Enabling a “RISC” Approach for Software-Defined Monitoring using Universal Streaming

Zaoxing Liu [†], Greg Vorsanger[†], Vladimir Braverman[†], Vyas Sekar[⋆]
[†] Johns Hopkins University [⋆]Carnegie Mellon University

ABSTRACT

Network management relies on an up-to-date and accurate view of many traffic metrics for tasks such as traffic engineering (e.g., heavy hitters), anomaly detection (e.g., entropy of source addresses), and security (e.g., DDoS detection). Obtaining an accurate estimate of these metrics while using little router CPU and memory is challenging. This in turn has inspired a large body of work in data streaming devoted to developing optimized algorithms for individual monitoring tasks, as well as recent approaches to make it simpler to implement these algorithms (e.g., OpenSketch). While this body of work has been seminal, we argue that this trajectory of crafting special purpose algorithms is untenable in the long term. We make a case for a "RISC" approach for flow monitoring analogous to a reduced instruction set in computer architecture—a simple and generic monitoring primitive from which a range of metrics can be computed with high accuracy. Building on recent theoretical advances in *universal streaming*, we show that this “holy grail” for flow monitoring might be well within our reach.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations—*Network Monitoring*

General Terms

Algorithms, Measurement, Design

1 Introduction

Network management requires accurate and timely statistics on a wide range of traffic measurements. Metrics such as the flow size distribution [29], heavy hitters [4], entropy measures [30, 42] are used for a range of management applications ranging from traffic engineering [5, 26], attack and anomaly detection [40], and forensic analysis [37].

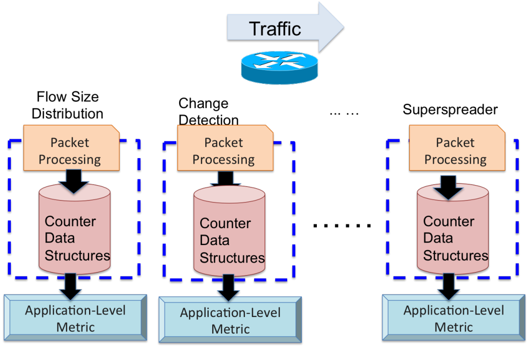


Figure 1: Sketch-based traffic monitoring requires careful algorithm and data structure design customized for each measurement task

Given the limited memory and computation power available for such measurements on routers, some form of sampling is inevitable. In this respect, there are two classes of approaches by which these metrics can be obtained. The first class of approaches rely on *generic flow monitoring* which may be deterministic (e.g., counters for active flows in OpenFlow [32]) or use classical packet sampling (e.g., NetFlow [16]). While generic flow monitoring is good for coarse-grained visibility, it provides poor accuracy for more fine-grained metrics [6, 22, 23, 33]. This has spawned a rich body of work in *sketching* or *streaming* methods, where sophisticated data streaming algorithms with provable resource-accuracy tradeoffs are designed for specific metrics of interest (e.g., [8, 9, 11, 23, 24, 28, 31, 33]).

A key practical challenge that has hindered the adoption of sketch-based algorithms is that each measurement task of interest requires carefully crafted algorithms. As the number of monitoring tasks grows, sketch-based monitoring requires significant investment in algorithm design and hardware upgrades to support new metrics of interest (Figure 1). While recent approaches like OpenSketch [38] provide libraries to reduce the implementation effort, they do not address the fundamental need to design new custom sketches for each task, nor do they provide a roadmap to support new sketching algorithms that require functions outside the library.

To break away from this undesirable trajectory of having to craft a custom algorithm for each monitoring task, we argue that we need a capability akin to “RISC” architectures or

HotNets ’15 November 16–17 2015, Philadelphia, PA USA
Copyright 2015 ACM 978-1-4503-4047-2 ...$15.00.
DOI: http://dx.doi.org/10.1145/2834050.2834098.

a reduced instruction set. Our vision is a UNIVMON (Universal Monitoring) architecture, where a simple and generic monitoring primitive runs on a router. Based on the data structures this UNIVMON primitive collects, we should be able to obtain high accuracy estimates for a broad spectrum of monitoring tasks, such that the accuracy is equivalent (if not better) than running custom sketches for each task running with comparable compute and memory resources.

To date, however, this vision has been an elusive open question in both theory [27] (Question 24) and in practice [35]. However, recent theoretical advances in *universal streaming* have, for the first time in 20 years of research in sketching algorithms, suggested a promising approach for creating a single universal sketch that is provably accurate for a broad spectrum of monitoring applications [7, 10, 13].

Our contribution in this paper is to bring to light these recent theoretical advances to the networking community and to show that universal streaming can be a viable basis for the UNIVMON vision. To this end, we provide the first implementation and evaluation of the universal streaming primitives from prior work [7, 10, 13]. We demonstrate that several canonical flow monitoring tasks [24, 28, 30] are naturally amenable to universal streaming. Finally, we use a trace-driven analysis to quantitatively show that the UNIVMON approach can provide comparable accuracy and space requirements compared to custom sketches.

The generality of universal streaming essentially delays the binding of data plane computation and primitives to specific monitoring tasks, and thus UNIVMON provides a truly software-defined monitoring approach that can fundamentally change network monitoring analogous to SDNs. If our research agenda succeeds, then it will naturally motivate hardware vendors to rally around and develop optimized hardware implementations, in the same way that a minimal data plane in the (original) SDN architecture was key to get vendor buy-in [25, 32]. Finally, this agenda opens up a range of exciting new research directions in both theory and practice: What other network monitoring tasks can we cover? Are there distributed versions of universal streaming [35]? Can we do universal streaming over multidimensional network aggregates [17, 41]? Can we use this primitive to adaptively zoom in to subspaces of interest [39]?

2 Background and Overview

In this section, we briefly review related work in the areas of streaming algorithms and network monitoring before outlining the overarching vision of the UNIVMON approach.

2.1 Related Work

Many network monitoring and management applications depend on sampled flow measurements from routers (e.g., NetFlow or sFlow). While these are useful for coarse-grained metrics (e.g., total volume) they do not provide good fidelity unless these are run at very high sampling rates, which is undesirable due to compute and memory overhead.

This inadequacy of packet sampling has inspired a large body of work in data streaming or sketching. This derives from a rich literature in the theory community on streaming algorithms starting with the seminal "AMS" paper [3] and has since been an active area of research (e.g., [10, 15, 18, 20]). At the high level, the problem they address is as follows: Given an input sequence of items, the algorithm is allowed to make a single or constant number of passes over the data stream while using sub-linear, usually poly-logarithmic space compared to the storage of the data to estimate desired statistical properties of the stream (e.g., mean, median, frequency moments, and so on). Streaming is a natural fit for network monitoring and has been applied to several measurement tasks including heavy hitter detection [24], entropy estimation [31], and change detection [28].

A key limitation that has stymied the practical adoption of streaming approaches is that the measurements from each algorithm are tightly coupled to the intended metric of interest. This forces vendors to invest time and effort in building specialized hardware components without knowing if these will be useful for their customers. Given the limited resources available on network routers and business concerns of router vendors, it is difficult to support a wide spectrum of monitoring tasks in the long term.

The efforts closest in spirit to our UNIVMON vision is the minimalist monitoring work of Sekar et al. [36] and OpenSketch by Yu et al., [38]. Sekar et al, showed empirically that flow sampling and sample-and-hold [25] can provide comparable accuracy to sketching when equipped with similar resources. However, this work is purely trace-driven and offers no analytical basis for this observation and does not provide guidelines on what metrics are amenable to this approach. OpenSketch [38] addresses an orthogonal problem of making it easier to implement sketches. Here, the router is equipped with a library of predefined functions in hardware (e.g., hash-maps or count-min sketches [18]) and the controller can reprogram these as needed for different tasks. While OpenSketch reduces the implementation burden, it does not address the problem that many concurrent sketch instances are needed to perform a diverse set of monitoring tasks and that future tasks may fall outside the library.

In summary, prior works present a fundamental dichotomy: generic approaches that offer poor fidelity and are hard to reason about analytically vs. sketch-based approaches that offer good guarantees but are practically intractable given the wide range of monitoring tasks of interest.

2.2 UNIVMON Vision

Drawing an analogy to reduced instruction set architecture, our overarching research agenda is to develop a truly "software-defined" monitoring approach called UNIVMON as shown in shown in Figure 2. In the *data plane*, UNIVMON employs an online streaming primitive that populates a compact data structure of counters. The key difference from traditional streaming is that the UNIVMON data plane delays the binding between the work done in the online stage and the ultimate measurement task. That is, given this set of counters

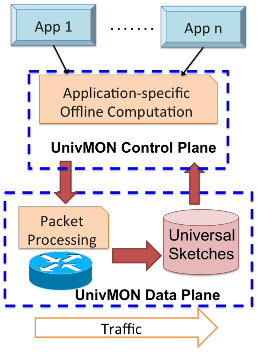


Figure 2: The architecture of Universal Monitoring

we can support a broad spectrum of statistical queries. In the *control plane*, UNIVMON periodically retrieves the counters being maintained by the data plane and then uses a set of *estimation* functions. These functions essentially translate the specific query of interest into a specific computation on the counters. Note that since this is "offline", we can potentially use more compute power in the control plane.

Given this vision, there are two fundamental challenges:

1. Do such data plane and control plane primitives exist to support this vision?

2. Will this architecture be comparable in terms of accuracy and overhead relative to custom sketches?

The answer to both questions, surprisingly, is yes. For (1), we show that recent advances in *universal streaming* can serve as an enabler for the UNIVMON vision. For (2), we show a preliminary trace-driven evaluation that empirically confirms that UNIVMON can offer comparable accuracy-resource tradeoffs vs. custom sketches.

3 UNIVMON Design

In this section, we discuss recent theoretical advances in universal streaming [7, 10, 13] and how this can serve as a basis for UNIVMON. We also show how several classical network measurement tasks are amenable to this approach.

3.1 Theory of Universal Streaming

For the following discussion, we consider an abstract data stream $D(m, n)$ which is a sequence of length m with n unique elements. Let f_i denote the frequency of the i-th unique element in the stream.

The intellectual foundations of many streaming algorithms can be traced back to the celebrated lemma by Johnson and Lindenstrauss [19]. This shows that N points in Euclidean space can be embedded into another Euclidean space with an exponentially smaller dimension while approximately preserving the pairwise distance between the points. Alon, Matias, and Szegedy used a variant of the Johnson Lindenstrauss lemma to compute the second moment (or the L_2 norm of a frequency vector) in the streaming model [3]. Namely, they approximate with high probability the function $\sum_i f_i^2$, while only using a small (poly logarithmic) amount of memory.

A natural question in a quest for universal streaming is whether such methods [3] can be extended to more general statistics of the form $\sum g(f_i)$. We denote this statistic as the function $G\text{-}sum$.

Class of *Stream-PolyLog* Functions: We begin by characterizing the class of $G\text{-}sum$ functions amenable to streaming. Informally, streaming algorithms using a polylogarithmic amount of space complexity, are known to exist for $G\text{-}sum$ functions, where g is monotonic and upper bounded by the function $O(f_i^2)$ [7, 10]. This is an informal explanation; the precise characterization is more technically involved and can be found in [10].[1] Note that this only suggests that *some custom* sketching algorithm exists if $G \in$ *Stream-PolyLog*; this makes no claim of universality.

Intuition Behind Universality: The surprising recent theoretical result of universal sketches is that for any function $g()$ where $G\text{-}sum$ belongs to the class *Stream-PolyLog* defined above can now be computed by using a *single universal sketch*.

The intuition behind universality stems from the following argument about heavy hitters in the stream. Informally, f_i is a heavy hitter if changing its value significantly affects the $G\text{-}sum$ value as well. For instance, consider the frequency vector $(\sqrt{n}, 1, 1, \ldots, 1)$; here f_1 is the L_2 heavy hitter. Let us call the set of these g-heavy elements the $G\text{-}core$.

Now, let us consider two cases:

1. There is one sufficiently large g-heavy hitter in the stream: Now, if the frequency vector does have one (sufficiently) large heavy hitter, then most of mass is concentrated in these value. Thus, we can simply use the output of some algorithm for detecting heavy hitters and this can estimate $G\text{-}sum$. It can be shown that a heavy hitter for a (computable) g is also a heavy hitter for the L_2 norm [7, 10]. Thus, to compute $G\text{-}core$ for g, we simply need to find L_2 heavy hitters. In our implementation, we use the Count Sketch [15] algorithm to find L_2 heavy hitters.

2. There is no single g-heavy hitter in the stream; i.e., no single element contributes significantly to the $G\text{-}sum$: When there is single large heavy hitter, it can be shown that $G\text{-}sum$ can be approximated w.h.p. by finding heavy hitters on a series of sampled *substreams* of increasingly smaller size. The exact details can be found in [10] but the main intuition comes from standard tail bounds such as Chernoff or Hoeffding. Each substream is defined recursively by the substream before it, and is created by sampling the previous frequency vector by replacing each coordinate of the frequency vector with a zero value with probability 0.5. Repeating this procedure k times reduces the dimensionality of the problem by a factor of 2^k. Then, summing across heavy hitters of all these recursively defined vectors, we create a single "recursive sketch' which gives a good estimate of $G\text{-}sum$ [12].

[1] While streaming algorithms are also known for functions that grow asymptotically faster than $g = f_i^2$ [8] they cannot be computed in polylogarithmic space due to the lower bound from [14].

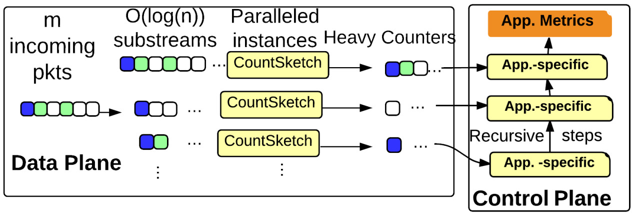


Figure 3: High-level view of the universal sketch construction that inspires the UnivMon Design

Universal Sketch Construction: Combining the ideas for the two cases described above, we now have the following universal sketch construction. In the online stage, we maintain $\log(n)$ parallel copies of Count Sketch, one for each substream as described in case 2 above. Observe that the number of unique elements in the input stream for each copy of Count Sketch is expected to be half the number of elements of the Count Sketch before it. This is shown in the "Data Plane" section of Figure 3. In the offline estimation stage, we use the output of these Count Sketches and applying the function g we wish to approximate, we are able to obtain our result as shown in the "Control Plane" section of Figure 3.

We can show that with this is in fact a *universal* sketch construction that can efficiently estimate any function in *Stream-PolyLog*. The proof of this theorem is outside the scope of this paper and we refer readers to the previous work of Braverman et al [10].

3.2 UnivMon Data Plane

The UnivMon data plane essentially follows the structure of the online phase described above. We maintain many parallel instances of our Count Sketch data structure. Next, we describe the procedure of the algorithm for the j-th instance, but the procedure is the same for instances.

For instance j, the algorithm processes incoming 5-tuples and uses an array of j hash functions $h_i : [n] \to \{0, 1\}$ to decide whether or not to sample the tuple. When 5-tuple tup arrives in the stream, if for all h_1 to h_j, $h_i(tup) = 1$, then the tuple is added to D_j, the sampled sub-stream. Otherwise, the value is thrown out. Then, for sub-stream D_j, we run an instance of Count Sketch as shown in Algorithm 1, and visualized in Figure 3. This creates substreams of decreasing lengths as the j-th instance is expected to have all of the hash functions agree to sample half as often as the $(j-1)$-th instance. This simple data structure is all that is required for the online computation portion of our algorithm. The output of the Count Sketches is then aggregated in the control plane.

Algorithm 1 UnivMon Data Plane Algorithm

```
1. Generate $\log(n)$ pairwise independent hash functions
   $h_1 \dots h_{\log(n)} : [n] \to \{0, 1\}$.
2. For $j = 1$ to $\log(n)$, in parallel:
   (a) when a packet $a_i$ in $D$ arrives, if all
       $h_1(a_i) \dots h_j(a_i) = 1$, sample and add $a_i$ to $D_j$.[2]
   (b) run Count Sketch on $D_j$.
3. Output heavy hitter counters $Q_j$ from Count Sketch.
```

Algorithm 2 UnivMon Control Plane Algorithm

```
1. Construct a set of $Q'_j$ with entries $w'_j(i) = g(w_j(i))$.
2. Compute $Y_{\log(n)} = |Q'_{\log(n)}|$.
3. For each $j$ from $\log(n) - 1$ to 0:
       Compute $Y_j = 2Y_{j+1} + \sum_{i \in Q'_j} (1 - 2h_{j+1}(i)) w'_j(i)$
4. Return $Y_0$.
```

3.3 Control Plane Design

The UnivMon control plane's goal is to estimate different *G-sum* functions of interest. The algorithm used by the control plane to aggregate the various sampled sketches for a given *G-sum* is shown in Algorithm 2. This method is based on the Recursive Sum Algorithm from [12].

The input to the control plane algorithm is the output of the data plane from Algorithm 1; i.e., a set of $\{Q_j\}$ buckets maintained by the Count Sketch from parallel instance j. Let $w_j(i)$ be the counter of the i-th bucket in Q_j. $h_j(i)$ is the hash of the value of the i-th bucket in Q_j corresponding to the hash function for instance j as instantiated in Algorithm 1. The output of Algorithm 2 is an unbiased estimator of *G-sum*.

In this algorithm, each Y is recursively defined, with Y_j is function g applied to each bucket of Q_j, the Count Sketch for substream D_j, and the sum taken on these values. Note that $Q_{\log(n)}$ is the sparsest substream, and we begin by computing $Y_{\log(n)}$. Thus, Y_0 can be viewed as computing *G-sum* in parts using these sampled streams.

3.4 Application to Network Measurement Tasks

As discussed earlier, if a function $G \in$ *Stream-PolyLog*, then it is amenable to estimation via the universal sketch. Here we show that a range of network measurement tasks can be formulated via suitable some $G \in$ *Stream-PolyLog*, and thus can be efficiently captured using the UnivMon approach. The network traffic is a stream $D(n, m)$ with m packets and at most n unique flows. For simplifying the discussion, we consider the case where all flows have the same destination address; however, this does not impact correctness in the general case.

Heavy Hitters: To detect 'heavy hitters' in the network traffic, our goal is to identify the flows that consume more than a fraction α of the total capacity [25]. Consider a function $g(x) = x$ such that the corresponding *G-core* outputs a list of heavy hitters with$(1 \pm \epsilon)$-approximation of their frequencies. For this case, G is in *Stream-PolyLog* and we have an algorithm that provides *G-core*. This is technically special case of the universal sketch; we are not ever computing a

[2] In this way, we obtain $\log(n)$ streams $D_1, D_2 \dots D_{\log(n)}$; i.e., for $j = 1 \dots \log n$, the number of unique items n in D_{j+1}, is expected to be half of D_j

G-sum function and using G-$core$ directly in all cases.

DDoS: Suppose we want to identify if a host X that may be experiencing a Distributed Denial of Service (DDoS) by checking if more than k unique flows from different sources are communicating with X [38]. To show that the simple DDoS problem is solvable by the universal sketch, consider a function g that $g(x) = x^0$. Here g is upperbounded by $f(x) = x^2$ and sketches already exist to solve this exact problem. Thus, we know G is in *Stream-PolyLog* and we approximate G-sum in polylogarithmic space using the universal sketch. In terms of interpreting the results of this measurement, if G-sum is estimated to be larger than k, a specific host is a potential DDoS victim.

Change Detection: Change detection is the process of identifying flows that contribute the most to traffic change over two consecutive time intervals. As this computation takes place in the control plane, we can store the output of the universal sketches from multiple intervals without impacting online performance.

Consider two adjacent time intervals t_A and t_B. If the volume for a flow x in interval t_A is $S_A[x]$ and $S_A[x]$ over interval t_B. The difference signal for x is defined as $D[x] = |S_A[x] - S_B[x]|$. A flow is a heavy change flow if the difference in its signal exceeds ϕ percentage of the total change over all flows. The total difference is $D = \sum_{x\in[n]} D[x]$. A flow x is defined to be a heavy change key iff $D[x] \geq \phi \cdot D$. The task is to identify these heavy change flows. We assume the size of heavy change flows is above some threshold T over the total capacity c.

We can show that the heavy change flows are L_1 heavy hitters on interval t_A $(a_1 \cdots a_{n/2})$ and interval t_B $(b_1 \cdots b_{n/2})$, where $L_1(t_A, t_B) = \sum |a_i - b_i|$. L_1 norm is in *Stream-PolyLog*, and G-sum/G-$core$ can be solved by universal sketch. The G-sum outputs the estimated size of the total change D and G-$core$ outputs the possible heavy change flows. By comparing the outputs from G-sum and G-$core$, we can detect and determine the heavy change flows that are above some threshold of all flows.

Entropy Estimation: We define entropy with the expression $H \equiv -\sum_{i=1}^{n} \frac{f_i}{m}\log(\frac{f_i}{m})$ [30] and we define $0\log 0 = 0$ here. The entropy estimation task is to estimate this H for source IP addresses (but could be performed for ports or other features).

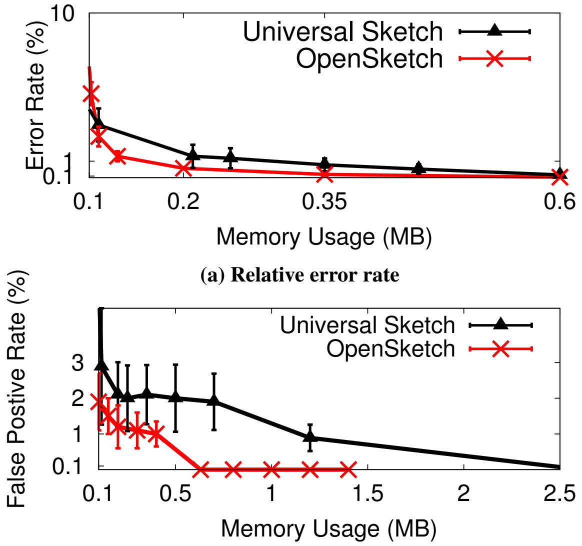
To compute the entropy, $H = -\sum_{i=1}^{n} \frac{f_i}{m}\log(\frac{f_i}{m}) = \log(m) - \frac{1}{m}\sum_i f_i \log(f_i)$. As m can be easily obtained, the difficulty lies in calculating $S = \sum_i f_i \log(f_i)$. Here the function $g(x) = x\log(x)$ is bounded by $g(x) = x^2$ and is sketchable, thus it is in *Stream-PolyLog* and S can be estimated by universal sketch.

4 Preliminary Evaluation

In this section, we present a preliminary trace-driven evaluation comparing the performance of our UNIVMON approach vs. specialized sketches.

Setup: We have implemented (in software) the key algorithms and data structures for the *online* and *offline* components of UNIVMON. This is a preliminary and unoptimized implementation and there is room for much improvement in memory and CPU consumption. We have implemented the offline translation functions for four monitoring tasks: Heavy Hitter detection (HH), DDoS detection (DDoS), Change Detection (Change), and Entropy Estimation (Entropy). We compare against corresponding optimized custom sketch implementations from the OpenSketch software library for HH, Change, and DDoS [2]. OpenSketch does not yet support Entropy and thus we do not report results for OpenSketch for Entropy. For brevity, we focus on each of these metrics computed over one feature, namely the source IP address.

For the purposes of this evaluation, we use a one-hour backbone trace collected at backbone link of a Tier1 ISP between Chicago, IL and Seattle, WA in 2015 [1]. We have repeated these experiments on other traces and found qualitatively similar results and do not report these due to space constraints. We assume that we have a single switch processing this trace and that in both cases the "controller" periodically polls the switch for the sketch every 5 seconds. That is, the memory numbers reported are roughly for a 5-second trace. Both UNIVMON and OpenSketch are randomized algorithms; we run the experiment 20 times and report the *median* and *standard deviation* over these 20 independent runs.



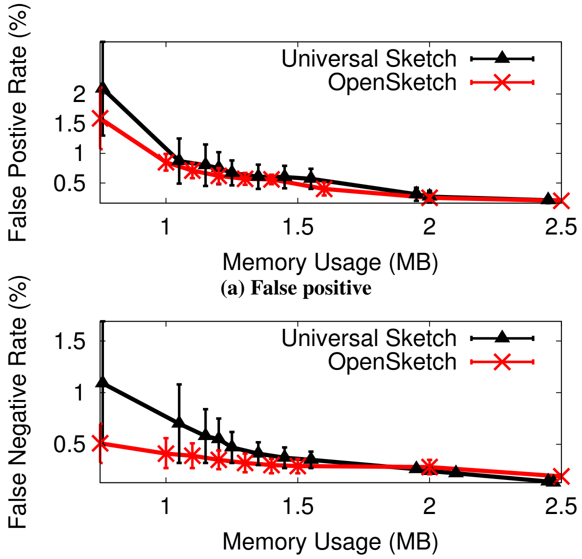
(a) Relative error rate

(b) False positive rate

Figure 4: HH result

Accuracy vs. Memory: First, we evaluate the accuracy vs. memory tradeoff of UNIVMON vs. OpenSketch for HH, DDoS, and Change. Figures 4 and 5 show the results for HH and DDoS respectively. In each case, we report the false positive and negative rates separately. For HH, we configured the sketches so that we are interested in reporting any destination that consumes more than $\alpha = 0.5\%$ of the link capacity. In both figures, we see that UNIVMON consumes slightly more memory to reach a similar error rate. The key takeaway however is that the absolute error rates are quite small for both applications and the error rates are comparable to OpenSketch once we cross 1MB. Furthermore, the small increased memory consumption of UnivMon comes at a dramatically increased flexibility and generality across the *suite* of applications. That is, OpenSketch is in effect using K-times as many resources if there are K distinct tasks as it instantiates a separate sketch when we look at the set of applications, whereas UnivMon uses a single universal sketch instance.

Figure 6 shows the error rate for Change Detection application. Here, we observe an interesting reversal of trends—UnivMon is actually better. This is because the universal streaming approach is inherently better suited to capturing "diffs" across sketches, whereas traditional sketches need to explicitly create a new complex sketch for the change detection task. Finally, Figure 7 shows that the error of UnivMon for the entropy estimation task is also quite low even with limited memory.



(a) False positive

(b) False negative

Figure 5: DDoS detection

Overhead: One concern might the computational cost of the universal streaming primitive vs. custom sketch primitives. To understand this, we used the Intel Performance Counter Monitor (PCM) [21] to evaluate various compute overhead measures (e.g., Total cycles on CPU). For data plane performance, UnivMon takes 1.407×10^9 total cycles on CPU to support all simulated applications while OpenSketch needs in total 2.941×10^9. In general, we found that the cost of OpenSketch was quite variable across applications. In the worst case, our unoptimized implementation was only 10-15% more expensive and in some cases it was more than 2X more efficient. We leave a more systematic analysis of the computation cost of universal streaming and techniques to optimize it as a direction for future work.

In summary, our preliminary evaluations are quite promising —we observe comparable or sometimes even better accuracy and overhead relative to custom sketches.

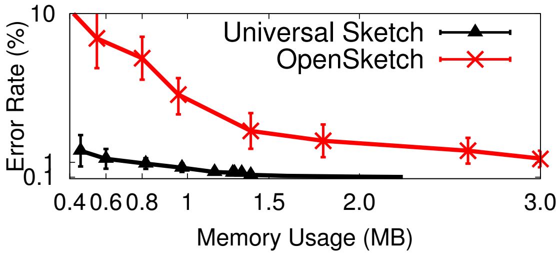


Figure 6: Change detection error

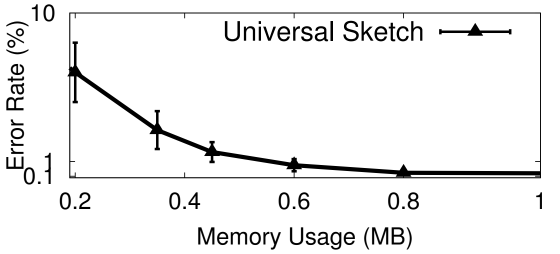


Figure 7: Entropy estimator error

5 Discussion

We conclude by highlighting a subset of new and exciting opportunities for further research that UnivMon opens up.

Reversibility: The reliance on hash functions, and a result of using hash functions with small memory algorithms, is that the 'keys' are thrown out to save space. For network monitoring tasks such as change detection, it is important to understand *which* keys have caused anomalous network traffic. A natural question is whether we can *reverse* the universal sketch using reversible hash functions [34].

Multidimensional data: We can consider the IP 5-tuple a piece of high-dimensional data and the various $n < 5$ tuples to be subsets. Ideally, we want to avoid explicitly creating a sketch per subset of features (e.g., src-dst combination). This area relates closely to hierarchical heavy hitters [17,41], and is a potential area for extending UnivMon.

Dynamic monitoring adjustments: Operators may want the ability to adjust the granularity of the measurement dynamically to adaptively zoom in on sub-regions of interest [39]. One approach may involve dynamically increasing or decreasing the size of the sketches per switch to increase resolution in certain parts of the network.

Distributed monitoring: In spite of the lightweight nature of sketches, some switches may get overloaded while other locations have a lighter load. A natural direction would be to distribute the workload across multiple devices [35].

Feasibility of hardware implementation: Finally, our current work shows that the in-software overhead of UnivMon is comparable to OpenSketch. A natural extension is to explore if this translates into efficient implementations on commodity switch hardware.

6 References

[1] The caida ucsd anonymized internet traces 2015 - sanjose dira. `http://www.caida.org/data/passive/passive_2015_dataset.xml`.

[2] Opensketch simulation library. `https://github.com/USC-NSL/opensketch`.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM.

[4] N. Bandi, A. Metwally, D. Agrawal, and A. El Abbadi. Fast data stream algorithms using associative memories. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 247–256, New York, NY, USA, 2007. ACM.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 8:1–8:12, New York, NY, USA, 2011. ACM.

[6] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 159–164, New York, NY, USA, 2006. ACM.

[7] V. Braverman and S. R. Chestnut. Universal Sketches for the Frequency Negative Moments and Other Decreasing Streaming Sums. In N. Garg, K. Jansen, A. Rao, and J. D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 591–605, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] V. Braverman, J. Katzman, C. Seidell, and G. Vorsanger. An optimal algorithm for large frequency moments using o(nˆ(1-2/k)) bits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 531–544, 2014.

[9] V. Braverman, Z. Liu, T. Singh, N. V. Vinodchandran, and L. F. Yang. New bounds for the CLIQUE-GAP problem using graph decomposition theory. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 151–162, 2015.

[10] V. Braverman and R. Ostrovsky. Zero-one frequency laws. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 281–290, New York, NY, USA, 2010. ACM.

[11] V. Braverman and R. Ostrovsky. Approximating large frequency moments with pick-and-drop sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 42–57, 2013.

[12] V. Braverman and R. Ostrovsky. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 58–70. Springer, 2013.

[13] V. Braverman, R. Ostrovsky, and A. Roytman. Zero-one laws for sliding windows and universal sketches. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 573–590, 2015.

[14] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117. IEEE Computer Society, 2003.

[15] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[16] B. Claise. Cisco systems netflow services export version 9. RFC 3954.

[17] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 464–475. VLDB Endowment, 2003.

[18] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, Apr. 2005.

[19] S. Dasgupta and A. Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, Jan. 2003.

[20] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, June 2002.

[21] R. Dementiev, T. Willhalm, O. Bruggeman, P. Fay, P. Ungerer, A. Ott, P. Lu, J. Harris, P. Kerly, P. Konsor, A. Semin, M. Kanaly, R. Brazones, and R. Shah. Intel performance counter monitor - a better way to measure cpu utilization. `https://software.intel.com/en-us/articles/intel-performance-counter-monitor`.

[22] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 325–336, New York, NY, USA, 2003. ACM.

[23] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 323–336, New York, NY, USA, 2002. ACM.

[24] C. Estan and G. Varghese. *New directions in traffic measurement and accounting*, volume 32. ACM, 2002.

[25] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 323–336, New York, NY, USA, 2002. ACM.

[26] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Trans. Netw.*, 9(3):265–280, June 2001.

[27] P. Indyk, A. McGregor, I. Newman, and K. Onak. Open problems in data streams, property testing, and related topics, 2011. *Available at: people. cs. umass. edu/ mcgregor/papers/11-openproblems. pdf*, 2011.

[28] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[29] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '04/Performance '04, pages 177–188, New York, NY, USA, 2004. ACM.

[30] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '06/Performance '06, pages 145–156, New York, NY, USA, 2006. ACM.

[31] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *ACM SIGMETRICS Performance Evaluation Review*, volume 34, pages 145–156. ACM, 2006.

[32] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[33] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani. Fast monitoring of traffic subpopulations. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 257–270, New York, NY, USA, 2008. ACM.

[34] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A. Dinda, M.-Y. Kao, and G. Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking (ToN)*, 15(5):1059–1072, 2007.

[35] 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*, volume 8, pages 233–246, 2008.

[36] V. Sekar, M. K. Reiter, and H. Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 328–341, New York, NY, USA, 2010. ACM.

[37] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang. Worm origin identification using random moonwalks. In *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*, pages 242–256. IEEE Computer Society, 2005.

[38] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 29–42, Berkeley, CA, USA, 2013. USENIX Association.

[39] L. Yuan, C.-N. Chuah, and P. Mohapatra. Progme: towards programmable network measurement. *IEEE/ACM Transactions on Networking (TON)*, 19(1):115–128, 2011.

[40] Y. Zhang. An adaptive flow counting method for anomaly detection in sdn. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 25–30, New York, NY, USA, 2013. ACM.

[41] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 101–114. ACM, 2004.

[42] H. C. Zhao, A. Lall, M. Ogihara, O. Spatscheck, J. Wang, and J. Xu. A data streaming algorithm for estimating entropies of od flows. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 279–290, New York, NY, USA, 2007. ACM.