IFIP Networking 2019 1570511943

A Black-box Method for Accelerating Measurement Algorithms with Accuracy Guarantees

Ran Ben Basat
Harvard University

Gil Einziger
Ben Gurion University

Marcelo Caggiani Luizelli
Federal University of Pampa

Erez Waisbard
Nokia Bell Labs

***Index Terms*—heavy hitters, frequency estimation, network measurement, NFV, OVS, high-performance**

***Abstract*—Network Function Virtualization (NFV) enables software implementations of middleboxes such as load balancing, traffic engineering and quality of service. These often rely on network measurement such as per-flow frequency estimation, bandwidth estimation, counting distinct elements and estimating the traffic entropy. Keeping up with the line speed is an active challenge for NFV measurement techniques, and library algorithms are simply too slow.**

Sampling is a natural technique to increase the measurement throughput, but it requires a certain amount of traffic before accuracy is guaranteed. In this work, we introduce a throughput acceleration method that preserves accuracy from the very first packet. This technique works with a variety of existing measurement algorithms (e.g., the ones mentioned above), and improves their throughput while guaranteeing their correctness throughout the entire measurement. Our work includes a rigors analysis, an extensive evaluation with real network traces, and a real DPDK enabled Open vSwitch implementation.

I. Introduction

Network algorithms such as traffic engineering, load balancing, quality of service, caching and anomaly detection [1]–[4] are used to optimize various aspects of networks and to provide users with better service. These algorithms base their decisions on network measurement; e.g., load balancers often place flows on the least congested path [4]. Measurement is a well-researched problem, as algorithms are required to handle rapid line speed and massive data volumes.

Emerging technologies such as *Network Function Virtualization (NFV)* can potentially reduce costs by running network algorithms on virtual machines. This improves over the current state where network functionalities are supplied with dedicated, proprietary, and typically expensive hardware. In addition, NFV enables dynamic deployment and modification of network functionalities without hardware changes [5].

NFV introduces a difficult challenge to network measurements, as we require software algorithms to match the performance of custom-made hardware. Moreover, the design assumptions of software algorithms differ from those of custom hardware. For example, in hardware, a common approach is to reduce space consumption in order to use the fast but scarce (SRAM) memory [6]. In software, we have no direct control over the memory type that is used and therefore we require different methods to improve operation speed. We argue that faster software measurements are essential for NFV.

This work is about *accelerating* the performance of software measurement algorithms while ensuring the accuracy guarantees of the original algorithm. Illustrated in Figure 1, our acceleration method reuses the code of existing algorithms as a black-box while adding an external sampler that is indifferent to the underlying algorithm. The sampling probability of our sampler varies dynamically, resulting in faster algorithms that can seamlessly replace the existing ones as they provide the same accuracy guarantees. Unlike traditional uniform sampling approaches [7], our algorithms are accurate throughout the entire measurement and have no convergence time. Further, we show that our approach allows estimating the number of distinct elements and entropy – types of measurements that cannot be reconstructed from uniform samples [7].

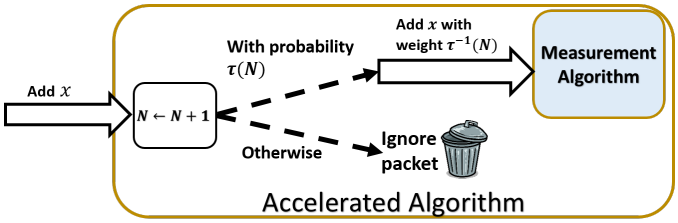


Fig. 1: An existing black-box algorithm is accelerated by ignoring a large portion of the traffic. The sample ratio is **dynamically** adjusted to boost performance while maintaining accuracy guarantees.

Our Contribution: We focus on the per-flow frequency estimation problem and introduce an acceleration method which is formally analyzed and extensively evaluated on real packet traces. We show a speedup of up 12x without a significant impact on empirical accuracy. Next, we integrate our method into the popular Open vSwitch (OVS) and evaluate two measurement options – in data plane and in a dedicated VM. We demonstrate +16% higher throughput for OVS data plane measurement and +19% for the VM alternative.

Finally, we show the generality of our approach by accelerating per-flow volume estimation, L_2-heavy hitters, hierarchical heavy hitters and universal monitoring – a unified solution for multiple measurements such as Count Distinct, Entropy, and Change Detection.

II. Related Work

Frequency Estimation Algorithms are used to find the most frequent elements in a stream and are traditionally divided into *sketch algorithms* and *counter algorithms*. Sketch algorithms use an array of counters and a set of hash functions and for every packet, each function selects a counter to be up-

dated. Sketches estimate the frequency of an element based on the values of its associated counters. Classical sketches include *Count Sketch (CS)* [8] and *Count Min Sketch (CMS)* [9]. They are widely used in practice as they are simple and efficient to implement both in hardware and software. Recent sketch algorithms include Counter Braids [10], Randomized Counter Sharing [11] and Counter Tree [12] which optimize the update time and accuracy at the expense of query speed.

Counter algorithms maintain a small flow cache with a dedicated line for each monitored flow. These methods accurately monitor the cached flows and thus attempt to identify the most frequent flows to maximize their precision and accurately monitor as much of the traffic as possible. Examples include *Lossy Counting* [13], Frequent [14] and *Space Saving (SS)* [15]. SketchVisor [16] combines counter and sketch algorithms in a fast path slow path technique. They observe that SS works faster than sketches, and when the sketch cannot meet the line rate they use it to aggregate updates that are then merged into the sketch. Our work is complimentary since it can make both the sketch and the counter algorithm fast enough to meet the line's rate.

Sampling Algorithms: Many networking devices offer the 1 out of N random packet sampling [17], [18]. The work of [19] shows that diverse statistical metrics can be monitored with sampling, and [20] shows how to efficiently partition a limited sample budget to satisfy multiple objectives. Other methods such as Sample and hold and Sticky sampling [13], [21] combine samples and counter algorithms. These methods update their data structures for every packet of each monitored flow. This design choice makes their improvement workload dependent.

III. Statistical Acceleration

Our goal is to enable a black-box acceleration of legacy frequency estimation algorithms while preserving their accuracy guarantees as much as possible. Toward this goal, our Skipper algorithm dynamically lowers the sampling probability as the stream progresses.

Intuitively, uniform sampling where each packet is sampled with a fixed probability can also be used to accelerate frequency estimation algorithms. While this approach is practical and is covered in our statistical analysis it is not a black-box acceleration method. The problem with uniform samples is that it requires a certain amount of traffic to converge before it provides accuracy guarantees. This means, that accelerating a legacy algorithm in this manner is not transparent to the user. Specifically, the consuming application needs to be aware of the concrete algorithm configuration to know when the measurement results can be reliably queried. Therefore, applications need to be updated to cope with the additional complexity of the acceleration method which is expensive.

In contrast, Skipper assures that the accuracy is preserved at any time during the measurement. Thus, the user application cannot distinguish between the output of the accelerated algorithm and that of a legacy algorithm. It supports measurements where the collected data is used during the measurement rather than just at the end. This kind of measurement is often referred to as "online" in the literature and is especially useful for near real-time decision making algorithms [22], [23].

Notations and Definitions: We start with the necessary notations and definitions. A stream $\mathcal{S}$ is a sequence of elements $x_0, x_1, \ldots, x_N$ such that every element is a member of the *domain* $\mathcal{U}$; We denote by N the current number packets in the stream. For example, a stream might contain the source IP addresses of incoming packets, and the domain will be all 32-bit binary vectors ($\mathcal{U} \triangleq \{0,1\}^{32}$). Alternatively, applications often consider 5-tuples (source ip, destination ip, source port, destination port and protocol) as the identifier of a flow. Given a flow id, (x) we denote the frequency of x by $f_x \triangleq |\{i \mid x_i = x\}|$. Our proposed algorithm, Skipper, which is described in Section III takes as input two parameters ε_s, δ_s which affect a trade-off between the accuracy and its update speed. We denote by $\Gamma \triangleq \lceil \varepsilon_s^{-2} 3 \ln \frac{1}{2\delta_s} \rceil$ a quantity which depends on the parameters ε_s, δ_s and frequently appears in the paper. This constant arises from the analysis and determines the minimal sampling rate needed for the accuracy guarantees. Additionally, Skipper takes a measurement algorithm $\mathbb{A}$ that is used as a black box; we assume that $\mathbb{A}$ takes as input parameters ε_a, δ_a which determines its accuracy, and their exact definition appears below; Table I summarize notations used in this work.

Symbol	Meaning
$\mathcal{S}$	Stream
N	Current number of packets (in all flows)
S_x^i	$S_x^i = 1$ if the i'th packet is sampled.
S_x	The number of sampled packets from flow x.
S_N	Total number of sampled packets from all flows.
$\mathcal{U}$	The domain of ids.
p_i	The i'th packet in the stream.
f_x	Total number of packets in flow x
$\widehat{f_x}$	An estimation of f_x
ε	Accuracy parameter
ε_a	Measurement algorithm's accuracy
ε_s	Sample's accuracy
δ	Confidence parameter
δ_a	Measurement algorithm's confidence
δ_s	Sample's error probability
τ	Average packet sampling probability
Γ	Sampling constant $\left(\lceil \varepsilon_s^{-2} 3 \ln \frac{1}{2\delta_s} \rceil\right)$

TABLE I: List of Symbols

Frequency Estimation Algorithms We consider algorithms with the following methods:
Update$(\mathbf{x})$: Reports an arrival of packet x.
Estimate$(\mathbf{x})$: returns an estimator $\widehat{f_x}$ of f_x, the total number of packets from flow x.

Some algorithms support *weighted updates*, where each arriving packet is associated with a weight. In this case, f_x is defined as the total weight of x, and the Update(x, w) operation receives a weight parameter w.

Definition III.1. *An algorithm solves the* (ε, δ) **- Frequency Estimation** *problem if for any flow* (x)*:*
$\Pr\left[\left|f_x - \widehat{f_x}\right| \leq \varepsilon N\right] \geq 1 - \delta.$

Note that frequency estimation is general and we can find heavy hitters by returning all flows whose estimated frequency

is as above $(\theta - \varepsilon) N$, where θ is a threshold provided by the user. This returned a set contains all flows whose frequency is above θN, and no flow whose frequency is below $(\theta - \varepsilon) N$.

Skipper: We now present the *Skipper* meta algorithm that provides black-box acceleration of legacy frequency estimation algorithms. When the user updates Skipper with a packet, Skipper may either update the underlying algorithm or ignore the packet altogether. The decision is done in a manner that preserves the accuracy from the first packet onward, which means that accelerated algorithms enjoy the performance benefits of sampling without many of its drawbacks.

Formally, Skipper accelerates an off-the-shelf (weighted) frequency estimation algorithm $\mathbb{A}$, that solves the FREQUENCY ESTIMATION problem (for ε_a and δ_a). We show that the accelerated version of $\mathbb{A}$ (ACC($\mathbb{A}$)), solves the (ε, δ) - FREQUENCY ESTIMATION problem while operating significantly faster. Here, $\varepsilon \triangleq \varepsilon_a + \varepsilon_s$ and $\delta \triangleq \delta_a + 2\delta_s$, where ε_s, δ_s are the input parameters for Skipper and affect a run time vs. accuracy trade-off.

Algorithm 1 provides pseudo code for Skipper. When packet N arrives, Skipper computes a sample probability $\tau(N)$ and ignores the packet with probability $1 - \tau(N)$. In case the packet is sampled, the corresponding flow id is updated in $\mathbb{A}$ with weight of $\tau^{-1}(N)$, so that each occurrence increases the *expected* weight added to $\mathbb{A}$ by 1. The sampling probability $\tau(N)$ is defined as $\min\left\{1, \lceil N/\Gamma \rceil^{-1}\right\}$ and decreases as the stream grow larger and we use smaller sampling probabilities.

Algorithm 1 Skipper($\mathbb{A}$, ε_a, δ_a, ε_s, δ_s)

Initialization:
$N \leftarrow 0$ ▷ Current number of packets.
$\Gamma \leftarrow \lceil \varepsilon_s^{-2} 3 \ln \frac{1}{2\delta_s} \rceil$ ▷ Skipper's decision interval.
$\mathbb{A} \leftarrow \mathbb{A}(\varepsilon_a, \delta_a)$ ▷ Initialize underlying algorithm.
1: **function** ADD(x)
2: $N \leftarrow N + 1$
3: $\tau(N) \leftarrow \min\left\{1, \left\lceil \frac{N}{\Gamma} \right\rceil^{-1}\right\}$
4: **if** $U(0,1) < \tau(N)$ **then** ▷ With prob $\tau(N)$
5: $\mathbb{A}$.UPDATE($x, \tau^{-1}(N)$)
6: **function** QUERY(x)
7: **return** $\mathbb{A}$.ESTIMATE(x)

IV. ANALYSIS

Formally, Skipper wraps an algorithm that solves the frequency estimation problem and the accelerated algorithm solves the (ε, δ) - FREQUENCY ESTIMATION problem for larger δ and ε that factors both the sampling error and the algorithmic error. This analysis formally proves this trade-off.

A. Frequency Estimation from Samples

We start by deriving bounds for frequency estimation from uniform samples with a (fixed) sampling probability τ. These would later allow us to dynamically set the sampling probability. Theorem IV.1 shows that for any given ε_s, δ_s and sampling probability (τ), the sample solves $(\varepsilon_s, \delta_s)$ - FREQUENCY ESTIMATION after enough packets have passed. The proof of this theorem is deferred to the full version of the paper due to lack of space.

Theorem IV.1. *If $N \geq \Gamma\tau^{-1}$, then a random sample solves the $(\varepsilon_s, \delta_s)$ - FREQUENCY ESTIMATION problem.*

Corollary IV.2. *Given ε_s, δ_s and N, random sample solves the $(\varepsilon_s, \delta_s)$ - FREQUENCY ESTIMATION problem as long as $\tau \geq \tau_{min}$ defined as $\tau_{min} \triangleq \min\{1, \frac{\Gamma}{N}\}$*

Proof. If $\tau = 1$ then we observe the entire stream and clearly solve the problem. Otherwise, the proof is derived by extracting τ as function of N, ε_s and δ_s. We use Theorem IV.1 and deduce that $\tau \geq \Gamma/N$. Therefore as long as $\tau \geq \Gamma/N$ we get that: $\Pr\left(\left|f_x - \widehat{f_x}\right| \leq \varepsilon_s N\right) \geq 1 - \delta_s$. □

B. Combining Samples and Estimation Algorithms

Measurement algorithms guarantee that their estimation error is proportional to the number of updates (or the number of packets). However, when we sample packets, the sample size varies and thus we need to prove an upper bound on the sample size (which we then use to bound the algorithmic error). We address this issue by allocating slightly more space to the algorithm, as explained below. We start by binding the probability of over sampling; we denote by S_N the sample size.

Corollary IV.3. *Consider random sample with probability τ, and stream of length $N \geq \Gamma\tau^{-1}$, then*
$\Pr\left(S_N \leq \tau N\left(1 + \varepsilon_s\right)\right) \geq 1 - \delta_s$

Proof. Note that the expected number of sampled packets is $\mathbb{E}(S_N) = \tau \cdot N$, and that $N \geq \Gamma\tau^{-1}$. Therefore, we use Theorem IV.1 to complete the proof. □

Using Corollary IV.3, we can guarantee that although the number of packets fluctuates the actual sample size is bounded. Given an error parameter ε'_a and algorithm $\mathbb{A}$ that solves the $(\varepsilon_a, \delta_a)$ - FREQUENCY ESTIMATION problem, we define $\varepsilon_a \triangleq \frac{\varepsilon'_a}{1+\varepsilon_s}$. According to Corollary IV.3, with probability $1 - \delta_s$ the number of sampled packets is at most $(1 + \varepsilon_s)N\tau$. According to the union bound with probability $1 - \delta_a - \delta_s$ we get: $ERR(\mathbb{A}) \leq \varepsilon'_a\left(1 + \varepsilon_s\right)\tau N = \frac{\varepsilon_a(1+\varepsilon_s)}{1+\varepsilon_s} N\tau = \varepsilon_a N\tau$. For example, Space Saving [15] normally requires $1,000$ counters for $\varepsilon_a = 0.001$; with a random sample that satisfies $\varepsilon_s = 0.001$ we now require 1001 counters.

Hereafter, we assume that the algorithm is configured correctly to handle the over sampling problem. Next, we prove that by applying an off the shelf algorithm on a random sample, we still solve the (ε, δ) - FREQUENCY ESTIMATION problem. Intuitively, we have two sources of error – first, the input is only a sample; second, we apply an approximation algorithm.

Theorem IV.4. *Consider an algorithm ($\mathbb{A}$) that solves the $(\varepsilon_a, \delta_a)$ - FREQUENCY ESTIMATION problem . If the input to $\mathbb{A}$ is a τ random sample and $N > \Gamma\tau^{-1}$, then for $\delta \geq \delta_a + 2 \cdot \delta_s$ and $\varepsilon \geq \varepsilon_a + \varepsilon_s$, $\mathbb{A}$ solves (ε, δ) - FREQUENCY ESTIMATION.*

Proof. Since $N > \Gamma\tau^{-1}$, we can use Theorem IV.1 to get:

$$\Pr\left[\left|f_x - S_x\tau^{-1}\right| \geq \varepsilon_s N\right] \leq \delta_s. \quad (1)$$

Similarly, $\mathbb{A}$ solves the $(\varepsilon_a, \delta_a)$ - FREQUENCY ESTIMATION problem and provides us with an estimator $\widehat{S_x}$ that approximates S_x – the number of samples from flow x. According to Corollary IV.3: $\Pr\left(\left|S_x - \widehat{S_x}\right| \leq \varepsilon_a N\tau\right) \geq 1-\delta_a-\delta_s$.Therefore: $\Pr\left(\left|S_x - \widehat{S_x}\right| \geq \varepsilon_a N\tau\right) \leq \delta_a+\delta_s$. Multiplying by τ^{-1} gives:

$$\Pr\left(\left|S_x\tau^{-1} - \widehat{S_x}\tau^{-1}\right| \geq \varepsilon_a N\right) \leq \delta_a + \delta_s. \quad (2)$$

We need to prove that: $\Pr\left(\left|f_x - \tau^{-1}\widehat{S_x}\right| \leq \varepsilon N\right) \geq 1 - \delta$. Recall that: $f_x = E(S_x)\tau^{-1}$ and that $\widehat{f_x} = \widehat{S_x}\tau^{-1}$ is the estimated frequency of x. Thus,

$$\begin{aligned}
&\Pr\left(\left|f_x - \widehat{f_x}\right| \geq \varepsilon N\right) = \Pr\left(\left|f_x - \tau^{-1}\widehat{S_x}\right| \geq \varepsilon N\right) \\
&= \Pr\left(\left|f_x + \left(\tau^{-1}S_x - \tau^{-1}S_x\right) - \tau^{-1}\widehat{S_x}\right| \geq (\varepsilon_a + \varepsilon_s)N\right) \quad (3) \\
&\leq \Pr\left(\left[\left|f_x - \tau^{-1}S_x\right| \geq \varepsilon_s N\right] \vee \left[\left|S_x\tau^{-1} - \widehat{S_x}\tau^{-1}\right| \geq \varepsilon_a N\right]\right)
\end{aligned}$$

Where the last inequality follows from the fact that in order for the error of (3) to exceed εN, at least one of the events has to occur. We use the union bound to get the following bound: $\Pr\left(\left|f_x - \tau^{-1}S_x\right| \geq \varepsilon_s N\right) + \Pr\left(\left|S_x\tau^{-1} - \widehat{S_x}\tau^{-1}\right| \geq \varepsilon_a N\right)$.

Thus, from Equation 1 and Equation 2 we get $\Pr\left(\left|f_x - \widehat{f_x}\right| \geq \varepsilon N\right) \leq \delta_a + 2\delta_s$. □

C. Analysis of Skipper

We are now ready to analyze Skipper. We first prove that $\tau(N)$ used by Skipper is always larger or equal to τ_{min} derived by Corollary IV.2.

Lemma IV.5. $\forall N, \tau(N) \geq \tau_{min}$.

Proof. According to Corollary IV.2, $\tau_{min} = \min\{1, \frac{\Gamma}{N}\}$, and $\tau(N) = \min\left\{1, \left\lceil\frac{N}{\Gamma}\right\rceil^{-1}\right\}$. If $N < \Gamma$ then $\tau_{min} = \tau(N) = 1$; else, $\tau_{min} = \Gamma/N \leq \lceil N/\Gamma\rceil^{-1} = \tau(N)$. □

We are now ready to prove Theorem IV.6 that shows that Skipper is correct.

Theorem IV.6. *Given an algorithm $\mathbb{A}$ that solves the $(\varepsilon_a, \delta_a)$ - FREQUENCY ESTIMATION problem, and parameters ε_s, δ_s, Skipper solves the (ε, δ) - FREQUENCY ESTIMATION problem for $\delta = \delta_a + 2\delta_s$ and $\varepsilon = \varepsilon_a + \varepsilon_s$ for any stream size N.*

Proof. We prove by induction on the values of $\tau^{-1}(N)$.
Base: Initially, $\tau(N) = 1$ and the claim holds trivially.
Hypothesis: For streams of length N such that $\tau^{-1}(N) \leq k$, Skipper solves the problem as needed.
Step: Assume that the claim holds until $\tau(N) = \frac{1}{k}$ and prove that it continues to hold for $\tau(N) = \frac{1}{k+1}$. For every packet that is sampled with the new $\tau(N)$, Lemma IV.5 satisfies that we can apply Theorem IV.4. When a packet is sampled, $\mathbb{A}$ is updated with weight $\tau^{-1}(N)$. Therefore, if the estimation was within margin at the beginning, Theorem IV.4 guarantees that it remains within the error margin for the new $\tau(N)$. □

Next, we show that Skipper samples a logarithmic number of packets. Recall that S_N is the random variable representing the number of packets sampled on an N-sized stream.

Theorem IV.7. *The number of sampled packets satisfies* $\mathbb{E}[S_N] = O(\Gamma\log(N/\Gamma)) = O\left(\log\left(\delta_s^{-1}\right)\varepsilon_s^{-2}\log\left(N\varepsilon_s^2\log\delta_s\right)\right)$.

Proof. Recall that the i'th packet is sampled with probability $\tau(i)$. The overall expected number of sampled packets is thus:

$$\begin{aligned}
\mathbb{E}[S_N] &= \sum_{i=1}^{N}\tau(i) = \sum_{i=1}^{N}\min\left\{1, \left\lceil\frac{i}{\Gamma}\right\rceil^{-1}\right\} \\
&= \sum_{i=1}^{\lfloor\Gamma\rfloor}1 + \sum_{i=\lfloor\Gamma\rfloor+1}^{N}\left\lceil\frac{i}{\Gamma}\right\rceil^{-1} \leq \sum_{i=1}^{\lfloor\Gamma\rfloor}1 + \sum_{i=\lfloor\Gamma\rfloor+1}^{N}\frac{\Gamma}{i} \\
&= \sum_{i=1}^{\lfloor\Gamma\rfloor}1 + \Gamma\sum_{i=\lfloor\Gamma\rfloor+1}^{N}\frac{1}{i} \leq \Gamma\cdot\left(1 + H_N - H_{\lfloor\Gamma\rfloor}\right) + 1 \\
&\leq \Gamma\left(\ln\left(N/\Gamma\right) + 2\right) = O(\Gamma\log\left(N/\Gamma\right)).
\end{aligned}$$

Here, H_n is the $n'th$ harmonic number which is known to satisfy $H_n \triangleq \sum_{i=1}^{n} i^{-1} = \ln n + \gamma + O(1/n)$, where $\gamma \approx 0.577$ is the Euler-Mascheroni constant. □

We now use the result derived in Theorem IV.7 to bound the expected amortized time of our algorithm.

Lemma IV.8. *Let $\mathbb{A}$ be an algorithm that solves the $(\varepsilon_a, \delta_a)$ - FREQUENCY ESTIMATION problem, whose update complexity is $O(f(\varepsilon_a, \delta_a))$. The expected run time complexity of the Skipper acceleration of $\mathbb{A}$ for a stream of length N, is $O\left(1 + f(\varepsilon_a, \delta_a)\cdot\frac{\Gamma\log(N/\Gamma)}{N}\right)$.*

Proof. The processing of each packet takes $\Omega(1)$ time regardless of whether it is sampled or not. Additionally, $\mathbb{E}[S_N]$ packets are sampled from a N-sized stream. This means that the expected overall processing time is $t(N) \triangleq O(N + \mathbb{E}[S_N]\cdot f(\varepsilon_a, \delta_a))$, and that the amortized time per packet is $\frac{t(N)}{N} = O(1 + f(\varepsilon_a, \delta_a)\cdot\mathbb{E}[S_N]/N)$. □

We use an additional lemma that is helpful in bounding the minimal stream size to obtain constant runtime.

Lemma IV.9. *Let $a, b \in \mathbb{R}^+$ such that $b > 10$ and $a \geq b\ln b$, then $2a \geq b\ln a$.*

Finally, we use this lemma to provide bounds on the minimal stream length that allows various algorithms to run in constant amortized time.

Theorem IV.10. *Let $\mathbb{A}$ be an algorithm that solves the $(\varepsilon_a, \delta_a)$ - FREQUENCY ESTIMATION problem, whose update runtime complexity is $O(f(\varepsilon_a, \delta_a))$ and let $N = \Omega\left(\Gamma f(\varepsilon_a, \delta_a)\log f(\varepsilon_a, \delta_a)\right)$ $= \Omega\left(\varepsilon_s^{-2}\log{\delta_s}^{-1} f(\varepsilon_a, \delta_a)\log f(\varepsilon_a, \delta_a)\right)$. The Skipper acceleration of $\mathbb{A}$ operates in expected amortized constant time for N sized streams.*

Proof. According to Lemma IV.8, our runtime is

$$\begin{aligned}
&O\left(1 + \frac{f(\varepsilon_a, \delta_a)\log\left(\delta_s^{-1}\right)\varepsilon_s^{-2}\log\left(N\varepsilon_s^2\log\delta_s\right)}{N}\right) \\
=&O\left(1 + \frac{f(\varepsilon_a, \delta_a)\Gamma\log\left(N/\Gamma\right)}{N}\right).
\end{aligned}$$

Our choice of N satisfies: $\frac{f(\varepsilon_a,\delta_a)\Gamma\log(N/\Gamma)}{N} = O(1)$ and thus the runtime is constant. Notice that this is equivalent to demanding $f(\varepsilon_a,\delta_a)\log(N/\Gamma) = O\left(\frac{N}{\Gamma}\right)$. We denote $b \triangleq f(\varepsilon_a,\delta_a)$, and $a \triangleq \frac{N}{\Gamma}$, and show that $b\log a = O(a)$. Recall that $N = \Omega\left(\Gamma f(\varepsilon_a,\delta_a)\log\left(f(\varepsilon_a,\delta_a)\right)\right)$, and thus $a = \Omega\left(f(\varepsilon_a,\delta_a)\log\left(f(\varepsilon_a,\delta_a)\right)\right)$. Hence, according to Lemma IV.9: $b\ln a = O(a)$. □

Next, we note that the heap implementation of Space Saving (SSH) [24] supports weights and operates at $f(\varepsilon_a,\delta_a) = O(\log 1/\varepsilon_a)$ time. Similarly, Count Sketch and Count Min Sketch operate at $f(\varepsilon_a,\delta_a) = O(\log 1/\delta_a)$. Finally, we conclude that our bounds on the stream size allows expected constant amortized time for the accelerated algorithms.

Corollary IV.11. *The Skipper acceleration of the Space Saving algorithm achieves expected amortized constant update time when $N = \Omega(\Gamma\log 1/\varepsilon_a \log\log 1/\varepsilon_a)$. Similarly, if $N = \Omega(\Gamma\log 1/\delta_a \log\log 1/\delta_a)$ then the amortized update time of accelerated CS and CMS to $O(1)$.*

V. Evaluation

Our evaluation includes four datasets, each containing a mix of 1 billion UDP, TCP, and ICMP packets. The datasets were collected from major backbone routers in Chicago [25] and San Jose [26] in the years 2014-2016. We use the On-Arrival error model suggested by [23], [27]–[29]. That is, we query each packet to estimate its flow's frequency. This metric is motivated by requiring high estimation accuracy throughout the entire measurement period and is attractive for applications that make real time decisions. We measure the *Root Mean Square Error (RMSE)* of the algorithm, i.e., $RMSE(Alg) \triangleq \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\widehat{f_{x_t}} - f_{x_t}\right)^2}$,where x_t is the flow identifier of the t^{th} packet. We compare with the following algorithms: Count min sketch [9] (CMS); Count Sketch [8] (CS); and Space Saving [15] (SS). SS has two implementations, a heap implementation that supports weights and runs in logarithmic complexity (SSH) [30] and an unweighted implementation that operates in $O(1)$ (SSL) [15]. We configure both CS and CMS to use 5 lines which is shown to be effective in practice [31]. This means that the sketches are allocated about $(5\cdot e)$ times as many counters as Space Saving. This additional space is partially compensated by not maintaining an explicit flow to counter association but means that they use $3-4$ times as much space in practice.

Skipper enhanced algorithms are denoted – ACC(CMS), accelerated CMS, ACC(CS) for accelerated CS, and ACC(SS) for accelerated SSH. Note that we cannot accelerate SSL since it does not support weights. Further, SSL implements the fast path of Sketchvisor [16], thus its performance is an upper bound on Sketchviosr's update speed. Our evaluation is done in C++ using the widely used library published by [30] for its CMS, SSH and SSL implementations; CS and Skipper were implemented by us. Skipper uses an algorithm *as a black-box* and runs same code for sampled packets. Our code is generic and receives a frequency estimation algorithm as parameter. The evaluation was performed on a PC with an Intel Core i7-3930K processor (@3.2GHz) and 32GB of RAM, running a Windows 7 platform. For each data point, we averaged the result of 50 runs. All algorithms used a single thread and did not run in parallel.

A. Evaluation of Operation Throughput

We evaluate the throughput in updates per second.

Speed as function of the number of packets. We start with a comparative evaluation of the update speed of various algorithms and their accelerated versions. Intuitively, we expect better performance on longer streams, as Skipper dynamically adjusts the measurement probability. Figure 2a and Figure 2b illustrate the operation speed for $\varepsilon = 0.01$ on two real traces. The figures shows that the operation speed of accelerated algorithms is indeed increased as expected with the number of packets. Overall, we improve the throughput by up to 12x.

Throughput and accuracy guarantee. When ε_s is small, Skipper needs to sample more packets in order to guarantee accuracy and therefore less speedup is gained. Figure 2c and Figure 2d present results for a range of error parameters ε_a. The number of packets is set to 1 Billion packets and $\delta_s = 0.01\%$ and $\varepsilon_s = \varepsilon_a$. This means that the accelerated algorithms are allocated the same number of counters but their error guarantee is doubled as $\varepsilon = \varepsilon_a + \varepsilon_s$. As the figure shows, the graphs are relatively similar and large values of ε allow for significant speedup. In these measurements, Γ is set to be roughly $14.38\cdot\varepsilon^{-2}$. This means that for ε value of 2^{-15}, we have $\Gamma > 1B$, and thus no sampling takes place. In this case, the "accelerated" algorithms degenerate to the unaccelerated versions and thus become slightly slower due to Skipper's overheads. Yet, for a wide range of ε values, the achieved speedup is between 2x and 12x. We can also see that the speed of the sketch-based algorithms (CS/CMS/ACC(CS)/ACC(CMS)) is barely affected by the skew. On the other hand, SS and ACC(SS) are faster on skewed traces and slower on heavy tailed ones. Note that ACC(CMS) has a slightly weaker guarantee than CMS – its estimator is no longer a *deterministic* lower bound on the true frequency.

B. Evaluation of Empirical Accuracy

On real traces, most measurement algorithms provide considerably better accuracy than guaranteed by their worst case analysis. Therefore, empirical accuracy is often the deciding factor favoring one algorithm over the other. Further, for real time algorithms, it is important to show that the accuracy is preserved throughout the entire trace. We note that CS uses $O(\varepsilon_a^{-2})$ counters but provides a stronger guarantee than (ε,δ) - Frequency Estimation. Its error is bounded by $F_2\cdot\varepsilon_a$, where F_2 is the second norm of the frequency vector. For a fair comparison, we use CS with the *same number of counters* as CMS, and not the same ε_a parameter. Further, we only show the error of SSH as SSL and SSH are different implementations of the Space Saving algorithm and have exactly the same error.

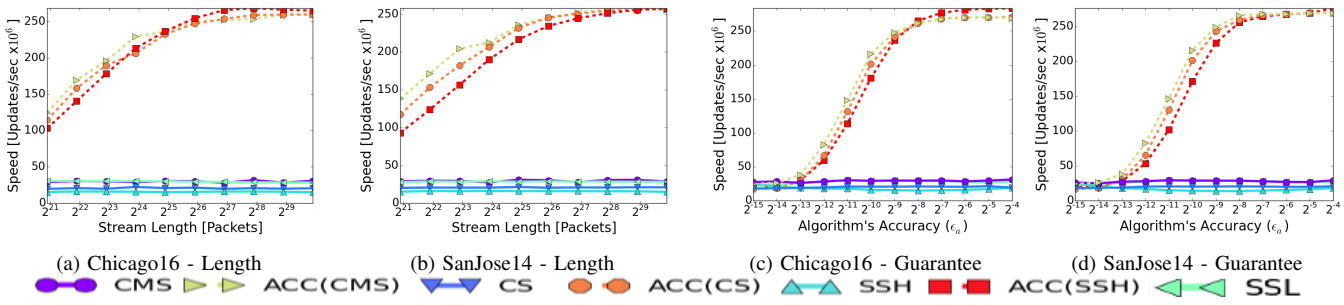


Fig. 2: (a) and (b): Relation between measurement length and throughput ($\varepsilon_s = \varepsilon_a = 1\%$). (c) and (d): Relation between ε_s and the throughput for a measurement length of 1 Billion packets.

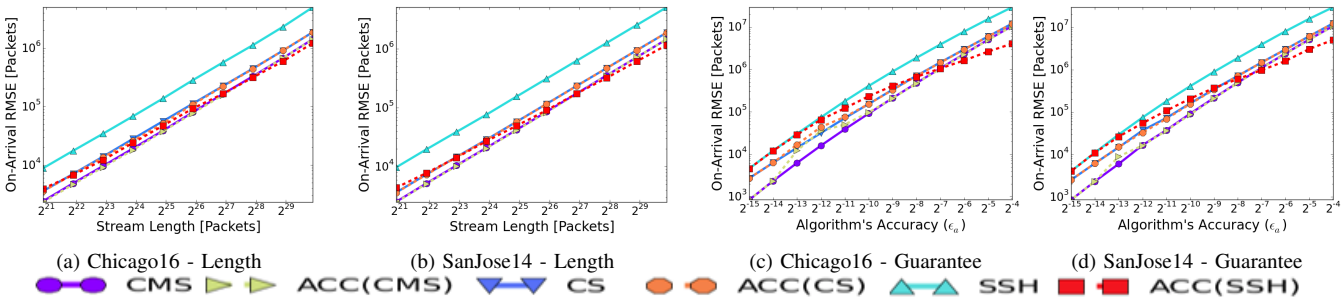


Fig. 3: (a), (b) Relation between the empirical error and the measurement length. (c),(d) Relation between the empirical error and the algorithm's error guarantee (ε_a) with a measurement length of 1 Billion packets. The accelerated algorithms are configured with the **same** number of counters as the unaccelerated ones.

Accuracy as function of the number of packets Figure 3a and Figure 3b show the empirical accuracy as function of the stream length. We use $\varepsilon_a = \varepsilon_s = 0.01$ and $\delta_s = 0.05$. As illustrated, ACC(CS) and ACC(CMS) achieve similar RMSE as CS and CMS for all tested values of N. Surprisingly, ACC(SSH) improves over SSH on all datasets and N values. In particular, in Chicago15, Chicago16, and SanJose14, it is the most accurate algorithm for large streams. A possible explanation for this phenomenon offered by [27] that showed how sampling can reduce the error of Space Saving.

Accuracy as function of guarantee (ε) Figure 3c and Figure 3d describe experiments to evaluate the empirical error for a given guarantee ε_a. We used $\varepsilon_s = \varepsilon_a$ and thus the guaranteed accuracy of the accelerated versions is only half as good. This means that each accelerated algorithm is allocated the same number of counters as its unaccelerated counterpart. Nevertheless, the accuracy of the accelerated algorithms is similar and sometimes better than the unaccelerated versions. As before, ACC(SSH) is more accurate than SSH when ε is large enough to allow sampling.

VI. Open vSwitch Implementation

Virtual switching is an essential building block in NFV environments. It allows interconnecting multiple Virtual Network Functions (VNFs) in a flexible manner and enables other traffic steering technologies such as SDN. Virtual switches are required to steer network traffic at line rate which is enabled by sophisticated optimizations. In practice, high-speed virtual switching is enabled by various software and hardware acceleration technologies such as Netmap [32] and Intel's DPDK [33], our implementation uses the DPDK-based OVS.

In that version, control and data planes are performed in user space. Network packets ingress the datapath either from a physical port connected to the physical NIC or from a virtual port connected to a remote host (e.g., a VNF). OVS then parses the headers and determines the set of actions to be applied (e.g., forwarding or rewrite a specific header). The interested reader is referred to [34] for additional information.

Design Choices: The OVS project does not have data plane sampling capabilities. Thus, we implemented this functionality through a simple forwarding action which is active for sampled packets. That is, we perform Skipper's sampling in data plane but evaluate two alternatives for the measurement itself. First, we perform it in the data plane to minimize overheads. The drawback of this alternative is that it creates coupling between the measurement algorithm and the virtual switch. Second, we perform the measurement in a separate machine. This option also offers more flexibility but has larger switching overheads. The two implementation options are illustrated in Figure 4.

Skipper's OVS Evaluation: Our evaluation consists of two identical HP ProLiant servers each with an Intel Xeon E3-1220v2 processor that has four 3.1 GHz physical cores, 8 GB RAM, and a DPDK-enabled Intel 82599ES 10 Gbit/s network

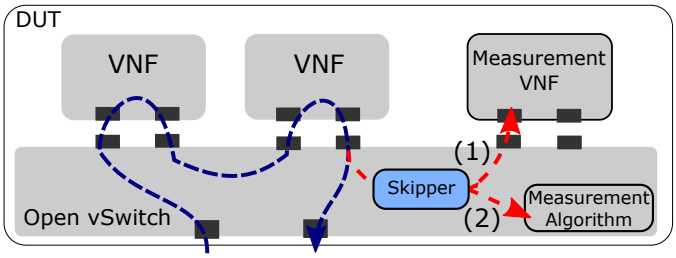
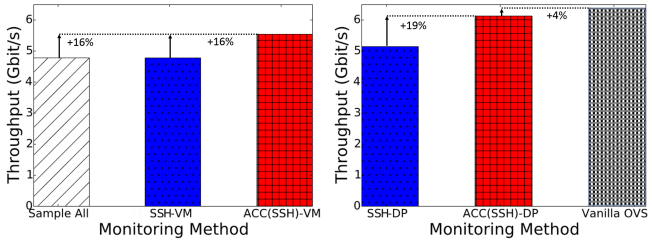


Fig. 4: Traffic enters OVS at the left port, traverses through two VNFs, and passed to Skipper. In (1), Skipper forwards the sampled packets to a measurement VNF. In (2) the measurement algorithm part of the OVS data plane. Finally, the processed traffic is then sent back to the traffic generator on a second port.



(a) VM (b) Data plane

Fig. 5: OVS throughput ($\delta_s = 0.01\%$, $\varepsilon_s = 1\%$).

device with two interconnected interfaces (physical ports). The servers run CentOS 7.2.1511, Linux kernel 3.10.0.

We test one server and use the other as a traffic generator. We use Open vSwitch 2.6 compiled with Intel DPDK 16.07. VNFs are configured with Fedora 22 operating system, Linux kernel 4.0.4 – running on top of qemu 2.6, configured with a single virtual CPU (pinned to a specific physical core) and 512MB of RAM. We use the MoonGen traffic generator [35] to send packets through one interface. We receive the packets on the other one after being processed by OVS.

OVS evaluation's considered scenario: We consider a case where an NFV service chain is deployed in parallel to network monitoring. The service chain has two VNFs in line which are forwarding the received network traffic back to the traffic generator. Additionally, traffic sent by the second VNF is sampled by Skipper which forwards sampled packets either to a measurement VNF or to an in-data plane algorithm. Figure 4 depicts this scenario.

OVS evaluation's results: We evaluate two deployment options, in the first the measurement is performed in a dedicated virtual machine and in the second it is performed in the data plane. We compare Skipper, to the fastest unaccelerated algorithm, SSH (Space Saving with a heap), we used ACC(SSH) for Skipper.

Figure 5 shows the average end-to-end OVS throughput on the SanJose14 trace for the two designs. Similar results were obtained for the other network traces.

The virtual machine deployment, in Figure 5a, shows that using an unaccelerated algorithm requires the switch to forward all packets which hurts the throughput. In contrast, using Skipper improves the OVS throughput by 16% as less traffic is pushed to the measurement VM.

Data plane deployment is evaluated in Figure 5b for both SSH and ACC(SSH). As can be observed, deploying the fastest unaccelerated algorithm in OVS's data plane still reduces its throughput by over 20%. However, when OVS uses ACC(SSH) its throughput is improved by 19%. This solution yields only 4% less throughput than not performing measurement at all! Thus, Skipper considerably mitigates measurement overheads and enables efficient network measurements.

VII. Extensions

Weighted Skipper: The weighted variant of the heavy hitter problem [28], [29], [36], [37] is about estimating per-flow **byte volume**. In these settings, every packet (z, w) is associated with a flow identifier z and a weight w, our goal is to to estimate the total weight per flow. The usage of uniform samples for this task is problematic, as uniform sampling samples small packets with the same probability as large packets and therefore the accuracy of the method depends on the size variance of the packet's [38]. Such a dependence is undesirable as the size variance is workload dependent and not under our control. Instead, we introduce *Weighted Skipper* (WS) that extends Skipper to weighted streams. For this, WS utilizes a legacy weighted heavy hitters algorithm $\mathbb{A}$ such as [28], [36]; that is, we accelerate an algorithm with constant update time for weighted streams. WS treats a packet (z, w) as a sequence of w weight 1 bytes and a flow-identifier z. Clearly, the frequency of each flow in the byte-stream is identical to its volume in the weighted stream. For each packet, WS samples each of its bytes with probability β until possibly one is sampled. This is implemented by drawing a geometric random variable $G \sim Geo(\beta)$ with mean β^{-1} that indicates how many bytes are *not sampled* in a row. Theorem IV.4 ensures that as long as the β is at least $\min\left\{1, \lceil N/\Gamma \rceil^{-1}\right\}$ the correctness is guaranteed. Once a byte is sampled, WS also samples the remaining bytes in the packet with probability 1 which is larger than β and is therefore also correct. The value of β is dynamically determined, similarly to τ in Skipper, according to the number of bytes (B) in the measurement in a way that satisfies Theorem IV.4. Algorithm 2 provides the technical details for this approach.

It is worth mentioning that while $\mathbb{E}[G] = \beta^{-1}$, inserting the packet with weight w instead of $\beta^{-1} + w - G$ in Line 6 is *incorrect* as the distribution of G **given that** $G \leq w$ is not geometric anymore and its expectation is lower than β^{-1}. Also note that if all packets are of unit weight then WS degenerates and becomes mathematically identical to Skipper.

Accelerating L_2-Heavy Hitter Sketches: An alternative accuracy guarantee for frequency estimation is the L_2 *guarantee*. Here, $L_2 \triangleq \sqrt{\sum_{x \in \mathcal{U}} f_x^2}$ is the second norm of the frequency vector and was first proposed in [8]. Formally, an algorithm solves the $(\varepsilon, \delta) - L_2$ - Frequency Estimation problem if for any flow (x):$\Pr\left[\left|(f_x)^2 - \left(\widehat{f}_x\right)^2\right| \leq \varepsilon L_2\right] \geq 1 - \delta$.

This problem requires $\Theta^*(\varepsilon^{-2})$ [8] space, where the Θ^* notation hides poly-logarithmic factors. Since $\sqrt{N} \leq L_2 \leq$

Algorithm 2 Weighted Skipper($\mathbb{A}$, ε_a, δ_a, ε_s, δ_s)

Initialization:
$B \leftarrow 0$ ▷ Current number of bytes.
$\Gamma \leftarrow \lceil \varepsilon_s^{-2} 3 \ln \frac{1}{2\delta_s} \rceil$ ▷ Skipper's decision interval.
$\mathbb{A} \leftarrow \mathbb{A}(\varepsilon_a, \delta_a)$ ▷ Initialize underlying algorithm.
function ADD(z, w)
  $B \leftarrow B + w$
  $\beta \leftarrow \min\left\{1, \lceil \frac{B}{\Gamma} \rceil^{-1}\right\}$
  $G \leftarrow Geo(\beta)$
  if $G \leq w$ **then**
    $\mathbb{A}$.UPDATE($z, \beta^{-1} + w - G$)
function QUERY(z)
  return $\mathbb{A}$.ESTIMATE(z)

N and sketches that provide an $N\varepsilon$ approximation require $\Theta^*(\varepsilon^{-1})$ space, L_2 sketches are more space efficient when $L_2 = O(\varepsilon N)$. In [7], McGregor et al. showed that it is possible to reconstruct the L_2 frequency estimation and heavy hitters from a *sub-sampled* stream. Namely, they showed one can solve L_2-Frequency Estimation if the stream satisfies $L_2 = \Omega^*(\varepsilon^{-3} p^{-3/2})$ by feeding a uniform sample (with sampling probability p) into a Count Sketch. The Count Sketch requires more space and is is configured to provide $L_2\varepsilon/\sqrt{p}$ approximation. Thus, we increase memory by a factor of p^{-1}.

Here, we propose an alternative approach that has no convergence time; as in Skipper, we start by sampling all packets and gradually reduce the sampling rate. Our algorithm can be summarized as follows: for a parameter $\underline{\tau} \in (0, 1]$ initialize a Count Sketch with $O(\varepsilon^{-2}\underline{\tau}^{-1})$ columns and $O(\log \delta^{-1})$ rows. At the N'th packet, we set $\underline{L_2} \leftarrow \sqrt{N}$ (a lower bound on the L_2 of the stream) and $\tau \leftarrow \min\left\{\max\left\{C\varepsilon^{-2}\left(\frac{\log(|\mathcal{U}|\delta^{-1})}{\underline{L_2}}\right)^{2/3}, \underline{\tau}\right\}, 1\right\}$, for the appropriate constant C. That is, we set the sampling probability τ to ensure that (1) it is always high enough to satisfy the convergence rate condition of [7], and (2) it is always larger than $\underline{\tau}$ to ensure that we allocated enough space. Each update with probability τ increases the counters by τ^{-1}. Intuitively, we start without sampling and gradually begin sampling until we reach a sampling probability of τ and our algorithm merges with [7]. Finally, by setting $\underline{\tau} = 1/\log \delta^{-1}$ we eventually get a constant update time, we conclude with the following:

Theorem VII.1. *For any $\underline{\tau} \in (0, 1]$ there exists an algorithm that uses $O\left(\varepsilon^{-2} \log \delta^{-1} \underline{\tau}^{-1}\right)$ space and solves the $(\varepsilon, \delta) - L_2$ - Frequency Estimation problem. Further, it performs updates in $O\left(\log \delta^{-1} \underline{\tau}\right)$ amortized time if the stream is long enough.*

Universal Monitoring: Universal Monitoring (UnivMon) [39] supports versatile measurements of multiple metrics within a single unified hierarchical sketch. Specifically, UnivMon supports many measurements such as L_2-heavy hitters [8], [40], Count Distinct [41], [42], Entropy [43], [44] and Change Detection [45]. UnivMon is composed of $\log |\mathcal{U}|$ instances of an L_2 algorithm, each applied to a different substream of the incoming packets. Its implementation utilizes Count Sketches for providing the L_2 guarantee. By replacing each Count Sketch with the algorithm suggested in Section VII, we get an accelerated version of UnivMon. No additional analysis is necessary since our algorithm provides the same guarantees as the original Count Sketch does (while requiring slightly more space).

Hierarchical Heavy Hitters Hierarchical Heavy Hitters (HHH) are used to identify *Distributed Denial of Service (DDoS)* attacks. In this scenario, many attacking devices attempt to bring down an Internet service. However, each device transmits a moderate amount of traffic and is not a heavy hitter. HHH is motivated by observations that attacking devices often share networks and sub-networks. Thus, it detect such attacks by detecting networks that suddenly transmit massive amounts of traffic [46]–[48].

Some HHH algorithms [37], [49]–[51] are composed of multiple heavy hitters algorithms as building blocks. Thus, they can be accelerated by Skipper. The state of the art method (RHHH) [49], utilizes a variant of uniform sampling to meet the line rates. It requires a large amount of traffic to provide accuracy guarantee, whereas using Skipper provides accuracy guarantee from the start and gradually improves throughput.

VIII. DISCUSSION

Our work is motivated by the increasing popularity of NFV, which drives the need for efficient software-based measurement techniques. We suggested a general black-box acceleration approach that increases the throughput of existing measurement algorithms for a variety of measurement tasks. Specifically, we showed algorithms for per-flow frequency estimation, per-flow volume estimation, L2-Heavy Hitters, Hierarchical Heavy Hitters, and for Universal Monitoring (UnivMon). Further, UnivMon is a unified sketch that concurrently solves a broad set of measurement problems including Count Distinct, Entropy, and Change Detection.

Our approach is based on non-uniform sampling where the sampling probability is dynamically adjusted as the measurement progresses, which has distinct advantages over uniform samples. For per-flow frequency estimation, our method samples asymptotically fewer packets than the uniform sampling method provided by applied tools such as sFlow [18] and sampled Netflow [17]. In the per-flow volume estimation problem, the accuracy of uniform samples depends on the packets' size variance which is an uncontrollable and workload dependent parameter. In contrast, our method has no such limitations. Further, it is a strict improvement of a previous non-uniform method [29] as it allows for more flexible sampling probabilities. Markedly, by accelerating UnivMon, our work allows for faster measurements for counting distinct elements and entropy – two problems that do not to admit *any* constant factor approximation given a uniform sample [7]! That is the use of non-uniform samples allows us to accelerate metrics that cannot be approximated from uniform samples.

We implemented a prototype of our algorithm for the fundamental problem of per-flow frequency estimation problem,

presented rigors mathematical analysis and an extensive evaluation over real packet traces. We also extended Open vSwitch (OVS) to quantify the throughput increase obtainable by applying our approach for software measurement. Our results show that by switching to the accelerated variants, the measurements throughout overheads are reduced 5-fold to mere 4%.

Our work allows maximum code reuse of legacy algorithms. Specifically, we use the existing code as a black-box without requiring fundamental changes. This capability is likely to shorten development time, testing time and integration time. We believe that it makes our method accessible for many network applications. Finally, we released our code as an open source library [52] to benefit the community.

Acknowledgements: This research is partially supported by the Cyber Security Research Center at Ben-Gurion University; the Zuckerman Institute; the Technion Hiroshi Fujiwara Cyber Security Research Center; and the Israel Cyber Bureau.

REFERENCES

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *ACM CoNEXT*, 2011.
[2] P. Garcia-Teodoro, J. E. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers and Security*, pp. 18–28, 2009.
[3] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," in *IEEE INFOCOM 2015*.
[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "Conga: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM 2014*.
[5] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network functions virtualization: Challenges and opportunities for innovations."
[6] N. Hua, B. Lin, J. J. Xu, and H. C. Zhao, "Brick: A novel exact active statistics counter architecture," ser. ACM/IEEE ANCS, 2008, pp. 89–98.
[7] A. McGregor, A. Pavan, S. Tirthapura, and D. P. Woodruff, "Space-efficient estimation of statistics over sub-sampled streams," *Algorithmica*, 2016.
[8] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *EATCS ICALP*, 2002.
[9] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, 2005.
[10] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," in *ACM SIGMETRICS*, 2008.
[11] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Trans. on Networking*, 2012.
[12] M. Chen and S. Chen, "Counter tree: A scalable counter architecture for per-flow traffic measurement," in *IEEE ICNP*, 2015, pp. 111–122.
[13] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB*, 2002.
[14] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions Database Systems*, 2003.
[15] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *ICDT*, 2005.
[16] Q. Huang, X. Jin, P. P. C. Lee, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *ACM SIGCOMM*, 2017.
[17] "Random sampled netflow," cisco IOS Software Release 12.2(33)XNE.
[18] "Sflow's homepage, http://sflow.org."
[19] A. Andoni, R. Krauthgamer, and K. Onak, "Streaming algorithms via precision sampling," in *IEEE FOCS*, 2011.
[20] N. Duffield, "Fair sampling across network flow measurements," *ACM SIGMETRICS*, 2012.
[21] Z. Z. W. B.-q. Chen and S.-q. Z. Ke, "A mechanism of identifying heavy hitters based on multi-dimensional counting bloom filter," *JEIT 2010*.
[22] L. Yang, W. Hao, P. Tian, D. Huichen, L. Jianyuan, and L. Bin, "Case: Cache-assisted stretchable estimator for high speed per-flow measurement," in *IEEE INFOCOM*, 2016.
[23] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *IEEE INFOCOM*, 2016.
[24] G. Cormode and M. Hadjieleftheriou, "Methods for finding frequent items in data streams," *J. VLDB*, 2010.
[25] P. Hick, "CAIDA Anonymized 2016 Internet Trace, equinix-chicago 2016-02-18 13:00-13:05 UTC, Direction A."
[26] ——, "CAIDA Anonymized 2014 Internet Trace, equinix-sanjose 2013-06-19 13:00-13:05 UTC, Direction B."
[27] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Randomized admission policy for efficient top-k and frequency estimation," in *IEEE INFOCOM*, 2017.
[28] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *IEEE INFOCOM*, 2017.
[29] E. W. G. Einziger, Marcelo Caggiani Luizelli, "Constant time weighted frequency estimation for virtual network functionalities," in *IEEE ICCCN*, 2017.
[30] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *VLDB*, vol. 1, no. 2, pp. 1530–1541, Aug. 2008.
[31] B. Manes, "Caffeine: A high performance caching library for java 8," *https://github.com/ben-manes/caffeine*.
[32] L. Rizzo, "Netmap: A novel framework for fast packet i/o," in *USENIX ATC*, 2012.
[33] "Intel dpdk, http://dpdk.org/," accessed: 04-19-2016.
[34] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *NSDI*, 2015.
[35] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *IMC*, 2015.
[36] D. Anderson, P. Bevan, K. Lang, E. Liberty, L. Rhodes, and J. Thaler, "A high-performance algorithm for identifying frequent items in data streams," in *ACM IMC 2017*.
[37] R. Ben Basat, G. Einziger, and R. Friedman, "Fast Flow Volume Estimation," in *ACM ICDCN*, 2018.
[38] B.-Y. Choi, J. Park, and Z.-L. Zhang, "Adaptive random sampling for load change detection," *ACM SIGMETRICS*, pp. 272–273, 2002.
[39] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *ACM SIGCOMM*, 2016.
[40] V. Braverman, S. R. Chestnut, N. Ivkin, J. Nelson, Z. Wang, and D. P. Woodruff, "Bptree: An l2 heavy hitters algorithm using constant memory," in *ACM PODS*, 2017.
[41] E. Cohen, "All-distances sketches, revisited: HIP estimators for massive graphs analysis," *IEEE Trans. Knowl. Data Eng.*, 2015.
[42] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *ACM EDBT*, 2013.
[43] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," *ACM SIGMETRICS Perform. Eval. Rev.*, 2006.
[44] E. Assaf, R. Ben-Basat, G. Einziger, and R. Friedman, "Pay for a sliding bloom filter and get counting, distinct elements, and entropy for free," in *IEEE INFOCOM*, 2018.
[45] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *ACM IMC*, 2003.
[46] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," *ACM SIGCOMM CCR*, 2015.
[47] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system." in *USENIX ATC*, 2006.
[48] R. B. Basat, G. Einziger, I. Keslassy, A. Orda, S. Vargaftik, and E. Waisbard, "Memento: Making sliding windows efficient for heavy hitters," in *ACM CoNEXT*, 2018.
[49] R. Ben-Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *ACM SIGCOMM*, 2017.
[50] M. Mitzenmacher, T. Steinke, and J. Thaler, "Hierarchical Heavy Hitters with the Space Saving Algorithm," in *ALENEX*, 2012.
[51] R. B. Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Volumetric hierarchical heavy hitters," in *IEEE MASCOTS)*, 2018.
[52] "Skipper C++ code, see https://gist.github.com/anonymous/cd7f48dbdd386261d493f3eecceb7b7a."