Elastic Sketch on OmniSketch

Jinqiao Hu*
Peking University
Beijing, China
2000013141@stu.pku.edu.cn

Qingyue Wu*
Peking University
Beijing, China
2000013057@stu.pku.edu.cn

Yuyang Zhou*
Peking University
Beijing, China
zhouyuyang2002@stu.pku.edu.cn

ABSTRACT

Network measurement plays an important role in today's Internet, both to the clients and servers. However, facing problems like fluctuation, congestion, scan attack, DDos attack, *etc.*, traffic characteristics may vary drastically. Good performance cannot be achieved in all possible situations using a fixed measurement strategy, so Elastic Sketch is proposed to deal with it. Elastic Sketch is **elastic** and **generic**, adapting to most of the platforms and traffic characteristics. We implement Elastic Sketch based on OmniSketch[1], a platform for testing sketch algorithms, and run experiments using a publicly available pcap file. We find that the main advantage of Elastic Sketch comes from the smaller upper bound of the light part, and Elastic Sketch has some issues not mentioned in the original paper.

CCS CONCEPTS

• **Networks** → **Network monitoring**; *Network measurement*.

KEYWORDS

Sketches; Network measurements; Elastic; Compression; Generic

1 INTRODUCTION

1.1 Motivation

Network traffic measurement is the process of measuring the amount and type of traffic on a particular network [16], and is the only efficient way to qualify and quantify how networks are being used, and how networks are behaving [4]. Without knowing the network behavior, there is no way for network maintenance and future network development.

However, due to the limitation of the network devices, we can neither store traffic information on the disk, nor traverse traffic information to answer the query. Under resource constraints, we are only able to record a sketch of the network, and gives an estimation on network characteristic queries as accurate as possible. Therefore, the goal of measurement is to adapt to different traffic characteristics and give an estimation of particular network, while achieving the highest possible accuracy under speed and space limitations.

Previous works have given an excellent solution under a stable network environment; however, it is remains challenging to be **elastic**. When network is undergoing problems such as congestion, DDoS attack, *etc.*, traffic characteristics including available bandwidth, packet rate, and flow size distribution vary drastically. For almost every previous works, a vary characteristic cause a significantly drop on the performance of measurements. So we implement *Elastic Sketch* [18], the first algorithm supporting effective measurement under vary traffic characteristics, and compared it with different network measurement systems.

1.2 Network Measurements

There are many classical network measurement tasks, including:

Flow Size Estimation: estimating the flow size for any flow ID. A flow ID can be any combinations of the 5-tuple, such as source IP address and source port, or only protocol. In this paper, we consider the number of packets of a flow as the flow size.

Heavy Hitter Detection: deciding whether the size of a flow is larger than a given threshold.

Heavy Change Detection: reporting flows whose sizes in two adjacent time windows increase or decrease beyond a predefined threshold, to detect anomalous traffic.

Flow size Distribution Estimation: estimating the distribution of flow sizes.

Entropy Estimation: estimating the entropy of flow sizes.

Cardinality Estimation: estimating the number of flows.

Due to the limitations of OmniSketch platform, we only focus on the first two tasks in this report.

Generic solutions can use one data structure to support all these measurement tasks. If we can record the IDs and sizes of all the flows, then we can support these tasks, but recording the information needs high memory usage. The original paper gives a way to handle this problem: discarding IDs of mouse flows, because the flow IDs of mouse flows are not necessary for these tasks. For this, we need to separate elephant flows from mouse flows, which leads to a new sketch algorithm. This algorithm is both generic and memory efficient.

Generic solutions should also be able to be implemented on different platforms. For small companies, the traffic speed might be limited, and measurement on CPU is a good choice. For large companies, the traffic speed could be very high, therefore to catch up with the traffic, hardware platforms should be used for measurements.

1.3 Previous Work

There are many well-known systems for measurements, including UnivMon [9], Trumpet [11], OpenSketch [19], FlowRadar [8], SketchVisor [7], Marple [14], Pingmesh [6], DREAM [10], Count-Min [3], HashPipe [15], *etc.* Among them, FlowRadar and UnivMon are generic, and thus are the most related work to this paper.

FlowRadar [8] records all flow IDs and flow sizes in a Bloom filter [1] and an Invertible Bloom Lookup Table (IBLT) [5]. To reduce memory usage, the authors propose an elegant solution of network-wide decoding. However, com- pared with sketches, its memory usage is still much higher.

UnivMon [9] is based on a key method named universal streaming [2]. Accuracy is guaranteed thanks to the the- ory of universal streaming. UnivMon is the first work to be generic, and achieves

*All authors contributed equally to this research.
[1]Github repository of OmniSketch: https://github.com/N2-Sys/OmniSketch

good performance. However, it does not handle the problem of variable traffic characteristics. To the best of our knowledge, *Elastic Sketch* is the first work that relies on a single data structure which is adaptive to bandwidth, packet rate, and flow size distribution.

1.4 Our Solution

In this report, we implemented Elastic Sketch based on OmniSketch platform. Elastic Sketch itself contains two parts, the `heavy part` and the `light part`. It proposed a separation method referred from ancient Athen Ostracism [17], which keeps the elephant flow in the heavy part, mouse flow in the light part.

To make the Elastic sketch "**Elastic**", It does the following.

- 1) **Bandwidth**. Since the light part hold the most of the storage space, Elastic Sketch provide algorithms which can compress and merge the light part. First, it can compress the sketch into an appropriate size which fit the bandwidth; second, it can merge sketch came from different measurement nodes.
- 2) **Packet Rate**. When the packet rate becomes extremely high, the incoming packet discard the light part update, find the balance between speed and accuracy.
- 3) **Flow size distribution**. The number of elephant flows is unknown in advance, thus Elastic Sketch provide algorithm which can increase the size of heavy part, reduce the probability of hash collisions.

We tested the performance of Elastic Sketch on the OmniSketch platform, and found that it mainly improved utilization of storage space compared to CountMin Sketch. Because of elephant/mouse flow separation, the counter array is only required to record mouse flows, which leads to a smaller structure and less memory usage.

1.5 Main Contribution

- 1). We implemented Elastic sketch based on OmniSketch platform, and test two basic measurement tasks supported by the platform. We then compared the results to the CountMin Sketch and HashPipe.
- 2). We found that, compared with CountMin sketch, the advantage of Elastic Sketch mainly comes from memory usage. Because of elephant-mouse flow separation, we can use a smaller data structure, which lead to a nearly half memory usage reduction, only at the cost of little performance lost.

2 ELASTIC SKETCH

2.1 Rationale

As we mentioned above, flow size measurement is one of the most extensively studied measurement tasks. The sketch algorithm is expected to be able to separate elephant flows and mouse flows.

We consider a simplified version of the separation problem: given a high-speed network stream, how to use only one bucket to select the flow with largest size? Due to the limitation of storage space, we generally cannot record the information of all streams, so it's impossible to achieve complete correctness. In order to achieve higher accuracy, we refer to the spirit of Ostracism (Greek: ostrakismos, where any citizen could be voted to be evicted from Athens for ten years).

In particular, we record the stream number, positive votes, and negative votes in the packet. We increment the positive votes if the flow ID f_1 of the incoming flow matches the stream number in the packet, and increment the negative votes otherwise. If we found $\frac{\texttt{\# positive votes}}{\texttt{\# negative votes}} \leq \lambda$ at some point, where λ is a predefined threshold up to 1, we will clear the packet, and insert the new flow f_1 into it.

2.2 Data structures

The Elastic Sketch can be split into two parts: the "`Heavy Part`" recording the information about elephant flows, and the "`Light Part`" recording the information about mouse flows.

2.2.1 Heavy Part. The heavy part $\mathcal{H}$ is a hash table with B buckets, and a hash function $h(\cdot)$. Each bucket contains a tuple $(f_1, \texttt{vote}^+, \texttt{flag}, \texttt{vote}^-)$, representing flow ID, positive votes, negative votes, and a boolean variable indicating whether the light part may contain positive votes for this flow.

2.2.2 Light Part. The light part $\mathcal{L}$ is a CountMin Sketch. The CountMin Sketch uses multiple hash arrays $(\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_d)$ of size w. To deal with the storage problem caused by hash collisions. Each hash array is associated with one hash function, and is composed of w counters.

Given the flow ID f and the size n of the incoming packet, the CountMin sketch computes d hash functions to locate the specific counter for each array (we call them *hashed counters*), and increment the d hashed counters by n. For the flow size query, we report the minimum one of the d hashed counter, which gives a upper bound of the real flow size.

2.2.3 Operations. Elastic Sketch provides two basic operations: insertion and query.

Insertion: Given a coming packet with flow id f, it is matched to bucket $\mathcal{H}[h(f) \bmod B] = (f_1, \texttt{vote}^+, \texttt{flag}, \texttt{vote}^-)$, where B means the number of buckets in the heavy part. Similar with Ostracism, we can get the following processing methods:

- 1). If the bucket is empty, put $(f, 1, F, 0)$ in it.
- 2). If $f = f_1$, then increment $\texttt{vote}^+$ by 1.
- 3). If $f \neq f_1$, and $\texttt{vote}^+ \geq (\texttt{vote}^- + 1)\lambda$, then insert $(f, 1)$ into the light part $\mathcal{L}$; increment $\texttt{vote}^-$ by 1;
- 4). If $f \neq f_1$, and $\texttt{vote}^+ < (\texttt{vote}^- + 1)\lambda$, then insert $(f_1, \texttt{vote}^+)$ into the light part $\mathcal{L}$; evict flow f_1 by replacing the contents of the bucket with $(f, 1, T, 0)$.

Query: First check if the flow belongs to the heavy part. If not, the problem is simplified to a CountMin Sketch query; Otherwise, there are two cases, depending on the boolean variable `flag`:

- 1). If `flag` is set to false, then $\texttt{vote}^+$ save the exact value.
- 2). If `flag` is set to true, then the answer should be the sum of $\texttt{vote}^+$ and the result of CountMin Sketch.

2.3 Optimization

The basic version performs well if the parameters are set reasonably. But when the packet size distribution or the bandwidth changes, a fixed size for heavy part and light part is not suitable for all

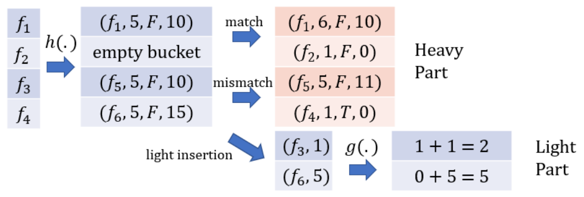

Figure 1: Data structure of Elastic. The four insertion flows matches four different cases. To insert flow f_4, after incrementing $\texttt{votes}^-$, $\frac{\texttt{votes}^-}{\texttt{votes}^+} > \lambda = 3$, hense f_6 is evicted from the heavy part and inserted into the light part.

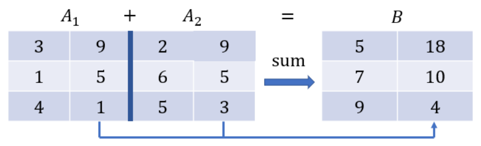

Figure 2: Compression algorithm for Elastic sketch. Split sketch $\mathbb{A}$ into sub-sketchs with same size, Then merge sub-sketchs using mering operators(*e.g.*, $\texttt{sum}$)

situations. Also, some portion of storage space are not used under elephant-mouse separation. This fact can be utilized to reduce memory usage.

2.3.1 Adaptation to bandwidth. To adapt to the available bandwidth, we can compress the light part $\mathcal{L}$ before sending them, which requires less bandwidth. As there will not be many elephant flows, the size of the heavy part is usually very small (*e.g.*, 150KB), thus the time overhead mainly comes from the light part. To compress the light part, we split the counter array into several groups with equal size, merge the groups and get a compressed counter.

Given a sketch $\mathbb{A}$ with size $zw' \times d$(counter size $w = zw'$, depth d), we can split it into z groups, where the i'th group $\mathbb{A}^i$ includes counters $\{A_j[iw' + k] : 0 \le j < d, 0 \le k < w'\}$. Then we can build a compressed sketch $\mathbb{B}$ with size $w' \times d$ using the following rule $B_j[k] = \texttt{OP}_{i=0}^{z-1} A_j^i[k] (0 \le j < d, 0 \le k < w')$, where $\texttt{OP}$ means the merging operator like $\texttt{sum}$ and $\texttt{max}$. The insertion and the query on compressed sketch $\mathbb{B}$ is almost the same as sketch $\mathbb{A}$, except that we replace the hash function $h(\cdot) \bmod w$ with $h(\cdot) \bmod w \bmod w'$, because w is the multiple of w'.

If we use the merging operator $\texttt{sum}$(called **sum compression**, **SC**), then the compressed sketch $\mathbb{B}$ is equal to the sketch $\mathbb{A}'$ with size $w' \times d$ under the same sequence of insertion, thus providing the same error analysis. However, if we use the merging operator $\texttt{max}$(called **maximum compression**, **MC**), the compressed sketch has smaller counter than **SC**. As the hashed counters (Defined in section 3.2.2) are increased by at least the flow size, the smaller the counter, the higher the accuracy, so it is proposed to use **maximum compression**.

2.3.2 Adaptation to packet rate. There are always input queues to buffer incoming packets in measurement nodes. However the packet rate (i.e., the number of incoming packets per second)is variable. In most cases, packet is low, so that the buffer responds well to instantaneous rate changes; but in the worst case (DDOS attack, *etc.*), large amount of data packets are pouring into the node, causing buffer overflow and massive information loss.

The Elastic Sketch uses a special approach inspired by the division of elephant flows and mouse flows to speedup the insertion when needed. The queued incoming packets only access the heavy part, and the information of mouse flows is discarded. The insertion process in the heavy part is almost the same, except the flow replacement: positive vote $\texttt{vote}^+$ remains unchanged rather than being set to 1. This does not mean that we discard the light part, as we still use it in queries. We drop the information in light part only when the packet rate is high, and it will not affect accuracy too much, as the packet rate is usually low.

2.3.3 Adaptation to flow size distribution. One of the main ideas of Elastic Sketch is splitting elephant flows and mouse flows, but it's usually difficult to determine size of the heavy part. So we need to estimate the number of elephant flows, and make the heavy part adaptive to changes in the traffic distribution.

We tried a method different from Elastic Sketch. We call the flow elephant if and only if its size is larger than threshold T_1, and make the assumption that elephants flows can always be found in the heavy part. For every bucket in heavy part $\mathcal{H}$, we record $\texttt{vote}^{\mathcal{L}}$ to indicate the result of CM Sketch additionally, which can give the estimation of flow size together with $\texttt{vote}^+$. If the number of elephant flow exceed some threshold, then we propose the **copy operation**: *simply copy the heavy part, concatenate the heavy part with the copied one, and replace the hash function $h(\cdot) \bmod w$ with $h(\cdot) \bmod 2w$.*

After the **copy operation**, exactly half of the buckets should be removed because of wrong hash value. The hash check is done in each insertion incrementally, and on average half of the bucket are removed. Although there may be "ghost" buckets remaining not cleaned, it does no harm to our algorithm.

2.3.4 Hash collision. A major problem of Elastic Sketch is elephant collision: when two or more elephant flows are mapped onto one heavy bucket, some of them are evicted to the light part. Some counters in CountMin Sketch becomes too large, causing significant light flow over-estimation. To resolve hash collisions, we can increment the depth of the heavy part rather than the light part, as the light part take more storage space.

The depth d of the heavy part means the numbers of flows saved in the bucket. Each flow in the bucket share the same $\texttt{vote}^-$, and we will always try to evict the flow with smallest $\texttt{vote}^+$. This allows multiple elephant flows to be saved in the same bucket, which helps to reduce the elephant collision rate significantly.

2.3.5 Memory usage. In CountMin Sketch, because of the existence of elephant flows, the maximum value in the counter can reach the magnitude of the number of all packets. However, most of the counters are untouched by elephants flows, and a huge order of magnitude difference exist between counters of elephant flows and mouse flows, bringing in a lot of useless leading 0s.

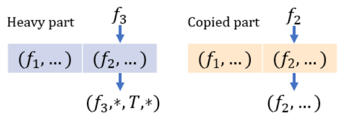

Figure 3: Duplication of the heavy part of Elastic with size 2. Because $h(f_2) \neq h(f_3) (\bmod 4)$, the 'ghost' flow f_2 is replaced by f_3.

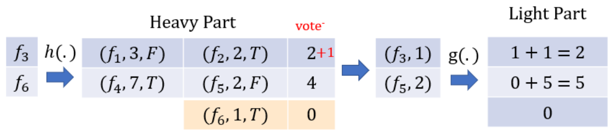

Figure 4: Evicted flow is always the smallest flow. Given packet from flow f_3, f_6, we can only check f_2, f_5 because they are the smallest flow in the packet.

However, after elephant-mouse separation, there is no need for Elastic Sketch to handle elephant flows in the light part. Even under the premise of hash collision, several mouse flows still looks insignificant in comparison to a single elephant flow. So we propose to use a smaller structure as the counter, which have a higher memory utilization and store more information under the same memory limits.

2.4 Implementations

We have implemented the software version of Elastic Sketch Based on OmniSketch. While writing our codes, we found two bugs of the Omnisketch platform: First[2] is about the timer, and second[3] is about heavy hitter testing. We fixed these bugs before running the experiments. Our implementation support the following applications:

- **Estimate flow size**: Elastic sketch can be directly used for flow size estimation, and it always return the correct answer if `flag` is set to false.
- **Heavy hitter detection**: Under the assumption that elephants flows are all stored in the heavy part, we can traverse every flow in the buckets. If the estimation is greater than a predefined threshold, we report the flow as a heavy hitter.

Due to limitations of the OmniSketch platform, we can neither get the information about the traffic characteristic, not save the sketch for compression and merging. So we set most of the thresholds before starting the measurement node, with compression, merging, and fast-insertion disabled. We enable the heavy part reallocation, since the number of elephant flows can be estimated during insertion.

3 EXPERIMENTS

3.1 Experimental Setup

Traces: We use a pcap file created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) [12, 13]. There are in total $\sim$ 3.5M records, composed of $\sim$ 85K flows.

Evaluation Metrics:

- *Update & Query Rate:* total time used.
- *AAE (Average Absolute Error):* $\frac{1}{n}\sum_{i=1}^{n}|\hat{f_i} - f_i|$, where n is the number of flows, and f_i and $\hat{f_i}$ are the actual and estimated flow sizes respectively.
- *ARE (Average Relative Error):* $\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{f_i} - f_i|}{f_i}$.
- *HH PRC (Heavy Hitter Precision):* The ratio of true heavy hitters reported, or $\frac{TP}{TP+FP}$, where TP means the reported heavy hitters, and FP means the reported non-heavy-hitters.
- *HH RCL (Heavy Hitter Recall):* The ratio of reported heavy hitters, or $\frac{TP}{TP+FN}$, where FN means the unreported heavy hitters.
- *F_1 score:* The harmonic mean of PRC and RCL.

Setup:

The other applications mentioned in Elastic Sketch (heavy change detection, flow size distribution, entropy) are not supported by the testing system provided by OmniSketch, so we focus on the following two tests:

- *Flow size estimation:* We compare Elastic Sketch to CountMin Sketch. For CountMin Sketch, we set `depth=3`, and adjust the width of the sketch to simulate the scenario of different memory usage. For Elastic Sketch, as there are multiple parameters (depth and width of the light part, number of heavy buckets, number of slots in each bucket, *etc.*), we adjust the parameters manually to obtain the near-optimal performance for each fixed memory size.
- *Heavy hitter detection:* We compare Elastic Sketch to HashPipe. Similarly, for HashPipe, we set `depth=5`, and adjust its width. For Elastic Sketch, we try different parameters manually to select the best parameters for a fixed memory size. We ask both algorithms to retrieve the top 300 flows as heavy hitters.

Since Elastic Sketch separates elephant flows from mouse flows, we may assume that the light part only need to store the mouse flows. Thus 8-bit unsigned integers should be enough for the counters in the light part. We are also curious about the "real" performance of Elastic Sketch had it not been able to use 8-bit integers, so we also ran a set of experiments where the counters in the light part are 32-bit integers, which is consistent with CountMin Sketch and HashPipe.

3.2 Results and corollaries

3.2.1 Flow Size Estimation. The test results we get are shown in Figure 5.

We find that the 8-bit Elastic Sketch does offer a better accuracy than CountMin Sketch under the same memory consumption, which is consistent with the original paper. When using 600KB of memory, the ARE of 8-bit Elastic Sketch is only about the half of the one of CountMin.

[2]See https://github.com/N2-Sys/OmniSketch/issues/2
[3]See https://github.com/N2-Sys/OmniSketch/issues/3

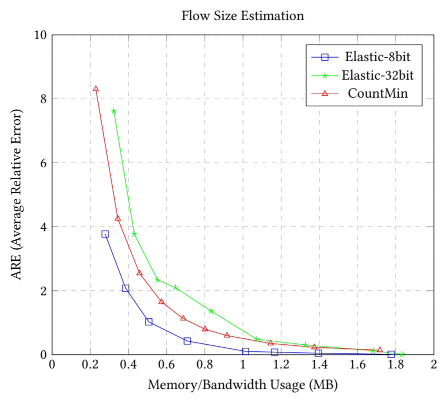

Figure 5: Performance for Flow size estimation under different memory limits

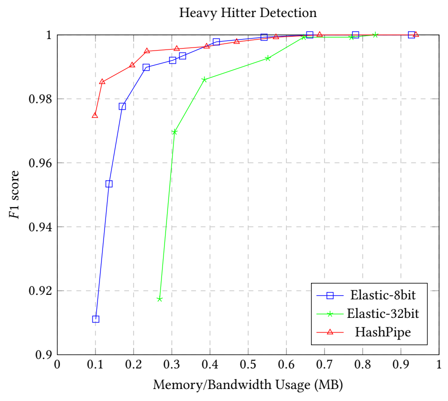

Figure 6: Performance for Heavy Hitter Detection under different memory limits

However, the ARE of 32-bit Elastic Sketch is even larger than CountMin Sketch. This is because in the case of similar counter size, 32-bit Elastic sketch needs extra memory to store the heavy part. Also, CountMin sketch usually use a counter array with depth greater than 1 and width far greater than the number of elephant flows. For any mouse flow, the probability that all hashed counter of the mouse flow is updated by elephant flow becomes negligible. Therefore, 32-bit Elastic Sketch itself is similar to a CountMin Sketch which uses more memory/bandwidth, and naturally, it cannot reach the performance of CountMin sketch.

The difference between 8-bit and 32-bit elastic sketch mainly comes from the size of the counter. According to the foregoing paragraphs, CountMin sketch usually use a counter array with width far greater than the number of elephant flows, which leads to a better performance, but brought the waste of memory. We found that the 300-th biggest flow in the pcap file only have size 746. Assume that the first 300 biggest flows are elephant flows, and the memory usage is 600KB, then only $(3 \times 300)/600\text{KB}/4 \approx 0.6\%$ of counters is updated by elephant flows, and 99.4% counters contains a large number of useless leading zeros. The design of Elastic sketch ensures that the information in those `0.6%` counters are recorded in the heavy part, and there is no need for the light part to record it. Without elephant flows, we can reduce the size of counter, and improve storage space utilization. Because of that, 8-bit Elastic provide a better performance, compared with CountMin Sketch.

3.2.2 Heavy Hitter Detection. The test results we get are shown in Figure 6.

For the F1 score, we can see that HashPipe outperforms both 8-bit and 32-bit Elastic Sketch under almost all memory usage. In the original paper, the author only drew the graph for memory usage > 0.2MB, where the precision and recall of both HashPipe and Elastic Sketch is 1. However, for even less memory consumption, our results show that HashPipe actually does better in finding out the heavy hitters.

This is mainly because Elastic Sketch has the light part. When the light part is small, many different flows are stored in the same slot, which leads to a larger reported size, causing the precision to be small. However, HashPipe just ignores the "light part", which means the precision of HashPipe is always 1. Only when the memory assumption becomes larger (greater than 400KB) can Elastic Sketch outperform HashPipe. This problem is even more obvious when the size of the test data becomes larger.

4 CONCLUSION

In network measurement, generality and accuracy are always the pursuit. Elastic Sketch adapts well to measurement tasks and works across different platforms. It also has a better accuracy compared with many existing methods.

We implemented Elastic Sketch on OmniSketch, and compared it with CountMin and HashPipe. The result showed that 8-bit Elastic sketch provides better performance on flow size estimation under same memory-usage compared with CountMin Sketch. At the same time, 8-bit Elastic sketch supports much more application than HashPipe (Estimation of Flow Size Distribution, Entropy, Cardinality *etc.*), with little loss of accuracy. At the same time, we found that Elastic Sketch improved memory usage by elephant-mouse separation. Without 8-bit counters array, Elastic sketch performs even worse than CountMin Sketch because of the extra memory of heavy part.

The original paper mainly focus on word "`elastic`" and "`generic`". However, some important issues are not mentioned:

- Compared to CountMin Sketch, the main improvement of Elastic Sketch is that it can use a counter array with smaller upper bound and smaller size. However, we found neither rule for upper bound setting, nor overflow probability analysis or overflow handling process in the original paper. Further research on mouse flow size distribution is still required for a better upper bound setting.
- For heavy hitter detection, in the result of original paper, the $F1$ score of all the algorithms but UnivMon is close to 100%. However, if we analyse the data carefully, we found that HashPipe performs better than Elastic significantly when the memory limit is no more than 400KB. It is because when the memory limit is small, the light part of Elastic has to be small, which leads to a low precision. However, HP just

ignores the mouse flows, which means the precision of HP is always 100%.

REFERENCES

[1] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13, 7, 422–426.

[2] Vladimir Braverman and Rafail Ostrovsky. 2013. 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*. Springer, 58–70.

[3] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55, 1, 58–75.

[4] Kim Ervasti. 2016. A survey on network measurement: concepts, techniques, and tools. *University of Helsinki*.

[5] Michael T Goodrich and Michael Mitzenmacher. 2011. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 792–799.

[6] Chuanxiong Guo et al. 2015. Pingmesh: a large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 139–152.

[7] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. Sketchvisor: robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 113–126.

[8] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. {Flowradar}: a better {netflow} for data centers. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, 311–324.

[9] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 101–114.

[10] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2014. Dream: dynamic resource allocation for software-defined measurement. In *Proceedings of the 2014 ACM conference on SIGCOMM*, 419–430.

[11] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 129–143.

[12] Nour Moustafa and Jill Slay. 2015. 'unsw-nb15', a comprehensive data setfor network instruction detection systems(unsw-nb15 network dataset).

[13] Nour Moustafa and Jill Slay. 2016. *The evaluation of Network Anomaly Detection Systems: Statical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set*. Information Security Journal: A Global Perspective(2016).

[14] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 85–98.

[15] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, 164–176.

[16] Wikipedia. 2021. Network traffic measurement. (Aug. 2021). Retrieved August 7, 2021 from https://en.wikipedia.org/wiki/Network_traffic_measurement.

[17] Wikipedia. 2021. Ostracism. (Jan. 2021). Retrieved Janurary 12, 2023 from https://en.wikipedia.org/wiki/Ostracism.

[18] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 561–575.

[19] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software {defined}{traffic} measurement with {opensketch}. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 29–42.