SCREAM: Sketch Resource Allocation for Software-defined Measurement

Masoud Moshref† Minlan Yu† Ramesh Govindan† Amin Vahdat*
†University of Southern California *Google

ABSTRACT

Software-defined networks can enable a variety of concurrent, dynamically instantiated, measurement tasks, that provide fine-grain visibility into network traffic. Recently, there have been many proposals for using sketches for network measurement. However, sketches in hardware switches use constrained resources such as SRAM memory, and the accuracy of measurement tasks is a function of the resources devoted to them on each switch. This paper presents SCREAM, a system for allocating resources to sketch-based measurement tasks that ensures a user-specified minimum accuracy. SCREAM estimates the instantaneous accuracy of tasks so as to dynamically adapt the allocated resources for each task. Thus, by finding the right amount of resources for each task on each switch and correctly merging sketches at the controller, SCREAM can multiplex resources among network-wide measurement tasks. Simulations with three measurement tasks (heavy hitter, hierarchical heavy hitter, and super source/destination detection) show that SCREAM can support more measurement tasks with higher accuracy than existing approaches.

CCS Concepts

•**Networks** → **Network resources allocation; Network monitoring; Data center networks;** *Programmable networks;*

Keywords

Software-defined Measurement; Sketches; Resource Allocation

1. INTRODUCTION

Traffic measurement plays an important role in network management. For example, traffic accounting, traffic engineering, load balancing and performance diagnosis require measuring traffic on multiple switches in the network [6, 11]. Software-defined Measurement (SDM) [42, 36] facilitates controller-directed network-wide measurement: with SDM, operators or tenants can submit measurement tasks to the SDN controller, and the SDN controller configures switches to monitor traffic for each task, then collects statistics and produces measurement reports.

A recent prior work in software-defined measurement [36] has relied on flow-based counters. These counters are often implemented using TCAM memory, which is expensive and power hungry. Moreover, flow-based counters are limited to supporting volume-based measurement tasks such as heavy hitter detection and often require a large number of counters. For example, a switch may need to count traffic from thousands of source IP addresses to find heavy users of a specific service, for each of which it would require a counter. To reduce counter usage, many solutions rely on counting traffic to/from prefixes (instead of specific IP addresses), and then iteratively zooming in and out to find the right set of flows to monitor [35, 43, 27]. Such prefix-based summarization has two drawbacks: it cannot be applied to many tasks such as flow-size distribution and entropy calculation, and it can take multiple measurement epochs to reconfigure counters (e.g., to zoom into 32 levels in the IP prefix tree) [35].

In contrast, this paper focuses on hash-based counters, or *sketches* [42]. Sketches are summaries of streaming data for approximately answering a specific set of queries. They can be easily implemented with SRAM memory which is cheaper and more power-efficient than TCAMs. Sketches can use sub-linear memory space to answer many measurement tasks such as finding heavy hitters [17], super-spreaders [18], large changes [29], flow-size distribution [30], flow quantiles [17], and flow-size entropy [32]. Finally, they can capture the right set of flow properties in the data plane without any iterative reconfiguration from the controller.

Any design for sketch-based SDM faces two related challenges. First, SDM permits multiple instances of measurement tasks, of different types and defined on different aggregates, to execute concurrently in a network. Furthermore, in a cloud setting, each tenant can issue distinct measurement tasks within its own virtual network.

The second challenge is that sketch-based measurement tasks may require significant resources. To achieve a re-

CoNEXT'15 December 01–04, 2015, Heidelberg, Germany
 ISBN 978-1-4503-3412-9/15/12…$15.00
DOI: http://dx.doi.org/10.1145/2716281.2836106

quired accuracy, each task may need up to a million counters, and the number of counters is bounded by resources such as the SRAM memory needed for saving sketch counters, the control datapath inside switches required to report counters from ASIC to CPU, and the control network bandwidth that is shared among many switches.

Therefore, an SDM design must ensure efficient usage of these resources. For many forms of sketches, it is possible to estimate the resources required to achieve a desired accuracy (i.e., there is a *resource-accuracy* trade-off). These resource estimates are also dependent on traffic. Prior work [42] has assumed *worst-case* traffic in allocating resources to sketches, and this can result in pessimistic overall resource usage, reducing the number of tasks that can be concurrently supported. In contrast, our key idea is to use a *dynamic* resource allocator that gives just enough resources to each task for the *traffic* it observes, and dynamically adapts the resource allocation as traffic changes over time and across switches.

We propose a sketch-based SDM system, called SCREAM (SketCh REsource Allocation for Measurement), which enables *dynamic* resource allocation of limited resources for many concurrent measurement tasks while achieving the required accuracy for each task. Our paper makes two contributions: *(1) Sketch-based task implementation across multiple switches:* Each task type implementation must gather sketch counters from *multiple* switches and prepare measurement results to the user. As switches see different traffic, each sketch may need *different* sizes for an efficient and accurate measurement. SCREAM uses novel techniques to merge sketches with *different* sizes from multiple switches. This extension of sketch-based measurement [42] to multiple switches is a critical step towards making sketches useful in practice. *(2) Accuracy estimator:* SCREAM incorporates a new method to estimate accuracy for measurement tasks on multiple switches without ground-truth or an a priori knowledge of traffic model with low estimation errors, rather than rely on the worst-case bounds of sketches. SCREAM feeds these *instantaneous accuracy estimates* (which can also give operators some insight into how trustworthy the measurements are) into a dynamic resource allocation algorithm [36] to support more accurate tasks by leveraging *temporal and spatial statistical multiplexing*.

We have implemented three measurement task types (heavy hitter, hierarchical heavy hitter and super source/destination) in SCREAM and improved their design for dynamic resource allocation on multiple switches. Our simulations demonstrate that SCREAM performs significantly better than other allocation alternatives. Compared to OpenSketch, which allocates resources on a single switch based on the worst-case bounds, SCREAM can support $2\times$ more tasks with higher accuracy. This result is valid across all task types and even when applying OpenSketch on multiple switches. This is because SCREAM can leverage traffic variations over time to multiplex resources across task instances and switches while OpenSketch reserves the same fixed resources for all tasks of the same type. SCREAM can support the same number of tasks with comparable accuracy as an oracle which is aware of future task resource requirements.

Finally, SCREAM builds upon our prior work on DREAM [36], which establishes a framework for dynamic resource allocation for TCAM-based measurement tasks. SCREAM deliberately preserves many of the elements of DREAM (Section 3), to permit a unified system that multiplexes resources across different types of measurement tasks.

2. BACKGROUND AND MOTIVATION

Sketch-based SDM. Sketches are memory-efficient summaries of streaming data for approximately answering a specific set of queries. Sketches often provide a provable trade-off between resources and accuracy, where the definition of accuracy depends on the queries. We focus on hash-based sketches because they can be implemented on hardware switches using commodity components (hashing, TCAM, and SRAM modules) as discussed in OpenSketch [42]. Note that the same accuracy estimation and similar resource allocation technique can be applied to software switches where cache for counters and CPU budgets per packet are limited. We leave software switches to future work but note that measurement in software switches or hypervisors does not extend to wide-area networks across datacenters, networks where operators do not have access to end hosts, and networks which devote most server resources to revenue-generating applications.

A commonly used sketch, the Count-Min sketch [17] can approximate volume of traffic from each item (e.g. source IP) and is used for many measurement tasks such as heavy hitter detection (e.g., source IPs sending traffic more than a threshold). Count-Min sketch keeps a two dimensional array, A, of integer counters with w columns and d rows. For each packet from an input item $x \in (0 \ldots D)$ with size I_x, the switch computes d pairwise independent hash functions and updates counters, $A[i,h_i(x)] += I_x, i \in (1 \ldots d)$. At the end of measurement epoch, the controller fetches all counters. When the controller queries the sketch for the size of an item, Count-Min sketch hashes the item again and reports the minimum of the corresponding counters. As the controller cannot query every item (e.g., every IP address), we need to limit the set of items to query. We can keep a sketch for each level of prefix tree (at most 32 sketches) and avoid querying lower levels of the tree by using the result of queries on upper levels (Section 4). Multiple items may collide on a counter and cause an over-approximation error, but Count-Min sketch provides a provable bound on the error. Using d hash functions each mapping to w entries bounds the worst-case error to: $e_{cm} \leq e\frac{T}{w}$ with probability $1-e^{-d}$, where T is the sum of packet sizes. Approaches to improve Count-Min sketch accuracy, for example by running the least-squares method [31] or multiple rounds of approximation over all detected prefixes [33], add more computation overhead to the controller, and their resource-accuracy trade-off is not known in advance.

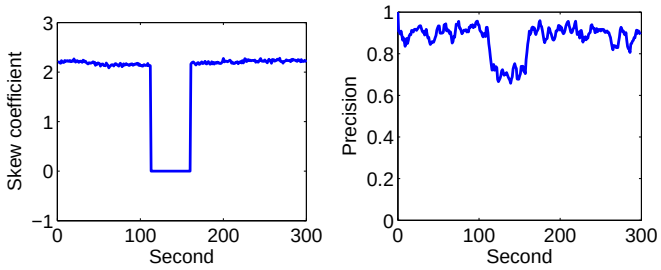
Many sketches have been proposed for counting distinct items [24, 23]. We use HyperLogLog [24] as its space usage is near-optimal, and it is easier to implement than the optimal algorithm [26]. First, we hash each item and count the num-

ber of leading zeros in the result, say 0_x. Intuitively, hash values with more leading zeros indicate more distinct items. By only keeping the count of maximum leading zeros seen over a stream, $M = max_i(0_{x_i})$, we can estimate the number of distinct items as 2^{M+1}. For this, a 5-bit counter is enough for a 32-bit hash function. We can replicate this sketch m times, and reduce the relative error of approximation with a factor of $\sqrt{m}$ but with no additional hashing overhead, by using the first p bits of hash output to select from $m = 2^p$ replicas and the other bits to update the replica counter. For example, a distinct counter with $m = 64$ replicas will require 320 bits, have a standard deviation of the relative error of $\frac{1.04}{8}$, and will use the first 6 bits of hash outputs to select a replica.

Why is sketch-based SDM resource constrained? SDM permits multiple instances of measurement tasks execute concurrently in a network which together require a lot of resources. These tasks can be of different types and defined on different traffic aggregates. For example, an operator may run different types of tasks in a (virtual) network, such as finding large flows for multi-path routing [6] and finding sources that make many connections for anomaly detection [41]. Operators may also instantiate tasks dynamically on different aggregates to drill down into anomalous traffic aggregates. Furthermore, in a cloud setting, each tenant can issue distinct measurement tasks for its virtual network; Amazon CloudWatch already offers simple per-tenant measurement services [1], and Google Andromeda allows SDN-based network functionality virtualization for tenants [40]. Besides, modern clouds service a large number of tenants (3 million domains used AWS in 2013 [2]), so SDM with many measurement tasks will be common in future clouds.

However, switches have limited memory and bandwidth resources for storing these counters. Today's switches have 128 MB SRAM capacity (HP 5120-48G EI [4]) which can support 4-128 tasks where each task needs 1-32 MB SRAM counters [42]. In practice, SRAM is used for other functions and only a small part of it is available for measurement. Moreover, there is limited bandwidth for the controller to fetch counters. First, inside a switch the control data-path that transfers counters from the ASIC to the switch CPU has low bandwidth (e.g., 80 Mbps) [20]. Second, there is limited bandwidth to send counters from many switches to the controller. For example, 12 switches each dumping 80 Mb per second can easily fill a 1 Gbps link. Thus, we need sketches with fewer counters to reduce memory usage, lower network overhead and send counters more frequently to the controller to report in a short time-scale.

Why dynamic allocation? Prior work [42] has proposed tuning the size of sketches based on their resource-accuracy trade-off at *task instantiation* to maintain a required accuracy. However, these resource-accuracy trade-offs are for the *worst-case*. For example, based on the formulation of Count-Min sketch, to not detect items sending less than 9 Mbps for a threshold of 10 Mbps in a 10 Gbps link ($e_{cm} = 1$ Mbps), we need about 27 K counters of 4 bytes for each row; with 3 rows and a prefix tree with 32 levels, this works out to



(a) Skew **(b) Sketch accuracy**

Figure 1: Variation of traffic skew and sketch accuracy over time

5.5 MB[1]. However, the total traffic T of a link may not always reach the link capacity. In addition, Cormode [19] showed that the bound is loose for skewed traffic (a not uncommon case in real world) and the sketch can be exponentially smaller when sized for known skew. For the above example, if the link utilization is 20% in average with a skew factor of 1.3 [19], each row will need only 1040 counters which require 260 KB of SRAM. Other sketches also exhibit traffic-dependent accuracy [14, 30].

These trade-off formulations are loose because the optimal resource requirement of a sketch for a given accuracy depends on the *traffic* that changes over time. For example, Figure 1a shows the skew of traffic from source IPs in CAIDA trace [3] over time. (Our skew metric is the *slope* of a fitted line on the log-log diagram of traffic volume from IPs vs. their rank (ZipF exponent).) The skew decreases from time 110 to 160 because of a DDoS attack. Figure 1b shows the accuracy of heavy hitter (HH) source IP detection of Count-Min sketch with 64 KB memory over time. Heavy hitter detection accuracy, *precision* (the fraction of detected true HHs over detected ones), decreases from 90% to 70% for less skewed traffic, which means that the sketch needs more counters *only* at this time period in order to maintain 90% accuracy. This presents an opportunity to *statistically multiplex* SRAM and bandwidth resources across tasks on a single switch by dynamically adjusting the size of sketch.

Besides, we may need sketches with different sizes on different switches for tasks that monitor traffic on multiple switches. For example, we may need to find heavy hitter source IPs on flows coming from two switches (say A and B). These switches monitor different traffic with different properties such as skew, thus they need different number of counters. This allows *spatial statistical multiplexing*: A sketch may need more counters on switch A vs. B while another may need more on switch B vs. A.

3. SCREAM OVERVIEW

SCREAM provides sketch-based software-defined measurement with limited resources (Figure 2). It allows users

[1] The number of hash functions, d, is usually fixed to 3 or 4 to simplify hardware and because of reduced marginal gains for larger values. The total is smaller than $27K \times 4 \times 3 \times 32$ because the sketches for the top layers of the prefix tree can be smaller [15].

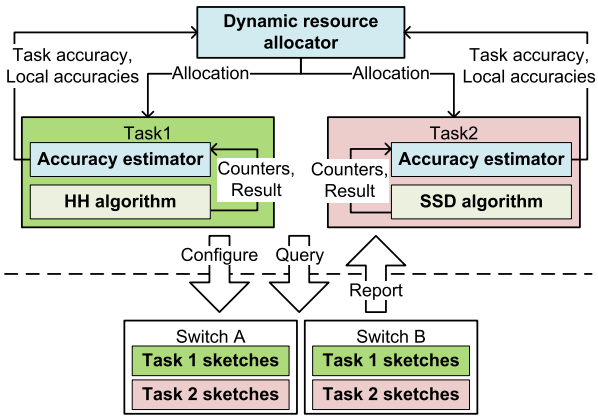


Figure 2: Resource allocation overview

to dynamically instantiate measurement tasks and specify a required accuracy. Specifically, the user instantiates a task by specifying its type, flow filter, its parameters and its accuracy bound. For example, she may instantiate a heavy hitter detection task on a five tuple flow filter <srcIP=10/8, dstIP=16.8/16,*,*,*> that reports sources sending traffic more than 10 Mbps with a precision (the fraction of detected items that are true heavy hitter) of at least 80%.

SCREAM can run multiple concurrent instances of different task types. Each task instance (henceforth, simply task) configures counters at switches and periodically queries counters from switches. Periodically, SCREAM distributes resources to each task on each switch based on the traffic observed at the switch to satisfy its requirement. As a result, tasks update the sketch parameters based on allocated resources and re-configure their counters at switches. In the following, we describe the two components of SCREAM (tasks and dynamic resource allocation).

Tasks: Sketches can support a diverse range of measurement tasks [42]. We consider three examples in this paper:

Heavy Hitter (HH): A heavy hitter is a traffic aggregate identified by a packet header field that exceeds a specified volume. For example, heavy hitter detection on source IP finds source IPs contributing large volumes of traffic.

Hierarchical Heavy Hitter (HHH): Hierarchical heavy hitters (HHHs), used for detecting DDoS [38], are defined by the longest prefixes that exceed a certain threshold, θ, in aggregate traffic volume even after *excluding* any HHH descendants in the prefix tree [16]. For example, if a prefix 10.1.0.0/16 has traffic volume more than θ, but all the subnets within the prefix have traffic volume less than θ, we call the prefix a HHH. In contrast, if one of its subnets, say 10.1.1.0/24, has traffic more than θ, but the rest of the IPs collectively do not have traffic more than θ, we view 10.1.1.0/24 as a HHH, but 10.1.0.0/16 is not a HHH.

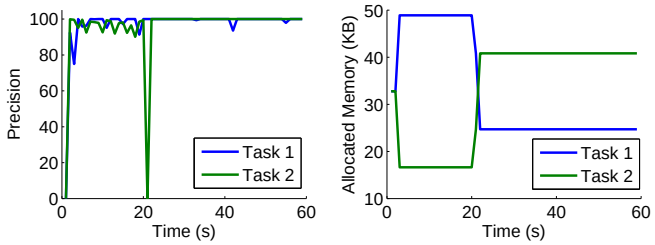
Super source and destination (SSD): A super source is a source IP that communicates with a more than a threshold number of *distinct* destination IP/port pairs. A super destination is defined in a similar way. SSDs are used for detecting worms, port-scans, P2P super nodes or DDoS targets.

Dynamic resource allocation: At the heart of SCREAM is its resource allocation mechanism. We use the *instantaneous accuracy* of a task as its feedback for an iterative allocation algorithm (Figure 2). Each task periodically shares its result and counters with an accuracy estimator module. The accuracy estimator estimates the accuracy of current results for resource allocation algorithm which in turn determines the number of counters for each sketch based on those estimates. Then tasks tune sketch parameters based on the number of assigned counters and re-configure switches. Thus, the resource allocation mechanism requires two components, a dynamic resource allocator and an accuracy estimator per task type.

SCREAM's resource allocator, borrowed from DREAM [36] (see below), requires tasks to provide a global task accuracy and local accuracies per switch, and runs parallel per-switch resource allocators that use the maximum of global and local accuracies as follows. If the accuracy estimate is below the specified accuracy bound ("poor" task), it receives more resources; these resources are taken away from "rich" tasks whose accuracy estimate is well above their bound. The allocator algorithm achieves fast but stable convergence by dynamically adapting the step size of resource exchange for each task. It also contains an admission control algorithm that rejects new tasks when necessary: without this, no task may receive sufficient resources. However, since resource demands for a task can change over time, resource overloading can occur even without the arrival of new tasks. In this case, the allocator assigns resources to poor tasks based on assigned priorities, and when it cannot, it may drop tasks.

We now illustrate how the resource allocator works using a simple example (Figure 3). We ran an experiment on two heavy hitter (HH) detection tasks on source IP using Count-Min sketch on a single switch. Each task monitors a chunk of a packet trace [3] starting from time 0. Figure 3a shows the estimated accuracy in terms of *precision* (the fraction of detected true HHs over detected ones) of tasks over time and Figure 3b shows the allocated resources per task. In the beginning, both tasks get equal resources (32 KB), but task 1 cannot reach the 80% accuracy bound at time 3 while task 2 has very high accuracy. Therefore, the resource allocator takes memory resources (16 KB) from task 2 and gives to task 1. At time 20, we increase the skew of volume of traffic from source IPs for task 1 and decrease it for task 2. As a result, task 2 requires more resources to reach 80% accuracy bound, thus its estimated accuracy degrades at time 20. The resource allocator responds to the accuracy decrease by iteratively allocating more resources to task 2 (first 8 KB then 16 KB) until it exceeds the bound.

An alternative approach to design an allocation mechanism would have been to find and quantify the effect of different traffic properties on the resource-accuracy trade-off of each task, and run parallel measurement tasks to find the value of traffic properties (e.g., skew, which can be used to tighten the accuracy bound for Count-Min sketch [28]). However, quantifying the effect of traffic properties (e.g., skew parameters) on the accuracy is complicated, and dynamically estimating them may require significant resources [28].



(a) Estimated precision **(b) Allocation**

Figure 3: Resource allocation example

Relationship to DREAM [36]. SCREAM is inspired by DREAM, our prior work on efficient and dynamic resource management for TCAM-based measurement tasks. In particular, SCREAM *deliberately* reuses the dynamic resource allocator (described above) proposed in DREAM [36]. This reuse has the advantage that it can enable a unified framework to support a variety of measurement tasks that uses either sketches or counters. Such a framework can simplify the configuration of measurement tasks and provide a unified interface for specifying these tasks and processing the results. Our next step in the near future is to develop such a unified framework.

However, SCREAM is different from DREAM in several ways because it is sketch-based instead of TCAM-based. First, by using sketches, SCREAM can support tasks that cannot be supported using TCAM-based counters. For example, in this paper, we implement the SSD task using a Count-Min sketch and a distinct counter instead of a volume counter (we leverage an existing technique [42, 18] for this, but adapt the design to provide unbiased results). Also, unlike flow-based counters, sketches do not need iterative reconfiguration. The iterative re-configuration is less accurate for time-varying traffic because it takes multiple measurement epochs to reconfigure counters (e.g., to zoom into 32 levels in IP prefix tree) [35].

Second, in SCREAM, different switches may be assigned different-sized sketches by the dynamic allocator because they see differing amounts of traffic. Combining different-sized sketches is non-trivial, and we present an approach to merge the counters from sketches of different sizes, together with a general algorithm to use Count-Min sketch in a prefix tree to run the three measurement task types (Section 4).

Finally, the primary enabler for dynamic resource allocation in SCREAM is the estimation of instantaneous accuracy of a task. We present a solution that does not assume an *a priori* traffic model or run parallel measurement tasks for each of the implemented task types (Section 5).

Generality: In this paper, we build three measurement tasks in SCREAM, which cover various important applications in data centers and ISP networks such as multi-path routing [6], optical switching [10], network provisioning [22], threshold-based accounting [21], anomaly detection [27], worm detection [41], P2P seed server detection [8], port scan [8] and DDoS detection [38].

Moreover, although we have implemented measurement tasks based on Count-Min sketch and its variants, SCREAM

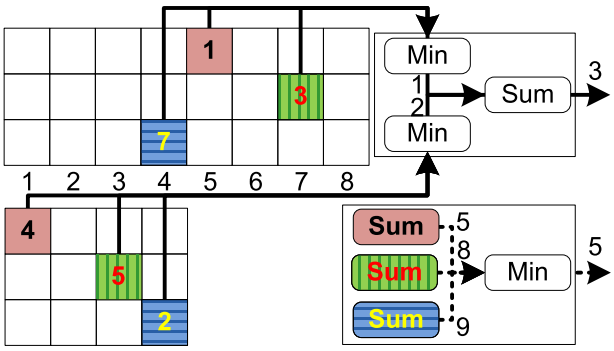


Figure 4: Merging two Count-Min sketches with different sizes

can support other sketches for a different resource-accuracy trade-off or for other measurement tasks. If a given sketch's accuracy depends on traffic properties, it also benefits from our dynamic resource allocation algorithm. For example, the error of Count-Sketch [9], that can also support our three tasks, depends on the variation of traffic (compared to the error of Count-Min sketch which depends on the size of traffic). However, its theoretical bound is still loose for a skewed distribution [14]. Kumar [30] proposed a sketch to compute the flow size distribution, but the accuracy of this sketch also depends on the traffic properties (number of flows). In order to add those sketches to SCREAM, we need to estimate their accuracy, which have left to future work.

4. DESIGN OF SKETCH-BASED TASKS

In SCREAM, tasks at the controller configure sketch counters at switches, fetch counters and prepare reports. In order to prepare reports, tasks need to find instances of HHs, HHHs, or SSDs. In this section, we first describe how to approximate traffic counts for HHs and HHHs, and connection counts for SSDs using Count-Min and HyperLogLog sketches on multiple switches. These algorithms execute at the controller, and are specific to a given task type. Then, we describe an algorithm independent of task type that, from the derived counts, estimates and reports instances of HHs, HHHs and SSDs that exceed the specified threshold.

Although there have been many sketch-based algorithms [42], we improve upon them in the following ways. We introduce novel techniques to merge sketches with different sizes from multiple switches, leverage hierarchical grouping with adjustable overhead to find instances of HHs, HHHs and SSDs, and adapt the design of the SSD task to be unbiased and provide stable accuracy. We describe these improvements below.

Heavy Hitter (HH): If a prefix has traffic on one switch, we estimate traffic size by the minimum of counts from different rows of the counter array in the Count-Min sketch approximation algorithm (Section 2). However, a heavy hitter prefix may have traffic from multiple switches. One approach [5] in this case is to simply sum up the Count-Min sketch arrays fetched from different switches into a single array ($A_{new} = \sum_s A_s$ for each switch s) and run the algorithms as if there is only one sketch. However, in SCREAM, the

resource allocator sizes the sketches at each switch differently, so each sketch may have an array of different widths and cannot be summed. For example, Figure 4 shows the counter arrays for two Count-Min sketches with three rows and different widths that cannot be directly summed.

A natural extension for sketches of different sizes is to find the corresponding counter for each prefix at each row and sum the counters at similar rows across sketches. The approximated count will be their minimum: $min_i(\sum_s A_s[i,h_i(x)])$. For example, say an item on the first sketch maps to counters with index 5, 7, and 4, and on the second sketch maps to 1, 3, and 4. The approximation will be: $min(A_1[1,5]+A_2[1,1]$, $A_1[2,7]+A_2[2,3]$, $A_1[3,4]+A_2[3,4])$. In Figure 4 (right bottom) we connect counters with the same color/pattern to the corresponding sum boxes to get 5 as the final result.

However, because Count-Min sketch always over-approximates due to hash collisions, we can formulate a method that generates smaller, thus more accurate, approximations. The idea is to take the minimum of corresponding counters of a prefix inside each sketch and then sum the minima: $\sum_s min_i(A_s[i,h_i(x)])$. For the above example, this will be $min(A_1[1,5],A_1[2,7],A_1[3,4])+min(A_2[1,1],A_2[2,3],A_2[3,4])$ (Figure 4, the top merging module with solid lines to counter arrays which approximates the size as 3 instead of 5). This approximation is always more accurate because the sum of minimums is always smaller than minimum of sums for positive numbers. In all of this, we assume that each flow is monitored only on one switch (e.g., at the source ToR switches).

Hierarchical Heavy Hitter (HHH): Recall that HHHs are defined by the longest prefixes exceeding a certain threshold in aggregate volume after *excluding* any HHH descendants in the prefix tree. Figure 5 shows an example of a prefix tree for four bits. With a threshold of $\theta = 10Mb$, prefix 010* is a HHH as IPs 0100 and 0101 collectively have large traffic, but prefix 01** is not a HHH because excluding descendent HHHs (010* and 0111), its traffic is less than the threshold.

We leverage multiple sketches to find HHHs [16]. We need a sketch for each layer of the prefix tree to estimate the size of prefixes at different levels. For a HHH without any descendant HHH, the approximation function works the same as HH detection task. However, for other HHHs, we need to exclude the size of descendant HHHs. Thus, during a bottom up traversal on the tree, SCREAM tracks the total size of descendant HHHs already detected and subtracts that size from the approximation for the current prefix [16].

Super Source/Destination (SSD): SSD detection needs to count distinct items instead of the volume of traffic, so we replace each counter in the Count-Min sketch array with a distinct counter [18]. Therefore, each sketch has $w \times d$ distinct counters; we used the HyperLogLog [24] distinct counter because its space usage is near-optimal and it is easy to implement [26]. However, distinct counters may under-approximate or over-approximate with same probability, so picking the minimum can cause under-approximation and result in many missing items even with a large Count-Min array. For example, suppose that there is no collision in a Count-Min sketch, we have d distinct counters for a source

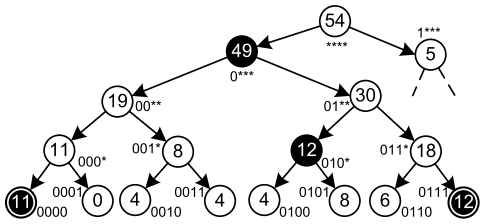
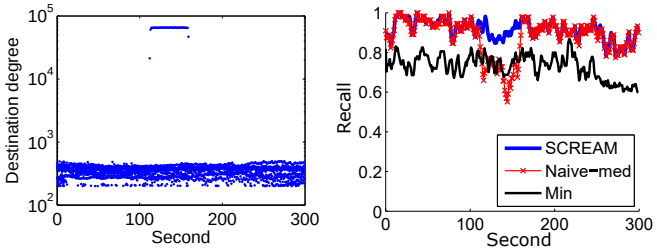


Figure 5: A prefix trie of source IPs where the number on each node shows the bandwidth used by the associated IP prefix in Mb in an epoch. With threshold 10, the nodes in double circles are heavy hitters and the nodes with shaded background are hierarchical heavy hitters.

IP, and we pick the minimum, it is more likely to pick the one that under-approximates. Figure 6b shows the recall (detected fraction of true SSDs) of SSD detection given fixed resources in simulation over a CAIDA traffic trace [3]. The under-approximation of picking the minimum resulted in missing more than 20% of SSDs. Unfortunately this will become worse for larger Count-Min sketches with fewer collisions. Thus, the SSD approximation, even for a single sketch, cannot use the minimum of corresponding counters.

To counter this bias, unlike prior work [18, 42, 25], we pick the median instead of minimum. Then, to remove the median's bias towards Count-Min hash collisions, we remove the average error of the sketch from it in Equation 1 where A is the Count-Min sketch array of width w and T is the sum of distinct items of prefixes. Equation 1 is unbiased since we can interpret the over/under-approximations of distinct counters as random positive and negative updates on the sketch and use the proof in [29].

$$\frac{median_i(A[i,h(x)]) - T/w}{1-1/w} \quad (1)$$



(a) Destination degree **(b) Recall**

Figure 6: Unbiasing detection of destination IPs contacted by > 200 source IPs on Count-Min sketch ($w = 1320, d = 3$) plus HyperLogLog ($m = 16$)

However, when the number of sources per destination IP is highly skewed (e.g., in a DDoS attack) removing the average error (T/w) in Equation 1 can result in missing SSDs. For example, Figure 6a shows the degree of super destinations over time in the ground-truth where a specific destination has a large number of distinct sources from time 110 to 160. Figure 6b shows that the recall for an approach that uses Equation 1, named Naive-med, drops during the DDoS attack. These missed SSDs result from the fact that Equation 1 compensates for the average Count-Min collision from every counter, but for skewed traffic a few large items that

```
1 Function approximate (prefix, sketches)
2     for i = 1...d do
3         M_new,* = 0
4         for s in sketches do
5             DC_s,i = distinct counter in A_s[i, h(prefix)]
6             for k = 1...m do
7                 M_new,k = max(M_new,k, M_DC_s,i,k)
8         c_i^est = hyperloglog(M_new,*).approximate()
9     return unbias(median_i(c_i^est))
```

Figure 7: Approximate function for SSD

increase average error significantly only collide with a few counters. Thus, reducing this large error from every median approximation causes an under-approximation of the total count, and results in missing true SSDs.

Instead, we refine the average error estimate by removing those very large prefixes from it, but must first detect them using two steps. In the first step, we use the average error just to detect very large prefixes, set L (as mentioned before, this causes an under-approximation, but is still sufficient to detect very large SSDs.). In the second round, we reduce the adjusted average error, $\frac{T-\sum_{k\in L} c_k^{est}}{w}$, from the medians, where c_k^{est} is the estimated count for item k. This results in high recall, independent of traffic skew (Figure 6b). This does not come at the cost of increased false positives, and SCREAM's precision is also high.

Multiple switches: If a prefix has traffic from multiple sketches, summing the number of distinct items from different sketches over-approximates the number of distinct items because two distinct counters may have counted similar items. For example, two switches that forward traffic from a source IP prefix may see traffic to a common destination IP. However, the common destination IP should only be counted once in the degree of the source IP prefix. We can combine two HyperLogLog distinct counters, DC_1 and DC_2, with m replica counters by taking the maximum of each corresponding replica counter to make a new distinct counter [24]: $M_{new,k} = max(M_{DC_1,k}, M_{DC_2,k})$ for $k = 1 \ldots m$.

To leverage this, we keep a fixed number of replica in distinct counters of different sketches and only change the width of the array in Count-Min sketch (w) based on the allocated resources. Again, having Count-Min sketches with different widths, we cannot use the traditional approach of merging distinct counters with the same index in Count-Min counter array [25]. Instead, we find corresponding distinct counters for each specific query in each row and merge them.

Figure 7 summarizes how we use this idea to approximate the degree of each prefix when Count-Min sketches may have different widths. For each d rows of Count-Min sketches, we find the corresponding distinct counters for a prefix in each sketch (lines 2-5). Then, we merge replicas from these distinct counters (lines 6-7) and approximate the number of distinct items using the new replica counters similar to a single HyperLogLog sketch [24] (line 8). Now similar to the case of a single switch, we approximate the degree of the SSD using the unbiased median approach (line 9).

Reporting HHs, HHHs and SSDs: So far, we have presented ways of approximating traffic volumes and connection counts. However, we also need an efficient way of determining which IP prefixes contain HHs, HHHs or SSDs. In the data plane of each switch, SCREAM uses Count-Min sketches to count traffic. A single Count-Min sketch can only approximate the count given a prefix. Exploring all prefixes at the controller is impossible, so SCREAM uses a hierarchy of Count-Min sketches to identify the actual prefixes [15]. It employs a Count-Min sketch for each level of prefix tree (e.g., 16 sketches for a task with flow filter of 10.5/16), where the sketch on level l (from leaves) ignores l least significant IP bits [2]. Note that to find HH/SSD IP prefixes that are not exact, we can start the tree from a level > 0.

```
1 Function createReport (prefix, output)
2     e=approximate(prefix, prefix.sketches)
3     if e≥threshold then
4         foreach child of prefix do
5             createReport(child, output)
6         updateOutput(prefix, output)
```

Figure 8: Generic algorithm to create output

Figure 8 shows an algorithm that does not depend on the task type. In line 2, the algorithm `approximates` the size of a prefix tree node by combining multiple sketches (using algorithms described above). Then, it traverses the prefix tree (lines 3-6). If the approximation is above the threshold, it goes deeper to find items for output. This algorithm relies on the observation that if a prefix's size is not over the threshold, its ancestors sizes are not too.

For example in Figure 5, it starts from the root and goes in the left branches until it finds heavy hitter 0000, but later when it reaches prefix 001*, it does not need to check its children. The `updateOutput` function for HH/SSD detection is simply to add the prefix for a leaf node (path from the root in the prefix tree) to the output. However, for HHH detection, we only add the prefix into output if its size remains larger than threshold after gathering its descendant HHHs and excluding their size.

Many techniques are proposed to identify items to query (reverse a sketch) [17, 7, 37]. At the core, all use multiple sketches and apply group testing to reverse the sketch, but their grouping is different. We use hierarchical grouping [17] because it is enough for our tasks, is fast and simple and has tunable overhead comparing to some alternatives. For example, OpenSketch used Reversible sketch [37] with fixed high memory usage of 0.5 MB. Our work generalizes prior work that has used hierarchical grouping for HHs and HHHs, but not for SSDs [15, 17].

5. ACCURACY ESTIMATION

To support dynamic resource allocation, we need algorithms that can estimate the instantaneous accuracy for individual tasks, even when the traffic for a task spans multiple

[2] It is possible to have a sketch for each $g > 1$ levels of the tree but with more overhead at the controller to enumerate 2^g entries at each level. Our implementation is easily extendible for $g > 1$.

switches. In addition to informing resource allocation, our accuracy estimates can give operators some understanding of the robustness of the reports. Our accuracy estimators discussed in this section consider two accuracy metrics: *precision*, the fraction of retrieved items that are true positives; and *recall*, the fraction of true positives that are retrieved.

The key challenge is that we do not have an *a priori* model of traffic and it takes too much overhead to understand traffic characteristics by measuring traffic. Instead, our accuracy estimator only leverages the collected counters of the task. There are two key ideas in our accuracy estimator: (1) applying probabilistic bounds on *individual* counters of detected prefixes, and (2) tightening the bounds by *separating* the error due to large items from the error due to small items.

Heavy hitters: Count-Min sketch always over-approximates the volume of a prefix because of hash collisions; therefore, its recall is 1. We compute precision by averaging the probability that a detected HH j is a true HH, p_j. We start with the case where each HH has traffic from one switch and later expand it for multiple switches. The strawman solution is to estimate the probability that an item could remain a HH even after removing the collision error of *any* other item from its *minimum* counter. The resulting estimated accuracy under-estimates the accuracy by large error mainly because, for skewed traffic, a few large items make the probabilistic bound on the error loose since the few large items may only collide on a few counters. Our approach treats the counters in each row separately and only uses the probabilistic bound for the error of small undetected items.

A strawman for estimating p_j. p_j is the probability that the real volume of a detected HH is larger than the threshold θ, $p_j = P(c_j^{real} > \theta)$. In other words, an item is a true HH, if the estimated volume remains above the threshold even after removing the collision error. We can estimate the converse (when the collision error is larger than the difference between estimated volume and threshold (Equation 2) using the Markov inequality. To do this, we observe that each counter has an equal chance to match traffic of every item, so the average traffic on each counter of each row is $\frac{T}{w}$ (T is the total traffic, and w is the number of counters for each hash function) [17]. Using the Markov inequality, the probability that the collision exceeds $c_j^{est} - \theta$ is smaller than $\frac{T}{w(c_j^{est}-\theta)}$. However, since Count-Min sketch picks the minimum of d independent counters, the collisions of all counters must be above the bound. Putting this together, we get Equation 3:

$$P(c_j^{real} > \theta) = P(c_j^{est} - e_{cm} > \theta) = 1 - P(e_{cm} \geq c_j^{est} - \theta) \quad (2)$$

$$P(c_j^{real} > \theta) > 1 - (\frac{T}{w(c_j^{est} - \theta)})^d \quad (3)$$

Unfortunately, as Figure 9 shows, the resulting estimated precision is far from the actual precision, which leads to inefficient resource allocation. The reason is that, for skewed traffic, a few large items can significantly increase average error $\frac{T}{w}$, but only collide with a few counters.

Our solution: separate the collision of detected HHs on each counter. We can leverage individual counters of detected HHs in two ways to tighten the p_j estimation. First, instead of using the final estimated volume for a HH (c_j^{est}) that is the smallest in all rows, we use the individual counters for each hash function h_i separately ($c_{i,j}^{est}$) that can be larger and provide tighter bounds in Equation 4.

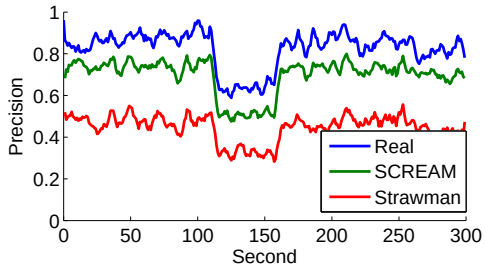


Figure 9: HH detection accuracy estimation for Count-Min sketch ($w = 340, d = 3$)

$$P(c_j^{real} > \theta) > 1 - \prod_{i=1}^{d} \frac{T}{w(c_{i,j}^{est} - \theta)} \quad (4)$$

Second, we know the counter indices for detected HHs and can find if they collide with each other. Therefore, we separate the collisions of detected HHs from collisions with other small items. Using this, we can lower the estimate for average collision traffic in the Markov inequality by removing the traffic of detected HHs, resulting in a tighter estimate[3]. We now describe the details of this technique.

There are two cases where a detected HH is not a real HH: (1) when detected HHs collide with each other; (2) when a detected HH does not collide with other detected HHs, but collides with multiple IPs with low counts, which together inflate the traffic count above the threshold. For case (1), we can easily check if a detected HH collides with other detected HHs by checking if the index of its counter is hit by another HH. If $B_{i,j}$ is the set of other HHs that collide on the ith counter of HH j, we just remove the estimated volume of those HHs from the counter by using $c_{i,j}^{est} - \sum_{k \in B_{i,j}} c_k^{est}$ instead of $c_{i,j}^{est}$. The estimated volume of HHs in set $B_{i,j}$ may be an over-approximation and removing them from $c_{i,j}^{est}$ makes our p_j estimate conservative. For case (2), we use the Markov inequality to bound the collision of undetected small items. However, instead of T, now we should use the traffic of only undetected small items. Let A be the set of detected HHs whose estimation is not affected by other HHs (no hit on minimum counter). Replacing T with $T - \sum_{k \in A} c_k^{real}$ in Equation 4, we can estimate p_j in Equation 5. However, we do not know $c_{k \in A}^{real}$ because of the over-approximations of counts in the sketch. Thus, as an estimate, we use $c_{k \in A}^{est}$ after reducing the average collision error of only small items from it. Figure 9 shows that our estimation based on Equation 5 is close to the real precision, even under traffic dynamics. In Section 6.4, we have validated that this improvement applies across all the traces we have used in our evaluations, and that

[3] Cormode [19] also used this technique to find a resource-accuracy trade-off for Count-Min sketch assuming the skew of traffic is known, but our goal is to estimate p_j for each HH without assuming a model of traffic.

this improvement is essential for SCREAM.

$$P(c_j^{real} > \theta) > 1 - \prod_{i=1}^{d} \frac{T - \sum_{k \in A} c_k^{real}}{w(c_{i,j}^{est} - \sum_{k \in B_{i,j}} c_k^{est} - \theta)} \quad (5)$$

Multiple switches: As described in Section 3, our resource allocator estimates a global accuracy for the task, as well as a per-switch local accuracy [36]. It uses these to add/remove resources from switches. Similar to the single switch case, we compute the global precision by finding p_j for each detected HH.

Markov's inequality is too loose when a HH has traffic from a set of switches, so the single-switch accuracy estimator does not work well. The reason is that the network-wide collision (a random variable) is the sum of collisions at individual switches (sum of random variables) [12]. However, since the collision on a sketch is independent from the collision on another, we can replace Markov's bound with Chernoff's bound [12] to get a more accurate estimation of p_j (see Appendix A.). We still use the Markov's inequality to estimate precision if a HH has traffic from a single switch.

Once we calculate p_j, we compute local accuracies by attributing the estimated precision p_j to each switch. If a given HH j has traffic at a single switch, the p_j is only used for the local accuracy of that switch. Otherwise, we attribute precision proportionally to each switch based on its average error as, intuitively, the switch that has smaller average error compared to others must have higher precision.

Hierarchical heavy hitters: If a detected HHH has no descendant HHH (e.g., 0000, 010*, 0111 in Figure 5), its p_j can be easily calculated using the Markov or Chernoff bound. However, if a detected HHH has descendant HHHs, we cannot just apply those equations to c_j^{est} (volume excluding descendant HHHs) as its p_j depends on the p_j of descendent HHHs, because even if the sketch approximated the volume of a HHH accurately, the over-approximation of the descendant HHHs can make it a false HHH. For example in Figure 5, if we detected 0000, 010*, and 0111 as HHHs and over-approximated only the volume of 010* as 17, the weight for 0*** excluding descendant HHHs will be $49-40=9$ and will not be detected. Instead, we detect **** as a HHH with volume $54-40=14$. In this scenario, although we may have approximated the volume of **** correctly, it will be incorrectly detected as a HHH. Thus, we need to find if the sum of over-approximations in a set of descendants could make a true descendant HHH below j and avoid j to become a true HHH.

Instead, SCREAM uses a simpler but conservative approach. First, we notice that in the worst case, the over-approximated traffic has been excluded from one of children of the detected HHH. For each child prefix, we check if these over-approximations could make it a HHH. If any child with a new volume becomes HHH, the parent cannot be, so as a heuristic, we halve p_j. Second, we find a conservative bound for the over-approximations of *each* descendant HHH and add them up instead of going through the probability distribution of the sum of over-approximations. The over-approximation error bound, say $\hat{e}_{D(j)}^{cm}$, for each descendant HHH of j, $D(j)$, is the upper bound on its error, $e_{D(j)}^{cm}$: $P(e_{D(j)}^{cm} < \hat{e}_{D(j)}^{cm}) > 0.1$. [4] We find this upper bound using Markov's inequality for HHHs originated from a single switch and Chernoff's bound otherwise. For example, Equation 6 derived from Equation 5 shows the maximum error that a descendant HHH at a single switch at level l can have while keeping $p_j \geq 0.1$.

$$e_{D(j)}^{cm} \leq \frac{T - \sum_{k \in A_l} c_k^{real}}{w \sqrt[d]{0.9}} \quad (6)$$

Multiple switches: For the global accuracy, we just replace the Markov inequalities in HH tasks and Equation 6 with Chernoff's bound. Finding the local accuracy on each switch is similar to HH with one difference: when the p_j of a HHH decreases because of its descendants, we need to consider from which switch the data for descendants come and assign lower accuracy to them. So in these cases, we also consider the average error of sketches per descendant for each switch and attribute the accuracy proportionally across switches.

Finally, we have found in our experiments (Section 6) with realistic traffic that, for HHH, recall is correlated with precision. Our intuition is that because the total size of detected HHHs is smaller than T [16] and no non-exact HHH prefix can have a size $\geq 2\theta$ [36], detecting a wrong HHH (low precision) will also be at the cost of missing a true HHH (low recall).

Super source or destination: The p_j of a SSD depends on both the distinct counter error (e^{dc}) and hash collisions (e^{cm}) because their errors add up [25]. For a false positive SSD, j, that has counter $c_{i,j}^{est}$ for ith hash function, the error, $e_{i,j}^{dc} + e_{i,j}^{cm}$ must have been greater than $c_{i,j}^{est} - \theta'$ where θ' is computed based on the threshold θ and our version of Equation 1 (see Appendix B.). If the SSD has $d' \leq d$ such counters (remember we choose median instead of minimum), p_j is computed using Equation 7. We can compute p_j, based on the *individual* error distributions of Count-Min sketch and the distinct counter (see Appendix B.). The error of HyperLogLog sketch has the Gaussian distribution with mean zero and relative standard deviation of $1.04/\sqrt{m}$ when it has m replica counters [24]. The collision error because of Count-Min sketch is also bounded using Markov inequality as before.

$$P(c_j^{real} > \theta') = 1 - \prod_{i=1}^{d'} P(e_{i,j}^{dc} + e_{i,j}^{cm} \geq c_{i,j}^{est} - \theta') \quad (7)$$

In contrast to HH tasks, the recall of SSD is not 1 because the distinct counters can under-approximate. However, the probability of missing a true SSD can be calculated based on the error of the distinct counter [25]. The error of HyperLogLog distinct counter depends on the number of its replica counters, and we can configure it based on the user requirement just at the task instantiation.

Multiple switches: We merged distinct counters on different switches into one distinct counter for each row of Count-Min sketch. Thus, for SSDs, accuracy estimation on mul-

[4] In practice, we found 0.1 a reasonable value.

tiple switches is the same as one switch. To compute local accuracies, we use the average error of sketches from different switches to attribute the computed global accuracy, p_j, proportionally across switches.

6. EVALUATION

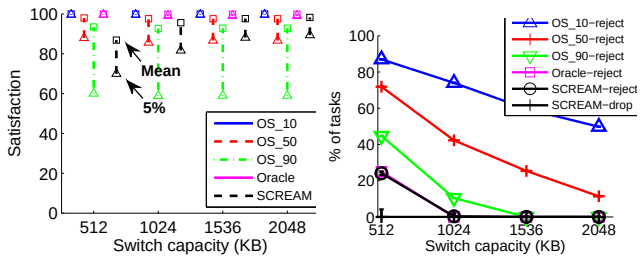
In this section, we use simulations driven by realistic traffic traces to show that SCREAM performs significantly better than OpenSketch, and is comparable to an oracle both on a single switch and on multiple switches.

6.1 Evaluation setting

Simulator: Our event-based simulator runs sketches on 8 switches and reports to the controller every second. Tasks at the controller generate task reports and estimate accuracy, and the resource allocator re-assigns resources among tasks every second. The reject and drop parameters of the resource allocator are set the same as DREAM [36]. The resource allocator is scalable to more switches, and the average number of switches that a task has traffic from is the dominating factor for controller overhead [36]. Therefore, we make each task to have traffic from all 8 switches and put the evaluation for more switches for future work.

Tasks and traffic: Our workload consists of three types of tasks: HHs, HHHs and SSDs. In a span of 20 minutes, 256 tasks with randomly selected types appear according to a Poisson process. The threshold for HH and HHH tasks is 8 Mbps and the threshold for SSD tasks is 200 sources per destination IP. We choose 80% as the accuracy bound for all tasks since we have empirically observed that to be the point at which additional resources provide diminishing returns in accuracy. Each task runs for 5 minutes on a part of traffic specified by a random /12 prefix. We use a 2-hour CAIDA packet trace [3] from a 10 Gbps link with an average of 2 Gbps load. Tasks observe dynamically varying traffic as each task picks a /4 prefix of a 5-min chunk of trace and maps it to their /12 filter. Thus, our workload requires dynamic resource adjustment because of traffic properties variations and task arrival/departure. For scenarios with multiple switches, we assign /16 prefixes to each switch randomly and replay the traffic of a task that matches the prefix on that switch. This means that each task has traffic from all 8 switches. In a data center, SCREAM would monitor traffic on the source switches of traffic (the ToRs), thus network topology is irrelevant to our evaluation.

Evaluation metrics: The *satisfaction* rate is the percentage of task lifetime for which accuracy is above the bound. We show the average and 5^{th}% for this metric over all tasks. The 5^{th}% value of 60 means that 95% of tasks had an accuracy above the bound for more than 60% of their lifetime: it is important as a resource allocator must keep all tasks accurate, not just on average. The *drop* ratio shows the percentage of tasks that the SCREAM resource allocator drops to lower the load if it cannot satisfy accepted tasks, and the *rejection* ratio shows the ratio of tasks that had been rejected at instantiation in each algorithm. These metrics are important because



(a) Satisfaction **(b) Rejection and drop**

Figure 10: Comparison for OpenSketch (different relative error%) and the oracle at a single switch

a scheme can trivially satisfy tasks by rejecting or dropping a large fraction of them.

Comparison with OpenSketch: OpenSketch [42] allocates resources to a task based on worst-case traffic to reach a given relative error at a single switch. To bound the error to x% of the threshold θ on a HH/HHH detection task that has an average traffic of T, it configures a Count-Min sketch with $w = \frac{eT}{x\theta}$. For example, if a task has 128 Mbps traffic, a sketch with $w = 435$ and $d = 3$ can guarantee that the relative error is lower than 10% of the threshold $\theta = 1$ MB with probability $1 - e^{-3} = 0.95$. In our experiments, we fix the number of rows, d, to 3 and find w based on the error rate. For SSD detection, OpenSketch solves a linear optimization to find the best value for distinct counter parameter (m) and Count-Min sketch width (w) that minimizes the size of sketch [42].

At task arrival, OpenSketch finds the required amount of resources for the relative error guarantee and reserves its resources if there is enough free resources; otherwise, it rejects the task. We run OpenSketch with a range of relative errors to explore the trade-off between satisfaction and rejection. OpenSketch was originally proposed for the single-switch case, so for comparison on multiple switches, we propose an extension as follows. We run a separate OpenSketch allocator for each switch and if a task cannot get resources on any switch it will be rejected. However, setting the resources based on the worst case traffic for all switches would require too many resources as some tasks may have most of their traffic from a single switch. Therefore, given the total traffic T for a task, we configure the sketch at each switch based on the average traffic across switches ($T/8$).

Comparison with Oracle: We also evaluate an *oracle* that knows, at each instant, the exact resources required for each task in each switch. In contrast, SCREAM does not know the required resources, the traffic properties or even the error of accuracy estimates. We derive the oracle by actually executing the task, and determining the resources required to exceed the target accuracy empirically. Thus, the oracle always achieves 100% satisfaction and never drops a task. It may, however, reject tasks that might be dropped by SCREAM, since the latter does not have knowledge of the future.

6.2 Performance at a single switch

SCREAM supports more accurate tasks than OpenSketch.

Figure 10 compares SCREAM to OpenSketch with three different relative error percentages for a combination of different types of tasks over different switch memory sizes [5]. We ran the experiment 5 times with different task arrival patterns and show the error bars in Figure 10b which are very tight. Note that OpenSketch uses the worst-case guarantee and even a relative error of 90% of threshold can result in high satisfaction. SCREAM has higher satisfaction rate than OpenSketch with high relative error (90%), but its rejection rate is lower. SCREAM can support 2 times more tasks with comparable satisfaction (e.g., the curve 50% error on switch capacity of 1024 KB). Finally, OpenSketch needs to reject up to 80% of tasks in order to get much higher satisfaction than SCREAM (in 10% relative error curve).

SCREAM can match the oracle's satisfaction for switches with larger memory. For switches with larger memory, SCREAM can successfully find the resource required for each task and reach comparable satisfaction as the oracle while rejecting no tasks (Figure 10). For switches with smaller memory, SCREAM has similar rejection rate as the oracle, but its satisfaction rate is smaller than the oracle. The reason is that, in this case, SCREAM needs to dynamically adjust task resources over time more frequently than for larger switches and waits until tasks are dropped, during which times some tasks may not be satisfied.

6.3 Performance on multiple switches

Figures 11a and 12a show that SCREAM can keep all task types satisfied. However, OpenSketch either has high rejection for strict error guarantee that over-allocates (OS_10), or cannot keep tasks accurate for relaxed error guarantees that admit more tasks (OS_50, OS_90). Note that the satisfaction of OpenSketch for multiple switches is lower than its satisfaction on a single switch especially for 5^{th}% because OpenSketch uses the error bounds that treat every switch the same. (OpenSketch is not traffic aware, and cannot size resources at different switches to match the traffic). As a result, it either sets low allocation for a task on all switches and reaches low satisfaction or sets high allocation and wastes resources. However, SCREAM can find the resource requirement of a task on each switch and allocate just enough resources to reach the required accuracy. Other diagrams in Figures 11 and 12 also show the superiority of SCREAM over OpenSketch for each individual task type. Note that SSD tasks need more resources and OpenSketch with 10% relative error cannot run any task on the range of switch capacity we tested.

Like the single switch case, SCREAM can also achieve a satisfaction comparable to the oracle for multiple switches (Figure 11). In Figure 12, SCREAM has a lower rejection rate than oracle for small switches but still has no drop. This is because SCREAM does not know the future resource requirements of tasks, thus it admits them if it can take enough headroom of free resources from highly accurate tasks. But if later it cannot support them for *a few epochs*, it drops them. Thus, SCREAM can tolerate tasks to be less accurate for a few epoch and does not reject or drop them. However, the oracle is strict and rejects these tasks at task instantiation; hence, it has higher rejection.

SCREAM supports more accurate tasks than OpenSketch over different traffic traces. Above, we showed SCREAM's superior performance for tasks with different traffic traces. Now, we explore the effect of traffic skew by changing the volume of traffic from each source IP at each second: Recall that the frequency (denoted by volume) of elements (source IPs) of rank i in ZipF distribution with exponent z is defined by i^{-z}. Thus to change the ZipF exponent to $s \times z$, it is enough to raise to the power of s the traffic volume from each source IP in each measurement epoch. Note that we keep the total traffic volume the same by normalizing the traffic volume per source IP by the ratio of new total volume over old total volume. Figure 13 shows that SCREAM can keep tasks satisfied in a wide range of skew. For example, if we reduce the skew to 60%, the mean (5^{th}%) of satisfaction is 98% (92%) and no task is rejected or dropped. However, as OpenSketch considers the worst case irrespective of traffic properties, it either ends up with low satisfaction for less skewed traffic (OS_50, OS_90) or over-provisioning and rejecting many tasks (OS_10).

6.4 Accuracy estimation

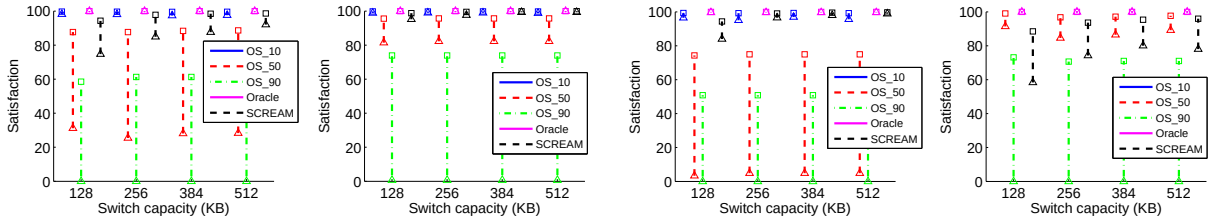
SCREAM's superior performance requires low accuracy estimation error. Our experiments show that our accuracy estimation has within 5% error on average. Although we define accuracy based on precision, SCREAM achieves high recall in most cases.

SCREAM accuracy estimation has low errors. We calculated the accuracy estimation error (the percentage difference of the estimated accuracy and the real accuracy) of tasks for the single switch and multiple switches cases. Figure 15 shows that SCREAM can estimate the accuracy of tasks with about 5% error on average. As an aside, using the strawman accuracy estimator for HH detection (Section 5), resulted in about 15%(40%) error in average (std) and forced SCREAM to reject or drop all tasks in this scenario.

The error of our accuracy estimator varies across different task types but goes down for switches with larger capacity. The reason is that the error of our accuracy estimators decreases for higher accuracies and with larger switches more and more tasks can reach higher accuracies. We found that the cases with high error in accuracy estimation usually only have very few detected items that are close to the threshold. For such items with small margin over the threshold, Markov inequality is loose, resulting in error in accuracy estimation.

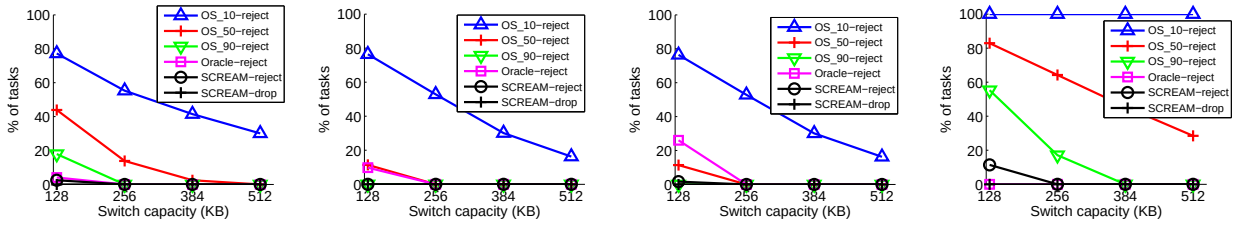
Tasks in SCREAM have high recall. Figure 14 shows that the satisfaction of HHH detection tasks based on recall is higher than that of OpenSketch. This is because the recall of HHH detection is correlated with its precision (see Section 5). Hence, the curves for satisfaction based on recall (Figure 14) are similar to the curves for satisfaction based on precision (Figure 11c). As mentioned in Section 5, the

[5] Switch memory size and network bandwidth overhead are linearly related and both depend on the number of counters in each sketch.



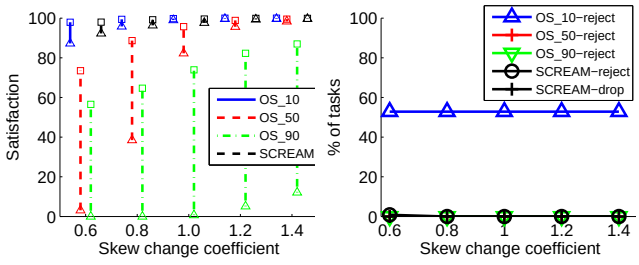
(a) Combine (b) HH (c) HHH (d) SSD

Figure 11: Satisfaction comparison for OpenSketch (different relative error%) and the oracle on multiple switches



(a) Combine (b) HH (c) HHH (d) SSD

Figure 12: Drop & reject comparison comparison for OpenSketch (different relative error%) and the oracle on multiple switches



(a) Satisfaction (b) Rejection and drop

Figure 13: Changing skew for HH detection at multiple switches with capacity 64 KB

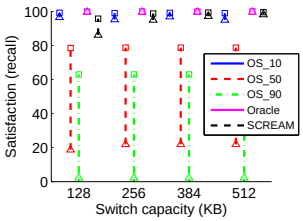
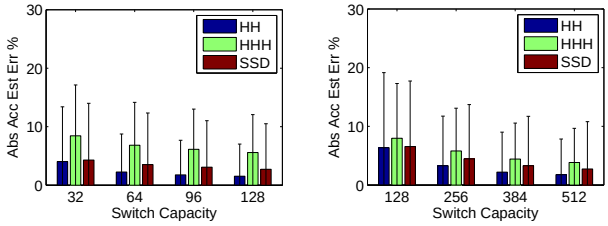


Figure 14: HHH satisfaction based on recall on multiple switches

recall of HH detection is always 1, and the recall for SSD detection depends on the number of replica counters in the distinct counter. In our experiments, the average recall was above 80% for all switch sizes.

7. RELATED WORK

Sketch-based measurement on individual tasks: There have been many works on leveraging sketches for individual measurement tasks. Some works propose sketch-based measurement solutions on a single switch for HH [15], HHH [16], and SSD [18]. Other works [5, 25] provide algorithms to run sketches on multiple switches with a fixed sketch size on each switch. Instead, SCREAM provides efficient resource



(a) Single switch (b) Multiple switches

Figure 15: Accuracy estimation error

allocation solutions for *multiple* measurement tasks on *multiple switches* with different sketch sizes at these switches.

Resource allocation for measurement tasks: Most resource allocation solutions focus on sampling-based measurement. CSAMP [39] uses consistent sampling to distribute flow measurement on multiple switches for a single measurement task and aims at maximizing the flow coverage. Volley [34] uses a sampling-based approach to monitor state changes in the network, with the goal of minimizing the number of sampling. Payless [13] decides the measurement frequency for concurrent measurement tasks to minimize the controller bandwidth usage, but does not provide any guarantee on accuracy or bound on switch resources.

OpenSketch [42] provides a generic data plane that can support many types of sketches with commodity switch components. It leverages the *worst-case* accuracy bounds of sketches to allocate resources on a *single* switch for measurement tasks at task instantiation. On the other hand, SCREAM dynamically allocates sketch resources on multiple switches by leveraging the instantaneous accuracy estimation of tasks, and thus can support more tasks with higher accuracy.

DREAM [36] focuses on flow-based counters in TCAM, and dynamically allocates TCAM resources to multiple measurement tasks to achieve their given accuracy bound. DREAM develops accuracy estimators for TCAM-based zoom-

in/out algorithms, and its paper's evaluations show that DREAM is better than simple *task-type agnostic* schemes such as equal TCAM allocation. In contrast, SCREAM explores the accuracy estimation for sketch-based tasks, where the sketch counters are not accurate compared to TCAM counters because of random hash collisions. We show that SCREAM supports 2 times more accurate tasks than a *task-type aware* allocation, OpenSketch [42], and has comparable performance as an oracle that knows future task requirements.

8. CONCLUSION

Sketches are a promising technology for network measurement because they require lower resources and cost with higher accuracy compared to flow-based counters. To support sketches in Software-defined Measurement, we design and implement SCREAM, a system that dynamically allocates resources to many sketch-based measurement tasks and ensures a user-specified minimum accuracy. SCREAM estimates the instantaneous accuracy of tasks to dynamically adapt to the required resources for each task on multiple switches. By multiplexing resources among network-wide measurement tasks, SCREAM supports more accurate tasks than current practice, OpenSketch [42]. In the future, we plan to add more sketch-based tasks such as flow-size distribution and entropy estimation.

9. ACKNOWLEDGEMENTS

We thank our shepherd Laurent Mathy and CoNEXT reviewers for their helpful feedback. This work is supported in part by the NSF grants CNS-1423505, CNS-1453662, and a Google faculty research award.

APPENDIX

A. Heavy hitter accuracy estimation using Chernoff's bound

If HH j has traffic on a set of sketches S, SCREAM returns $\sum_{s\in S} c_j^{est_s}$ as an approximation of its volume. In Equation 8, however, we prove that this approximation is always smaller than $c_j^{real} + min_i(\sum_{s\in S} e_{i,j}^{cm_s})$ where $e_{i,j}^{cm_s}$ is the Count-Min sketch error on sketch s in row i for HH j.

$$c_j^{est} = \sum_{s\in S} c_j^{est_s} = \sum_{s\in S} min_i(c_{i,j}^{est_s}) \leq min_i(\sum_{s\in S} c_{i,j}^{est_s})$$
$$= min_i(\sum_{s\in S} c_{i,j}^{real_s} + e_{i,j}^{cm_s}) = c_j^{real} + min_i(\sum_{s\in S} e_{i,j}^{cm_s}) \quad (8)$$

Now for each row of Count-Min sketch, i, $e_{i,j}^{cm_s}$ are independent random variables on different switches. We apply Chernoff's bound on their sum to compute the complement probability of p_j in Equation 9 where μ is the summation of average errors on sketches and $(1+\delta)\mu = c_j^{est} - \theta$. The exponent d in Equation 9 is because of the minimum over d summations in Equation 8.

$$P(c_j^{real} > \theta) \leq 1 - P(min_i(\sum_{s\in S} e_{i,j}^{cm_s}) \geq c_j^{est} - \theta)$$
$$= 1 - exp(-\frac{\delta_j^2}{2+\delta_j}\mu d) \quad (9)$$

B. Super source or destination accuracy estimation

Following Equation 7, we simplify the probability that a counter for SSD j in the ith hash function has large enough error to make it a false positive. This happens if sum of Count-Min error and distinct counter error, $e_{i,j}^{dc} + e_{i,j}^{cm}$, is greater than $c_{i,j}^{est} - \theta'$ for threshold θ and $\theta' = (1-\frac{1}{w})\theta + \frac{T - \sum_{k\in L} c_L^{est}}{w}$. We can rewrite this probability based on an indicator function (I) and the joint probability distribution of $e_{i,j}^{dc}$ and $e_{i,j}^{cm}$ in Equation 10. As the two errors are independent, we can separate the joint distribution in Equation 11. Now running the integration on the indicator function and the probability distribution of Count-Min sketch error, $e_{i,j}^{cm}$, we reach Equation 12. Then we can calculate each term of Equation 12 separately: The error of HyperLogLog follows the Gaussian distribution and the probability on Count-Min sketch can be bounded using Markov inequality.

$$P(e_{i,j}^{dc} + e_{i,j}^{cm} \geq c_{i,j}^{est} - \theta') = \int_t \int_s I_{t+s\geq c_{i,j}^{est} - \theta'} \; f_{e_{i,j}^{dc}, e_{i,j}^{cm}}(t,s) \; dt \; ds \quad (10)$$

$$= \int_t \int_s I_{t+s\geq c_{i,j}^{est} - \theta'} \; f_{e_{i,j}^{dc}}(t) \; f_{e_{i,j}^{cm}}(s) \; dt \; ds \quad (11)$$

$$= \int_t f_{e_{i,j}^{dc}}(t) \; P(e_{i,j}^{cm} \geq c_{i,j}^{est} - \theta' - t) \; dt \quad (12)$$

10. REFERENCES

[1] Amazon CloudWatch. http://aws.amazon.com/cloudwatch/.
[2] Amazon Web Services' Growth Unrelenting. http://news.netcraft.com/archives/2013/05/20/amazon-web-services-growth-unrelenting.html.
[3] CAIDA Anonymized Internet Traces 2014. http://www.caida.org/data/passive/passive_2014_dataset.xml.
[4] HP 5120 EI Switch Series: QuickSpecs. http://h18000.www1.hp.com/products/quickspecs/13850_na/13850_na.PDF.
[5] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi. Mergeable Summaries. In *PODS*, 2012.
[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.
[7] T. Bu, J. Cao, A. Chen, and P. P. C. Lee. Sequential Hashing: A Flexible Approach for Unveiling Significant Patterns in High Speed Networks. *Computer Networks*, 54(18):3309–3326, 2010.
[8] J. Cao, Y. Jin, A. Chen, T. Bu, and Z.-L. Zhang. Identifying High Cardinality Internet Hosts. In *INFOCOM*, 2009.
[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.
[10] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. In *NSDI*, 2012.

[11] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *WREN*, 2009.

[12] H. Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *The Annals of Mathematical Statistics*, 23(4), 1952.

[13] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks. In *IEEE/IFIP Network Operations and Management Symposium*, 2014.

[14] G. Cormode. Sketch Techniques for Approximate Query Processing. In P. H. G. Cormode, M. Garofalakis and C. Jermaine, editors, *Synposes for Approximate Query Processing: Samples, Histograms, Wavelets and Sketches, Foundations and Trends in Databases*. NOW publishers, 2011.

[15] G. Cormode and M. Hadjieleftheriou. Finding Frequent Items in Data Streams. In *VLDB*, 2008.

[16] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding Hierarchical Heavy Hitters in Data Streams. In *VLDB*, 2003.

[17] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1), 2005.

[18] G. Cormode and S. Muthukrishnan. Space Efficient Mining of Multigraph Streams. In *PODS*, 2005.

[19] G. Cormode and S. Muthukrishnan. Summarizing and Mining Skewed Data Streams. In *SIAM International Conference on Data Mining*, 2005.

[20] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *SIGCOMM*, 2011.

[21] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. *SIGCOMM Computer Communication Review*, 32(4):323–336, 2002.

[22] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. *Transactions on Networking*, 9(3), 2001.

[23] P. Flajolet and G. N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[24] É. Fusy, G. Olivier, and F. Meunier. Hyperloglog: The Analysis of a Near-optimal Cardinality Estimation Algorithm. In *Analysis of Algorithms(AofA)*, 2007.

[25] M. Hadjieleftheriou, J. W. Byers, and J. Kollios. Robust Sketching and Aggregation of Distributed Data Streams. Technical report, Boston University Computer Science Department, 2005.

[26] S. Heule, M. Nunkesser, and A. Hall. HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. In *International Conference on Extending Database Technology*, 2013.

[27] F. Khan, N. Hosein, C.-N. Chuah, and S. Ghiasi. Streaming Solutions for Fine-Grained Network Traffic Measurements and Analysis. In *ANCS*, 2011.

[28] F. Korn, S. Muthukrishnan, and Y. Wu. Modeling Skew in Data Streams. In *SIGMOD*, 2006.

[29] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based Change Detection: Methods, Evaluation, and Applications. In *IMC*, 2003.

[30] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. In *SIGMETRICS*, 2004.

[31] G. M. Lee, H. Liu, Y. Yoon, and Y. Zhang. Improving Sketch Reconstruction Accuracy Using Linear Least Squares Method. In *IMC*, 2005.

[32] P. Li and C.-H. Zhang. A New Algorithm for Compressed Counting with Applications in Shannon Entropy Estimation in Dynamic Data. In *COLT*, 2011.

[33] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: A Novel Counter Architecture for Per-flow Measurement. *SIGMETRICS Performance Evaluation Review*, 36(1):121–132, 2008.

[34] S. Meng, A. K. Iyengar, I. M. Rouvellou, and L. Liu. Volley: Violation Likelihood Based State Monitoring for Dataceners. *ICDCS*, 2013.

[35] M. Moshref, M. Yu, and R. Govindan. Resource/Accuracy Tradeoffs in Software-Defined Measurement. In *HotSDN*, 2013.

[36] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. DREAM: Dynamic Resource Allocation for Software-defined Measurement. In *SIGCOMM*, 2014.

[37] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible Sketches for Efficient and Accurate Change Detection over Network Data Streams. In *IMC*, 2004.

[38] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang. LADS: Large-scale Automated DDoS Detection System. In *ATC*, 2006.

[39] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. CSAMP: A System for Network-Wide Flow Monitoring. In *NSDI*, 2008.

[40] A. Vahdat. Enter the Andromeda zone - Google Cloud Platform's Latest Networking Stack. http://goo.gl/smN6W0.

[41] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders. In *NDSS*, 2005.

[42] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *NSDI*, 2013.

[43] Y. Zhang. An Adaptive Flow Counting Method for Anomaly Detection in SDN. In *CoNEXT*, 2013.