This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Designing Heavy-Hitter Detection Algorithms for Programmable Switches

Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich

***Abstract*— Programmable network switches promise flexibility and high throughput, enabling applications such as load balancing and traffic engineering. Network measurement is a fundamental building block for such applications, including tasks such as the identification of heavy hitters (largest flows) or the detection of traffic changes. However, high-throughput packet processing architectures place certain limitations on the programming model, such as restricted branching, limited capability for memory access, and a limited number of processing stages. These limitations restrict the types of measurement algorithms that can run on programmable switches. In this paper, we focus on the Reconfigurable Match Tables (RMT) programmable high-throughput switch architecture, and carefully examine its constraints on designing measurement algorithms. We demonstrate our findings while solving the heavy hitter problem. We introduce PRECISION, an algorithm that uses *Partial Recirculation* to find top flows on a programmable switch. By recirculating a small fraction of packets, PRECISION simplifies the access to stateful memory to conform with RMT limitations and achieves higher accuracy than previous heavy hitter detection algorithms that avoid recirculation. We also evaluate each of the adaptations made by PRECISION and analyze its effect on the measurement accuracy. Finally, we suggest two algorithms for the hierarchical heavy hitters detection problem in which the goal is identifying the subnets that send excessive traffic and are potentially malicious. To the best of our knowledge, our work is the first to do so on RMT switches.**

***Index Terms*— Software defined networking, measurement.**

I. Introduction

PROGRAMMABLE network switches enable rapid deployment of network algorithms such as traffic engineering, load balancing, quality-of-service optimization, anomaly detection, and intrusion detection [9], [21], [25], [28], [41] Measurement capabilities are often at the core of such applications, as they extract information from traffic to make informed decisions [22]. The current trend is to allow application designers to code applications by invoking measurement tasks as primitives [26], [31], [42], [45], [52], [55]. For example, traffic engineering [9] may want to optimize the steering of the largest (heavy hitter) flows [5], [32], and an intrusion detection system [23], [40], [43], [48] may be interested in hierarchical heavy hitters [6], [39], or in the flows' entropy [34], [35].

Zooming in on the implementation of a single measurement task, the main challenge is the traffic volume. Typically there are millions of network flows to monitor [46], [49] and ideally each flow is allocated some memory for storing its measurement statistics. Coping with the 100 Gbps line rate requires various optimization techniques that depend on the target platform; e.g., utilizing sampling [34], SIMD instructions [44], or FPGAs [53], [54] to accelerate the performance.

Heavy Hitter algorithms only store flow state for the largest flows to overcome this limitation. This approach exposes a trade-off between memory space and accuracy, where additional space improves the accuracy.

There are two types of solutions for heavy hitter detection problem — *counter-based* algorithms and *sketch-based* algorithms. Counter-based algorithms maintain a bounded-size flow cache. Only a small portion of the flows are measured, and each monitored flow has its own counter [16], [37]. Examples of counter-based algorithms include *Lossy Counting* [37], *Frequent* [29], *Space-Saving* [38], and *RAP* [5]. In sketch-based algorithms, counters are implicitly shared by many flows. Examples of sketch-based algorithms include *Multi Stage Filters* [24], *Count-Min Sketch* [19], *Count Sketch* [12], *Randomized Counter Sharing* [32], *Counter Tree* [13], and *UnivMon* [35].

Heavy hitter measurement has two closely related goals. In the *frequency estimation* problem, we wish to approximate the size of a flow whose ID is given at query time. Alternatively, in the *top-k* problem, the algorithm is required to list the k top flows. In general, sketch algorithms solve the frequency estimation problem but require additional efforts to address the top-k problem. For example, UnivMon [35] uses heaps alongside the sketches to track the top flows. FlowRadar [33] and Reversible Sketch [47] encode flow ID in the sketch, and have a small probability to fail to decode. In contrast, counter algorithms already store flow identifiers and can directly solve the top-k problem. While sketch algorithms are readily implementable in programmable switches, supporting top-k measurements is a strong motivation for deploying counter algorithms in such switches. Unfortunately, high-performance

Manuscript received March 6, 2019; revised September 16, 2019 and February 15, 2020; accepted February 26, 2020; approved by IEEE/ACM Transactions on Networking Editor P. Giaccone. This work was supported in part by the NSF under Grant CCF-1535948, in part by the Taub Family Foundation, in part by the Technion Hiroshi Fujiwara Cyber Security Research Center, in part by the Cyber Security Research Center at Ben-Gurion University, in part by the Israel National Cyber Directorate, in part by the Zuckerman Foundation, in part by the Alon Fellowship, in part by the German-Israeli Science Foundation (GIF) Young Scientists Program, and in part by the Gordon Fund for System Engineering. *(Corresponding author: Ori Rottenstreich.)*

Ran Ben Basat is with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: ran@seas.harvard.edu).

Xiaoqi Chen is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA (e-mail: xiaoqic@cs.princeton.edu).

Gil Einziger is with the Department of Computer Science, Ben Gurion University, Beer Sheva 84105, Israel (e-mail: gilein@bgu.ac.il).

Ori Rottenstreich is with the Technion, Haifa 3200003, Israel (e-mail: or@technion.ac.il).

Digital Object Identifier 10.1109/TNET.2020.2982739

1063-6692 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

packet processing imposes severe restrictions on the programming model which makes implementing counter algorithms a daunting task.

Some applications such as attack mitigation and intrusion detection require something more sophisticated than (plain) heavy hitters [23], [40], [43], [48]. For example, in a Distributed Denial of Service (DDoS) attack, a large number of devices collaborate to overwhelm an Internet service. In many cases, the source IP addresses of the attacking devices are different from the legitimate traffic. That is, the attack shares common prefixes which correspond to several sub-networks that do not deliver much legitimate traffic. *Hierarchical Heavy Hitters (HHH)* identify frequently appearing sub-networks. These can be used to either white-list traffic from the most frequent sub-networks of the legitimate traffic or to detect that traffic from specific sub-networks is likely due to an attack and blacklist them. Since programmable switches are powerful enough to cope with the current volume of DDoS attacks, performing HHH analysis on such switches offers exciting opportunities. Unfortunately, even (plain) heavy hitters are non trivial to implement in programmable switches.

Contribution: Our work summarizes the restrictions of the Reconfigurable Match Tables (*RMT*) [11] switch programming model in the context of measurement algorithm design. The RMT breakthrough design allows a pipeline multiple match-action tables of different widths and depths and was recently described as a "key enabling technology for the support of the emerging P4 data plane programming" [20]. We divide the RMT restrictions into four easy-to-understand rules: *limited branching*, *limited concurrent memory access*, *single stage memory access*, and *fixed number of stages*. We used several techniques to fit our implementation into RMT restrictions, and these techniques are also applicable to implementing other measurement algorithms in the data plane.

We present *Partial RECirculation admisSION (PRECISION)* – a heavy hitter algorithm that is fully compatible with the RMT high-performance programmable switch architecture. We implemented PRECISION in the P4 language [10] on the newly released Barefoot Tofino [1] programmable switch that achieves multiple Tbps of aggregated throughput, and deployed it in Princeton University's campus network to measure real-world heavy hitter flows. The P4 source code of PRECISION can be found at [2]. The core idea behind PRECISION is *Partial recirculation*; PRECISION recirculates a small portion of packets from unmonitored flows; we decide probabilistically or deterministically if the packet should be recirculated and pass again through the programmable switching pipeline. In the first pipeline pass, we try to match a packet to an existing flow entry; if this succeeds, we increment its counter. If unmatched, we sometimes recirculate it to claim an entry with the new packet's flow ID. Using the packet recirculation feature greatly simplifies the memory access pattern without significantly degrading throughput, while by carefully setting the recirculation portion we achieve high monitoring accuracy.

Previous suggestions include HashParallel and HashPipe [51], two counter-based heavy hitter detection algorithms proposed specifically for running on high-throughput programmable switches. They both maintain a d-stage flow table tailored to the pipeline architecture of programmable switches but differ in whether to recirculate an unmatched packet.HashPipe never recirculates packets and always inserts the new entry, which yields high throughput but lower accuracy. Instead, HashParallel recirculates **every** unmatched packet, which achieves much better accuracy but lowers the throughput. In contrast, PRECISION only recirculates a tiny portion of the unmatched packets with a minimal impact on performance. This approach allows PRECISION to conform to the RMT memory access restrictions and also improves accuracy over HashPipe, especially for heavy-tailed workloads. We then analyze the impact of each RMT constraint individually and find that most limitations have little effect in practice. We also show that HashPipe [51] cannot satisfy both the *limited branching* rule and the *single stage memory access* rule. Within the terminology of Domino [50], HashPipe requires the more complex Paired atoms (one pipeline stage supports reading two values, performing two nested branching, and writing two values back) while PRECISION can be implemented with the simpler RAW atoms (one stage supports reading one value, add to it, and write it back).

Next, we suggest two methods to implement Hierarchical Heavy Hitters (HHH) detection on programmable switches. These, utilize PRECISION as a black box, and demonstrate the feasibility of HHH detection entirely in the data plane of a high performance switch. Such a capability is an important enabler for attack mitigation systems.

Finally, we evaluate PRECISION on real packet traces and show that it improves on the state-of-the-art for high-performance programmable switches (HashPipe) for the two variants of the heavy hitter problem. It is up to 1000 times more accurate than HashPipe for the frequency estimation problem and reduces the space required to correctly identify the top-128 flows by a factor of up to 32 times. When compared to general (software) heavy hitter algorithms, PRECISION often has similar accuracy compared to Space-Saving and RAP. Interestingly, approximating the desired recirculation probability appears very important, with a stage-efficient 2-approximate solution PRECISION requires at most four times as much memory as RAP. When we dedicate more hardware pipeline stages to achieve a better approximation, the performance gap between PRECISION and RAP diminished.

Paper Outline: The paper is structured as follows. Section II outlines the programming restrictions of the RMT high-performance switch architecture and their impact on designing data plane algorithms. Section III introduces the reader to the heavy hitter detection problem and surveys related work. Section IV discusses the implementation of PRECISION, specifically how we adapt to the limitations imposed by the RMT architecture to achieve probabilistic recirculation; we also discussed a deterministic variant of PRECISION. We present theoretical analysis on bounding the amount of recirculation in Section V. Section VI shows an extension of our work to perform the Hierarchical Heavy Hitters (HHH) measurement. In Section VII, we evaluate PRECISION, by first quantifying the impact of each adaptation on the accuracy, and then position it within the field by

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

comparing it with other heavy-hitter detection algorithms. Finally, we conclude in Section VIII.

II. Constraints of Programmable Switches

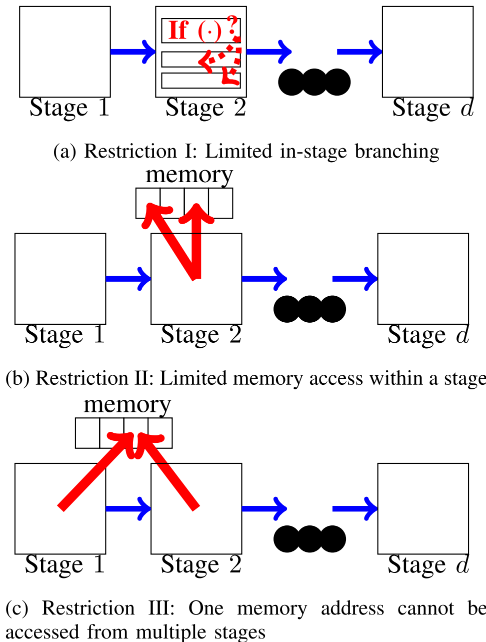
The emergence of P4-based programmable data plane [10] is an exciting opportunity to push network algorithms to programmable switches. In this section, we give a brief introduction of the recently developed RMT [11] high-performance programmable switch architecture and explain its programming model and restrictions in the context of network measurement algorithm design.

The RMT architecture uses a pipeline to process packets. At a glance, the packet first goes into a programmable packet header parser that extracts the header fields, and then traverses a series of pipeline stages, and finally is emitted through a deparser. Each stage includes a *Match-Action Table*, which first performs a *Match* that reads some packet header fields and matches them with a list of values. Then, it performs the corresponding *Action* on the packet, which can be routing decisions or modifying header field variables. RMT promises Tbps-level throughput which is achieved by limiting the complexity of pipeline stages. These typically run at a fixed clock cycle $\geq 1\,GHz$ (i.e., $\leq 1\mu s$ processing time), permitting only elementary actions. Flexibility is achieved by allowing many parallel actions in the same stage, and by connecting many simple stages into a pipeline. Our case-study of heavy hitter measurement in this model exposed the following restrictions which we survey below.

Simple Per-Stage Actions (Limited Branching): Also, branching operations are expensive, and the hardware pipeline may only support very limited branching within stages (but can perform complex branching across stages using match-action tables), as illustrated in Figure 1a. Therefore, we cannot perform arbitrary computation and have to redesign the algorithm to fit the architecture. We expand on this limitation in Section IV-B.

Limited Concurrent Memory Access: A small amount of static random access memory (SRAM) is attached to each hardware stage for stateful processing. As illustrated in Figure 1b, when a packet arrives, it can access one, or a few, addresses in the memory region but not read or write the entire memory block, again due to per-stage timing requirement. From an algorithm design perspective, this means we can only read from or write to memory at specific addresses, and therefore cannot compute even the most straightforward functions globally, e.g., find a minimum across many array elements.

Single Stage Memory Access: Each stage is processing a different packet at any given time. Therefore, allowing two packets to access the same memory region may cause a read-write hazard, shown in Figure 1c. The RMT architecture avoids this by allowing access to stateful memory blocks only from one particular pipeline stage. Thus, our algorithm can only access each memory region once as the packet is going through the pipeline. We need to *recirculate* a packet, causing it to go through the entire pipeline again, in order to access the same memory block twice. Recirculation is expensive as it reduces the rate that incoming packets can access the pipeline.



(a) Restriction I: Limited in-stage branching

(b) Restriction II: Limited memory access within a stage

(c) Restriction III: One memory address cannot be accessed from multiple stages

Fig. 1. Illustration of some restrictions imposed by RMT pipeline model for designing measurement algorithm.

Even in more recently proposed architecture like dRMT [14] where memory resources are dynamically allocated to different hardware stages, we still cannot allow accessing the same memory region from two different pipeline stages. Therefore, the restriction we describe seems fundamental.

Fixed Number of Stages: For guaranteeing a low per-packet latency, the switch cannot have too many pipeline stages. In our case, since the pipeline is not very long, the total number of operations performed on a packet cannot exceed a hardware-imposed constant. Again, we can circumvent the limit by recirculating some packets, with a throughput impact.

Limited Support for Arithmetic Operations: Each pipeline stage can only execute primitive arithmetic. For example, division is much slower than addition; thus the switching hardware usually does not support division or multiplication.

Discussion: While these restrictions target specifically the newly proposed RMT architecture, we believe that future high-throughput switching architectures are likely to have similar constraints due to the throughput and latency requirements they need to satisfy.

We also note that capabilities prepared for packet forwarding can be exploited by measurement algorithms as well. The Match-Action Table model specifies that each pipeline stage will use a part of packet header data (e.g., a network address) to perform a lookup in a match table, and subsequently executes the corresponding action in the table (e.g., a forwarding destination). In our algorithm design perspective, this means we can perform parallel lookups on intermediate variable cheaply. Beyond *exact* matching, the architecture also supports *ternary* and *longest-prefix* matching.

Note that the TCAM memory used in table lookup is different from the memory used for stateful processing (SRAM) mentioned earlier. TCAM allows for parallel reads, but writing may not finish in constant time. Hence it can only be modified by the switch control plane but not within the data plane (by

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

the packet being processed, in one pipeline clock cycle). Thus, the parallel-readable lookup tables are "read-only" for the packet, and writable memory must be accessed by addresses.

III. The Heavy Hitters Problem and Existing Solutions

This section formally defines the problems addressed in this work as well as surveys the relevant related work.

A. Problem Statement

Our work targets two common measurement forms, the *frequency estimation* problem and the *top-k* problem. For both, we refer to a quasi-infinite packet stream, where each packet is associated with a flow as explained below.

A flow refers to a particular subset of the packet stream that we choose to combine and analyze as a whole. For example, a flow may apply to a TCP connection or a UDP flow, in which case the five-tuple (source and destination IP, protocol, source and destination port) becomes the flow identifier. Alternatively, a flow may refer to just the source IP address, or just the destination IP and port pair. In any case, we assume that a flow identifier is available from some fields of the packet header, and that flows partition the stream such that each packet belongs to a single flow.

We denote the frequency of a network flow with ID s, or the total number of packets belonging to flow s, as f_s. For the *frequency estimation* problem, we use the OnArrival model [5], which requires an algorithm to estimate the flow frequency for each new packet it sees, and evaluates the estimation error upon each packet arrival. Formally, we reveal packets in a stream $(p_1, p_2, \ldots)$ one packet at a time, and on each packet arrival, with packet p_t belonging to some flow s. An algorithm *Alg* is required to provide an estimate $\widehat{f_s}$ for $f_s \triangleq |\{p_i \in s | 1 \leq i \leq t\}|$ — the number of packets belonging to flow s in $p_1, \ldots, p_t$.

The *top-k* identification problem is defined as follows: Given a stream $(p_1, p_2, \ldots)$ and a query parameter k, the algorithm outputs a set of flows containing as many of the k largest flows as possible.

B. Existing Approaches

The Space-Saving Algorithm: Space-Saving (SS) [38] is a heavy hitter algorithm designed for database applications and software implementations. Space-Saving maintains a fixed-size flow table, where each entry has a flow identifier and a counter. When a packet from an unmonitored flow arrives, the identifier of the minimal table entry is replaced with the new flow's identifier, and its counter is incremented. Space-Saving uses a sophisticated data structure named stream-summary which allows it to maintain the entries ordered according to counter values in constant time as long as all updates are of unit weight.

Space-Saving was designed for database workloads, which often exhibit a heavily concentrated access pattern, i.e. most of the traffic comes from a few heavy hitters. In contrast, networking traces are often heavy-tailed [5], [27]. That is, a non-negligible percentage of the packets belong to tail flows or those other than heavy hitters. Unfortunately, Space-Saving works poorly on such workloads. For conciseness, we present only the Space Saving algorithm between all classical heavy hitter algorithms, as it is often considered to be the most accurate [15], [16], [36].

Optimization for Heavy-Tailed Workloads: To deal with heavy-tailed workload, Filtered Space-Saving [27] attempts to filter out tail flows before inserting into flow table. It utilizes a bitmap alongside a Space-Saving instance. When a packet arrives, a hash function is used to map its flow ID into a bitmap entry. If the entry is zero, it merely sets the entry to one. Otherwise, we update the Space-Saving instance.

Maintaining additional data structures to filter tail flows may be wasteful. Therefore, *Randomized Admission Policy (RAP)* [5] suggests using randomization instead. When an unmonitored flow arrives, it is admitted only with a small probability. Thus, most tail flows are filtered while heavy hitters that appear many times are eventually admitted. Specifically, if the minimal entry has a counter value of c, RAP requires the competing flow to win a coin toss with a probability of $\frac{1}{c+1}$ to be added. The idea of RAP can be applied to the Space-Saving algorithm for software implementations. For hardware efficiency, the authors evaluate a limited associativity variant.

Unfortunately, the programming model of high-performance programmable switches is too restrictive to implement these algorithms directly. Specifically, Space-Saving evicts the minimal flow entry across all monitored flows, whereas the architecture of programmable switches does not permit finding (and replacing) the minimum element among all counters. Even for the limited associativity variant of RAP, it is still difficult to implement the randomize replacement after finding the approximate minimum value, due to same-stage memory access restriction.

High-Performance Switch Algorithms: HashPipe [51] adapts Space-Saving to meet the design constraints of the P4 language and PISA programmable switch architecture [10]. The authors suggest partitioning the counters into d separate stages to fit the programmable switch pipeline. They use d hash functions that dictate which counter can accommodate each flow on each stage. They first propose a strawman solution, *HashParallel*, which makes each packet traverse all d stages while tracking the minimal value among the counters associated with its flow. If the flow is monitored, HashParallel increments its counter. If not, it recirculates the packet to replace the minimal entry among the d. The authors explain that HashParallel potentially recirculates all the packets, which halves the throughput.

Hence, they suggest HashPipe as a practical variant with no recirculation. In HashPipe, each packet's flow entry is always inserted in the first stage. They then find a rolling minimum — the evicted flow proceeds to the next stage where its counter is compared with the flow monitored there. The flow with the larger counter remains, while the smaller flow's entry is propagated further. Eventually, the smaller counter on the dth stage is evicted. This allows HashPipe to avoid recirculation but introduces the problem of duplicates — some flows may occupy multiple counters, and small flows may still evict other flows.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

FlowRadar [33] is another P4 measurement algorithm that follows a different design pattern. The main design difficulty to overcome is the lack of access to a fully associative hash table in programmable switches. While HashPipe and this work implement a fixed associativity table using multiple pipeline stages, FlowRadar potentially stores multiple flows within the same table entry. That is, upon hash collision the new flow identifier is XORed into the existing identifier. FlowRadar works best when the measurement is distributed, where multiple programmable switches can share their state to decode flow entries. Initially, FlowRadar recovers all flow entries that had no collision. Recovered flows are then recursively removed from the data structure, enabling for more flows to be recovered.

This approach is differentiated from our own as it attempts to perform an accurate measurement and therefore requires space which is proportional to the number of flows. In contrast, our approach provides an approximation of the flow sizes, and the required memory is independent of the number of flows. Also, FlowRadar requires multiple measurement devices each encoding a different subset of flows whereas our solution can also be implemented on a single device.

Sampling: Instead of running algorithms in the data plane, one may also sample a fraction of packets and run sophisticated algorithms elsewhere [7]. This approach simplifies the hardware implementation but the problem migrates elsewhere. Namely, to process the samples in real time, we need additional computation and bandwidth overheads. Also, achieving high monitoring accuracy on smaller flows requires high sampling rate.

Hierarchical Heavy Hitters: MST is an HHH algorithm that utilizes an independent (plain) heavy hitter instances for each prefix length [39]. Once in a while, MST calculates the set of HHH prefixes from the heavy hitters of each prefix length. The RHHH algorithm [6] optimizes the performance of MST in software by updating a single random prefix. These algorithms are non-trivial to implement in programmable switches due to the limited programming model.

IV. Design and Implementation of PRECISION

We now present several hardware-friendly adaptations that address the restrictions imposed by RMT switch architecture.

A. From Fully Associative to d-Way Associative Memory Access

Building on top of Space-Saving [38] and RAP [5], we first tackle the fact that a programmable switch cannot perform the fully-associative memory access to evict the minimum item. At any given pipeline stage, the algorithm can specify an index to access some location in the register array. The switch may allow accessing a small number of positions simultaneously but definitely cannot compute a global minimum across an entire register array.

We adopt the limited-associativity idea from HashParallel and HashPipe [51] to approximately evict a small element, by choosing the minimum across d randomly selected elements from d separate register arrays. With this relaxation, we can naturally spread the memory access across different hardware stages, and at each hardware stage, we only access one memory location. Specifically, we use d independent hash functions $h_1, \ldots, h_d$ to compute a different index for each stage, and at each stage, we access the $h_i(key)^{th}$ element of the ith register array. Note that PRECISION performs d flow entry reads, but it does not consume exactly d hardware pipeline stages, as processing each read involves two branchings, and costs three hardware stages. We also discuss how to reduce the total number of hardware stages required in Section IV-F.

Algorithm 1 HashPipe [51] Heavy Hitter Algorithm

```
1  l_1 ← h_1(iKey)                    ▷ Always insert in the first stage;
2  if key_1[l_1] = iKey then
3  |   val_1[l_1] ← val_1[l_1] + 1;
4  |   end processing;
5  else if l_1 is an empty slot then
6  |   (key_1[l_1], val_1[l_1]) ← (iKey, 1);
7  |   end processing;
8  else
9  |   (cKey, cVal) ← (key_1[l_1], val_1[l_1]);
10 |   (key_1[l_1], val_1[l_1]) ← (iKey, 1);
11 for i ← 2 to d do
12 |                     ▷ Track a rolling minimum;
13 |   l_i ← h_i(cKey);
14 |   if key_i[l_i] = cKey then
15 |   |                   ▷ Read key_i ;
16 |   |   val_i[l_i] ← val_i[l_i] + cVal        ▷ R/W val_i ;
17 |   |   end processing;
18 |   else if [l_i] is an empty slot then
19 |   |   (key_i[l_i], val_i[l_i]) ← (cKey, CVal)   ▷ Write key_i,
   |   |   val ;
20 |   |   end processing;
21 |   else if val_i[l_i] < cVal then
22 |   |      ▷ Condition on val_i; Violating Restriction I;
23 |   |   swap (cKey, cVal) ⇔ (key_i[l_i], val_i[l_i])    ▷ R/W
   |   |   key_i ;
```

B. Simplified Memory Access

a) Implementation requirements of HashPipe: Although the design of HashPipe has already satisfied many restrictions imposed by RMT structure, its memory access pattern prevents us from implementing it in today's programmable switch hardware (that has a limited support for Paired atoms). The high-level idea of the HashPipe algorithm (see pseudocode in Algorithm 1) is to always evict the minimum out of d elements, by "carrying" a candidate eviction element through the pipeline. At each stage, we compare the counter read from register memory with that of the carried element. Then, the smaller of which is propagated further onward.

We now scrutinize the register memory access to different arrays of HashPipe, as highlighted in Algorithm 1. If we look at Line 14 and Line 23, they both access the register array key holding flow identifiers. The single stage memory access restriction requires that line 14 through line 23 would be placed within the same hardware pipeline stage.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

However, the execution flow is branched in line 21 based on the values in another register array (val). Such branching violates the limited in-stage branching restriction. Referring to the model presented in [50], to implement HashPipe, the simple *RAW*[1] action atoms at each stage are inadequate, and at least *Paired*[2] action atoms are required. While the RMT architecture [11] does not specifically define what features the action units need to support, Paired action atoms are more expensive to implement than RAW atoms and require 14x larger chip area than RAW atoms [50]. Therefore, today's RMT programmable switches do not support Paired atoms. We strive to design our measurement algorithm to only require the simpler RAW atoms.

With only the simple RAW atoms, it is not possible to conditionally update a flow entry while simultaneously incrementing the corresponding counters. As long as we place flow identifier and counter in two separate register arrays, this seemingly innocuous set of operations has some inevitable in-stage branching: if we access flow identifiers first, we need to: (i) Read flow ID from flow entry array; (ii) If ID matched, increment counter; otherwise, compare the counter to the carried counter value; (iii) If the condition is satisfied, replace flow ID. This leads to a write to flow entry register memory conditioned on reading from another counter register memory. Therefore, two nested branching within the stage is inevitable.

Some may argue that we can cleverly rearrange the operations to mitigate the branching; however, even if we access the counter first, we still encounter the same restriction: (i) Read a counter from the counter register memory; (ii) Read flow ID; if ID not matched, check if the counter is smaller than the carrier counter to decide whether to replace the flow ID; (iii) Write the incremented counter value, if the ID matched. Again, the conditional write after reading another register forces two nested branching within a hardware pipeline stage (requiring Paired atom). Therefore, we cannot implement HashPipe on the first generation programmable switches available on the market.

b) PRECISION's solution: The implementation of PRECISION is even more challenging. We decide to replace an entry after knowing the minimum sampled counter value, but we only know this value after reaching the end of the pipeline, at which point it is too late to write to the register memory of earlier stages.

We resolve this difficulty using the recirculation feature on switches [4], [10], that allows packets to traverse the pipeline again, removing all conditional branching for register access. When a packet leaves the last stage of the pipeline, instead of leaving the switch, we may choose to bring it to the beginning of the pipeline and go through all stages again. We can use metadata to distinguish between recirculated packets (which should be dropped) and regular packets that should be forwarded to their next hop.

Using recirculation allows more versatile packet processing at the cost of packet forwarding performance, as the recirculated packet will compete for resources with new incoming packets. However, we believe it's a necessary trade-off to satisfy the no-branching-within-stage constraint for high-performance programmable switches.

At the end of the pipeline, we ignore those packets already matched to flow entries and probabilistically recirculate the other packets using probability $\frac{1}{carry_min+1}$, where $carry_min$ is the value of the minimum sampled entry. The recirculated packet will evict and replace the minimum sampled entry. It will traverse the pipeline again to write its flow identifier into the corresponding register array when it arrives at the right pipeline stage, and also update the corresponding counter to a new value $carry_min + 1$. In expectation, for every unmatched packet we increased the count for its flow by 1.

As a packet recirculates, it introduces a delay between the point in which we chose to admit it, and when it writes its flow ID on its second pipeline traversal. During this period other packets may increment the counter, an effect that will be overridden. Thus, the recirculation delay may have some impact on PRECISION's accuracy. The duration of such delay is architecture-specific and depends on both the queuing before entering the pipeline and the length of the pipeline. In Section VII-B, we evaluate its impact on PRECISION's accuracy and show that PRECISION is insensitive to such delay.

C. Efficient Recirculation

We avoid packet reordering and minimize application-level performance impact by using the *clone-and-recirculate* primitive, which routes the original packet out of the switch as usual, and drops the cloned packet after it finishes the second pipeline traversal. This implies that in-flow packet order is preserved and that a packet can only be recirculated once.

Since recirculated packets compete for resources with incoming packets, we would like to minimize the number of recirculated packets. Fortunately, recirculation happens only for unmatched packets, with a probability of $\frac{1}{carry_min+1}$, where $carry_min$ is the minimal counter value the packet saw in all pipeline stages. Thus, recirculation becomes less frequent as the measurement progresses and the counters grow. In Section V we show that the expected number of recirculated packets is asymptotically bounded by the square root of the number of packets.

We can further bound the expected recirculation ratio at the beginning of the execution by initializing all counter registers to a non-zero minimum value. For example, if we initialize all counters to 100, we also set an upper bound 1% for the expected recirculation probability. Subsequently, because of concentration bound, the probability for having more than $(1+\varepsilon)\%$ recirculation becomes negligible. In Section VII-C we show that adding an appropriate initial value has a negligible accuracy impact. We also note that in hardware switches, recirculating 1% of packets leads to at most 1% impact on throughput.

D. Approximating the Recirculation Probabilities

Recall that the original RAP algorithm admits packets from new flows with probability $\frac{1}{carry_min+1}$. Intuitively, a flow

[1]The RAW action unit is capable of Reading an element from register memory, Add a value to it, and Write it back. See [50].

[2]The Paired action unit is capable of reading two different elements from register memory, conditionally branch twice (two nested *if* s), perform addition or subtraction to the elements, and write two new values back. See [50].

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

needs to arrive $carry_min + 1$ times on average to capture a counter with a value of $carry_min + 1$.

It is straightforward to achieve this probability if a random arbitrary-range integer generator is available: we can generate an integer within $[0, carry_min]$ and check if it's 0. However, sometimes we can only obtain random bits from programmable switch's hardware random source, and this effectively limits us to generate random integers within $[0, 2^x - 1]$ range. Without the capability to do division or multiplication, we cannot accurately sample with desired probability $\frac{1}{carry_min+1}$. As we show in Section VII-D, we can work around this limit without affecting accuracy.

The most simple workaround is to only use probabilities of the form 2^{-x}. Achieving this probability is done by comparing x random bits to zeroes. That is, we recirculate unmatched packets with probability $\frac{1}{carry_min+1}$ rounded to the next smallest 2^{-x}. This is a 2-approximation of the desired recirculation probability. The recirculated packet will update the counter to 2^x. Rounding is achieved by using a ternary matching over bits of $carry_min$ variable to find the highest 1 bit. The evaluation in Section VII-D shows that this method has a noticeable but acceptable impact on accuracy.

We now introduce a tighter method for approximating the desired recirculation probability. Inspired by floating point arithmetic, we may decompose $carry_min + 1 = 2^y \times T$, $T \in [8, 16)$ and use a probability of the form $\frac{1}{2^y} \times \frac{1}{\lceil T \rceil}$ to approximate $\frac{1}{carry_min+1}$. We can directly implement the $\frac{1}{2^y}$, while the $\frac{1}{\lceil T \rceil}$ is approximated by randomly generating an integer between $[0, 2^N]$ and comparing it against a pre-computed constant $\lfloor \frac{2^N}{\lceil T \rceil} \rfloor$, via a lookup table. Further, to avoid non-integer number representation, we always increment the counter value by 1 upon recirculation. This achieves a $9/8$-approximation of the desired recirculation probability. Our evaluation shows that the accuracy gains are significant. Yet, this method requires additional pipeline stages.

E. Putting All Adaptations Together

With all the aforementioned hardware-friendly adaptations in mind, we assemble the PRECISION algorithm, which satisfies all hardware-imposed constraints of the RMT architecture. Algorithm 2 is a pseudocode version of PRECISION. Line 1 reflects PRECISION's d-way associative memory access, iterating through each way. In Line 7 we increment the counter for matched packets, while unmatched packets handled between Line 15 and Line 19. We flip a coin in Line 17, and the 2-approximation of recirculation probability manifests in Line 16. Recirculated packets update register memory corresponding to their minimal entries. This is described between Line 20 to Line 24. We highlighted accesses to register memory in color, note that registers are only accessed once per stage. Each branching fits in a transition between hardware pipeline stages, removing the need to perform in-stage branching.

F. Parallelizing Actions to Reduce Hardware Stages Used

Algorithm 2 presented PRECISION in its most straightforward arrangement, iterating through the d-way in tandem, while each uses three pipeline stages. This costs as much as $d \times 3$ hardware pipeline stages for register memory reads. Since the total number of stages is very limited, we explain how to optimize the required number of stages further, and fit a larger d on the same hardware. This optimization may also be applicable to other algorithms with a similar repeated register array access pattern.

Algorithm 2 PRECISION Heavy Hitter Algorithm

```
1  for i ← 1 to d do
2      l_i ← h_i(iKey) ;
3      if key_i[l_i] = iKey then
4              ▷ Hardware stage i_A: access key_i register;
5          matched_i ← true;
6      if matched_i then
7          val_i[l_i] ← val_i[l_i] + 1  ▷ Hardware stage i_B: access
           val_i register ;
8      else
9          oval_i = val_i[l_i]
10     if (¬matched_i) ∧ (oval_i < carry_min) then
11             ▷ Hardware stage i_C: maintain carry minimum;
12         carry_min ← oval_i;
13         min_stage ← i
14 if ⋀_{i=1}^{d} (¬matched_i) then
15         ▷ iKey not in cache; do Probabilistic Recirculation.
       new_val = 2^{⌈log_2(carry_min)⌉};
16     Generate random integer R ∈ [0, new_val − 1],
       by assembling ⌈log_2(carry_min)⌉ random bits;
17     if R = 0 then
18         clone and recirculate packet;
19 if packet is recirculated then
20     i ← min_stage;
21     l_i ← h_i(iKey);
22     key_i[l_i] ← iKey      ▷ Hardware stage i_A: access key_i
       register ;
23     val_i[l_i] ← new_val  ▷ Hardware stage i_B: access val_i
       register;
24     Drop the cloned copy;
```

Intuitively, each 'if' in the pseudocode is a branching, separating the algorithm into different hardware stages. However, it may be possible to group independent stages and reduce the total number of hardware stages needed.

In our implementation, PRECISION requires two branching for each of the d ways. That is, it requires three pipeline stages for each way. The stages in each way are:

Stage A: Read flow ID from flow entry array. *(branching: does entry's ID match my ID?)*

Stage B: Read/Update from the counter array. *(branching: is counter smaller than the current minimum?)*

Stage C: Compute and "carry" the new minimum value.

If we indeed require three hardware stages for each pair of flow entry array and counter array, a switch with X physical stages can at most implement PRECISION with $d = X/3$. This assumes that all pipeline stages serve for heavy-hitter detection. In practice, we would like to leave enough pipeline stages for other network applications.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

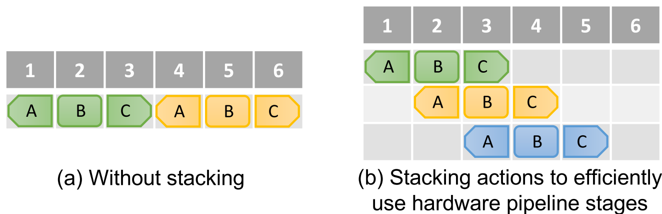


Fig. 2. We reduce the number of pipeline stages used by stacking together independent actions between different ways. For d-way PRECISION, this reduces the number of pipeline stages required from $d \times 3$ to $d + 2$.

However, our algorithm does not have a hard dependency between different groups of stages. If we denote the d ways as 1, 2, 3 and the three pipeline stages for each action as A, B, and C, we can observe that (for example) 2_A and 1_C are independent. Thus, it's not necessary to serialize everything into the pattern shown in Figure 2(a). Instead, we can "stack" operations from different groups together, as shown in Figure 2(b). Specifically, reading the flow identifier for the next flow entry array can be parallelized with incrementing a counter for the previous way's counter array and so forth. Therefore, we can parallelize different execution stages of multiple ways as there is no direct causal relation or data dependency between stage action $(i+1)_B$ and i_C, or between $(i+1)_A$ and i_B. Thus, by using the stacking pattern shown in Figure 2(b), we reduce the number of required stages to implement d-way PRECISION from $d \times 3$ to $d+2$, amortizing to one stage per way. [3]

For a programmable switch with a limit of X hardware stages, the actual maximum d we can implement will be smaller, because we need extra stages before and after the core algorithm for setup and teardown, such as extracting flow ID and performing random coin-tossing. Furthermore, a network switch will need to fulfill its regular duties like routing, access control, etc., and would not devote all its resources to the PRECISION algorithm. Nevertheless, we can expect any commodity programmable switch to run the $d = 2$ version of PRECISION smoothly, alongside its regular duties. When extra resources are available, we may increase d to improve accuracy as shown in Section VII-A.

G. A Deterministic PRECISION Variant

In this section, we consider a variant that replaces the probabilistic recirculation mechanism of PRECISION by a deterministic one. Intuitively, instead of admitting each packet with probability p we can admit the $1/p$'th packet. To implement this, we change the hashing scheme of PRECISION so that each flow is only hashed once (i.e., all l_i are the same, see Line 2 in Algorithm 2). We add a single counter per row that is initialized to zero and incremented for every packet that is mapped to this row. Then, if the counter equals the minimal value observed by the packet, we recirculate the packet and reset the counter.

[3]There is indeed a causal dependency between stage $(i+1)_C$ and i_C when computing the carried minimum value $carry_min$, thus using only a constant number of 3 hardware stages is not possible. Also, other hardware constraints that limit the number of parallel actions in one hardware stage exists, but these are less stringent than the limit on the total number of hardware stages.

V. Bounding the Amount of Recirculation

Here we show a bound on the total number of packet-recirculations. Our main result, Theorem 3, shows that the number of recirculated packets is sublinear. Combined with our approach for setting initial values to counters to avoid high recirculation ratio at the beginning, we maintain recirculation at acceptable levels throughout the measurement. While the main theorem deals with the randomized PRECISION, it also applies to the deterministic version (discussed in Section IV-G), as the expected number of recirculated packets is roughly the same in both versions.

We first present an auxiliary lemma about summing random variables.

Lemma 1: Fix some $p \in (0,1]$, $T \in \mathbb{N}^+$ and let $X_1, X_2, \ldots \sim Geo(p)$ be independent geometric random variables with mean $1/p$. Denote by $Z \triangleq \min\{n \in \mathbb{N} \mid \sum_{i=1}^{n} X_i \geq T\}$ the minimal number n such that the sum of $X_1, \ldots, X_n$ exceeds the threshold T. Then $\mathbb{E}[Z] = p(T-1)+1$.

Proof: For $n \in \{1, \ldots, T\}$, let $S_n \triangleq \sum_{i=1}^{n} X_i$ denote the sum of the first i random variables. Next, let $Y \triangleq |\{1, \ldots, T\} \setminus \{S_n \mid n \in \{1, \ldots, T-1\}\}|$ denote the number of integers between 1 and T that are *not* a sum of prefix of the X_i's. Observe that since the variables X_i are i.i.d., geometric variables, we have that $Y \sim Bin(T-1, p)$; that is, Y is a binomial random variable with mean $p(T-1)$. But observe that the value of Z is simply one plus the number of n values for which $S_n < T$. This establishes that $\mathbb{E}[Z] = 1 + \mathbb{E}[Y] = p(T-1)+1$. □

Next, we show a bound on the expected number of packets that would be sampled by a single-counter PRECISION instance. For this, we denote by X_i the number of packets between the time that the counter has reached a value of i and the time it first reaches of $i+1$.

Lemma 2: Fix some $T \in \mathbb{N}^+$ and let $\{X_i \sim Geo(1/i) \mid i \in \mathbb{N}\}$ denote independent geometric variables such that the expectation of X_i is i. Similarly to the above lemma, let $A \triangleq \min\{n \in \mathbb{N} \mid \sum_{i=1}^{n} X_i \geq T\}$ denote the number of variables needed to cross the threshold T. Then $\mathbb{E}[A] \leq 2\sqrt{T}$.

Proof: Intuitively, since $\mathbb{E}[X_i] = i$ and $\sum_{i=1}^{n} i = O(n^2)$, we can expect that $\Omega(\sqrt{T})$ variables are needed to cross the T threshold. To prove this, first notice that we are looking for an asymptotic bound rather than computing the expectation exactly. This allows us to "ignore" the first $\sqrt{T}-1$ random variables. Formally:

$$\mathbb{E}[A] \leq \sqrt{T} - 1 + \mathbb{E}\left[\min\left\{n \in \mathbb{N} \mid \sum_{i=\sqrt{T}}^{\sqrt{T}+n} X_i \geq T\right\}\right]. \quad (1)$$

Next, let $X'_{\sqrt{T}}, \ldots, X'_T \sim Geo(T^{-1/2})$ denote a set of i.i.d. geometric variables (independent of $X_1, \ldots, X_{\sqrt{T}}$) with expectation of $\sqrt{T}$. Notice that for $i \geq \sqrt{T}$, we have that the parameter for X_i is smaller than that of X'_i. Together with (1),

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

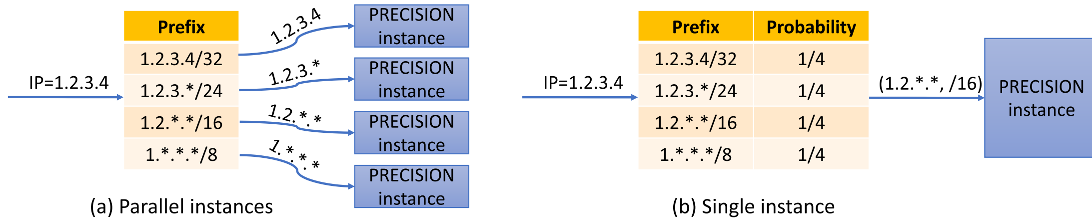


Fig. 3. An illustration of our two implementation approaches for Hierarchical Heavy Hitters.

this allows us to further write:

$$\mathbb{E}[A] \leq \sqrt{T} - 1 + \mathbb{E}\left[\min\left\{n \in \mathbb{N} \mid \sum_{i=\sqrt{T}}^{\sqrt{T}+n} X_i' \geq T\right\}\right]. \quad (2)$$

Finally, we use Lemma 1 to write $\mathbb{E}\left[\min\left\{n \in \mathbb{N} \mid \sum_{i=\sqrt{T}}^{\sqrt{T}+n} X_i' \geq T\right\}\right] = \sqrt{T} + 1$. Together with (2) we conclude that $\mathbb{E}[A] \leq 2\sqrt{T}$. □

We now present the main theorem. Note that here we assume ideal random recirculation probability $1/i$, and the approximation techniques only reduce recirculation further.

Theorem 3: Denote the number of packets in the stream by N and the number of counters by C. The expected number of recirculated packets is $O(\sqrt{NC})$.

Proof: For $i \in \{1, \ldots, C\}$, let R_i denote the number of times PRECISION recirculates a packet to update the i'th counter and by R the overall recirculation. Next, let N_i denote the number of times this counter was probabilistically modified (that is, a packet traversed the entire pipeline, and this counter was the minimal along its path). We have that $\sum_{i=1}^{C} N_i \leq N$ (this is inequality as some packets update their flow counter and are surely not recirculated). According to Lemma 2 we have that $\mathbb{E}[R_i] \leq 2\sqrt{N_i}$. This gives

$$\mathbb{E}[R] = \sum_{i=1}^{C} \mathbb{E}[R_i] \leq \sum_{i=1}^{C} 2\sqrt{N_i} \leq 2\sqrt{\sum_{i=1}^{C} N_i} = 2\sqrt{NC},$$

where the last inequality follows from the concaveness of the square root. □

VI. Hierarchical Heavy Hitters

In this section, we suggest two implementations of Hierarchical Heavy Hitters (HHH) on programmable switches. HHH is a generalization of the heavy hitter / top-k problems in which the goal is to identify *subnets* that send an excessive amount of traffic. HHH is motivated by the need to identify the attackers in a distributed denial of service (DDoS) attacks. Intuitively, the attack has access to many devices, each of which only sends a moderate amount of traffic, eliminating detection by standard top-k solutions. If the malicious devices share a common subnet (or a small number of subnets), HHH algorithms are able to identify them by considering the aggregated traffic that originates from each network. The full definition of the HHH problem is complex and appears in [17], [18].

Here, we present two possible solutions for finding HHH with PRECISION with different accuracy-resources tradeoffs. The *Independent Stacking* suggestion is based on the MST algorithm [39] and requires running multiple instances of PRECISION in parallel. Each such instance is updated in parallel with one of the prefixes of the current packet as illustrated in Figure 3(a). For example, in the common use case of source hierarchies in byte granularity, we are required to monitor (i) the total number of packets (i.e., /0 sub-network), (ii) the number of packets from each sub-network of size 8 bits, (iii) from each sub-networks of size 16 bits, (iv) from each sub-network of size 24 bits, and (v) sub-networks of size 32 bits. Clearly (i), and (ii) can be accurately counted using one and 256 counters, which requires three parallel instances of PRECISION for (iii)-(v). Such a suggestion is efficient in a pipeline architecture, as the independent PRECISION instances can be stacked together without requiring additional stages. However, Independent Stacking does not scale very well for larger hierarchies. Further, some architectures may limit the amount of stacking. Thus, our second suggestion is focused on implementing HHH using fewer hardware resources.

The basic idea of our second suggestion is to use RHHH as a model. We use a single PRECISION instance to monitor all prefix lengths, as illustrated in Figure 3(b). We add a pipeline stage that counts the total number of packets and then pick a prefix uniformly at random. The number for possible prefix lengths is 4 (for byte-level) or 32 (for bit-level), which are both power of 2 and easy to sample from. We update the single PRECISION instance in the same way as PRECISION does. The controller then receives the heavy hitters of the unified instance, separate them by prefix types and calculate the HHH list in the same manner as the previous work [6]. The work of RHHH showed that randomization works upon having a large number of packets. This is a reasonable assumption to make under our setup with high-throughput programmable switches. Further, the second suggestion requires only a single instance of PRECISION (with more allocated memory), therefore is applicable whenever PRECISION can work.

VII. Implementation and Evaluation

This section presents an evaluation of PRECISION's accuracy and adaptation mechanisms. We implement PRECISION in 800 lines of P4 code on an Barefoot Tofino Wedge-100 programmable switch, with $d = 2$ stages each tracking 64k flows, saving a total of 128k heavy hitter flows (defined as source-destination IP pairs). The implementation used 15% of header metadata memory and 20% of total register memory available to save flow IDs and counters. It computed $d = 2$ hash functions, less than 10% of totally available.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Our PRECISION prototype was deployed in Princeton University's campus network to report heavy flows to network operators, and has correctly reported the flows with empirically largest volume. The prototype processes mirrored traffic from campus Internet border, and implements recirculation using ingress pipeline resubmit.

We also run PRECISION on a Python-based simulator, as simulation allows us to choose parameters freely and independently manipulate each hardware restriction. We start by studying the effect of each hardware-friendly adaptation on PRECISION's accuracy. Next, we compare PRECISION to related work, including HashPipe [51], as well as Space-Saving [38] and RAP [5] that are not directly implementable on programmable switches. We obtain the code of HashPipe from its authors, and run it on a Java-based simulator.

For evaluating frequency estimation, we then measure the *Mean Square Error (MSE)* of the algorithm, i.e.,

$$MSE(Alg) \triangleq \frac{1}{N}\sum_{t=1}^{N}(\widehat{f_s} - f_s)^2.$$

We judge the quality of the top-k based on the standard *Recall* metric that measures how many top flows it identifies. Specifically, denoting the kth largest flow's frequency by F_k, when the algorithm outputs a flow set $\mathbf{C}$, quality using:

$$\text{Recall}(\mathbf{C}) \triangleq \quad |e \in \mathbf{C} : f_e \geq F_k|/k.$$

Our evaluation utilizes the following datasets:

CAIDA: The CAIDA Anonymized Internet Trace 2016 [3] (in short, *CAIDA*). Data is collected from the Equinix-Chicago backbone link with a mix of UDP, TCP, and ICMP packets.

UWISC-DC: A data center measurement trace recorded at the University of Wisconsin [8].

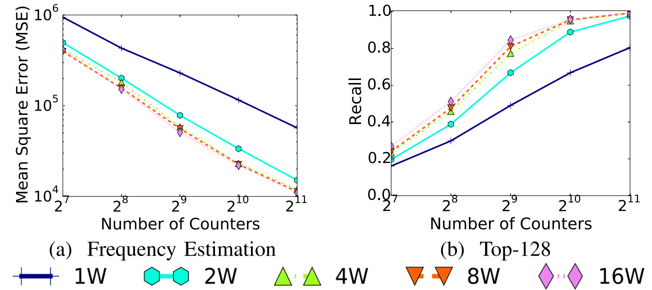
UCLA: The University of California, Los Angeles Computer Science department packet trace (denoted *UCLA*) [30].

We truncate each trace to its first 2 million packets, and use packets' Source-Destination IP address pair as their flow ID. In general, the CAIDA trace is heavy tailed, while the UWISC-DC trace and the UCLA traces are skewed.
We also tested our algorithm using synthetic trace with Zipf distribution and observed similar results.

All experiments were performed using a software emulated version of PRECISION, and we repeated each experiment 10 times with different random hash functions. Unless specified otherwise, the default associativity for PRECISION is 2-way.

A. Limited Associativity

We start with the frequency estimation problem and measure OnArrival error. In this measurement, we evaluate PRECISION with a varying number of ways (d) and use the same amount of total memory for all trials. Our results in Figure 4a show that for this problem 1-way associativity ($d = 1$, abbreviated as 1W) is a bit too low, but 2-way is already reasonable and further increasing d has diminishing returns. Figure 4b evaluates how d affects the Recall in top-k problem, using 512 counters to find top-128 flows. In this metric, we see that associativity is more important than in frequency



(a) Frequency Estimation (b) Top-128

Fig. 4. Effect of limited associativity on the frequency estimation error and top-k recall, on CAIDA trace. Using $d = 2$-way is a right balance between achieving good accuracy and saving pipeline stages usage.

estimation. $d = 2$ requires up to $2\times$ more counters than $d = 16$ to achieve the same recall. Changing to smaller or larger k yields similar observation.

We conclude that limited associativity incurs minimal accuracy loss in frequency estimation and is more noticeable in top-k. Our suggestion is to use $d = 2$ as it achieves the right balance between accuracy and the number of pipeline stages.

B. Entry Update Delay

We now evaluate the impact of update delay between the decision to recirculate and the actual flow entry update. Instead of using empirical evidence on one particular programmable switch, we simulate various possible delay values in terms of pipeline length. Figure 5a shows results for the MSE (Mean Square Error) in the frequency estimation problem and Figure 5b shows the Recall in top-k problem when trying to find the top-128 flows. As can be observed, the lines are almost indistinguishable. That is, update delay has a minor impact on accuracy for both metrics, even for a delay of 100 packets. We assume that practical switching pipelines would have shorter recirculation delays, as today's programmable switches have much less than 100 stages. A possible reason for this insensitivity to update delays is that replacing flow entries is already a rare and random event. Thus, the actual replacement time barely affects the accuracy even if it slightly deviates from the decision time.

C. Initial Value

We now evaluate the impact of having an initial value larger than zero set to all counters. Intuitively, the initial value limits the number of recirculated packets, but also requires some time to converge. This is because having a non-zero initial value means that we need to see more unmatched packets before we claim an entry — even if that entry is empty. Figure 6a show results for the frequency estimation metric. As can be observed, the initial value does affect the accuracy, and the effect is small until initial value 100, but initial value 1,000 causes a large impact. A similar picture can be observed in Figure 6b that evaluates Recall in the top-128 problem using 512 counters. As depicted, initial value also has a little impact up to 100, but an initial value of 1,000 results in a poor Recall.

Figure 6c completes the picture by showing the change of the Recall over time when trying to find top-128. As shown, the convergence time is inversely correlated with the initial

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

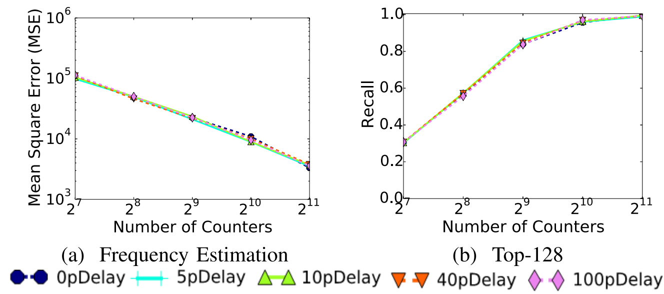


Fig. 5. Effect of the delayed update on the frequency estimation error and top-k Recall, on CAIDA trace. Even a delay of 100 packets has minimal impact on the accuracy.

value. In most cases, 1 million packets are enough for converging with an initial value of 100. We observed similar behavior for different packet traces. It appears that an initial value of 1,000 requires more packets to converge.

We conclude that a small initial value has a limited impact on the performance when the measurement is long enough. To facilitate quick convergence, we suggest an initial value of 100 (and use it in the following experiments), as it seems reasonable to upper bound recirculation to at most 1% of the packets, and the convergence time is shorter than 1 million packets, which translates to less than 10 milliseconds on fully-loaded 100 Gbps links.

D. Approximating the Desired Recirculation Probability

We now evaluate the impact of only using random bits as random source. This limits us to approximate the ideal recirculation probability $\frac{1}{carry_min+1}$ with a probability of the form 2^{-x} or $2^{-y} \times \frac{1}{\lceil T \rceil}$. Figure 7 shows results for frequency estimation problem (a) and (b), and for the top-k problem (c) and (d). We evaluated four variants: "NoAdaptation" is the algorithm without any hardware-friendly adaptation beyond limited associativity; "2-Approximate" is the variant added with an approximate recirculation probability of 2^{-x} form; "PRECISION (2-Approximate)" is the standard PRECISION algorithm with all other hardware-friendly adaptations also added; and "9/8-Approximate PRECISION" is the PRECISION algorithm using the $2^{-y} \times \frac{1}{\lceil T \rceil}$ form of approximate recirculation probability.

For frequency estimation, the 2-approximation in recirculation probability increases the error noticeably (in both workloads) possibly due to counters are always bumped to the next power of 2 when replacing a flow entry, causing some overestimation. Meanwhile, using the 9/8-approximation is almost as accurate as having no restriction on the recirculation probability.

For the top-k problem, we continue with our ongoing evaluation of how many counters are needed to identify the top-32 flows. Notice that recirculation probabilities are less impactful in this metric and in both workloads we need $\approx 2\times$ as many counters as NoAdaptation to achieve the same Recall.

It is surprising at first to notice that approximating the recirculation probability has a minimal performance impact in the UWISC-DC trace for the top-k metric. The reason is the highly-concentrated nature of this trace. In such workload where heavy hitters dominate, the sizes of tail flows are too small compared with the large counters maintained for heavy hitters, thus the tail flows have little chance to evict heavy hitters regardless of how we approximate probability.

E. Comparison With Other Algorithms

Next, we evaluate PRECISION with $d = 2$ and compare it with Space-Saving [38], and HashPipe [51] with $d = 2, 4, 6$ associativity. Similarly, we also compare with a 2-way set associative RAP [5]. Note that RAP was originally designed with a less restrictive programming model, and PRECISION adapts it to the RMT architecture.

Figure 8 shows results for the frequency estimation and top-k problems on the CAIDA (a), UCLA (b), and UWISC-DC (c) traces. Figures 8(a)-(c) shows that, for the frequency estimation problem, 2-way RAP and Space-Saving are the most accurate algorithm. They are followed by (2W-)PRECISION, which is orders of magnitude more accurate than 2W-HashPipe. PRECISION also has better performance than 4W- and 6W-HashPipe. We note that PRECISION also improves using higher associativity, as shown in Figure 4. Thus, we conclude the frequency estimation evaluation by saying that PRECISION is a dramatic improvement over HashPipe and is not much worse than the state-of-the-art algorithms despite its restricted programming model.

Figures 8(d)-(f) show the Recall performance for the top-k problem. In our top-32 setup, we see similar trends in all the traces, in which the best Recall is achieved by the 2-way RAP algorithm followed by PRECISION and Space-Saving. The algorithm with the lowest Recall is HashPipe, especially for $d = 2$-way. We see that PRECISION is on par with Space-Saving and not far behind 2-way RAP. PRECISION yields similar performance in all traces and requires at most $2\times$ more space than RAP or Space-Saving. Compared to 2W-HashPipe it requires up to $8\times$ less space for the same Recall. PRECISION also improves over 4W- and 6W-HashPipe by up to an $4\times$ factor.

F. Hierarchical Heavy Hitters

Next we show results for our Independent Stacking, and Randomized Prefix Selection algorithms. Recall that Independent Stacking implements the MST algorithm [39] where each Space Saving instance is replaced by PRECISION. Thus, as PRECISION's accuracy is similar to that of Space Saving we use it as a baseline. That is, the accuracy of Randomized Prefix Selection can only be as good as the baseline. In Figure 9 we show results for these options, where each PRECISION instance is configured with the default parameters (4-way, initial value of 100, delay of 10 packets, $9/8$-approximation of the sampling probability). The figure shows the obtained accuracy for different prefix lengths when varying the number of counters. As can be observed both algorithms obtain similar accuracy and indeed the light-weight Randomized Prefix Selection is slightly worse. However, the differences are small and the two algorithms are comparable for each prefix size.

To illustrate the attrativeness of our solution, we add a comparison to the state of the art software HHH algorithm,

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

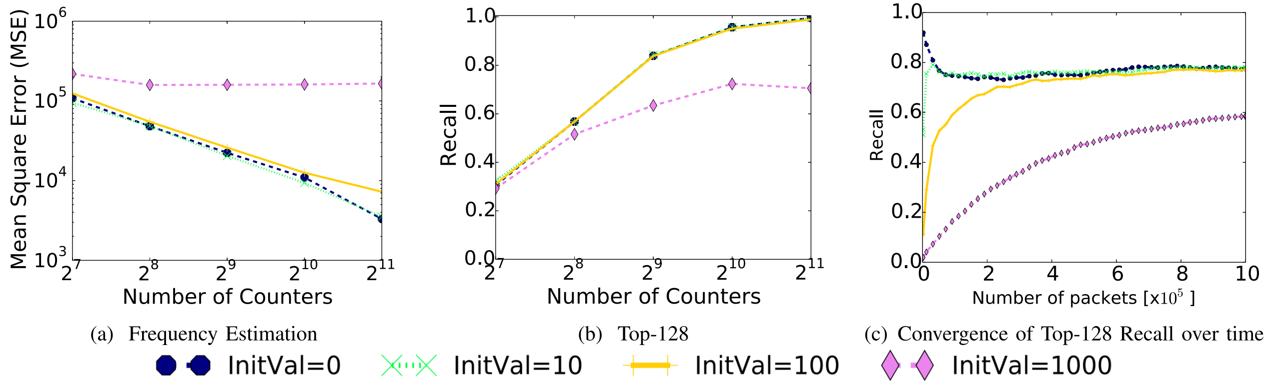


Fig. 6. Effect of initial value on the overall frequency estimation error and top-k recall, on CAIDA trace. An initial value of 100 leads to fast convergence and does not hurt accuracy, while upper-bounding recirculation to 1%.

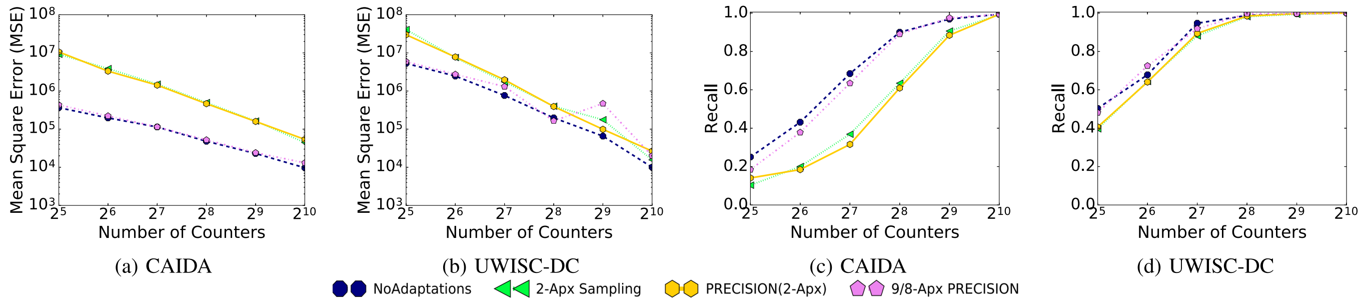


Fig. 7. The effect of approximating the recirculation probabilities on the accuracy for frequency estimation and top-32.

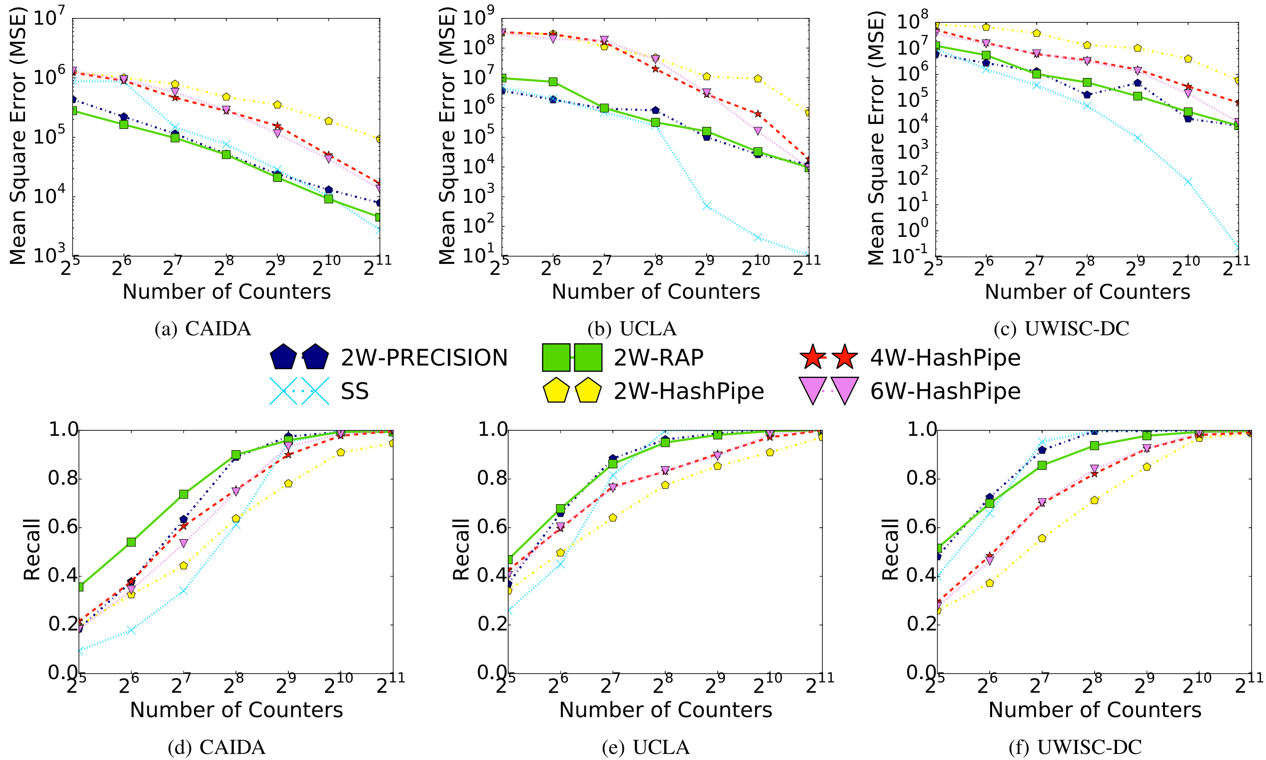


Fig. 8. Comparative evaluation of the frequency estimation and top-32 problems.

Randomized HHH (RHHH) [6]. As shown, our solutions are not far behind in terms of accuracy and it is usually enough to use PRECISION with double the space for getting comparable, or better, results. The exception is for the 8-bit networks, where RHHH gets significantly better accuracy. However, one can avoid using approximation algorithms altogether for these and count each of them separately using only $2^8 = 256$ counters.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

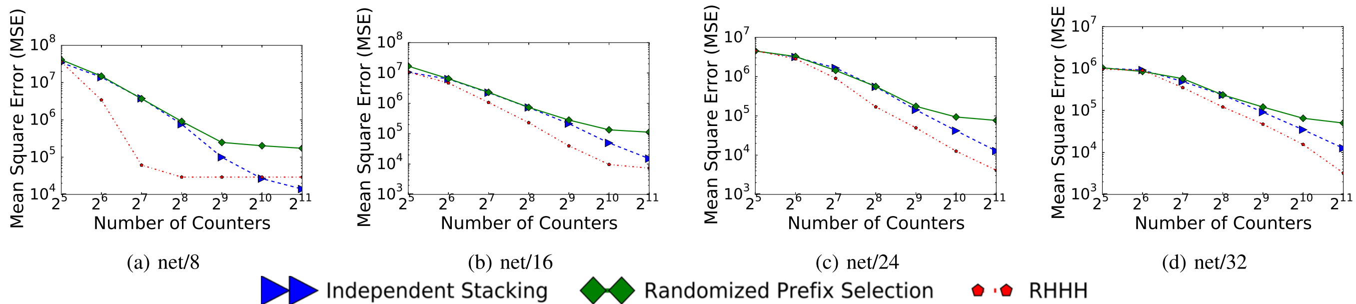


Fig. 9. The error of the "Independent Stacking" and "Randomized Prefix Selection" implementations of HHH-PRECISION for a given number of counters. We also compare with the state of the art *software* solution, RHHH [6].

VIII. Conclusions

This paper outlined the programming capabilities of the recently-developed RMT high-performance programmable switch architecture, and abstracted its programming restrictions into three easy-to-understand rules. Following these rules, we designed a novel heavy hitter detection algorithm, PRECISION, that can be compiled to the newly released Tbps-scale Barefoot Tofino programmable switch. PRECISION recirculates a small fraction of the packets for a second pipeline traversal, inducing a small (e.g., 1%) throughput overhead in order to follow the programming restrictions. We studied the impact of each RMT architectural restriction on our algorithms' accuracy. We concluded that the most severe impact comes from the lack of access to an unrestricted random integer generator, and specifically build a better approximation technique to mitigate the impact. We also present a deterministic variant of PRECISION, and further generalized PRECISION to solve the Hierarchical Heavy Hitters problem.

We performed extensive evaluation using real and synthetic packet traces, and demonstrated PRECISION is up to 100× more accurate when estimating per-flow frequency, or saves up to 8× memory space when identifying the top-128 flows, compared with HashPipe [51], a recently suggested alternative for programmable switches.

PRECISION enables heavy-hitter measurements at Tbps-scale aggregated throughput on today's high-performance programmable switches, at competitive accuracy compared to the state-of-the-art algorithms. Furthermore, we hope that our detailed case study of adapting a measurement algorithm to the RMT architecture would provide useful insights for implementing various other algorithms on such switches.

References

[1] *Tofino*. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/

[2] *Precision Implementation*. [Online]. Available: https://github.com/p4lang/p4-applications/tree/master/research_projects/PRECISION

[3] C. Walsworth, E. Aben, K. Claffy, and D. Andersen, "The CAIDA UCSD anonymized Internet traces," Center Appl. Internet Data Anal., La Jolla, CA, USA, Tech. Rep., Feb. 2015.

[4] Arista Networks. *Arista 7050X Switch Architecture ('A Day in the Life of a Packet')*. Accessed: Sep. 1, 2018. [Online]. Available: https://www.corporatearmor.com/documents/Arista_7050X_Switch_Architecture_Datasheet.pdf

[5] R. Ben Basat, X. Chen, G. Einziger, R. Friedman, and Y. Kassner, "Randomized admission policy for efficient top-k, frequency, and volume estimation," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1432–1445, Aug. 2019.

[6] R. Ben Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 127–140.

[7] R. B. Basat, G. Einziger, S. L. Feibish, J. Moraney, and D. Raz, "Network-wide routing-oblivious heavy hitters," in *Proc. Symp. Archit. Netw. Commun. Syst. (ANCS)*, 2018, pp. 66–73.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 267–280.

[9] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. ACM CoNEXT*, 2011, pp. 1–12.

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

[11] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, 2013.

[12] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. Int. Colloq. Automata, Lang., Program.*, 2002, pp. 693–703.

[13] M. Chen and S. Chen, "Counter tree: A scalable counter architecture for per-flow traffic measurement," in *Proc. IEEE ICNP*, Nov. 2015, pp. 111–122.

[14] S. Chole *et al.*, "dRMT: Disaggregated programmable switching," in *Proc. ACM SIGCOMM*, 2017, pp. 1–14.

[15] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *VLDB*, vol. 1, no. 2, pp. 1530–1541, 2008.

[16] G. Cormode and M. Hadjieleftheriou, "Methods for finding frequent items in data streams," *J. VLDB*, vol. 19, pp. 9–20, Feb. 2010.

[17] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *Proc. VLDB*, 2003, pp. 464–475.

[18] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *ACM SIGMOD*, 2004, pp. 155–166.

[19] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[20] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1701–1725, 3rd Quart., 2017.

[21] G. Dittmann and A. Herkersdorf, "Network processor load balancing for high-speed links," in *Proc. SPECTS*, Sep. 2002, pp. 1–9.

[22] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better NetFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, p. 245, Oct. 2004.

[23] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun. (SIGCOMM)*, 2003, pp. 137–148.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[24] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, p. 323, Oct. 2002.

[25] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, nos. 1–2, pp. 18–28, Feb. 2009.

[26] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in Proc. ACM SIGCOMM, 2018, pp. 357–371.

[27] N. Homem and J. P. Carvalho, "Finding top-K elements in data streams," *Inf. Sci.*, vol. 180, no. 24, pp. 4958–4974, Dec. 2010.

[28] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers," in *Proc. 18th IEEE Symp. High Perform. Interconnects*, Aug. 2010, pp. 58–65.

[29] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, no. 1, pp. 51–55, Mar. 2003.

[30] *Laboratory For Advanced Systems Research, UCLA*. [Online]. Available: http://www.lasr.cs.ucla.edu/ddos/traces/

[31] P. Laffranchini, L. Rodrigues, M. Canini, and B. Krishnamurthy, "Measurements as first-class artifacts," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 415–423.

[32] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1622–1634, Oct. 2012.

[33] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *Proc. USENIX NSDI*, 2016, pp. 311–324.

[34] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. ACM SIGCOMM*, 2019, pp. 334–350.

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

[36] N. Manerikar and T. Palpanas, "Frequent items in streaming data: An experimental evaluation of the state-of-the-art," *Data Knowl. Eng.*, vol. 68, no. 4, pp. 415–430, Apr. 2009.

[37] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," *Proc. VLDB Endowment*, vol. 5, no. 12, p. 1699, Aug. 2012.

[38] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proc. ICDT*, 2005, pp. 398–412.

[39] M. Mitzenmacher, T. Steinke, and J. Thaler, "Hierarchical heavy hitters with the space saving algorithm," in *Proc. Meeting Algorithm Eng. Exp. (ALENEX)*, 2012, pp. 160–174.

[40] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: Dynamic resource allocation for software-defined measurement," *ACM SIGCOMM*, 2014, pp. 419–430.

[41] B. Mukherjee, L. Heberlein, and K. Levitt, "Network intrusion detection," *IEEE Netw.*, vol. 8, no. 3, pp. 26–41, May/Jun. 1994.

[42] S. Narayana *et al.*, "Language-directed hardware design for network performance monitoring," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 85–98.

[43] K. Nyalkalkar, S. Sinhay, M. Bailey, and F. Jahanian, "A comparative study of two network-based anomaly detection methods," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 176–180.

[44] O. Polychroniou and K. A. Ross, "High throughput heavy hitter aggregation for modern SIMD processors," in *Proc. 9th Int. Workshop Data Manage. New Hardw. (DaMoN)*, 2013, pp. 1–6.

[45] D. A. Popescu, G. Antichi, and A. W. Moore, "Enabling fast hierarchical heavy hitter detection using programmable data planes," in *Proc. Symp. SDN Res. (SOSR)*, 2017, pp. 191–192.

[46] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, p. 261, Jun. 2003.

[47] R. Schweller *et al.*, "Reversible sketches: Enabling monitoring and analysis over high-speed data streams," *IEEE/ACM Trans. Netw.*, vol. 15, no. 5, pp. 1059–1072, Oct. 2007.

[48] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system," in *Proc. USENIX ATC*, 2006, pp. 171–184.

[49] D. Shah, S. Iyer, B. Prahhakar, and N. McKeown, "Maintaining statistics counters in router line cards," *IEEE Micro*, vol. 22, no. 1, pp. 76–81, Aug. 2002.

[50] A. Sivaraman *et al.*, "Packet transactions: High-level programming for line-rate switches," *Proc. ACM SIGCOMM*, 2016, pp. 15–28.

[51] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res. - SOSR*, 2017.

[52] B. Stephens, A. Akella, and M. M. Swift, "Your programmable NIC should be a programmable switch," in *ACM HotNets Workshop*, 2018, pp. 36–42.

[53] D. Tong and V. Prasanna, "Online heavy hitter detector on FPGA," in *Proc. IEEE Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, Dec. 2013, pp. 1–6.

[54] D. Tong and V. Prasanna, "High throughput sketch based online heavy hitter detection on FPGA," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 4, pp. 70–75, Apr. 2016.

[55] B. Turkovic, J. Oostenbrink, and F. A. Kuipers, "Detecting heavy hitters in the data-plane," *CoRR*, vol. abs/1902.06993, Feb. 2019.

Ran Ben Basat received the B.Sc. *(summa cum laude)*, M.Sc. *(cum laude)*, and Ph.D. degrees from the Computer Science Department, Technion. He is currently a Post-Doctoral Fellow with Harvard University, where he is involving in network monitoring and algorithms. He is partially sponsored by the Zuckerman and the Hiroshi Fujiwara Cyber Security Research Center Post-Doctoral Fellowships.

Xiaoqi Chen received the bachelor's degree from the Institute for Interdisciplinary Information Sciences, Tsinghua University, in 2013. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Princeton University, NJ, USA.

Gil Einziger received the B.Sc. and Ph.D. degrees in computer science from Technion. He worked as a Researcher with Nokia Bell Labs and a Post-Doctoral Research Fellow with the Polytechnic University of Turin, Italy. He is currently an Assistant Professor with the Department of Computer Science, Ben Gurion University of the Negev, Beer Sheva, Israel. His research interests include networked systems, algorithms, and security.

Ori Rottenstreich received the B.Sc. degree *(summa cum laude)* in computer engineering and the Ph.D. degree from Technion, Haifa, Israel, in 2008 and 2014, respectively. From 2015 to 2017, he was a Post-Doctoral Research Fellow with Princeton University. He is currently an Assistant Professor with the Department of Computer Science and the Department of Electrical Engineering, Technion.

Authorized licensed use limited to: Princeton University. Downloaded on April 18,2020 at 00:31:09 UTC from IEEE Xplore. Restrictions apply.