A Feasibility Study on Time-aware Monitoring with Commodity Switches

Yiming Qiu, Kuo-Feng Hsu, Jiarong Xing, Ang Chen
Rice University

ABSTRACT

Network monitoring and measurement are important tasks for operating large-scale cloud networks. Recently, the confluence of programmable networking hardware and streaming algorithms has given rise to a class of memory-efficient algorithms that can run entirely in the switch data plane.

However, existing systems cannot support the notion of time, and therefore are oblivious to data recency. Generally, capturing *recent* events is essential for reasoning about the most relevant trends, and the same holds for network monitoring. Recent data, whether for SLA monitoring or attack detection, is more useful and actionable. The key question we consider in this paper is how to perform *time-aware monitoring on commodity switches* with programmable data planes. Our contribution is a feasibility study that: a) identifies a class of hardware-friendly algorithms for time-aware monitoring, b) customizes their key operations to the P4 model, c) develops a Tofino hardware prototype as concrete evidence, and d) obtains promising early results on real-world datasets.

CCS CONCEPTS

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

KEYWORDS

Network monitoring, Programmable data plane, Time Awareness

ACM Reference Format:
Yiming Qiu, Kuo-Feng Hsu, Jiarong Xing, Ang Chen and Rice University. 2020. A Feasibility Study on Time-aware Monitoring with Commodity Switches. In *ACM SIGCOMM 2020 Workshop on Secure Programmable Network Infrastructure (SPIN 2020) (SPIN '20), August 14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3405669.3405821

1 INTRODUCTION

Understanding traffic trends in cloud networks is a common and essential task. Traditionally, this was done via the use of Netflow records and SNMP counters, which needed aggressive down-sampling in order to keep overhead low. The recent emergence of programmable switches has enabled the adaptation of memory-efficient streaming algorithms for a wide range of monitoring tasks. For instance, algorithms like Bloom filters [6] and count-min sketches [9] can support membership tests and counting estimation, respectively; more advanced algorithms can also perform tasks like heavy-hitter detection [30], super-spreader detection [34], and change detection [28]. The resulting systems enable a set of powerful monitoring tasks by matching the right "lean algorithms" [23] with new capabilities of emerging network hardware.

Take count-min sketches (CMS) as a concrete example. Instead of keeping a distinct counter per key (e.g., IP or flow), in which case the memory requirement would grow linearly with the number of keys, a count-min sketch uses a constant amount of memory to keep count for all keys. Upon insertion, it applies a number of hash functions to the key, and uses the hash outputs as indexes to the sketch memory to increment the corresponding counters. Upon query, it retrieves values from the same indexes and returns their minimum. Theoretical results show that this can achieve high accuracy in terms of approximation; and the desired level of accuracy can be increased by the amount of memory assigned to the sketch or keeping the number of insertions low [9].

However, a notable gap is that most existing systems are oblivious to the passage of time. Consider the algorithm for count-min sketches—none of the update or query operations has any notion of time. Indeed, many of these algorithms only support a uniform level of (in)accuracy for all events, regardless of when they took place. This means that, as time goes by, the sketches will suffer from higher inaccuracy with more insertions [9]. One could apply simple remedies such as periodically resetting the sketch or using a very large memory, but these are essentially just workarounds.

Ideally, network monitoring should support time-awareness as an *intrinsic* property. Moreover, the algorithm should also be "hardware-friendly", implementable on commodity programmable switches. To understand the challenges imposed by hardware constraints, consider a class of algorithms that instantiate a sketch per time interval, allocate more memory to recent intervals, and gradually move older data to smaller sketches [25]. While conceptually simple, this cannot be easily supported by programmable switches. Copying data from one chunk of memory to another requires loops, which goes beyond the P4 programming model. Creating many sketches would also require a large number of inter-dependent match/action tables, which is challenging to support within a limited number of hardware stages.

Our contribution is a feasibility study that identifies suitable algorithms for time-aware monitoring, customizes them to the P4 model, and demonstrates a real hardware implementation. First, we identify a recent proposal from the streaming algorithms community—time-adaptive sketches [29]—as a promising candidate. This class of algorithms are particularly attractive, because they keep the sketch structures almost unchanged and rely on simple yet effective methods to be time-aware. The key idea is similar to *Dolby noise reduction* [12, 32]: when inserting a key to the sketch, it inflates the update using a pre-emphasis function; when querying from the sketch, it reverses the artificial inflation by applying a de-emphasis function to restore the values. Therefore, older events are gradually aged out over time, and recent events enjoy a higher

SPIN '20, August 14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-8041-6/20/08…$15.00
https://doi.org/10.1145/3405669.3405821

accuracy. Applying this technique to a CMS would result in time-adaptive CMS; and similar enhancements apply to other types of sketches. Building upon this starting point, we customize the algorithms for programmable switches, by carefully choosing pre- and de-emphasis functions, approximating floating point operations, discretizing timestamps, and pre-computing in software certain types of needed values. We have implemented a hardware Tofino prototype, and conducted a set of evaluations using realistic traffic workloads. We intend this feasibility study to promote more discussions in the community on time-aware monitoring.

2 TIME-AWARE MONITORING

In this section, we provide an overview on sketch-based monitoring, motivate the need for time-awareness, and introduce how we leverage recent developments in time-adaptive sketching for this goal.

2.1 Background: Sketches

Sketches are probabilistic data structures used in streaming algorithms, which can provide accurate estimates on item frequency in data streams. Compared to exact data structures, where memory consumption grows with the number of insertions, sketches use constant memory at the cost of bounded amounts of inaccuracy. These properties have proven to be a good fit for network monitoring, because network switches need to process high-speed, high-volume packet streams with very limited switch memory.

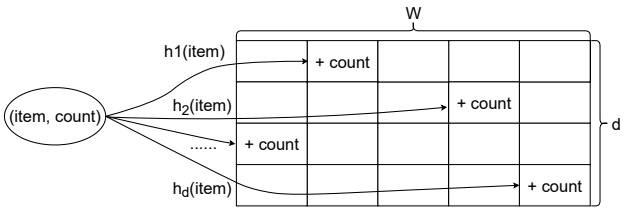


Figure 1: An example count-min sketch (CMS).

Count-min sketch (CMS) [9] is one of the most popular variants, and it can further serve as a building block for more advanced monitoring. As shown in Figure 1, it is a two-dimensional data structure consisting of d rows and w columns. Every row i in the CMS has an array of counters, as well as a distinct hash function h_i. Across rows, the hash functions $h_1 - h_d$ need to be pairwise independent [8]. Insertion of an item k to the CMS would result in d writes—one write to each of these rows. Concretely, the CMS would compute $h_i(k)$ for the i-th row, and use the hash value as the index to increment the corresponding counter. Upon a query on item k, the CMS uses the same set of hash functions to retrieve d counters and return the minimum. Since the hash functions may produce collisions, the CMS might suffer from *over-counting*. The inaccuracy of the CMS, however, has strong theoretical bounds, and it is typically low enough for many practical tasks [20, 27, 31].

Count-min sketches are particularly amenable to programmable data plane implementations. They only require simple arithmetic operations, CRC-based hash functions, and other operations that can be easily supported in the P4 model [7]; their memory-efficiency also plays a key role in network monitoring [35]. Consider a monitoring task that keeps track of the size of each TCP flow. A naïve implementation that keeps per-flow state would quickly grow beyond available switch memory, which is on the order of 10MB in modern switches [26]. Using CMS, we could instead use megabytes or even kilobytes of memory to approximate per-flow sizes with high accuracy.

2.2 The need for time-awareness

However, most existing projects on sketch-based network monitoring have a shared limitation—the monitoring tasks cannot capture the elapse of time. As typical networks remain operational for a long period of time, it is often important to understand the most *recent* network condition instead of some aggregate statistics over all past behaviors. For instance, consider pulsewave DDoS attacks [1], where an adversary generates short-lived traffic bursts to disrupt normal forwarding. Detectors that only rely on aggregate traffic volume may fail to detect high-strength bursts as a distinguishing feature. Similarly, many other low-rate DDoS attacks [18, 24] do not have anomalous statistics in aggregate. These, in fact, are just concrete instances of a more general pattern—as long as events of interest may change quickly, reports on the most recent data would be much more valuable than a simple aggregate [29]. Since network conditions are highly dynamic, explicitly capturing time properties would enable fundamentally new monitoring capabilities unavailable before.

One could design strawman solutions to approximate this task. For instance, one basic solution is to use a vanilla CMS that is completely unaware of time, but periodically dump its content per interval. This alleviates the problem that the inaccuracy of vanilla CMS increases with more insertions. However, this would require continuous data collection from the switches to the controller, which could become a bottleneck. For tasks that require fine-grained intervals (e.g., to detect millisecond-level pulses [18]), the overhead would be extremely high. A stronger strawman solution would be to allocate sketches of different sizes, dedicating larger sketches to more recent intervals [25]. When data becomes older, it gets moved to smaller and smaller sketches; and query inaccuracy would degrade over time. This approach, however, requires copying chunks of memory from one sketch to another—a task that is challenging to implement without loops. Moreover, it similarly cannot easily handle fine-grained intervals, as the number of sketches (therefore match/action tables and required stages) would quickly grow beyond what programmable switches can provide (10–20 stages). Therefore, supporting time-aware monitoring not only requires identifying the most suitable algorithm, but also customizing them to fit into the hardware constraints of programmable data planes.

2.3 Candidate: Time-adaptive sketches

We believe that a recent proposal from the streaming algorithms community is a promising starting basis: time-adaptive sketches [29]. Different from the first strawman solution, time-adaptive sketches support the notion of time explicitly. An operator, therefore, does not have to "work around" the problem by storing all data from all intervals. Also, unlike the second strawman solution, time-adaptive sketches do not require keeping multiple sketches for different intervals or moving data across sketches. The same sketch data structure will naturally remember recent trends with more accuracy.

The key idea of time-adaptive sketches is to leverage a technique similar to Dolby noise reduction [12, 32]. Upon an insertion of

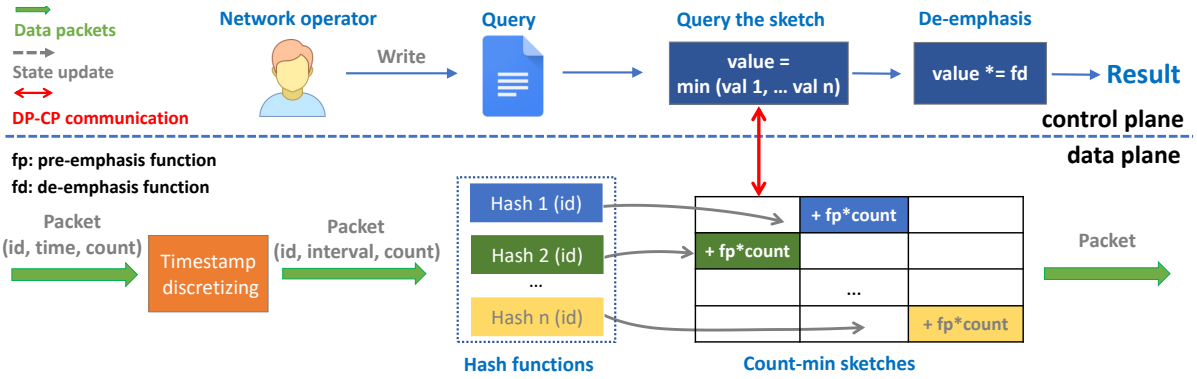


Figure 2: Customizing time-adaptive sketches to programmable data planes.

an item with count c, we apply a pre-emphasis function, which monotonically increases with larger timestamps, to the inserted count; in effect, more recent insertions are artificially enlarged and thus emphasized. Upon a query, we apply a de-emphasis function—the inverse of the previous pre-emphasis—to restore the original count. This algorithm provides theoretical guarantees on accuracy, and its accuracy increases for more recent data.

Consider a time-adaptive version of CMS, which works as follows. Suppose that we would like to insert the count for $flow_i$ at time t, which we denote as $flow_i^t$. We multiply this count with a pre-emphasis function $f(t)$, and increment the corresponding counters in the CMS using this enlarged value. When querying the count for $flow_i$ for time t, we retrieve the corresponding counters from the CMS, compute the minimum, and apply a de-emphasis function to this value. The only requirement for $f(t)$ is that it needs to be a monotonically increasing function with regard to t.

Algorithm 1: Time-adaptive count-min sketch [29]

Input: Any monotonically increasing $f(t), t \geq 0$
$w \leftarrow \lceil \frac{e}{\epsilon} \rceil$
$d \leftarrow log \frac{e}{\delta}$
$M \leftarrow d \times w$ array initialized to 0

Update $flow_i^t$:
for $j = 1$ ***to*** d **do**
$M(j, h_j(i,t)) \leftarrow M(j, h_j(i,t)) + f(t) * flow_i^t$
end

Query $flow_i^t$:
$\widehat{flow_i^t} \leftarrow min_{j \in 1,2,\ldots,d} \frac{M(j, h_j(i,t))}{f(t)}$
return $\widehat{flow_i^t}$

2.4 Customizing to switch hardware

Customizing time-adaptive sketches to programmable data planes requires several types of considerations, most of which revolve around the pre- and de-emphasis functions. We discuss the feasibility requirements one by one.

Pre-emphasis vs. de-emphasis. Our first observation is that insertions to the sketches are performed at a per-packet granularity, but queries are only occasional. In terms of hardware support, this means that the P4 data plane only needs to support the *pre-emphasis* function. The de-emphasis function, on the other hand, could reside in software (e.g., either at the switch control plane, or at a central controller). The querier can process the (pre-emphasized) counts from the hardware sketches, and then apply the de-emphasis function as a postprocessing step.

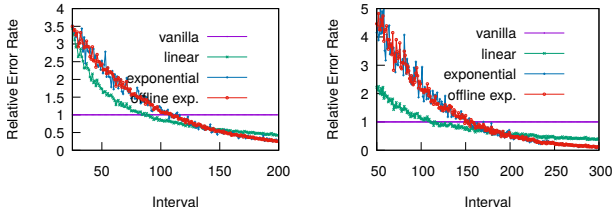
Linear vs. exponential. Depending on the choice of pre-emphasis functions, there are *linear* and *exponential* variants of time-adaptive sketches. The first class of variants use some linear function $f(t) = a \times t$, and the second class of variants use some exponential function $f(t) = a^t$; in both cases, a is a pre-defined constant for emphasis and de-emphasis. The common feature across both choices is that measurements at larger (therefore more recent) timestamps will be naturally emphasized more than those at smaller (therefore more distant) timestamps. Since programmable data planes do not support sophisticated operations such as exponentiation or multiplication, linear variants are more natural candidates. Moreover, instead of supporting arbitrary a, we can restrict the choices to the powers of two. Then we can easily support $a \times t$ using simple bitshifts in data plane.

Floating point numbers. Programmable data planes also lack support for floating point numbers. Therefore, our design approximates these functions in today's hardware by *discretizing* the timestamps. Instead of using wallclock timestamps, we use discrete logical steps that can be computed from the switch timer (e.g., one step per millisecond or one million packets). Supporting exponential functions a^t is more complex: we need to precompute a^t in advance for a range of discrete steps, and install approximate values in a match/action table; at runtime, insertions will first index this table using t and retrieve a^t for the pre-emphasis.

Enhancing the existing time-adaptive sketches with the above support allows us to arrive at a feasible hardware implementation. Figure 2 shows the high-level architecture of our design, where the control plane runs in software, and the data plane is P4-programmable hardware.

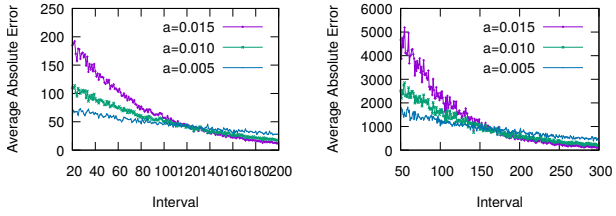
3 EMPIRICAL EVIDENCE

To demonstrate feasibility, we have implemented a time-adaptive CMS in two versions of P4: a) a P4_16 version that runs in Mininet v2.3.0; and b) a Tofino version that runs in hardware switches. Unless otherwise noted, we have used $a = 1$ for the linear function and



(a) DCTCP dataset (b) VL2 dataset

Figure 3: Relative error rate over time for different sketches on DCTCP and VL2; software implementation.



(a) CAIDA dataset (b) VL2 dataset

Figure 4: Choice of parameters a in exponential time-adaptive count-min sketch; software implementation.

$a = 1.015$ for the exponential function (for which we pre-compute the values as match/action entries). For each set of results below, we note whether they are obtained using hardware or software versions. In addition, we have also obtained the ground truth by analyzing the packet traces using exact counting without sketch-based approximation as further comparison.

We have downloaded the CAIDA Anonymized Internet Traces 2019 Dataset from an Equinix-NYC Internet router, using its first 6M TCP/UDP packets; and we have also used the DCTCP [2] and VL2 [14] workloads, generating 2.5M and 13M packets, respectively. We compare the performance of linear time-adaptive CMS, exponential time-adaptive CMS, and the vanilla count-min sketch, using the same amount of memory for each implementation. For the hardware version, the discretization relies on wallclock timestamps (2 seconds per interval); for the software version, since the processing speed is much slower and is not reflective of highspeed networks, we use the number of packets processed as the logical intervals (12k-30k packets per interval depending on the traces). We have adopted two metrics from the original design of time-adaptive sketch [29]. The first metric is average absolute error, which is computed by first obtaining the difference between a sketch estimate and the ground truth (exact count) for the i-th interval, i.e., $\widehat{c_i} - c_i$, and then computing an average across all intervals. The second metric is relative error rate, which is computed by dividing the average absolute error of the time-aware sketch by that of the vanilla CMS.

3.1 Performance comparisons

First, we compare the performance of time-adaptive CMS with the vanilla version, using the DCTCP and VL2 datasets. We queried the top-20 heavy hitters for each interval, and compared the results with the ground truths.

As Figure 3 shows, across the two datasets, time-adaptive CMS can consistently provide higher accuracy for recent trends. As tradeoff, this comes at the cost of lower accuracy for historical data. For the most recent interval, time-adaptive CMS achieves 3-10 times higher accuracy than the vanilla version. Moreover, we found that the exponential version achieves higher accuracy than the linear version; this is because $f(t) = a^t$ with $a = 1.015$ increases faster than the linear function $f(t) = t$, therefore putting a stronger emphasis on more recent data. As a third observation, the pre-computed values for the exponential function—despite accuracy loss due to conversion to integers—have enough precision to achieve almost identical results with the "offline" baseline, which implements the exponential functions in software without precision loss.

Table 1: Resource usage of a hardware Tofino switch

	Vanilla	Linear	Exponential
Stages	6	8	7
VLIWs (%)	2.34	3.12	2.86
ALU (%)	10.42	10.42	10.42
SRAM (%)	8.65	8.65	8.96

3.2 Choices of pre-emphasis functions

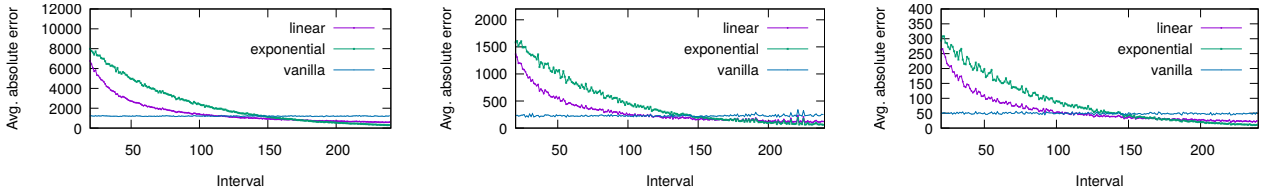
Our next set of experiments is designed to understand the choice of pre- and de-emphasis functions. Figure 4 shows different values of a for exponential functions. As we can see, a larger a in exponential time-adaptive CMS would put a stronger emphasis (de-emphasis) on more recent (older) data. This is because when we first compute $f(t_0) = a^{t_0}$, and then query the result by performing $g(t_1) = \frac{1}{a^{t_1}}$, where $t_0 < t_1$, the overall difference factor is $g(t_1) \times f(t_0) = a^{t0-t1}$. For linear sketches, we have $f(t_0) = a \times t_0$ and $g(t_1) = \frac{1}{a \times t_1}$ and the difference is $f(t_0) \times g(t_1) = t_0/t_1$; the coefficient a does not contribute to the overall difference.

3.3 Memory vs. accuracy

Next, we have used the CAIDA dataset to evaluate the influence of memory usage on the accuracy of time-aware monitoring. We varied the sketch size from 64KB to 1MB, and ran top-50 heavy hitter queries for all intervals. As Figure 5 shows, time-adaptive CMS has higher accuracy at larger memory sizes, and it always outperform vanilla sketches for recent intervals. As a closer view, Figure 6 shows the results for the most recent 10 intervals: every doubling of the memory usage would roughly cut the error rate in half, which aligns with the theoretical proofs in [29]. This is because vanilla CMS achieves higher accuracy with larger memory for all intervals uniformly, but time-adaptive CMS leverages the larger memory to achieve higher accuracy for recent data.

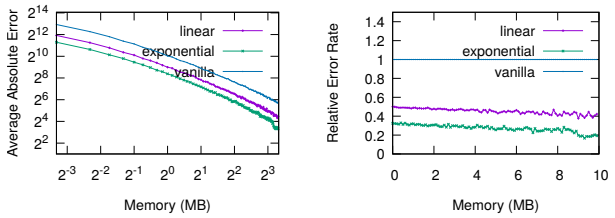
3.4 Hardware utilization

We have also measured the resource usage of vanilla CMS and the time-aware variants. As shown in Table 1, time-aware sketches only require 1–2 more stages for implementing the timing features; we note that the exponential variant uses one fewer stage, because the coefficients are precomputed in advance. For VLIWs (Very Long Instruction Words) and ALUs (Arithmetic Logic Units), which are used for arithmetic operations, we also only see modest or no increase. In terms of SRAM (static RAM), when the sketches have 2^{16} entries and each entry has 4 bytes, vanilla CMS uses 8.65%



(a) $w = 2^{12}$ (64kB) (b) $w = 2^{14}$ (256kB) (c) $w = 2^{16}$ (1MB)

Figure 5: Average absolute error over time for different sketches with various memory sizes; hardware implementation.



(a) Avg. absolute (b) Relative error

Figure 6: Results for the most recent ten intervals; software implementation.

memory while linear and exponential time-aware versions use 8.65% and 8.96% respectively.

4 RELATED WORK

Recent work on programming data planes [5, 7, 10, 33] has developed a range of network monitoring and measurement tasks in hardware [13, 16, 21, 22]. However, they do not support time-aware monitoring tasks. Existing work on time-aware data structures has used either sliding windows or their generalizations [4, 11, 15, 19], or hierarchical approaches with one sketch per interval [25].

State-of-the-art sliding window approaches [3] and their generalizations [15] require complex data structures that programmable network hardware cannot easily support. For instance, [15] proposed a set of powerful sliding window algorithms with L2 estimation guarantees. However, it extensively used Exponential Histogram which could not fit into programmable switches easily. Hierarchical approaches, on the other hand, would also create multiple sketches of different sizes for different levels of accuracy, but programmable data planes cannot easily support a large number of sketches or migrating data across sketches.

Other work has considered using hardware-supported low-pass filters (LPF) [17] or EWMA (Exponentially Weighted Moving Average) to support time-aware measurements. However, compared to sketch-based network monitoring, these mechanisms only support a single measurement key.

5 DISCUSSION

Handling overflow. Counter overflows are not unique to time-aware monitoring. However, the pre-emphasis functions would artificially enlarge counter values if we have $a >> 1$, so overflows may become more frequent. One tentative solution is to augment the time-aware sketches with auxiliary data structures that record overflows. For instance, whenever a counter is larger than a threshold, we right shift the current value by one bit, and record the number of bit shifts we have performed so far in an auxiliary counter. This causes some accuracy loss, but only in the least significant bit of each entry. When an entry needs to be updated, we first apply a right-shift to the value to be added before updating the entry. We plan to explore the feasibility of this design in a hardware implementation in the future.

SmartNICs and FPGAs. We have mostly focused on leveraging programmable switch hardware for time-aware monitoring, assuming the P4 programming model. However, programmable hardware in cloud networks exists in other forms, such as SmartNICs and FPGAs at the end hosts. They also have significantly different resource characteristics (e.g., more abundant RAM, lower processing speeds), and programming models (e.g., using a subset of C or Verilog/VHDL). We would also like to explore these alternative hardware platforms as future work.

6 SUMMARY

In this paper, we have presented a feasibility study on supporting time-aware queries in network monitoring. We have identified a class of algorithms called *time-adaptive sketches* to be a suitable starting basis, and proposed a set of customizations to adapt these algorithms to programmable data planes. We have developed software and hardware prototypes, and performed an initial set of experiments for validation. As next steps, we plan to investigate supporting a wider range of sketches that can be supported by this class of algorithms. We intend this paper to promote more discussions in the community on time-aware cloud network monitoring.

7 ACKNOWLEDGMENTS

We thank Dingming Wu and the anonymous reviewers for their valuable feedback. This work was partially supported by NSF grants CNS-1942219 and CNS-1801884.

REFERENCES

[1] Attackers Use DDoS Pulses to Pin Down Multiple Targets. https://www.imperva.com/blog/pulse-wave-ddos-pins-down-multiple-targets/.
[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM Computer Communication Review*, 41(4):63–74.
[3] R. B. Basat, R. Friedman, and R. Shahout. Stream frequency over interval queries. *Proceedings of the VLDB Endowment*, 12(4):433–445, 2018.
[4] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner. Heavy hitters in streams and sliding windows. In *IEEE International Conference on Computer Communications*. IEEE, 2016.
[5] P. Benácek, V. Pu, and H. Kubátová. P4-to-vhdl: Automatic generation of 100 Gbps packet parsers. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2016.
[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3), 2014.

[8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 2002.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon. Whippersnapper: A p4 language benchmark suite. In *Proceedings of the Symposium on SDN Research*. ACM, 2017.

[11] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[12] R. Dolby. Noise reduction systems, Nov. 5 1974. US Patent 3,846,719.

[13] M. Ghasemi, T. Benson, and J. Rexford. Dapper: Data plane performance diagnosis of tcp. In *Proceedings of the Symposium on SDN Research*. ACM, 2017.

[14] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 51–62. ACM, 2009.

[15] N. Ivkin, R. Ben Basat, Z. Liu, G. Einziger, R. Friedman, and V. Braverman. I know what you did last summer: Network monitoring using interval queries. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems*. ACM, 2019.

[16] N. Ivkin, Z. Yu, V. Braverman, and X. Jin. Qpipe: Quantiles sketch fully in the data plane. In *International Conference on emerging Networking Experiments and Technologies*, 2019.

[17] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé. Life in the fast lane: A line-rate linear road. In *Proceedings of the Symposium on SDN Research*, 2018.

[18] A. Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants. In *Processings of the 2003 ACM SIGCOMM Conference*, 2003.

[19] L.-K. Lee and H. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, 2006.

[20] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. ACM, 2006.

[21] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the 2019 ACM SIGCOMM Conference*. ACM, 2019.

[22] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016.

[23] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, and J. Rexford. Memory-efficient performance monitoring on programmable switches with lean algorithms. In *Proceedings of the SIAM Symposium on Algorithmic Principles of Computer Systems*. ACM, 2020.

[24] X. Luo and R. K. C. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *Proceedings of Network and Distributed System Security Symposium*, 2004.

[25] S. Matusevych, A. Smola, and A. Ahmed. Hokusai-sketching streams in real time. *arXiv preprint arXiv:1210.4891*, 2012.

[26] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the 2017 ACM SIGCOMM Conference*. ACM, 2017.

[27] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. Scream: Sketch resource allocation for software-defined measurement. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015.

[28] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. ACM, 2004.

[29] A. Shrivastava, A. C. Konig, and M. Bilenko. Time adaptive sketches (ada-sketches) for summarizing data streams. *Proceedings of the 2016 International Conference on Management of Data*, 2016.

[30] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*. ACM, 2017.

[31] D. Tong and V. Prasanna. High throughput sketch based online heavy hitter detection on FPGA. *ACM SIGARCH Computer Architecture News*, 43(4):70–75, 2016.

[32] S. V. Vaseghi. *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.

[33] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon. P4FPGA: A rapid prototyping framework for P4. In *Proceedings of the Symposium on SDN Research*. ACM, 2017.

[34] T. Wellem, G.-W. Li, and Y.-K. Lai. Superspreader detection system on netfpga platform. In *Proceedings of the tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2014.

[35] Y. Yu, C. Qian, and X. Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. ACM, 2014.