. Zheng, T. Benson, C. Hu, P4visor: Lightweight virtualization and composition primitives for building and testing modular programs, in: *Proceedings of the International Conference on Emerging Networking EXperiments and Technologies*, 2018.

[19] D. Hancock, J. Van der Merwe, Hyper4: Using p4 to virtualize the programmable data plane, in: *Proceedings of the International Conference on Emerging Networking EXperiments and Technologies*, 2016.

[20] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, J. Wu, Hyperv: A high performance hypervisor for virtualization of the programmable data plane, in: *International Conference on Computer Communication and Networks*, 2017.

[21] H. Namkung, Z. Liu, D. Kim, V. Sekar, P. Steenkiste, Sketchovsky: Enabling Ensembles of Sketches on Programmable Switches, in: *USENIX symposium on networked systems design and implementation*, 2023.

[22] X. Chen, Q. Huang, P. Wang, H. Liu, Y. Chen, D. Zhang, H. Zhou, C. Wu, Mtp: Avoiding control plane overload with measurement task placement, in: *IEEE Conference on Computer Communications*, 2021.

[23] X. Chen, Q. Xiao, H. Liu, Q. Huang, D. Zhang, X. Liu, L. Hu, H. Zhou, C. Wu, K. Ren, Eagle: Toward scalable and near-optimal network-wide sketch deployment in network measurement, in: *USENIX symposium on networked systems design and implementation*, 2024.

[24] A. Agarwal, Z. Liu, S. Seshan, HeteroSketch: Coordinating network-wide monitoring in heterogeneous and dynamic networks, in: *USENIX symposium on networked systems design and implementation*, 2022.

[25] P4 language consortium. p4runtime: A control plane framework and tools for the p4 programming language, https://github.com/p4lang/p4c, Accessed on: February 17, 2025.

[26] Q. Huang, P. P. Lee, Y. Bao, Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference, in: *Proceedings of the ACM special interest group on data communication*, 2018.

[27] J. Sonchack, A. J. Aviv, E. Keller, J. M. Smith, Turboflow: Information rich flow record generation on commodity switches, in: *Proceedings of the EuroSys Conference*, 2018.

[28] M. Durand, P. Flajolet, Loglog counting of large cardinalities, in: *European Symposium on Algorithms*, 2003.

[29] Y. Li, R. Miao, C. Kim, M. Yu, FlowRadar: A better NetFlow for data centers, in: *USENIX symposium on networked systems design and implementation*, 2016.

[30] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Elastic Sketch: Adaptive and fast network-wide measurements, in: *Proceedings of the ACM special interest group on data communication*, 2018.

[31] L. Tang, Q. Huang, P. P. Lee, SpreadSketch: Toward invertible and network-wide detection of superspreaders, in: *IEEE Conference on Computer Communications*, 2020.

[32] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, in: *Journal of Algorithms*, 2005.

[33] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: *Theoretical Computer Science*, 2004.

[34] P. Flajolet, É. Fusy, O. Gandouet, F. Meunier, Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm, in: *Discrete mathematics & theoretical computer science*, 2007.

[35] P. Flajolet, G. N. Martin, Probabilistic counting algorithms for data base applications, in: *Journal of computer and system sciences*, 1985.

[36] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman, One sketch to rule them all: Rethinking network flow monitoring with univmon, in: *Proceedings of the ACM special interest group on data communication*, 2016.

[37] Gurobi, https://www.gurobi.com/, Accessed on: February 17, 2025.

[38] T. Benson, A. Akella, D. A. Maltz, Network traffic characteristics of data centers in the wild, in: *Proceedings of the ACM special interest group on data communication*, 2010.

[39] AboveNet, http://www.topology-zoo.org/maps/Abvt.jpg, Accessed on: February 17, 2025.

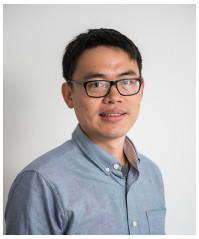
[40] AT&T, http://www.topology-zoo.org/maps/AttMpls.jpg, Accessed on: February 17, 2025.

[41] Internet2, https://internet2.edu/network/operations-and-support, Accessed on: February 17, 2025.

Mimi Qian is currently working toward the Ph.D. degree in Jinan University. Her current research interests include stateful data plane/programmable data plane, software-defined networking, and machine learning in computer networks.

Lin Cui is currently a professor in the Department of Computer Science at Jinan University, Guangzhou, China. He received the Ph.D. degree from City University of Hong Kong in 2013. He has broad interests in networking systems, with focuses on software defined networking (SDN), programmable networking, NFV, cloud networking, distributed systems and so on.

Xiaoquan Zhang is currently working toward the Ph.D. degree in Jinan University. His current research interests include programmable data planes, software-defined networking, and machine learning in computer networks.



Fung Po Tso received his B.Eng., M.Phil. and Ph.D. degrees from City University of Hong Kong in 2006, 2007 and 2011 respectively. He is currently a senior lecturer in the Department of Computer Science at the Loughborough University. His research interests include: network policy management, network measurement and optimisation, service chaining, data centre networking, software defined networking (SDN), and edge computing.

Yuhui Deng received the Ph.D. degree from the Huazhong University of Science and Technology in 2004. He is currently a Professor with the Computer Science Department, Jinan University. He worked with the EMC Corporation as a Senior Research Scientist from 2008 to 2009. He worked as a Research Officer with Cranfield University, UK, from 2005 to 2008. His research interests cover green computing, cloud computing, information storage, computer architecture, and performance evaluation.

Zhetao Li is a professor and dean of the College of Information Science and Technology, Jinan University, Guangzhou. He received the B.Eng. degree from Xiangtan University in 2002, the M.Eng. degree from Beihang University in 2005, and the Ph.D. degree in computer application technology from Hunan University in 2010. He was a visiting researcher with Ajou University from May to August 2012. His research interests include cloud computing, artificial intelligence, and multimedia signal processing.

Weijia Jia is currently a Chair Professor and Director of BNU-UIC Institute of Artificial Intelligence and Future Networks, VP for Research of BNU-HKBU United International College. His research interests include smart cities, IoT, knowledge graph, multicast and anycast QoS routing protocols, wireless sensor networks, and distributed systems. He has over 500 publications in prestigious international journals/conferences and research books/chapters.