FlyMon: Enabling On-the-Fly Task Reconfiguration for Network Measurement

Hao Zheng[★], Chen Tian[★*], Tong Yang[△], Huiping Lin[△], Chang Liu[★]
Zhaochen Zhang[★], Wanchun Dou[★], Guihai Chen[★]
[★]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[△]*School of Computer Science, Peking University, China*

ABSTRACT

Network measurement is important to data center operators. Most existing efforts focus on developing new implementation schemes for measurement tasks. Little attention is paid to on-the-fly task reconfiguration. Due to resource constraints, it is impossible to configure all needed tasks at start-up and dynamically turn on/off them. To support real-time reconfiguration of many different tasks, a key observation is that it is unnecessary to bind a task and its implementation at the compilation phase. We design FlyMon, the first sketch-based measurement system that can make on-the-fly reconfigurations on a large set of measurement tasks. FlyMon introduces the concept of Composable Measurement Units (CMUs), which are general operation units that support reconfigurable implementation for measurement tasks combined from different flow keys and flow attributes. FlyMon maps the design of CMUs to programmable switches' data planes so that the number of compacted CMUs can be maximized. FlyMon also provides dynamic memory management. We prototype FlyMon on Tofino and currently enable four frequently used flow attributes. Each CMU Group (with 3 CMUs) can concurrently perform up to 96 isolated measurement tasks with less than 8.3% hardware resources. The tasks can be deployed with configurable memory size at the millisecond level. By cross-stacking, FlyMon can deploy up to 27 CMUs in one pipeline of Tofino.

CCS CONCEPTS

• **Networks** → **Network monitoring**; *Programmable networks.*

KEYWORDS

Network Monitoring, Sketching, Programmable Switch

ACM Reference Format:
Hao Zheng, Chen Tian, Tong Yang, Huiping Lin, Chang Liu, Zhaochen Zhang, Wanchun Dou, Guihai Chen. 2022. FlyMon: Enabling On-the-Fly Task Reconfiguration for Network Measurement. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3544216.3544239

*Chen Tian is the corresponding author.

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544239

1 INTRODUCTION

Network measurement is important to data center operators. Many management tasks (*e.g.*, troubleshooting [8]) and scheduling decisions (*e.g.*, traffic engineering [3]) rely on monitoring real-time traffic events/statistics sensed by underlying measurements. To monitor a specific event or collect a particular statistic, an operator issues a measurement task, defined as a combination of a given flow **key** (*e.g.*, SrcIP, SrcIP/24, IP-pair, 5-tuple) and a targeted **attribute** (*e.g.*, frequency [61], distinct [54]). The measurement system then feeds qualified flow packets as input to an arithmetical operation unit, which outputs values for the attribute of the task.

Most existing efforts focus on developing new implementation schemes for measurement tasks [34, 54, 60]. Such a scheme combines dedicated data structures to store attribute values and the corresponding arithmetical operations over the data structures [68]. Nowadays, sketches are the most popular measurement schemes as they can achieve a good balance between accuracy and resource usage [63]. Running on the data plane, a sketch summarizes traffic statistics of observed packets into a form of lossy compression. Recent works have made significant progress on result accuracy [25, 52, 69], resource efficiency [24, 33], and functionality [48, 64] of various measurement tasks.

Little attention is paid to **on-the-fly task reconfiguration**. For a practical data center, an ideal measurement system should flexibly switch among different measurement tasks without interrupting network traffic [51]. There are tens of common measurement tasks, which could have different keys and attributes [68]. Currently, a switch can usually support only a few tasks (*e.g.*, 3 or 4) simultaneously [34, 60, 65]. Suppose a tenant complains that the performance of their network service is abnormally degraded. The operator needs to switch among various measurement tasks such as flow cardinality [49], DDoS detection [58], and flow congestion detection [55] of all related devices to locate the root cause of the problem gradually. Suppose the problem is eventually pinpointed as their flows congested in a particular switch. The operator then measures the heavy hitters in this switch to detect if any elephant flows dominate this device and then evenly schedules these flows to eliminate the congestion. Such task-switching requirements are common in practical data centers, and thus it is highly desired to design a measurement system that can achieve on-the-fly task reconfiguration.

Due to resource constraints, it is impossible to configure all needed tasks at start-up and turn on/off them on the fly. Currently, the task implementations on programmable switches are hardwired at the compilation phase. Suppose an operator needs to reconfigure tasks with m different flow keys and n different attributes. In the worst case, the hardware resource usage can be as large as $O(m \cdot n)$. Given the limited hardware resources at programmable switches,

it is impractical to pre-allocate exclusive computing and memory resources for every task [12]. It is reported that a Tofino [27] switch cannot support more than four single-key sketches in a typical scenario [65].

To support on-the-fly reconfiguration of a large number of different tasks, a critical observation is that it is unnecessary to bind a task and its implementation at the compilation phase. Instead, we can *dynamically compose* the implementation when a new task is assigned. The basic idea is: *(i)* decomposes task execution into a key-selection phase and an attribute-operation phase, *(ii)* separately minimizes each phase's resource consumption for supporting different tasks, and *(iii)* at runtime reconfigures two phases for a given task and links them together. Thus, different tasks could share key-selection/attribute-operation resources. The resource occupancy could be reduced from $O(m \cdot n)$ to near-constant.

We design FlyMon, the first sketch-based measurement system that can make on-the-fly reconfigurations on a large set of measurement tasks. Unlike existing efforts [2, 43, 50, 65] that develop novel measurement algorithms with programmable hardware, FlyMon explores how to efficiently extend the compilation-phase programmability to runtime to cope with the diverse measurement tasks. Without reloading the data-plane program, FlyMon can switch the data-plane measurement tasks by installing runtime rules using southbound APIs (*e.g.*, P4 Runtime [45]). Our contributions are listed below.

Firstly, FlyMon introduces the concept of **Composable Measurement Units (CMUs)**. By adding a front-placed match-action table, a key can be dynamically selected and copied from packet header fields according to the matched task. FlyMon further adopts a less-copy strategy that only copies a compressed key to reduce the copy burden. We propose a reduced operation set, which can support most attributes with only a few stateful operations. Thus, both the key-selection and attribute-operation phases are reconfigurable at runtime, and their resource occupancy is minimized. Putting them together, we define the CMUs as general operation units that support reconfigurable implementation for multiple concurrent measurement tasks combined from different flow keys and flow attributes.

Secondly, FlyMon maps the CMU design to programmable switches' data plane so that the number of compacted CMUs can be maximized. Inspired by the Instruction Pipeline [15] in the CPU, FlyMon implements CMUs in a grouped fashion called a CMU Group and expands the CMU-Group pipeline into four independent stages. Each stage has a different dominant resource demand. FlyMon can arrange multiple CMU Groups in a cross-stacking view to significantly improve hardware resources utilization.

Thirdly, FlyMon provides dynamic memory management. The configuration of the stateful memory (*i.e.*, size and bit-width) cannot be changed at runtime, which poses a significant challenge to allocate resources flexibly for different tasks and adapt to the changing network conditions [16]. To realize dynamic memory management on the fixed stateful memory, FlyMon introduces an address translation mechanism to convert the dynamic allocation of measurement tasks to the dynamic mapping of the address range. We propose two strategies for implementing the address translation mechanism in programmable switches and analyzing their respective resource overheads.

Table 1: Abstraction of measurement tasks. The 'FlowID' is an abstract identifier that can be any combination or part of protocol fields.

Task	Key	Attr.	Param.	Sketch
DDoS Victim [58]	DstIP	Distinct	SrcIP	HLL [20] BeauCoup [12] UnivMon [34] LC [57]
Worm [29]	SrcIP		DstIP	
Port Scan [21]	IPpair		DstPort	
Cardinality [49]	N/A		FlowID	
Per-flow Size [60]	FlowID	Frequency	Const(1) or Pkt Bytes	CMS [14] UnivMon MRAC [30] Counter Braids [36]
Heavy Hitter [37]				
Heavy Changer [53]				
Black List [38]	N/A	Existence	FlowID	B.F [6]
Congestion [55]	FlowID	Max	Queue Length	SuMax [66]
HOL [47]	FlowID		Queue Delay	
Maximum Interval [66]	FlowID		Packet Interval	

We prototype FlyMon on Tofino [27] and currently enable four frequently used attributes (see Table 1). Each CMU Group can concurrently perform up to 96 isolated measurement tasks composed of different keys and attributes with less than 8.3% hardware resources. By cross-stacking, FlyMon can deploy up to 9 CMU Groups (including 27 CMUs) in one pipeline of Tofino. Besides, network operators can dynamically deploy the measurement tasks with configurable memory space at the millisecond level. Our reference implementation of FlyMon is available at [67].

This work does not raise any ethical issues.

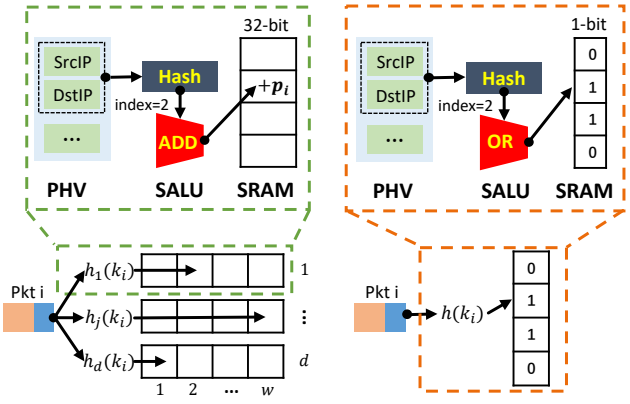
2 BACKGROUND

We start by providing some background on measurement tasks and associated sketching algorithms (§2.1). We then present the conventional way to implement sketches on reconfigurable match-action tables (RMT [7]) and discuss why it is not scalable for covering all measurement tasks (§2.2).

2.1 Measurement Tasks and Sketches

At a high level, a measurement task can be abstracted as a flow key and a flow attribute. The flow key can be a specific protocol field (*e.g.*, SrcIP), its subset (*e.g.*, SrcIP/16, SrcIP/24), or combinations of several fields (e.g., SrcIP-DstIP, SrcIP-SrcPort, 5-tuple). The flow attribute can be frequency, distinct, etc. According to the flow attribute, the measurement system uses specific algorithms to perform a single pass over the packets and group the statistics according to the flow key. As shown in Table 1, the different combinations of flow keys and the attributes with associated parameters cause the diversity of measurement tasks.

Inspired by BeauCoup [12], we define a measurement task as two parts: a key, and an attribute with associated parameters. The key indicates how to group the packets into multiple flows (*e.g.*,



(a) Count-Min Sketch (CMS) (b) Inserting Bloom Filter (k=1)

Figure 1: Sketch Implementation on RMT switches

by SrcIP, SrcIP/24, 5-tuple). The attribute with specific parameters indicates what kind of flow statistics to be measured among each flow's packets. We focus on four frequently used flow attributes in Table 1:

- **Distinct(param)** counts different numbers of the parameter for each key.
- **Frequency(param)** accumulates the parameter for each key.
- **Existence(param)** checks if the parameter exists in a given set for each key.
- **Max(param)** finds the maximum parameter for each key.

BeauCoup focuses on the Distinct attribute in this definition. For example, DDoS victim detection needs to count the different numbers of SrcIP for each DstIP. The task can be abstracted into taking DstIP as the key and Distinct(SrcIP) as the attribute. In the data plane, we maintain a distinct counter for each DstIP. When a packet arrives, DstIP (*i.e.*, the key) is used to locate a distinct counter, and we add 1 to the counter if the packet's SrcIP (*i.e.*, the parameter) is a new value not seen before.

Sketching algorithms summarize traffic statistics of all observed packets into a form of lossy compression on the original traffic data. Although the sketches lose some accuracy, they make a great trade-off between accuracy and resource usage [63]. As shown in Table 1, we can use existing sketching algorithms [68] to efficiently perform the above attributes. For example, Count-Min Sketch (CMS) [14] is suitable for performing the Frequency attribute. As shown in Figure 1a, it maintains $d \times w$ array of counters with d hash functions $h_1, h_2, \cdots, h_d$. For each key k_i with parameter p_i, CMS adds p_i to the counters $C[j, h_j(k_i)]$ for all rows j. The estimation result is the minimum count among $C[j, h_j(i)]$.

2.2 Sketch Implementation on RMT Hardware

Programmable switches (*e.g.*, Tofino [26]) based on the RMT paradigm support implementing various sketching algorithms [43]. The switches consist of several Match-Action Unit (MAU) stages arranged in a pipeline. Each MAU stage contains the same hardware resources for implementing network functions (*e.g.*, L3 forwarding). The key hardware resources used to implement sketches are

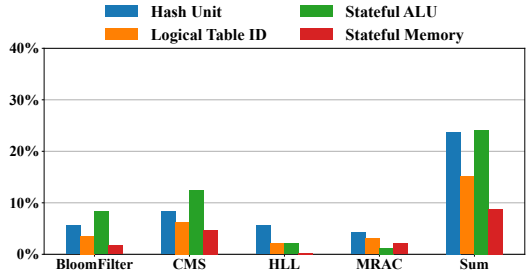


Figure 2: The resource footprint of four single-key sketching algorithms and their coexistence (denoted as Sum).

Packet Header Vector (PHV), hash units [1], stateful ALUs (SALU), and stateful memory (*i.e.*, static random access memory (SRAM) used to reserve states between packets).

Figure 1 shows how to implement Count-Min Sketch (CMS) and Bloom Filter (B.F [6]) in the RMT-based switches. Specifically, we need to allocate a set of buckets in SRAM for the algorithms to store frequency attributes (CMS) or existence attributes (B.F), respectively. The buckets for the CMS are configured to 32 bits, while the buckets in the B.F are configured to 1 bit. We can input a flow key (SrcIP-DstIP pair) stored in PHV into the hash unit to get the address of a corresponding bucket in SRAM. Then, the SALU performs a particular stateful operation to update the bucket according to the address. In particular, the CMS needs to perform the *ADD* operation to update the counter. The B.F needs to perform the *OR* operation for inserting an item.

Limitation of RMT Hardware. With the RMT hardware, users can customize the header fields input into the hash units, the stateful operations on the SALUs, and the bit width and length of the buckets in SRAM. However, once the switch is ready to forward packets, it requires interrupting the traffic to reconfigure the functionality of the hardware. It is not scalable to support on-the-fly reconfiguration of measurement tasks by deploying them in advance with exclusive hardware resources. We evaluate the critical resource footprint of four sketches and their coexistence (denoted as Sum) in Figure 2. We find that the solution can not support more than four different keys.

3 FLYMON DESIGN

Overview. FlyMon is a measurement system that realizes on-the-fly reconfiguration of measurement tasks combined from various keys and attributes. As shown in Figure 3, when a new measurement task is issued, FlyMon only needs to install runtime rules (*e.g.*, P4 Runtime interfaces [45]) from the control plane according to the task's key and attribute. Then the measurement task can be deployed on the hardware data plane with specified memory space. The design of FlyMon consists of four parts. We first propose the Composable Measurement Unit (CMU) to accommodate more measurement tasks (§3.1). We then discuss how to map CMUs on RMT Hardware (§3.2) efficiently. In §3.3, we introduce how to realize dynamic memory management based on CMUs. Finally, we introduce the control plane of FlyMon in §3.4.

[1]There are multiple types of hash resources (*e.g.*, hash bits, hash calculation units, hash distribution units) in the data plane. For the convenience of discussion, we refer to them collectively as hash units and consider the bottleneck resources (usually, the hash distribution units) as the main optimization target.

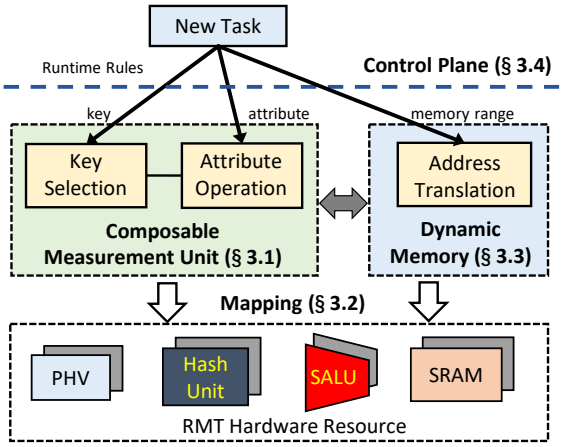


Figure 3: Overview of FlyMon.

3.1 Composable Measurement Unit

To support the on-the-fly reconfiguration of a large number of tasks, we decompose the task execution into a key-selection phase and an attribute-operation phase. The composition of the two phases constitutes the concept of a Composable Measurement Unit, a general operation unit to accommodate more kinds of measurement tasks.

3.1.1 The Key-selection Phase

The static task deployment cannot dynamically set the key for different tasks' packets because the hash units in RMT hardware cannot change the input PHV fields to other fields at run time. Fortunately, we have the opportunity to modify the value of the input PHV fields in advance [4, 70]. As shown in the top half of Figure 4, we can allocate an extra PHV field named 'Dynamic Key,' which is used as the fixed input of the hash unit. When a packet arrives, we can use a front-placed match-action table (denoted as 'Select Key') to dynamically select a key from the candidate key set according to the task to be performed. The hash unit then calculates the 'Dynamic Key' to get a memory address and passes this memory address to the attribute-operation phase.

Challenge: PHV Copy. The dynamic selection of the key needs to allocate extra PHV memory and copy the original header fields to this memory. PHV resources are statically allocated in the data plane and cannot be changed at runtime. This means that if we want to support 5-tuple (*i.e.*, SrcIP, DstIP, SrcPort, DstPort, and Protocol) as a key, we must statically allocate extra 104 bits in PHV for each SALU, even though a measurement task may only need to set SrcIP as the key. However, the PHV memory is a precious resource in RMT switches. Most data-plane behaviors in RMT switches are closely related to the PHV [7]. As the number of deployed SALUs increases, there will be a significant overhead on PHV occupancy. The situation worsens if we additionally consider other protocols' fields (*e.g.*, IPv6 addresses) as candidate keys.

Optimization: Less-copy Strategy. To enable the per-packet dynamical key selection according to the task type, the PHV Copy operation is unavoidable. We try to reduce the PHV occupancy by adopting a 'less-copy' strategy: setting a compressed key for each SALU. As shown at the bottom of Figure 4, we use several hash units to produce a set of compressed keys (*e.g.*, $C(k_1), C(k_2)$) in one

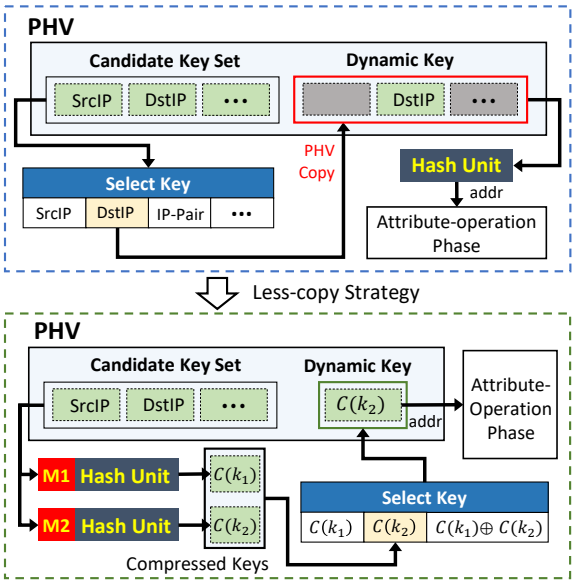


Figure 4: The key-selection phase and its optimization.

MAU stage. Although the one-way compression [2] will lead to hash collisions among different flows, it has little effect on the accuracy of network measurements.

Firstly, there is a small percentage of collisions in a compressed key since the number of flows is limited within a measurement epoch. Existing work [1] reports that there are up to 8K concurrent flows (identified by 5-tuples) over 1 ms intervals in a heavily loaded leaf switch. Based on this, there are around 400K distinct 5-tuples during a 50 ms measurement epoch. Theoretically, the mapping from n distinct flows to a log m-bit compression domain will result in a hash collision probability approximating $1 - e^{-n/m}$ for each flow (see more details in Appendix B). Given a 24-bit compressed key, the percentage of collision flows among all flows is about 2.35% in the above scenario. Secondly, a compressed key or its subparts can be directly used for memory updating of sketching algorithms. Therefore, the accuracy is still directly related to the allocated memory space of the algorithms. The sketching algorithms (*e.g.*, CMS [14], Bloom Filter [38]) are insensitive to hash collisions because they usually use multiple hash functions to reduce the impact of collisions [10]. Thirdly, in some extreme scenarios, FlyMon supports separating a heavy task into multiple subtasks by task filters (*e.g.*, separate a task with $filter[SrcIP : 10.0.0.0/8]$ to subtask 1 with $filter[SrcIP : 10.0.0.0/9]$ and subtask 2 with $filter[SrcIP : 10.128.0.0/9]$), which can reduce the probability of collisions in each subtask while taking more hardware resources.

However, few keys can be compressed since the number of hash units in the data plane is limited. We address this limitation by using the latest feature of RMT switches named dynamic hashing [3], which supports changing the hash units' parameters to mask portions of fields at runtime. We can set the input of the hash unit as the whole

[2]The one-way compression, also known as a message digest, fingerprint, is not related to conventional data compression, which instead can be inverted exactly (lossless compression) or approximately (lossy compression) to the original data [17].
[3]The corresponding control plane functions and example codes (*i.e.*, tna_dyn_hashing) appear in SDE version 9.7.0

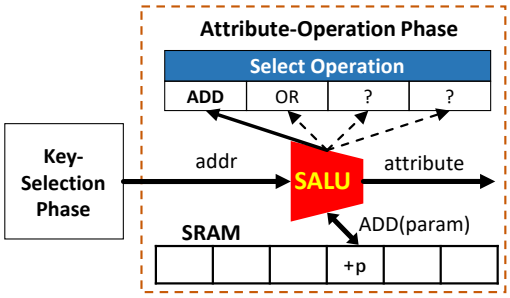


Figure 5: The attribute-operation phase. The 'OR' denotes a bit-wise OR operation.

candidate key set (*i.e.*, any partial key of 5-tuple). Then we can install hash-masking rules (denoted as 'M1' and 'M2') from the control plane to configure which keys to be compressed by the hash units. In addition, we support performing bit-wise exclusive or (XOR) operations between the compressed keys to increase the number of available keys. For example, if we generate two compressed keys $C(SrcIP)$ and $C(DstIP)$ with two hash units, we can set IP-pair as a key by performing $C(SrcIP) \oplus C(DstIP)$. Currently, RMT switches can only perform binary XOR in one match-action stage. Therefore, we can select at most $\frac{k(k+1)}{2}$ different keys with k hash units. In §3.2, We will discuss how to save hashing resources by sharing hash units among multiple CMUs.

3.1.2 *The Attribute-operation Phase*

FlyMon supports attribute measurement by implementing several built-in algorithms. At a high level, measurement algorithms can be abstracted into one or several building blocks [63]. A SALU and corresponding SRAM can implement one of the building blocks. In RMT-based programmable switches, a SALU is bound to a fixed-size memory, collectively called a register. To realize the dynamic selection of attributes, FlyMon uses the feature of RMT switches that each register can have several pre-loaded operations (*i.e.*, register actions). As shown in Figure 5, we can dynamically choose different operations with a 'Select Operation' table to implement the register as the building block of various algorithms. For example, suppose we choose the 'ADD' operation. In that case, we regard the register as a building block of CMS, which can be used to measure the Frequency attribute.

Challenge: Limited Stateful Operations. Although SALUs can dynamically select a specific operation to execute, the number of stateful operations that can be pre-loaded is limited. With the abstraction of measurement tasks in Table 1, only one algorithm is required for each attribute. However, to deal with different scenarios flexibly, it's better to support multiple algorithms for each attribute. In particular, BeauCoup [12] is more resource-efficient than HyperLogLog [20] for multi-key distinct counting (*e.g.*, detecting DDoS victims). For the Frequency attribute, MRAC [30] is dedicated to estimating flow size distribution and flow entropy. Under the same memory size, TowerSketch [59], SuMax [66], and Counter Braids [36] can replace CMS to obtain higher accuracy at the cost of more other resources (*e.g.*, MAU stages). Currently, each SALU in Tofino can only pre-load four different operations [26]. Therefore, implementing various algorithms for an extensible set of flow attributes is a potential challenge.

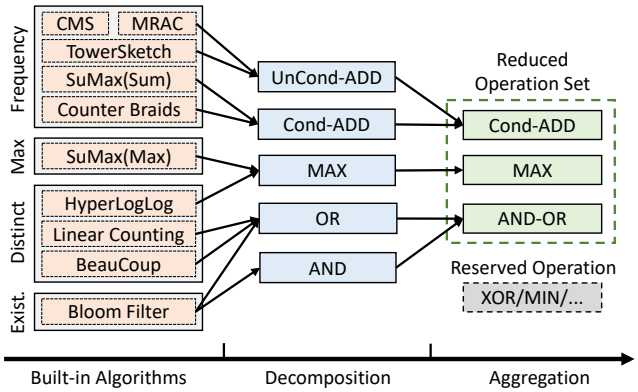


Figure 6: Generation process of the reduced operation set.

Optimization: Reduced Operation Set. As shown in Figure 6, we use a reduced operation set to implement various sketching algorithms. The generation process of the reduced operation set includes two steps:

- **Decomposition.** We decompose the measurement algorithms into data-plane operations and control-plane analysis. Only the data-plane operations need to be implemented with the stateful operations. We find some algorithms share some joint data-plane operations, which means we can let multiple measurement algorithms multiplex some stateful operations. For example, counter-based algorithms such as CMS [14] and MRAC [31] can share the unconditional ADD (UnCond-ADD) operation.
- **Aggregation**. The computing power of SALU is usually larger than the operations required by some measurement algorithms. In particular, SALUs can make a conditional judgment, but some algorithms (*i.e.*, Bloom Filter) do not utilize this resource. We can aggregate two simple operations into one stateful operation. For example, the bit-wise AND and bit-wise OR can be aggregated into a bit-wise AND-OR operation, and the SALU can decide which to run by the conditional judgment.

In the current implementation, we implement 10 algorithms (in Figure 6) with the reduced operation set, covering all the flow attributes listed in Table 1. The details of these operations can be found in Appendix A. In addition, we introduce a preparation stage to assist in implementing these algorithms (see §3.2). We discuss the detailed implementation of these algorithms in §4. We only occupy three stateful operations, which means FlyMon has expansion room to support more attributes or algorithms (see §6).

3.2 Mapping CMUs on RMT Hardware

Compared to the static deployment method, a significant problem introduced by CMU is that it always needs multiple MAU stages to perform a measurement task. However, the number of MAU stages in the data plane is limited (*i.e.*, only 12 in Tofino [27]). A straightforward implementation could easily lead to the underutilization of various hardware resources (*e.g.*, Hash, SALU). In this subsection, we first introduce how CMUs are mapped in RMT switches. We then discuss why such a design can efficiently utilize various hardware resources in the MAU stages.

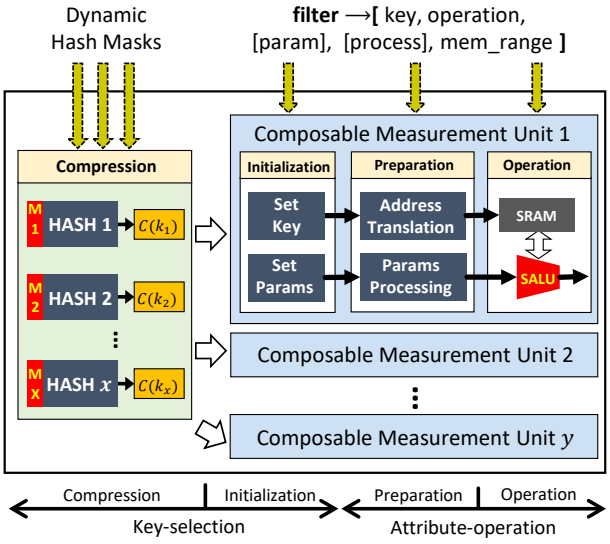


Figure 7: Composable Measurement Unit Group.

CMU Groups. We implement multiple CMUs in a grouped fashion, called a CMU Group. As shown in Figure 7, a CMU Group consists of four stages: a compression stage, an initialization stage, a preparation stage, and an operation stage. The functions of the four stages are listed in Table 2. Each CMU completes the key-selection phase through the compression and initialization stages, while the attribute-operation phase is completed through the preparation and operation stages. To make full use of hash resources while ensuring the flexibility of the key-selection phase. We let multiple CMUs in the same Group share a compression stage. Therefore, each CMU can select $\frac{k(k+1)}{2}$ different keys with k hash units (as discussed in §3.1)). Usually, CMUs need to perform independent hash calculations in some sketches to reduce the impact of hash collisions (*e.g.*, Bloom Filter, CMS). Inspired by SketchLib [43], we let CMUs in the same CMU Group select different sub-parts of the compressed key to simulate the independent hash calculations. For example, given a 32-bit compressed key, we select 0-15 bits for CMU 1, 8-24 bits for CMU 2, and 16-32 bits for CMU 3. Our experimental results demonstrate that the strategy has a negligible impact on measurement accuracy (see §5).

CMUs also set one or two parameters in their initialization stage. The parameters are used to distinguish the concrete measurement tasks for an attribute (see Table 1). Take the Frequency attribute as an example. If we want to count the number of packets for each flow, we need to set the parameter to 1. If we want to count per-flow bytes, we must set the parameter to the size of the packet. The parameters can be constant values or standard metadata such as packet size, timestamp, queue length, and delay. Besides, CMUs can also set parameters as the compressed keys to support the Distinct and Existence attributes (see §4).

The preparation stage processes the key and parameters set in the initialization stage separately. The processing of the key (*i.e.*, Address Translation in Figure 7) is for realizing dynamic memory management (see §3.3). The processing of the parameters is to enable CMUs to implement more sketching algorithms flexibly. With

Table 2: Functions of the four stages in CMU Group and their key resource occupancy.

Stage	Function	Key Resource
Compression (C)	Generates multiple compressed keys according to hash masks installed from the control plane	Hash Unit
Initialization (I)	Selects the key and parameters according to the matched task	VLIW Instruction
Preparation (P)	Performs address translation and parameter preprocessing	TCAM
Operation (O)	Selects a stateful operation to update the flow attributes based on the key and parameters	SALU, SRAM, Hash Unit [4]

a TCAM-based table, a CMU can dynamically establish a mapping function between the input and output parameters, bringing great flexibility in memory updating. For example, some algorithms like Bloom Filter [6] and BeauCoup [12] operate on a bit-level rather than a whole counter. However, CMUs need to use a uniform memory configuration (*e.g.*, all SRAM buckets are 16-bit) for generality. With the preparation stage, we can map a hashing parameter to a one-hot encoding (*i.e.*, only one bit is '1' and the other bits are '0') used in these algorithms. We discuss the detailed implementation of these algorithms in FlyMon with corresponding measurement tasks in §4. After the preparation stage is completed, the operation stage takes the key and parameters as inputs and perform specific stateful operations for a given task.

Cross-stack CMU Groups in RMT Pipeline. As shown in Table 2, we find that the four stages in the CMU Group have different dominant resource demands. Specifically, the compression stage occupies most hash units to generate compressed keys. The initialization stage requires more very long instruction words (VLIWs [7]) to select different keys and parameters dynamically. The preparation stage occupies more ternary content-addressable memory (TCAM [62]) resources to translate addresses (see §3.3) and process parameters (*e.g.*, one-hot encoding). The operation phase occupies all SALU and SRAM resources. Therefore, deploying multiple CMU Groups one by one will cause uneven utilization of various resources on each MAU stage.

Our inspiration comes from the Instruction Pipeline [15] in the CPU. To fully use the various components, the CPU divides an instruction into several micro-instructions (*e.g.*, instruction fetch, memory access) and executes different micro-instructions of multiple instructions simultaneously in each clock cycle. In FlyMon, a match-action unit (MAU) stage is similar to a CPU clock. As shown in Figure 8, we can make multiple CMU Groups cross-stacked to maximize the number of deployed CMU Groups within given MAU stages. The cross stacking means each stage of a CMU Group occupies its dominant resource as much as possible, and the subsequent CMU Groups are shift-one-stage placed to use various resources in

[4]Note that SALUs of current RMT switches always use a hash distribution unit for addressing, even though we input a correctly-processed memory address. In the current implementation, we compromise to allocate half of the hash distribution units in the compression stage and half in the operation stage. We expect future RMT hardware to equip the SALUs with standalone memory access units.

S1	S2	S3	S4	S5	S6	S7	...
C_0	I_0	P_0	O_0	C_4	I_4	P_4	...
	C_1	I_1	P_1	O_1	C_5	I_5	...
		C_2	I_2	P_2	O_2	C_6	...
			C_3	I_3	P_3	O_3	...

Figure 8: Cross-stacking view of CMU Groups. The S_i denotes the i-th MAU stage. The C_j, I_j, P_j, and O_j denote the four stages of CMU Group j listed in Table 2.

each MAU stage fully. There are two direct benefits of the cross stacking. Firstly, for dedicated measurement equipment, maximizing the number of deployable CMUs will increase the number of measurement tasks executed in parallel. Secondly, the cross stacking costs fewer MAU stages and utilizes various resources evenly, which is the main optimization goal in some network systems based on RMT switches [56].

However, the cross-stacking will increase the occupation of PHV resources because multiple CMU Groups are overlapped to share a set of MAU stages. In FlyMon, such stacking is supported because we reduce the PHV occupancy of CMUs with compression stages. Since one CMU Group occupies 4 stages, a maximum of 9 CMU Groups can be deployed in a 12-stage pipeline. We discuss how to further utilize the remaining resources (*i.e.*, triangle areas in Figure 8) in §6 and Appendix E

3.3 Dynamic Memory Management

FlyMon provides two levels of memory management. Firstly, each CMU can switch to any supported measurement tasks. Thus, we can achieve unit-level memory management by adjusting the number of CMUs for the measurement tasks. However, since the number of CMUs is limited, it cannot achieve fine-grained memory management only by the unit-level adjustment.

We implement fine-grained memory management inside the CMUs through an address translation mechanism in the preparation stage. Our basic idea is that while we cannot change the size of memory allocations at runtime, we can modify the memory address range for measurement tasks. During the initialization stage, we select one key from the compressed keys. Suppose the CMU has a total of m buckets. The selected key is an address representing its range as $[0, m]$. The address translation mechanism is that we narrow this range down to a fixed sub-range, such as $[0, \frac{m}{4})$, $[\frac{m}{2}, m)$, according to the measurement task to be performed. We propose two different address translation methods based on RMT hardware.

Shift-based address translation. After the key is selected, we can right shift the key to modify the access range of the memory. By right shifting, we can get an offset address of different sub-ranges, such as $[0, \frac{m}{2}]$, $[0, \frac{m}{4}]$, $[0, \frac{m}{8}]$. Further, we use another stage to add a base address to this address to get the physical address of the task. As shown in Figure 9, if we want to access the address space of Task 2 (*i.e.*, $[\frac{m}{2}, \frac{3m}{4}]$), we need to shift two bits right and then add $\frac{m}{2}$ as the base address. This method requires two MAU stages to complete the address translation. We can optimize it within one MAU stage by pre-calculating all possible ranges of offsets in the initialization stage while costing additional PHV resources.

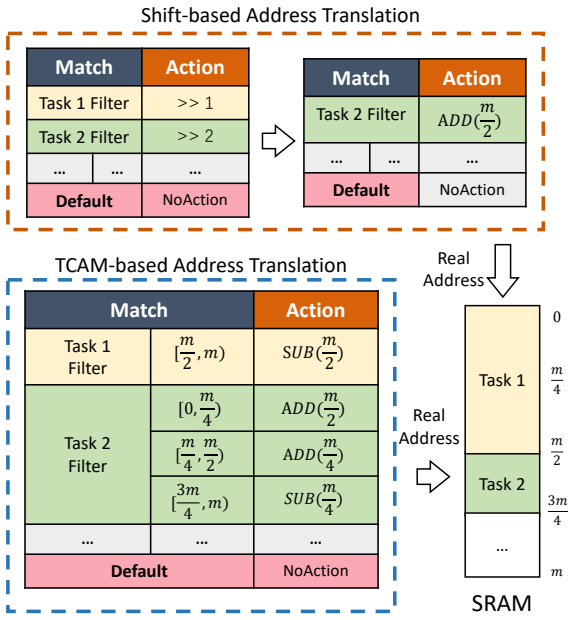


Figure 9: Two different Address Translation Mechanisms.

TCAM-based address translation. We can use the range matching function of TCAM to complete the address translation within a stage. To get the address range $[\frac{m}{2}, \frac{3m}{4}]$, We can use TCAM to judge different cases. As shown in Figure 9, if the current address range is already in $[\frac{m}{2}, \frac{3m}{4}]$, we do not need any processing. If the current address range is in $[0, \frac{m}{4}]$, we can map it to the target address range by adding $\frac{m}{2}$ to the original address. In other words, we need to add three TCAM entries and a default entry for this task in this case.

The above two methods use different types of resources. The shift-based method requires an additional stage or PHV occupation, while the TCAM-based method requires more TCAM resources. Network operators can implement the address translation based on currently available resources. We evaluate the resource overhead of these two methods in §5. We find that 32 memory partitions can be achieved using less than 15% TCAM resources within a single MAU stage. In other words, we can realize the dynamic adjustment of memory space size into $m, m/2, m/4, ..., m/32$. At the same time, up to 32 isolated measurement tasks can be performed on one CMU.

Limitation of Address Translation. There are two limitations to implementing dynamic memory allocation via the address translation. Firstly, only 2^n $(n \geq 0)$ partitions can be efficiently supported. The shift-based address translation cannot realize memory partitions that are not a power of two. And the TCAM-based address translation requires more TCAM entries if the ranges are not a power of two. Secondly, a SALU can only execute one measurement task per packet since it can only access the memory once in current RMT hardware. In other words, measurement tasks with traffic intersections (*e.g.*, a task with $filter[SrcIP = 10.0.0.0/24]$ and another with $filter[SrcIP = 10.0.0.0/16]$) cannot directly coexist in the same CMU. When deploying measurement tasks, we avoid traffic intersections on the same CMU. We evaluate a probabilistic execution approach to solve this limitation for heavy-hitter detection (§5.3) and make a discussion in §6.

3.4 Control Plane Implementation

Existing Software-defined Measurement (SDM) controllers (*e.g.*, DREAM [42], SCREAM [41]) already realize rich control-plane functions (*e.g.*, memory management strategies, network-wide measurements) based on software switches [23, 46] or simulations. FlyMon fills the gap of the flexible hardware measurement data plane for these SDM controllers. FlyMon provides two class interfaces for compatibility with existing SDM controllers: task management interfaces and resource management interfaces.

The task management interfaces are used to define new measurement tasks and modify running tasks' configurations. The task definition in FlyMon includes a filter, a key, an attribute, and a memory size (*i.e.*, number of buckets). A dedicated compiler selects a built-in algorithm according to the attribute and translates the task definition into runtime rules. The resource management interfaces maintain the occupancy status of various resources (*e.g.*, compressed keys, memory) and allocate or recycle these resources. Since FlyMon can only allocate the memory discretely (*i.e.*, power of 2), the control plane provides two modes of memory allocation: accurate and efficient. The accurate mode always allocates memory greater than or equal to needed, while the efficient mode always allocates memory closest to the required memory.

FlyMon's control plane takes a CMU Group as the basic unit to manage the functions and resource allocations of the data plane. A CMU Group can concurrently perform multiple different measurement tasks. In particular, the three tasks in Figure 10 execute (per-SrcIP) flow size estimation, DDoS victims, and congestion detection, respectively. If a new measurement task arrives, the control plane decomposes the task into the three parts (*i.e.*, key, attribute, memory size) and then installs the task to the CMU Group that meets the resource requirements (*e.g.*, compressed keys, memory). We adopt a greedy strategy: prioritize deploying new tasks to the CMU groups that already have (part of) the required compressed keys. For the CMU Group in Figure 10, if the new task requires SrcIP-SrcPort as the key, the control plane needs to configure the third compressed key (*i.e.*, H3) as SrcIP-SrcPort directly or configures it as SrcPort and then sets SrcIP-SrcPort by performing XOR between HASH 1 and HASH 3. If the task requires another $3 * 16384$ buckets in SRAM, the control plane will assign this task to other CMU Groups because there is not enough memory in this CMU Group.

4 FLYMON USAGE

In this section, we introduce how to use CMUs to implement parts of built-in algorithms through several classic measurement tasks. We use p_1 and p_2 to denote flow attributes' first and second parameters, respectively. Note that the tasks that FlyMon can perform are not limited to the below use cases. We discuss FlyMon's expressiveness in §6 and present the implementation of the rest algorithms (*e.g.*, Counter Braids [36]) in Appendix D.

Flow Cardinality [49] measures the number of distinct flows in the network, which is a single-key distinct counting task. We use HyperLogLog (HLL) to perform this task with a small memory footprint. Given a flow key (*e.g.*, 5-tuple), HLL uses a hash function to generate a random binary string, which can be represented as the pattern of $0^{\rho-1}1$, where the ρ denotes the position of the leftmost 1-bit in the binary string. By tracking the binary string with the

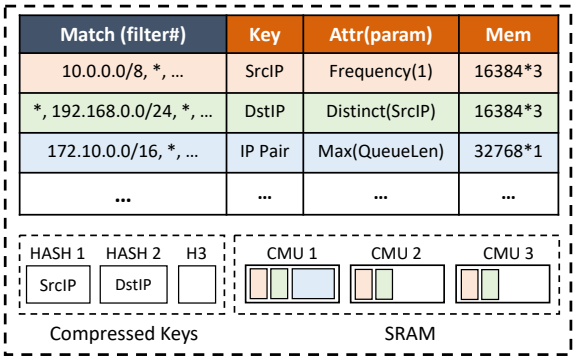


Figure 10: The abstraction of one CMU Group in the control plane. The 'Mem' in the figure denotes the number of buckets in registers.

largest ρ, the distinct count (*i.e.*, cardinality) of a multi-set can be estimated as at least 2^{ρ}. FlyMon simulates this process through the MAX operation. We first change to track the leftmost '1', equivalent to leftmost '0' tracking. In this way, we can track the largest ρ by finding the maximum value of these binary strings. Besides, HLL uses the technology called *stochastic averaging* to improve the accuracy further. By randomly dividing the binary strings into 2^b buckets, HLL calculates the cardinality based on the harmonic mean of the 2^b estimates. In FlyMon, we set both the key and p_1 to the flow key. The key is used to locate a bucket, and p_1 is used to track the largest ρ. When a measurement epoch ends, we read the measurement data and calculate the cardinality of the traffic in the control plane. Existing RMT-based HLL implementations [11, 12] track the largest ρ with TCAM entries. FlyMon can also support this implementation in the preparation stage. We prefer to implement HLL with the Max operation to save TCAM entries.

DDoS Victim Detection [58] is a multi-key distinct counting task. FlyMon can support the task with BeauCoup [12] algorithm. In the initialization stage of CMUs, we can set the key and p_1 to $C(DstIP)$ and $C(SrcIP)$, respectively. In the preparation stage, we can choose different coupons (a one-hot encoding) according to p_1 (*i.e.*, the hash value of SrcIP). Finally, we select the 'OR' in the AND-OR operation in the operation stage with p_2 (see Appendix A for detail). In the original BeauCoup algorithm, additional checksum information is maintained to detect hash collisions. In FlyMon, we adopt a mechanism similar to Count-Min Sketch to reduce the impact of hash collisions. We use multiple coupon tables (each with a CMU) and report a victim when all coupons in the multiple tables have been collected. This is reasonable because the algorithm overestimates a particular key when hash collisions occur. We evaluate the accuracy of our BeauCoup implementation (*i.e.*, FlyMon-BeauCoup) and the original BeauCoup algorithm in §5.3.

Heavy Hitter Detection [37] finds the flows whose total size dominates the whole network traffic in a measurement epoch. In FlyMon, we can use a Count-Min Sketch (CMS) or SuMax(Sum) to implement the threshold-based heavy hitter detection (*e.g.*, find the flows with a frequency larger than 1024). The Cond-ADD operation adds the value of p_1 to the counter in SRAM if the value of p_2 is greater than the counter (see Appendix A for detail). We can use the Cond-ADD operation with the second parameter as

positive infinity to regard the operation as an Uncond-ADD operation required by CMS. FlyMon can use SuMax(Sum) [66] to improve the measurement accuracy further. Unlike CMS, SuMax(Sum) only updates the counters with the current minimum value (*i.e.*, an approximate conservative updating strategy [19]). Therefore, implementing SuMax(Sum) requires cooperation among CMUs in non-overlap CMU Groups. We can store the current minimum value in p_2. The Cond-ADD operation will add p_1 to the counter only when the counter is less than the current minimum value.

Existence Check [6] detects the existence of specific flows (*e.g.*, the flows in blacklist) in the network. FlyMon uses Bloom Filter to perform this task. However, to support counter-based algorithms, the CMU's counters need to be set to a uniform memory configuration (*e.g.*, all SRAM buckets are 16-bit or 32-bit). Simply using 16 (or 32) bits as 1 bit is not memory-efficient. In FlyMon, we set both the key and p_1 to the compressed key that we want to check. In the preparation stage, the key is used to locate a bucket, while p_1 is used to select one bit from the 16 (or 32) bits of the bucket. In this way, we can make full use of all the bits in the CMUs to implement Bloom Filter.

Maximum Inter-arrival Time checks the maximum packet interval for each flow [66]. However, the packet interval time is not directly available in the data plane. Thus we must record the arrival time of each packet on each flow. We can record the arrival time of each packet through a CMU (denoted as CMU_1) using the Max operation and calculate the packet interval in CMU_1's downstream CMU (denoted as CMU_2) by performing a subtract operation in CMU_2's preparation stage. Finally, the CMU_2 performs another Max operation in its operation stage to track the maximum packet interval. However, the straightforward implementation will produce significant accuracy loss when a new flow reads the last arrival time of an old flow (*i.e.*, produces a large interval because of hash collisions). We use a Bloom Filter (with another CMU) to detect if the current flow is a new flow. If a new flow comes, we initialize the interval to 0. To conclude, this is a combinatorial task that requires three CMUs from three CMU Groups.

5 EVALUATION

We prototype FlyMon on a Wedge 100BF Tofino-based programmable switch and generate traffic on two Intel Xeon CPU E5-2650 servers, both with a Mellanox ConnectX-5 100 Gbps NIC. We conduct extensive experiments to evaluate the performance and resource overhead of FlyMon. We summarize the following experimental results:

- **Functionality.** FlyMon uses only one CMU Group to concurrently perform up to 96 isolated measurement tasks composed of all supported attributes with an extensive range of keys. These tasks can be deployed at the millisecond level with configurable memory space. The task reconfigurations have no performance impairments on traffic forwarding and the accuracy of existing measurement tasks.
- **Resource Usage.** FlyMon introduces less than 8.3% resource overhead for each CMU Group. By cross-stacking, FlyMon can deploy 9 CMU Groups within 12 MAU stages. More than 3 CMU Groups can be integrated into the Tofino baseline switch project.

Table 3: Built-in algorithms in FlyMon. 'CMUG' denotes CMU Group. The 'd' denotes the number of buckets rows.

Algorithm on CMU	Attribute	CMUG Usage	Deployment Delay (ms)
CMS (d=3)	Frequency	1	16.93
BeauCoup (d=3)	Distinct (multi-key)	1	40.18
Bloom Filter (d=3)	Existence	1	13.67
SuMax(Max) (d=3)	Max	1	19.68
HyperLogLog	Distinct (single-key)	1	5.98
SuMax(Sum) (d=3)	Frequency	3	19.47
MRAC	Frequency (distribution)	1	6.51

- **Accuracy.** With dedicated algorithms for each attribute, FlyMon can achieve higher or comparable accuracy to the state-of-the-art algorithms.

Setting. We set the set of candidate keys as 5-tuple together with a timestamp. Based on Tofino's resource distribution, we configure 6 hash (distribution) units and 3 CMUs (*i.e.*, 3 SALUs) for each CMU Group. Currently, Tofino always uses a hash (distribution) unit when accessing SRAM, even though FlyMon's design does not require the hashing calculation in the operation stage. We compromise to allocate half of the hash (distribution) units in the compression stage and half in the operation stage. We evaluate the resource utilization of the setting in §5.2.

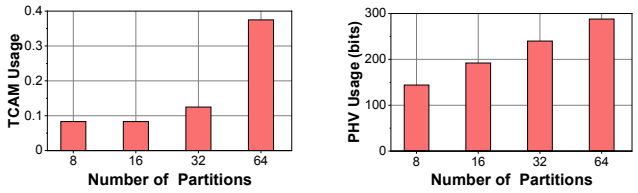
5.1 Functionality

Support for measurement tasks. We find that one CMU Group is enough for supporting all different combinations of keys and attributes. As shown in Table 3, most measurement algorithms can be implemented with a single CMU Group except for SuMax(Sum), which is optional for improving the accuracy of frequency attributes. These algorithms can cover our focused attributes and can be input with any partial key of the candidate-key set.

Task deployment delay. We find that all algorithms can be deployed within 100 ms without interrupting network traffic (in Table 3). FlyMon-BeauCoup introduces a higher delay because they need to install multiple one-hot encoding entries in the preparation stage. Specifically, it takes around 3 ms to install a common table rule and about 16 ms to install a hash mask rule. Note that the above delays include the software processing latency of FlyMon's control plane. Besides, the control plane supports batching multiple rules to mask the deployment delay. Therefore, when multiple rules are issued for a measurement task, the deployment latency does not increase exponentially.

Dynamic memory and multitasking. We find that a CMU can be split into 32 memory partitions with acceptable resource overhead, which brings great flexibility to dynamic memory management and multitasking parallelism. We evaluate the resource overhead of TCAM-based and shift-based address translation in Figure 11. We find only 12.5% of the TCAM is needed in the preparation stage to split a CMU into 32 memory partitions. It means that we can allocate 5 levels (*i.e.*, $m, m/2, ..., m/32$) of memory space on a CMU for each measurement task. More importantly, the memory partitions enable

a CMU Group to concurrently perform up to 96 (*i.e.*, 32*3) isolated measurement tasks.



(a) TCAM-based Method (b) Shift-based Method

Figure 11: Resource overhead of the two address translation methods. The TCAM usage is based on one MAU stage.

Impacts on traffic forwarding. We connect two servers to a Tofino switch and set up 12 server-client pairs with iPerf [28], which generate 80-93 Gbps TCP traffic for 100 seconds. During this period, we initiate 9 reconfiguration events in FlyMon every 10 seconds (labeled e_1 to e_9 in Figure 12a). We compare the server-side throughput of FlyMon and the static deployment method (*i.e.*, makes reconfigurations by reloading p4 codes). Since the static deployment method (labeled 'Static') will interrupt the running traffic, we make two optimizations on it: *(i)* no reconfiguration when there is a task deletion event because it is not critical. *(ii)* batch two critical events (i.e., add, reallocation) to a single reconfiguration. Besides, to rule out the performance impact of FlyMon itself, we add a data plane implementation without measurement functions (labeled 'Bare'). As shown in Figure 12a, we find that the reconfigurations in FlyMon have no performance impairment on traffic forwarding. In contrast, the reconfigurations in the static deployment method interrupt the traffic for 4-8 seconds.

Impacts on measurement accuracy. We divide a 20 seconds trace [13] into 20 discrete epochs. Each epoch has about 10K distinct flows. We emulate a traffic spike by injecting an additional 30K flows between epochs 6 and 15. We evaluate the average relative error (ARE) of a frequency measurement task (task A in Figure 12b). We compare the static deployment method and FlyMon without reloading the data plane program (*i.e.*, P4 codes). For FlyMon, we insert and remove another measurement task (*i.e.*, task B) in the same CMU Group at epochs 3 and 10, respectively. We find that the insertion and removal of measurement tasks in FlyMon don't impact the accuracy of the existing measurement task (*i.e.*, task A). Besides, we add and reduce memory for task A at epochs 6 and 16, respectively, to adapt to the changes in the traffic scale. We find that FlyMon maintains high accuracy through the on-the-fly memory reallocations, while the static deployment method introduces 15x higher ARE when facing the traffic surge.

5.2 Resource Usage and Scalability

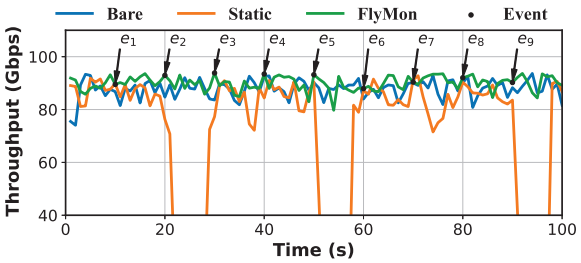
Resource overhead. We evaluate the usage of 6 critical resources by integrating CMU Groups into Tofino's baseline switch project (denoted as switch.p4) in Figure 13a. We find that the average resource overhead of a single CMU Group is less than 8.3% (the hash resources are the bottleneck). More than 3 CMU Groups can be integrated into Tofino's baseline switch project.

Cross-stacking. As shown in Figure 13b, we find that the more MAUs are allocated, the higher the resource utilization ratio can

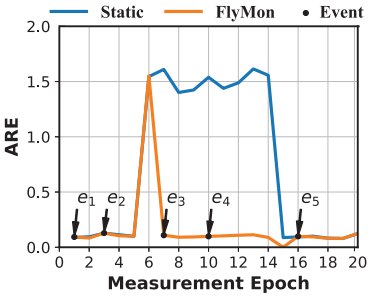
ID	Event
1	Add Task 1
2	Add Task 2
3	Del Task 1

ID	Event
4	Add Task 3
5	Resize Task 2
6	Del Task 3

ID	Event
7	Add Task 4
8	Del Task 2
9	Resize Task 4



(a) Impacts on traffic forwarding.



ID	Event
1	Add Task A
2	Add Task B
3	Add Memory of Task A
4	Del Task B
5	Reduce Memory of Task A

(b) Impacts on measurement accuracy.

Figure 12: Impacts of reconfiguration events. The 'Bare' denotes no measurement functions in the data plane. The 'Static' denotes making reconfigurations by reloading P4 codes.

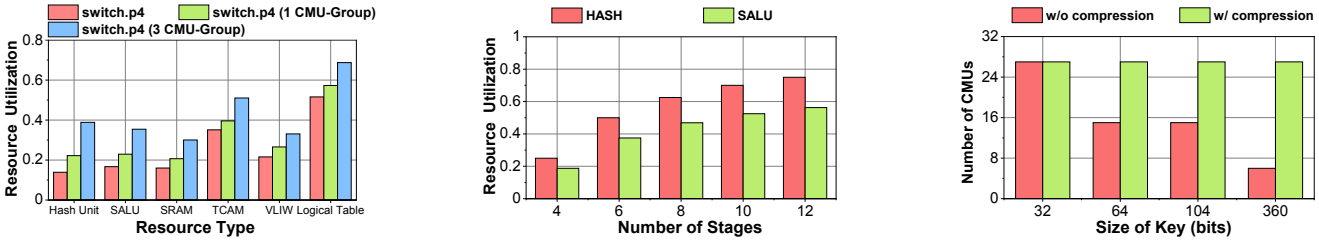
be achieved by stacking. When 12 MAU stages are allocated, the utilization of Hash and SALU resources reaches 75% and 56.25%, respectively. The reason for low SALU utilization is that current programmable switches use a hash (distribution) unit to access SRAM for SALU. We expect future RMT hardware to equip the SALUs with separate memory access units.

Scalability for key size. As shown in Figure 13c, we find that FlyMon is highly scalable to the size of candidate keys. With the optimization of PHV less-copy (*i.e.*, compression), FlyMon can deploy 5x more CMUs when the candidate key size reaches 350 bits (*i.e.*, including IPv4 addresses, IPv6 addresses, src_port, dst_port, and protocol).

5.3 Accuracy

We evaluate the measurement accuracy of FlyMon with six measurement tasks, which cover four different attributes mentioned in the paper. We use a real-world packet trace collected by the WIDE Project in 2020 [13]. We use the trace with a monitoring interval of 15 s and 30 s, which contains around 9M and 18M packets, respectively. Our metrics include relative error (RE), average relative error (ARE), F1 Score, and false positive. Their explanation can be found in Appendix C.

Heavy Hitter Detection. We set threshold=1024 and compare the accuracy of six algorithms. As shown in Figure 14a, we find that the



(a) Overhead of CMU Groups (b) Cross-stacking (c) Scalability for Key Size

Figure 13: Evaluation of resource usage.

counter-based measurement algorithms can achieve an F1 Score over 0.99 with 100 KB memory and FlyMon-SuMax is the most memory-efficient algorithm to achieve the accuracy. For BeauCoup-based algorithms, we perform this task by counting the number of different timestamps. FlyMon-BeauCoup can reach an F1 Score of over 0.9 faster original BeauCoup algorithm. We also evaluate the effect of probabilistic execution on the accuracy (Figure 14b). We find that probabilistic execution has little effect on the accuracy of heavy hitters.

DDoS Victim Detection. We use 30 seconds of traffic and set the DDoS threshold to 512. We compare the FlyMon-BeauCoup algorithm (see §4) with the original BeauCoup algorithm. As shown in Figure 14c, we find that FlyMon-BeauCoup achieves a higher F1 Score when the memory allocation exceeds 100 KB.

Flow Cardinality. We compared BeauCoup with HyperLogLog implemented on a CMU (denoted as FlyMon-HLL). As shown in Figure 14d, we find that the RE of BeauCoup can be less than 0.2 with only 16 bytes of memory. The advantage of HLL is that it can achieve a higher degree of accuracy (less than 0.1%) with more memory usage (*e.g.*, 8 KB). UnivMon cannot execute with such small memory space [12].

Flow Entropy. We deploy the MRAC algorithm in FlyMon (denoted as FlyMon-MRAC) for flow size distribution measurement, which further can be used for calculating flow entropy. We compare the results with UnivMon. As shown in Figure 14e, we find that the MRAC algorithm can achieve the RE less than 0.2 with only 200 KB of memory, while UnivMon needs 340 KB in our experiments.

Maximum inter-arrival time. We reduce the impact of hash collisions by taking the minimum value from multiple such instances. When setting d=3, we find that it can achieve an ARE of less than 4 with 5 MB of memory (Figure 14f). The result is similar to LightGuardian [66].

Existance Check. We compare the Bloom Filter before and after fully utilizing all of the memory bits (see §4). We inject 20k keys into the Bloom Filter and evaluate the performance of the Bloom Filter with about 95k elements (75k of which do not belong to the set). As shown in Figure 14g, we find that the optimized Bloom Filter achieves a False Positive (FP) of less than 0.1% using 40 KB of memory.

6 DISCUSSIONS

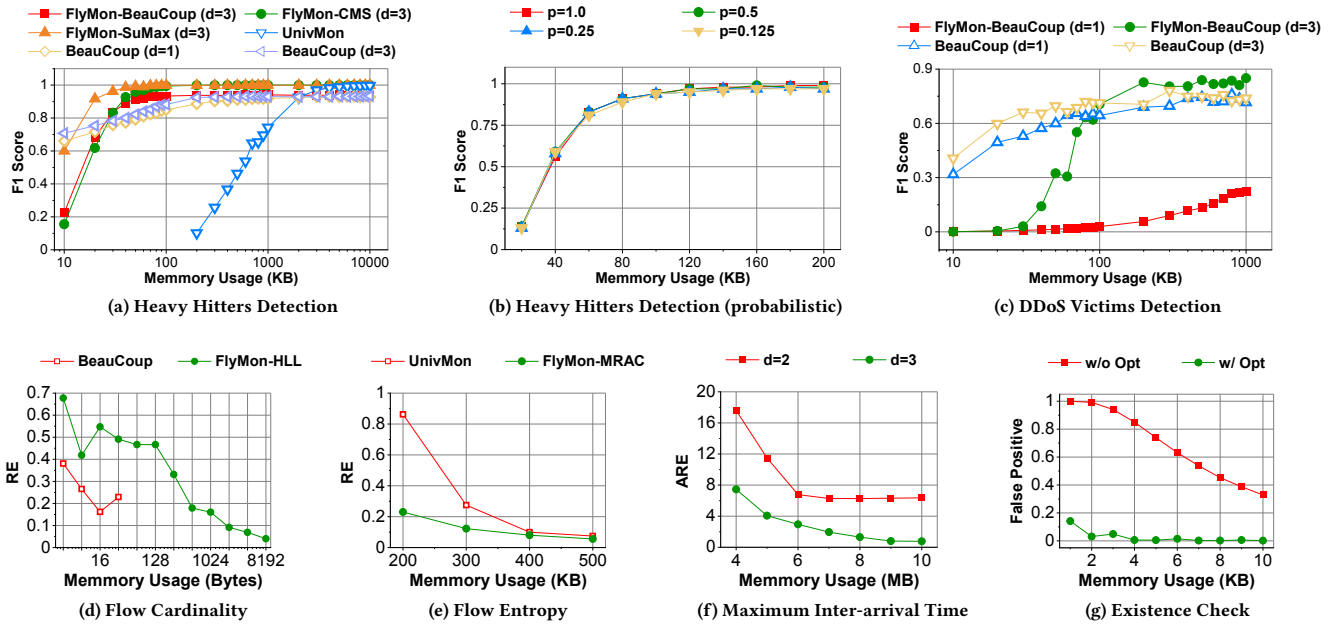
Expressiveness of FlyMon. FlyMon focuses on supporting measurement tasks based on the task abstraction described in §2.1. Given a candidate key set and a set of supported attributes, FlyMon can support the measurement tasks composed of any partial key of the candidate key set and the attributes. Although the less-copy strategy limits the number of co-existing keys, the expressiveness of FlyMon is unaffected since we introduce the dynamic hashing feature of RMT hardware to generate reconfigurable compressed keys.

In the current implementation, we do not occupy all stateful operations, which means FlyMon has expansion room to support more attributes or algorithms. For example, we can add an XOR stateful operation to implement Odd Sketch [40] for evaluating the similarity between two traffic sets. There are two ways to further increase the number of supported attributes. First, enhance hardware capabilities to allow a SALU to support more stateful operations. Second, restrict each CMU to support a portion of the attributes comes at the cost of reducing the expressiveness of the CMU.

Multitasking Parallelism. FlyMon supports the parallel execution of multiple measurement tasks running on different CMUs. Since a CMU can only perform one measurement task for a packet, only the tasks with no traffic intersection (*e.g.*, SrcIP=10.0.0.0/8 and SrcIP=20.0.0.0/8) can be performed on the same CMU. A straightforward method to solve this problem is through sampling among the tasks (*i.e.*, toss coins for each packet). In §5.3, we find that the sampling had little effect on the accuracy of heavy hitters. However, this straightforward sampling approach results in varying accuracy losses on other tasks in our preliminary experiments. The research community already has many sampling recovery methods on flow size distribution [18], distinct [9], etc. We leave a general and accurate task-parallel approach in one CMU as future work.

Memory reallocation strategy. FlyMon's built-in algorithms do not support dynamic adjustment of the memory during a measurement period without interfering with the accuracy. Our current strategy is to allocate a new task and freeze the original task. We divert the original traffic to the new task and reclaim the old task's resources.

Other optimizations. We also make some optimizations in the implementation. Firstly, a task id is assigned after firstly matching the task filter so that TCAM resources are not occupied in all CMU-Group stages. Secondly, when performing the TCAM-based address translation, we use only an ADD action to simultaneously support the ADD and SUB of address offsets (the SUB can be done by ADD overflow). Thirdly, the cross-stacking of CMU Groups cannot utilize some resources at the beginning and end of the pipeline (*i.e.*, triangle areas in Figure 8). To utilize these resources, we can splice additional three CMU groups by mirroring original packets and recirculating them while introducing extra bandwidth overhead (see Appendix E for details).



(a) Heavy Hitters Detection (b) Heavy Hitters Detection (probabilistic) (c) DDoS Victims Detection

(d) Flow Cardinality (e) Flow Entropy (f) Maximum Inter-arrival Time (g) Existence Check

Figure 14: Evaluation of measurement accuracy.

7 RELATED WORK

Existing efforts propose generic or versatile algorithms which support multiple measurement tasks simultaneously. These algorithms can be divided into two classes: multi-attribute algorithms and multi-key algorithms. The multi-attribute algorithms (*e.g.*, FlowRadar [32], UnivMon [34], Elastic [60]) can support the measurement of multiple attributes for a given key (*e.g.*, SrcIP). In opposite, the multi-key algorithms (*e.g.*, BeauCoup [12], CocoSketch [65]) can support the measurement of multiple keys for a given flow attribute. None of them can cover all measurement tasks combined from different keys and attributes. SketchLib [43] analyzes the resource bottlenecks of sketching algorithms in RMT-based switches and proposes an easy-to-use library to optimize resource occupancy without impacting fidelity. However, it focuses on optimizing the static deployment of measurement algorithms, which still allocates exclusive hardware resources for specific measurement tasks (*i.e.*, fixed keys and fixed attributes).

Unlike sketch-based measurements, Marple [44] and Sonata [22] are novel systems that use data flow operators (*e.g.*, map, filter, reduce) to represent telemetry queries and compile the queries into programmable switches. Unfortunately, the reconfiguration of RMT hardware will interrupt the running network traffic [51]. Newton [70] and DynamiQ [5] make significant contributions to enable dynamic queries based on them. Despite having a similar goal on dynamism, they focus on query-based monitoring rather than sketches. Their designs introduce massive PHV overhead [5] and do not consider the reduction of operations when accommodating diverse sketches. In contrast, FlyMon is an extensible measurement framework that focuses on the runtime reconfiguration of various sketching algorithms, the algorithms' inputs, and memory spaces. DynATOS [39] proposes a clever time-division scheduling strategy of dynamic telemetry queries to achieve higher accuracy and flexibility. The efficient data plane implementation of the queries is not their design goal.

Software-defined measurement (SDM) [41, 42, 63] is an architecture that uses the control plane's software to define the data plane's measurement functions. OpenSketch [63] proposes a novel data plane design based on NetFPGA [35], which can be configured to implement various sketching algorithms. However, OpenSketch focuses on the flexibility and resource efficiency of the measurement data plane rather than dynamics. In other words, it still needs to suspend the traffic to support undeployed measurement tasks [34]. DREAM [41] and SCREAM [42] are orthogonal works to FlyMon which provide rich control-plane functions for SDM.

8 CONCLUSION

We present FlyMon, a system enabling on-the-fly monitoring for versatile network measurements. FlyMon is implemented on RMT hardware and enables dynamic change of measurement tasks and resource allocations without interrupting running network traffic. Our future work includes the following three parts. Firstly, we intend to support more flow attributes in FlyMon. Secondly, we will explore a general and accurate task parallelism method within a CMU. Thirdly, we will enrich the functionality of FlyMon's control plane.

ACKNOWLEDGMENTS

We thank our shepherd (Robert Soulé) and the anonymous reviewers for their constructive feedback. We also thank Yuhan Wu, Kaicheng Yang, Ruijie Miao, Rui Qiu for their valuable comments. This research is supported by the Key-Area Research and Development Program of Guangdong Province under Grant number 2020B0101390001, the National Natural Science Foundation of China under Grant Numbers 92067206, 62072228 and 61972222.

REFERENCES

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 503–514. https://doi.org/10.1145/2619239.2626316

[2] Ran Ben-Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. 2018. Efficient Measurement on Programmable Switches Using Probabilistic Recirculation. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. 313–323. https://doi.org/10.1109/ICNP.2018.00047

[3] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies* (Tokyo, Japan) *(CoNEXT '11)*. Association for Computing Machinery, New York, NY, USA, Article 8, 12 pages. https://doi.org/10.1145/2079296.2079304

[4] Rohan Bhatia, Arpit Gupta, Rob Harrison, Daniel Lokshtanov, and Walter Willinger. 2021. DynamiQ: Planning for Dynamics in Network Streaming Analytics Systems. https://doi.org/10.48550/ARXIV.2106.05420

[5] Rohan Bhatia, Arpit Gupta, Rob Harrison, Daniel Lokshtanov, and Walter Willinger. 2021. DynamiQ: Planning for dynamics in network streaming analytics systems. *arXiv preprint arXiv:2106.05420* (2021).

[6] Burton H. Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (jul 1970), 422–426. https://doi.org/10.1145/362686.362692

[7] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. *SIGCOMM Comput. Commun. Rev.* 43, 4 (aug 2013), 99–110. https://doi.org/10.1145/2534169.2486011

[8] Frank Cangialosi, Dave Levin, and Neil Spring. 2015. Ting: Measuring and Exploiting Latencies Between All Tor Nodes. In *Proceedings of the 2015 Internet Measurement Conference* (Tokyo, Japan) *(IMC '15)*. Association for Computing Machinery, New York, NY, USA, 289–302. https://doi.org/10.1145/2815675.2815701

[9] Anne Chao and Shen-Ming Lee. 1992. Estimating the Number of Classes via Sample Coverage. *J. Amer. Statist. Assoc.* 87, 417 (1992), 210–217. https://doi.org/10.1080/01621459.1992.10475194 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1992.10475194

[10] Peiqing Chen, Yuhan Wu, Tong Yang, Junchen Jiang, and Zaoxing Liu. 2021. Precise Error Estimation for Sketch-Based Flow Measurement. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, New York, NY, USA, 113–121. https://doi.org/10.1145/3487552.3487856

[11] Xiaoqi Chen. 2020. HLL Implementation. https://github.com/Princeton-Cabernet/p4-projects/tree/master/HyperLogLog-tofino.

[12] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. 2020. BeauCoup: Answering Many Network Traffic Queries, One Memory Update at a Time. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 226–239. https://doi.org/10.1145/3387514.3405865

[13] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. 2000. Traffic Data Repository at the WIDE Project. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (San Diego, California) *(ATEC '00)*. USENIX Association, USA, 51.

[14] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[15] J.H. Crawford. 1990. The i486 CPU: executing instructions in one clock cycle. *IEEE Micro* 10, 1 (1990), 27–36. https://doi.org/10.1109/40.46766

[16] Vitalii Demianiuk, Sergey Gorinsky, Sergey I. Nikolenko, and Kirill Kogan. 2021. Robust Distributed Monitoring of Traffic Flows. *IEEE/ACM Transactions on Networking* 29, 1 (2021), 275–288. https://doi.org/10.1109/TNET.2020.3034890

[17] Bert den Boer and Antoon Bosselaers. 1994. Collisions for the compression function of MD5. In *Advances in Cryptology — EUROCRYPT '93*, Tor Helleseth (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 293–304.

[18] Nick Duffield, Carsten Lund, and Mikkel Thorup. 2003. Estimating Flow Distributions from Sampled Flow Statistics. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Karlsruhe, Germany) *(SIGCOMM '03)*. Association for Computing Machinery, New York, NY, USA, 325–336. https://doi.org/10.1145/863955.863992

[19] Cristian Estan and George Varghese. 2002. New Directions in Traffic Measurement and Accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Pittsburgh, Pennsylvania, USA) *(SIGCOMM '02)*. Association for Computing Machinery, New York, NY, USA, 323–336. https://doi.org/10.1145/633025.633056

[20] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *AofA: Analysis of Algorithms (DMTCS Proceedings, Vol. DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07))*, Philippe Jacquet (Ed.). Discrete Mathematics and Theoretical Computer Science, Juan les Pins, France, 137–156. https://doi.org/10.46298/dmtcs.3545

[21] Jayant Gadge and Anish Anand Patil. 2008. Port scan detection. In *2008 16th IEEE International Conference on Networks*. IEEE, 1–6. https://doi.org/10.1109/ICON.2008.4772622

[22] Arpit Gupta, Rüdiger Birkner, Marco Canini, Nick Feamster, Chris Mac-Stoker, and Walter Willinger. 2016. Network Monitoring as a Streaming Analytics Problem. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (Atlanta, GA, USA) *(HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 106–112. https://doi.org/10.1145/3005745.3005748

[23] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. *SoftNIC: A Software NIC to Augment Hardware*. Technical Report UCB/EECS-2015-155. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-155.html

[24] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. SketchVisor: Robust Network Measurement for Software Packet Processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 113–126. https://doi.org/10.1145/3098822.3098831

[25] Qun Huang and Patrick P.C. Lee. 2015. A hybrid local and distributed sketching design for accurate and scalable heavy key detection in network data streams. *Computer Networks* 91 (2015), 298–315. https://doi.org/10.1016/j.comnet.2015.08.025

[26] INTEL. 2020. OpenTofino. https://github.com/barefootnetworks/Open-Tofino

[27] Intel. 2021. Tofino2. https://www.intel.cn/content/www/cn/zh/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[28] iperf.fr. 2015. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/iperf-download.php

[29] N. Kamiyama, T. Mori, and R. Kawahara. 2007. Simple and Adaptive Identification of Superspreaders by Flow Sampling. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. 2481–2485. https://doi.org/10.1109/INFCOM.2007.305

[30] Abhishek Kumar, Minho Sung, Jun (Jim) Xu, and Jia Wang. 2004. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA) *(SIGMETRICS '04/Performance '04)*. Association for Computing Machinery, New York, NY, USA, 177–188. https://doi.org/10.1145/1005686.1005709

[31] Abhishek Kumar, Minho Sung, Jun (Jim) Xu, and Jia Wang. 2004. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. *SIGMETRICS Perform. Eval. Rev.* 32, 1 (jun 2004), 177–188. https://doi.org/10.1145/1012888.1005709

[32] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA) *(NSDI'16)*. USENIX Association, USA, 311–324.

[33] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and General Sketch-Based Monitoring in Software Switches. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 334–350. https://doi.org/10.1145/3341302.3342076

[34] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 101–114. https://doi.org/10.1145/2934872.2934906

[35] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianying Luo. 2007. NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 160–161. https://doi.org/10.1109/MSE.2007.69

[36] Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. 2008. Counter Braids: A Novel Counter Architecture for per-Flow Measurement. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (Annapolis, MD, USA) *(SIGMETRICS '08)*. Association for Computing Machinery, New York, NY, USA, 121–132. https://doi.org/10.1145/1375457.1375472

[37] Gurmeet Singh Manku and Rajeev Motwani. 2012. Approximate Frequency Counts over Data Streams. *Proc. VLDB Endow.* 5, 12 (aug 2012), 1699. https://doi.org/10.14778/2367502.2367508

[38] Yuxin Meng and Lam-For Kwok. 2014. Adaptive Blacklist-Based Packet Filter with a Statistic-Based Approach in Network Intrusion Detection. *J. Netw. Comput. Appl.* 39, C (mar 2014), 83–92.

[39] Chris Misa, Walt O'Connor, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. 2022. Dynamic Scheduling of Approximate Telemetry Queries. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 701–717. https://www.usenix.org/conference/nsdi22/presentation/misa

[40] Michael Mitzenmacher, Rasmus Pagh, and Ninh Pham. 2014. Efficient Estimation for High Similarities Using Odd Sketches. In *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea) *(WWW '14)*. Association for Computing Machinery, New York, NY, USA, 109–118. https://doi.org/10.1145/2566486.2568017

[41] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2014. DREAM: Dynamic Resource Allocation for Software-defined Measurement. *Computer Communication Review* 44 (08 2014). https://doi.org/10.1145/2619239.2626291

[42] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2015. SCREAM: Sketch Resource Allocation for Software-Defined Measurement. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies* (Heidelberg, Germany) *(CoNEXT '15)*. Association for Computing Machinery, New York, NY, USA, Article 14, 13 pages. https://doi.org/10.1145/2716281.2836099

[43] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. 2022. SketchLib: Enabling Efficient Sketch-based Monitoring on Programmable Switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 743–759. https://www.usenix.org/conference/nsdi22/presentation/namkung

[44] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-Directed Hardware Design for Network Performance Monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 85–98. https://doi.org/10.1145/3098822.3098829

[45] ONF. 2017. P4 Runtime. https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[46] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 117–130. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[47] Michael Scharf and Sebastian Kiesel. 2006. NXG03-5: Head-of-line Blocking in TCP and SCTP: Analysis and Measurements. In *IEEE Globecom 2006*. 1–5. https://doi.org/10.1109/GLOCOM.2006.333

[48] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A. Dinda, Ming-Yang Kao, and Gokhan Memik. 2007. Reversible Sketches: Enabling Monitoring and Analysis Over High-Speed Data Streams. *IEEE/ACM Transactions on Networking* 15, 5 (2007), 1059–1072. https://doi.org/10.1109/TNET.2007.896150

[49] Jingsong Shan, Yinjin Fu, Guiqiang Ni, Jianxin Luo, and Zhaofeng Wu. 2017. Fast counting the cardinality of flows for big traffic over sliding windows. *Frontiers of Computer Science* 11 (02 2017), 119–129. https://doi.org/10.1007/s11704-016-6053-x

[50] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research* (Santa Clara, CA, USA) *(SOSR '17)*. Association for Computing Machinery, New York, NY, USA, 164–176. https://doi.org/10.1145/3050220.3063772

[51] John Sonchack, Oliver Michel, Adam J. Aviv, Eric Keller, and Jonathan M. Smith. 2018. Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 823–835. https://www.usenix.org/conference/atc18/presentation/sonchack

[52] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. 2020. *FCM-Sketch: Generic Network Measurements with Data Plane Support.* Association for Computing Machinery, New York, NY, USA, 78–92. https://doi.org/10.1145/3386367.3432729

[53] Lu Tang, Qun Huang, and Patrick P. C. Lee. 2019. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, 2026–2034. https://doi.org/10.1109/INFOCOM.2019.8737499

[54] Lu Tang, Qun Huang, and Patrick P. C. Lee. 2020. SpreadSketch: Toward Invertible and Network-Wide Detection of Superspreaders. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE, 1608–1617. https://doi.org/10.1109/INFOCOM41043.2020.9155541

[55] Belma Turkovic, Fernando Kuipers, Niels van Adrichem, and Koen Langendoen. 2018. Fast Network Congestion Detection and Avoidance Using P4. In *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies* (Budapest, Hungary) *(NEAT '18)*. Association for Computing Machinery, New York, NY, USA, 45–51. https://doi.org/10.1145/3229574.3229581

[56] Shuhe Wang, Chen Sun, Zili Meng, Minhu Wang, Jiamin Cao, Mingwei Xu, Jun Bi, Qun Huang, Masoud Moshref, Tong Yang, Hongxin Hu, and Gong Zhang. 2020. Martini: Bridging the Gap between Network Measurement and Control Using Switching ASICs. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 1–12. https://doi.org/10.1109/ICNP49622.2020.9259415

[57] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. 1990. A Linear-Time Probabilistic Counting Algorithm for Database Applications. *ACM Trans. Database Syst.* 15, 2 (jun 1990), 208–229. https://doi.org/10.1145/78922.78925

[58] Yang Xu and Yong Liu. 2016. DDoS attack detection under SDN context. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524500

[59] Kaicheng Yang, Yuanpeng Li, Zirui Liu, Tong Yang, Yu Zhou, Jintao He, Jing'an Xue, Tong Zhao, Zhengyi Jia, and Yongqiang Yang. 2021. SketchINT: Empowering INT with TowerSketch for Per-flow Per-switch Measurement. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. 1–12. https://doi.org/10.1109/ICNP52444.2021.9651940

[60] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 561–575. https://doi.org/10.1145/3230543.3230544

[61] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. 2017. Pyramid Sketch: A Sketch Framework for Frequency Estimation of Data Streams. *Proc. VLDB Endow.* 10, 11 (aug 2017), 1442–1453. https://doi.org/10.14778/3137628.3137652

[62] Fang Yu, R.H. Katz, and T.V. Lakshman. 2004. Gigabit rate packet pattern-matching using TCAM. In *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004.* 174–183. https://doi.org/10.1109/ICNP.2004.1348108

[63] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (Lombard, IL) *(nsdi'13)*. USENIX Association, USA, 29–42.

[64] Yinda Zhang, Jinyang Li, Yutian Lei, Tong Yang, Zhetao Li, Gong Zhang, and Bin Cui. 2020. On-off Sketch: A Fast and Accurate Sketch on Persistence. *Proc. VLDB Endow.* 14, 2 (oct 2020), 128–140. https://doi.org/10.14778/3425879.3425884

[65] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: High-Performance Sketch-Based Measurement over Arbitrary Partial Key Query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 207–222. https://doi.org/10.1145/3452296.3472892

[66] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. 2021. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 991–1010. https://www.usenix.org/conference/nsdi21/presentation/zhao

[67] Hao Zheng. 2022. A Reference Implementation of FlyMon System. https://github.com/NASA-NJU/FlyMon.

[68] Hao Zheng, Yanan Jiang, Chen Tian, Cheng Long, Qun Huang, Weichao Li, Yi Wang, Qianyi Huang, Jiaqi Zheng, Rui Xia, Yi Wang, Wanchun Dou, and Guihai Chen. 2021. Rethinking Fine-grained Measurement from Software-defined Perspective: A Survey. *IEEE Transactions on Services Computing* (2021), 1–1. https://doi.org/10.1109/TSC.2021.3103968

[69] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. 2018. Cold Filter: A Meta-Framework for Faster and More Accurate Stream Processing. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 741–756. https://doi.org/10.1145/3183713.3183726

[70] Yu Zhou, Dai Zhang, Kai Gao, Chen Sun, Jiamin Cao, Yangyang Wang, Mingwei Xu, and Jianping Wu. 2020. *Newton: Intent-Driven Network Traffic Monitoring.* Association for Computing Machinery, New York, NY, USA, 295–308. https://doi.org/10.1145/3386367.3431298

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A REDUCED STATEFUL OPERATIONS

Operation 1 The Cond-ADD stateful operation

Input: register, key, p_1, p_2
Output: result

```
function COND-ADD(p1, p2)
    if register[key] < p2 then
        register[key] ← register[key] + p1
        return register[key]
    else
        return 0
    end if
end function
```

Operation 2 The MAX stateful operation

Input: register, key, p_1, p_2
Output: result

```
function MAX(p1, p2)
    if register[key] < p1 then
        register[key] ← p1
        return register[key]
    else
        return 0
    end if
end function
```

Operation 3 The AND-OR stateful operation

Input: register, key, p_1, p_2
Output: result

```
function AND-OR(p1, p2)
    if p2 == 0 then
        register[key] ← register[key]&p1
    else
        register[key] ← register[key] | p1
    end if
    return register[key]
end function
```

B ACCURACY ANALYSIS

In this section, we prove that the mapping from n distinct flows to a log m-bit compression domain will result in a hash collision probability approximating $1 - e^{-n/m}$ for each flow.

Firstly, we look at the case where a key is hashed to m buckets. The probability of one bucket being hashed to is $1/m$. Let $P(X = k)$ be the probability density function of a bucket containing X keys after all n keys are inserted into m buckets. Obviously $P(X = k)$ obeys the binomial distribution $B(n, \frac{1}{m})$, so we have

$$P(X = k) = \binom{n}{k}(\frac{1}{m})^k(1 - \frac{1}{m})^{n-k}$$

Let C denotes the event of a hash collision. The number of keys that do not encounter any collision $N(\neg C)$ is equal to the number of buckets that contain only one key. Therefore, the number of non-collision keys is

$$N(\neg C) = mP(X = 1) = m\binom{n}{1}(\frac{1}{m})(1 - \frac{1}{m})^{n-1}$$

When n and m are large enough, the above equation can be approximated as

$$N(\neg C) = n(1 - \frac{1}{m})^{m\frac{n-1}{m}} \approx ne^{-(\frac{n}{m})}$$

So the number of collision keys is equal to the total number of keys n minus the number of non-collision keys $ne^{-(\frac{n}{m})}$. Finally, we derive the probability of collision for each key as:

$$P(C) = N(C)/n = (n - ne^{-(\frac{n}{m})})/n = 1 - e^{-(\frac{n}{m})}$$

C EVALUATION METRICS

Metrics. We use different metrics for different aspects of the performance evaluation. To evaluate the measurement accuracy of FlyMonin different measurement tasks. We use the following four metrics:

(1) *ARE (Average Relative Error)*: $\frac{1}{n}\Sigma_{i=1}^{n}\frac{|f_i - \hat{f}_i|}{f_i}$, where N is the number of distinct flows, f_i and $\hat{f}_i$ are true and estimated metrics for flow f_i, respectively. We use *ARE* to evaluate the accuracy of per-flow size estimation.
(2) *F1-score*: $\frac{2\times PR\times RR}{PR+RR}$, where PR (Precision Rate) denotes the ratio of true instances reported and RR (Recall Rate) denotes the ratio of reported true instances. We use *F1-score* to evaluate the accuracy of heavy-hitter detection and DDoS victim detection.
(3) *RE (Relative Error)*: $\frac{|\hat{x}-x|}{x}$, where x and $\hat{x}$ are true and estimated metrics, respectively. We use *RE* to evaluate the accuracy of flow entropy and flow cardinality.
(4) *FP (False Positive)*: $\frac{N_{fp}}{N_{fp}+N_{tn}}$, where N_{fp} is the number of negative events wrongly categorized as positive. The N_{np} is the number of true negatives.

D SKETCH IMPLEMENTATION

Since MRAC [30], TowerSketch [59], Counter Braids [36], and Linear Counting [57] are not described in detail in the paper, we add their implementations in FlyMon here. MRAC and Count-Min Sketch implementations are identical in the data plane and are only differentiated in the control plane analysis. The same is true for Linear Counting and Bloom Filter. Therefore, we focus on the implementation of TowerSketch and Counter Braids here.

TowerSketch [59] adapts to high-skewed traffic by holding more small bit-width counters for mice flows. There are two challenges to implementing TowerSketch in FlyMon. The first challenge is how to support bucket arrays of different sizes. FlyMon can overcome this challenge with the address translation mechanism (*i.e.*, dynamic memory). The second challenge is how to implement different bit-width counters in different arrays under a uniform memory configuration (*e.g.*, all buckets are 16-bit). FlyMon solves this challenge by considering part of the bucket bits as a counter. As shown in Figure 15a, we can use several significant (*i.e.*, left-side) bits of a bucket to act as a flexible bit-width counter. To update the

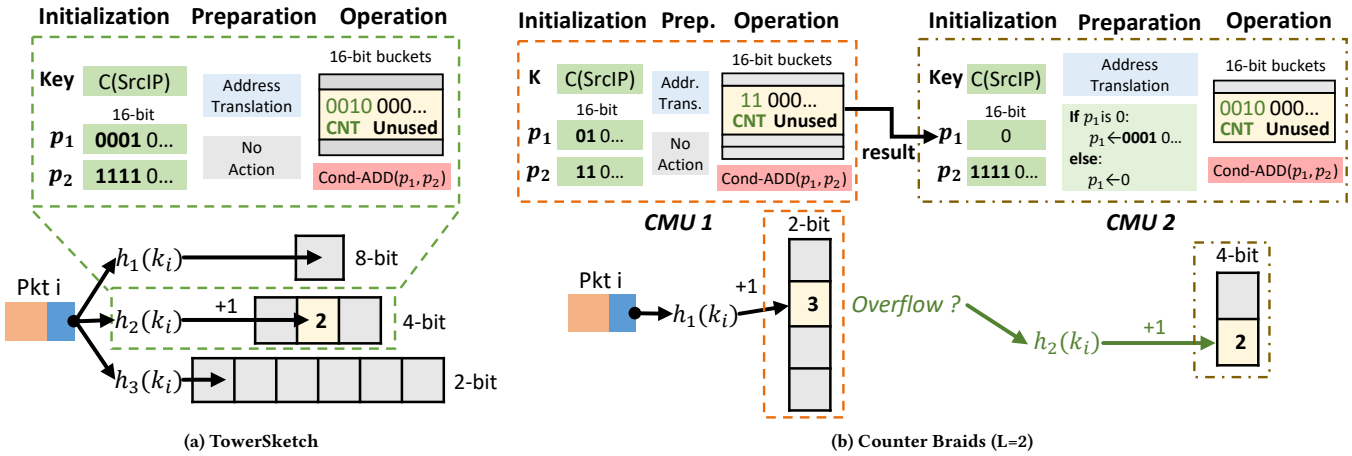


Figure 15: Implementation of TowerSketch and Counter Braids in FlyMon.

counter, p_1 is set to a value that represents '1' under the bit range of the flexible counter. For example, if we want to implement a 4-bit counter, we need to use four left-side bits of the bucket. The p_1 is set to the binary number '00010...'. We use the Cond-ADD operation and set p_2 to the binary number '11110...' to avoid overflow (*i.e.*, the counter is only increased when it is less than p_2). The unused bits (*i.e.*, 'Unused' in Figure 15a) are not wasted because they can be used by other measurement tasks (*e.g.*, the tasks using Bloom Filter or BeauCoup).

Counter Braids [36] is an accurate per-flow measurement architecture towards realizing zero-error measurement. Counter Braids also use several different size arrays with different bit-width counters. The difference is that the arrays of Counter Braids are arranged in multi-layer, and a high-layer counter is updated when the low-layer counter is overflowed. Therefore, it introduces another challenge. That is how to judge the overflow in the low-layer counter and update the high-layer counter accordingly. As shown in Figure 15b, we use the Cond-ADD operation to update the counters. Note that the Cond-ADD will not update the counter and output '0' if the counter is not less than p_2 (see Appendix A). In CMU 1, we can set p_2 to the binary number '110...' to judge if the 2-bit counter is overflowed. In CMU 2, we set p_1 as the result of CMU 1 and use two TCAM entries in the preparation stage to judge different situations. If the p_1 equals 0, we set p_1 to the binary number '00010...' to update the 4-bit counter in CMU 2. Otherwise, we set p_1 to 0 to avoid updating the counter in CMU 2, since the low-layer counter (*i.e.*, the counter in CMU 1) is not overflowed.

E OPTIMIZATION OF PIPELINE RESOURCES

There are two triangle areas at the beginning and end of the pipeline. We cannot directly use these resources because the remaining MAU resources cannot implement a complete CMU Group. As shown in Figure 16, we can build 3 additional CMU Groups by splicing these resources. If some packets need to perform the measurement tasks on these spliced CMU Groups (labeled as 10, 11, and 12 in Figure 16), we need to mirror the packets to a recirculate port. Additional metadata must be carried on these mirrored packets to help complete the unfinished CMU-Group actions. Only packets that need to perform the tasks on these spliced CMU Groups will incur additional bandwidth overhead.

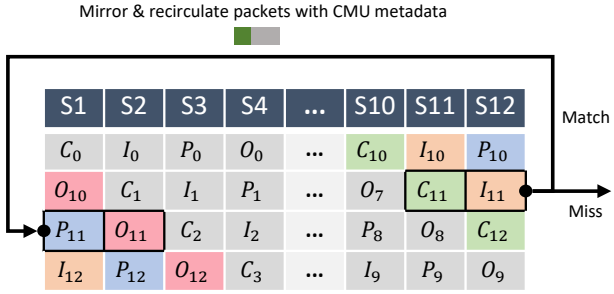


Figure 16: Fully utilize pipeline resources through mirror and recirculation.

F ARTIFACT APPENDIX

Abstract

FlyMon is an on-the-fly measurement system that can accommodate many measurement tasks. Without reloading the P4 program on RMT hardware switches, FlyMon can dynamically change measurement tasks and resource allocations without interrupting running traffic. We have fully implemented FlyMon on Tofino, a programmable switch with extremely high throughput (i.e., 6.5 Tbps). We have open-sourced the artifacts of FlyMon, which contains data plane codes, an interactive control plane framework, and simulations to evaluate built-in algorithms' accuracy fast.

Scope

The hardware implementation can be used to validate the dynamic nature of FlyMon, including making on-the-fly reconfigurations on sketching algorithms, the algorithms' inputs, and memory sizes (i.e., by adding new tasks with specified memory sizes). The artifact also supports validating the deployment delay of the algorithms

and the resource usage of CMU Groups. With the simulation framework, the artifact supports fast validate the accuracy of the built-in algorithms.

This artifact serves as an early exploration for academics purpose. This is not an implementation with industrial-grade reliability.

Contents

The artifact includes following four parts:

- A P416-based hardware implementation supporting four flow attributes: Frequency, Max, Distinct, and Existence.
- An interactive control plane realizing task reconfiguration, resource management, and data collection.
- A simulation framework to fast explore built-in algorithms' accuracy.
- Some easy-to-follow manuals used to introduce how to use our codes.

We consider FlyMon an extensible framework, and we will add the support for more sketching algorithms and flow attributes in future work. We will also improve the functionality and stability of the control plane.

Hosting

Our reference implementation of FlyMon is located in the main branch of the GitHub repository NASA-NJU/FlyMon [67].

Requirements

This artifact has strict hardware and software requirements. For hardware requirements, a Tofino-based hardware switch or a Tofino model are needed to deploy the P4 codes. At least one server with QSFP28 connectors and wires if you need to generate packets to a hardware switch. For the software requirements, the operating system of our switch is OpenNetworkLinux 4.14.151. We use Python version 3.8.10 and Intel SDE version 9.7.0.