




HBL-Sketch: A New Three-Tier Sketch for Accurate Network Measurement

Keyan Zhao¹, Junxiao Wang¹, Heng Qi¹(✉) , Xin Xie¹, Xiaobo Zhou²,
and Keqiu Li²

¹ Dalian University of Technology, Dalian, China
hengqi@dlut.edu.cn

² Tianjin University, Tianjin, China

Abstract. Network measurement is critical for many network functions such as detecting network anomalies, accounting, detecting elephant flow and congestion control. Recently, sketch based solutions are widely used for network measurement because of two benefits: high computation efficiency and acceptable error rate. However, there is usually a tradeoff between accuracy and memory cost. To make a reasonable tradeoff, we propose a novel sketch, namely the HBL (Heavy-Buffer-Light) sketch in this paper. The architecture of HBL sketch is three-tier consisting of heavy part, buffer layer and light part, which can be viewed as an improved version of Elastic sketch which is the state-of-the-art in network measurement. Compared to the Elastic sketch and other typical work, HBL sketch can reduce the average relative error rate by 55%–93% with the same memory capacity limitations.

Keywords: Network measurement · Sketch · Tradeoff · Average Relative Error

1 Introduction

Network measurement is often viewed as the basis of network anomaly detection, quality of service, capacity planning, accounting, congestion control, and so on. [5, 24, 30]. Therefore, network operators must use measurement tools to meet the growing demands on daily network operations and network management. Traditional network measurement methods are mostly based on packet sampling, such as sFlow [22] and NetFlow [2]. However, packet sampling will lose some information, which leads to the inaccuracy of network measurement. Several studies have shown that packet sampling is not sufficient for fine-grained network measurement [7, 18]. Recently, sketching techniques have found a widespread use in network measurements such as flow sizes estimation [15, 24, 26], elephant flow detection [5, 13, 21, 28], and flow number estimation [12, 14, 27]. Compared to the sampling methods, the sketch methods have higher accuracy [14, 16].

In recent work, some typical sketch-based network measurement methods are proposed, such as CM sketches [9], CU sketches [11], counting sketches [8],

Bloom sketch [29], FlowRadar [15]. In existing work, state-of-the-art method named Elastic sketch [25] is proposed to provide with a more speedy processing and a less memory usage. The Elastic sketch consists of two parts: a heavy part and a light part. The heavy part mainly records elephant flows. Because the flow ID of the packet is recorded in heavy part, the number of packets in the flow can be accurately recorded. The light part is a simple CM sketch, which mainly records mouse flows, does not record the flow ID of the packet, and can only roughly estimate the number of packets in the flow.

In Elastic sketch, the incoming packet is first stored in heavy part by hashing. When a hash collision occurs in the heavy part, an elephant flow is also possibly removed from the heavy part. Then a part of the elephant flow leaves the precise heavy part and enters the rough light part, which we argue is negative for keeping up accuracy of measurement. On another hand, the part of the elephant flow which moved into the light part will destroy the correctness of mouse flows. By sharing position in the light part, all the mouses flows relating to the elephant flow are excessively recorded, which we argue is also harmful to accuracy of measurement.

The low accuracy of measurement will affect functions [3, 17, 19], which may cause network operators to make wrong decisions and affect the normal operation of the network. To improve the accuracy of Elastic sketch, additional memory is needed. However, network measurement is generally used in network devices such as switches and routers. Memory on these network devices is an extremely precious resource. Therefore, we should address a tradeoff issue between the accuracy and the memory cost.

In this paper, we propose a novel sketch based on the buffer layer and the Elastic sketch, namely the HBL (Heavy-Buffer-Light) sketch. The main idea of our HBL sketch is to leverage the buffer layer to avoid the part of the elephant flow muted in the light part, while avoiding the part of the elephant flow disturbing the correctness of the light part. With the HBL sketch, we can improve the accuracy of network measurement without additional memory cost.

In summary, the contributions of this paper are as follows:

- (1) We design a novel sketch, namely HBL (Heavy-Buffer-Light) Sketch. With a buffer layer, not only the number of accesses to the light part is reduced, but also largely avoiding the heavy part wrongly passing the elephant flow records to the light part. The accuracy of network measurement can be improved by the HBL sketch.
- (2) We implement a network measurement mechanism based on HBL Sketch, which includes insertion algorithm and query algorithm. Based on the proposed mechanism, we further implement two measurement modules which could be leveraged to estimate flow sizes and to detect elephant flows.
- (3) We examine the HBL Sketch in terms of accuracy, memory usage and speed by plenty of experiments. The experimental results show that our HBL sketch significantly outperforms other sketches and decreases the error rate by 55%–93% on average with the same memory cost. Moreover, to obtain the same level of error rate, the memory usage of the HBL Sketch is about $1/5$ – $2/3$ of existing methods. In terms of processing speed, the HBL Sketch can match state-of-the-art method.

Code and Data Sharing. We implement HBL Sketch based on the code of Elastic Sketch¹. We make the HBL Sketch data snapshot and code used for this paper available to the research community as well².

2 Related Work

Recently, sketch is widely used in network measurement, such as Elastic sketch [25], count-min (CM) sketch [9], conservative-update (CU) sketch [11], Count sketch [8], Bloom sketch [29], SF sketch [26] and FlowRadar [15].

Among them, CM Sketch, CU Sketch and Count sketch are three classic sketches with the same data structure. A CM sketch [9] consists of d arrays, each of which consists of w counters and corresponds with a hash function. When inserting a flow f , the CM sketch first computes the d hash functions and locates the d counters. Then it increments all the d hashed counters. When querying a flow f , the CM sketch reports the minimum value of the d hashed counters. The CU sketch [11] makes a slight but effective modification compared with the CM sketch. During insertions, the CU sketch only increases the smallest counter(s) among the d hashed counters while the query process keeps unchanged. The Count sketch [8] is also similar to the CM sketch except that each array uses an additional hash function to smooth the accidental errors. They all have the shortcomings of slow speed and poor accuracy when using small memory.

Bloom sketch [29] consists of sketch layers and Bloom filter [6] layers. SF sketch [26] is a two-stage sketch based on CM sketch. They all have faster processing speeds, but the accuracy is poor. FlowRadar [15] maps flows to counters through XOR operations, such that new flows can be reconstructed by repeatedly XOR-ing the counters with known flows. However, such extensions incur heavy computational overhead and its memory usage is still much higher than other sketches.

The state-of-the-art solution Elastic sketch [25] is proposed to provide with a more speedy processing and a less memory usage. And it can adapt to current traffic characteristics. However, Elastic sketch is composed of the precise heavy part and the rough light part. If there are too many flows, especially elephant flows, removed from heavy part and stored in light part, the error of network measurement will be greatly increased. Therefore, the design goal of this paper is to improve the accuracy of Elastic Sketch without increasing memory usage while keeping its advantages.

3 HBL Sketch

3.1 Data Structure

As shown in Fig. 1, the data structure consists of three parts: the heavy part, the buffer layer and the light part.

¹ <https://github.com/BlockLiu/ElasticSketchCode>.

² <https://github.com/FlowAnalysis/HBLSketch>.

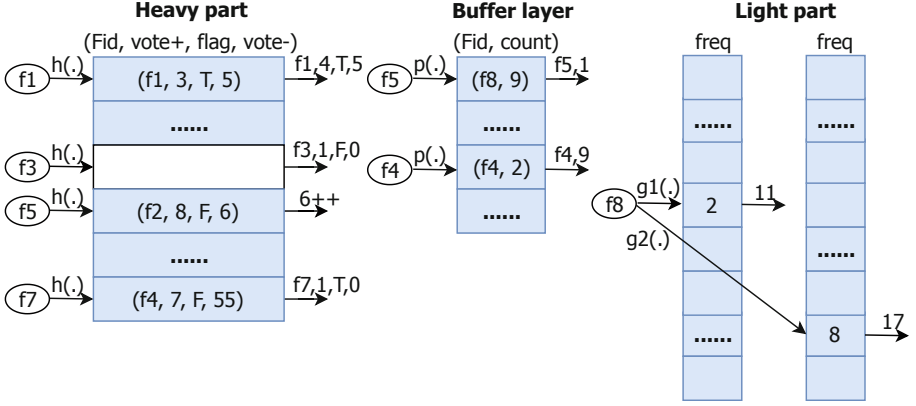


Fig. 1. The structure of our HBL sketch.

Heavy Part: The heavy part is a hash table associated with a hash function $h(\cdot)$. Each position (bucket) of the heavy part records the information of a flow by: flow ID (Fid), positive votes (vote+), negative votes (vote-), and flag. Each packet arriving the system is calculated its Fid (according to five-tuples) and its position in the hash table. When the calculated Fid matches the one that belonging to the position, vote+ increases by 1, otherwise, vote- increases by 1. When the value of vote-/vote+ is more than a fixed threshold λ , the record originating on the position will be evicted from the heavy part. The flag indicates whether such an eviction has occurred on the position.

Buffer Layer: The buffer layer is a hash table associated with a hash function $p(\cdot)$. Each position of the buffer layer records the information of a flow by: flow ID (Fid), packet counter (count). The record evicted from the heavy part is the input of the buffer layer. When the record arrives and the calculated Fid matches the one that belonging to the position, count increases by vote+, otherwise, the record originating on the position will be evicted from the buffer layer.

Light Part: The light part is a CM Sketch [9], consisting of d arrays. Each array is associated with one hash function, and is composed of w counters. Each counter records the information of a flow by: event frequency (freq). The record evicted from the buffer layer is the input of the light part and is eventually stored into the arrays.

3.2 HBL-Based Network Measurement Mechanism

Basically, Network measurement mechanism based on HBL sketch includes insertion and query. HBL sketch relies on a series of hash operations to update memory and to record the information of flows, just like working in a manner of pipeline processing. When the packet arrives the system, it's first processed by the hash operations defined in the heavy part. The processing finishes with two sets *heavy_part_left* and *heavy_part_evicted*. The first set contains the records

updated in the heavy part and the another contains the records evicted from the heavy part. The set *heavy-part-evicted* is also the input of the processing defined in the buffer layer. Similarly, the processing of buffer layer finishes with the sets *buffer-layer-left* and *buffer-layer-evicted* and the later is the input of the light part.

Insertion

Heavy Part: Given an incoming packet with flow ID f , it will be hashed into the bucket (position) $H[h(f)\%N_1]$, where N_1 is the number of buckets in the heavy part. According the memory usage of HBL Sketch, the bucket should have a record $(f_1, vote+, flag_1, vote-)$ or null. Then, there are three cases by different situations,

Case 1, the record is null: Add $(f, 1, F, 0)$ into the set *heavy-part-left*, where $flags = F$ is a boolean value to indicate no eviction has happened in the bucket. *Case 2, f matches f_1 :* Add $(f_1, sum(1, vote+), flag_1, vote-)$ into the set *heavy-part-left*. *Case 3, f mismatches f_1 :* If $\frac{sum(1, vote-)}{vote+} < \lambda$, add $(f_1, vote+, flag_1, sum(1, vote-))$ into the set *heavy-part-left* and add $(f, 1)$ into the set *heavy-part-evicted*. Otherwise, add $(f, 1, T, 1)$ into the set *heavy-part-left* and add $(f_1, vote+)$ into the set *heavy-part-evicted*. The $flags = T$ indicates the bucket has happened eviction before flow f is elected.

Buffer Layer: For the flow ID f and the number of packets val evicted from the heavy part, it will be hashed into the bucket $B[p(f)\%N_2]$, where N_2 is the number of buckets in the buffer layer. The bucket is supposed to have a record $(f_1, count)$ or null. Then, there are three cases by different situations,

Case 1, the record is null: Add (f, val) into the set *buffer-layer-left*. *Case 2, f matches f_1 :* Add $(f_1, sum(val, count))$ into the set *buffer-layer-left*. *Case 3, f mismatches f_1 :* Add (f, val) into the set *buffer-layer-left*, and add $(f_1, count)$ into the set *buffer-layer-evicted*.

Light Part: For the flow ID f and the number of packets val evicted from the buffer layer, it will be d -hashed into d buckets (in d arrays) with records $[(freq)]_d$, and $[(sum(val, freq))]_d$ is added into the set *light-part-left*.

To further clarify the above mappings, let us get back to the example depicted in Fig. 1, where $\lambda = 8$. The packet of flow f_1, f_3, f_5 and f_7 arrive the heavy part, triggering the record eviction of flow f_4 and f_5 . Then, they arrive the buffer layer and trigger the record eviction of flow f_8 . At last, the evicted record arrives the light part and is updated by Count-Min Sketch [9]. Now, let's zoom in the processing conducted in the heavy part, buffer layer and light part. In heavy part, f_1 matches the Fid of the record $(f_1, 3, T, 5)$ in the hashed bucket, so the record in the bucket is updated to $(f_1, 4, T, 5)$. The record in the bucket hashed by f_3 is empty, so the record in the bucket is updated to $(f_3, 1, F, 0)$. f_5 does not match the Fid of the record $(f_2, 8, F, 6)$ in the hashed bucket and $\frac{6+1}{8} < \lambda$, so the record in the bucket is updated to $(f_2, 8, F, 7)$ and $(f_5, 1)$ is passed to the

buffer layer. f_7 does not match the record $(f_4, 7, F, 55)$ in the hashed bucket and $\frac{55+1}{8} \geq \lambda$, so the record in the bucket is updated to $(f_7, 1, T, 0)$ and $(f_4, 7)$ is passed to the buffer layer. In buffer layer, f_5 does not match the record $(f_8, 9)$ in the hashed bucket, so the record in the bucket is updated to $(f_5, 1)$ and $(f_8, 9)$ is passed to the buffer layer. f_4 matches the Fid of the record $(f_4, 2)$ in the hashed bucket, so the record in the bucket is updated to $(f_4, 9)$. In light part, the value of all counters hashed by f_8 is increased by 9.

Query. For the flow recorded with HBL Sketch, since it might be stored in three parts, the system thus need to conduct (hash) query operations respectively in the heavy part, buffer layer and light part. The results returned from each part will be aggregated then to obtain the original record of the flow. Obviously, this can work well as a naive method. The problem is, how to optimise the query process with a less operations.

Optimized Query Process: The system begins the query process with the heavy part. For any flow f in the heavy part, first find the bucket by the hash mapping of flow f . Then, there are two cases by different situations,

Case 1, The flag of the bucket is false: The size of f is exactly equal to the value of vote+ (on the bucket), because no eviction has happened. All the packets of flow f shoot on the bucket, and no any other flows' packets touch the bucket.

Case 2, The flag of the bucket is true: The size of f is equal to the sum of the value of vote+, the value in the buffer layer and the value in the light part.

The value in the buffer layer is found by two steps. First, find the bucket by the hash mapping of flow f . Next, get the value of count (on the bucket). The value in the light part can be found by referring to the query of CM Sketch [9]. By d-hashed mapping, the $[(freq)]_d$ of d buckets (in d arrays) is found. Then, get the minimal freq (on the bucket).

3.3 Performance Analysis

Notice there should be no inaccuracy in the heavy part and the buffer layer if only no eviction happens both on them. But in practice it's often impossible. There are always some records evicted from the heavy part or the buffer layer and entering the light part. Since the light part doesn't record the flow ID, but rather the flow size, there will be some estimation error in the light part. Towards, we design the buffer layer to reduce the number of accesses to the light part and improve the accuracy of measurement.

Access to the Light Part. We use an example to demonstrate how the buffer layer reduces the number of accesses to the light part. Suppose now have 9 incoming packets evicted from the heavy part with flow IDs: $f_1, f_2, f_3, f_1, f_3, f_1, f_2, f_1, f_3$. If we just use memory with the heavy part and the light part (just like what has been done in Elastic Sketch [25]), the number of accesses to the light part is 9. In HBL Sketch, before the 9 packets enters the light part, they

are first aggregated into 3 flows ($f_1 \times 4$, $f_2 \times 2$, $f_3 \times 3$) in the buffer layer and then with a fewer possibilities to be evicted to the light part. In this case, the light part can be accessed at most 3 times. This is very similar to the role of the multi-level caches in CPU, which has been widely proven its effectiveness [20]. By this way, we can significantly reduce the number of accesses to the light part.

Error Bound. To analyze the error bound of HBL Sketch, we refer to the following theorem, whose proof can be found in [9].

Theorem 1. *Given two parameters ε and δ , the reported size \hat{f}_i by HBL sketch for flow i is bounded by*

$$\hat{f}_i \leq f_i + \varepsilon \|f_{l_1}\|_1 < f_i + \varepsilon \|f_{l_2}\|_1 \quad (1)$$

with probability at least $1 - \delta$, where f_{l_1} and f_{l_2} denote the size vector of the sub-streams recorded by the light part in HBL sketch and Elastic sketch, respectively.

According to Theorem 1, the estimation error of HBL sketch is bounded by $\|f_{l_1}\|_1$, instead of $\|f_{l_2}\|_1$ which is provided by Elastic sketch. When the total memory and the memory occupied by the heavy part are fixed, the buffer layer can reduce the packets of a stream recorded in the light part. Therefore, $\|f_{l_1}\|_1$ is usually significantly smaller than $\|f_{l_2}\|_1$, and the HBL sketch has a much tighter error bound than Elastic sketch when the parameters are the same.

4 Experiments

4.1 Experimental Setup

Trace: We use the dataset used in the latest work Elastic sketch, which is the public traffic traces collected in Equinix-Chicago monitor from CAIDA [1]. There are 11 traces in the dataset, namely 0.dat, 1.dat, ...10.dat. Each trace is in dat format and contains about 2.5M packets and 110K flows. Each 13 bytes in a trace is 5-tuple in the format of (srcIP, srcPort, dstIP, dstPort, protocol).

Testbed: We performed all the experiments on a server with 6-core CPUs (12 threads, Intel®Core™ i7-8700K CPU @ 3.70 GHz) and 62.9 GiB total memory running Ubuntu 16.04.

Metrics: The evaluation metrics we used are as follows:

ARE (Average Relative Error): ARE denotes the difference between estimated value and real value, which is defined as

$$\frac{1}{n} \sum_{i=1}^n \frac{|f_i - \hat{f}_i|}{f_i} \quad (2)$$

where n is the number of values. For flow size estimation, f_i and \hat{f}_i are the real flow sizes and estimated sizes. For elephant flow detection, f_i and \hat{f}_i are the real amount of elephant flows and estimated amount. We use ARE to evaluate the accuracy of flow size estimation and elephant flow detection.

Throughput: We calculate the throughput with unit of million packets per second (Mpps). Throughput is used to evaluate the processing speed of the flow size estimation.

Average Latency: We calculate the average delay for each trace with unit of millisecond (ms). Average delay is used to evaluate the processing speed of the flow size estimation.

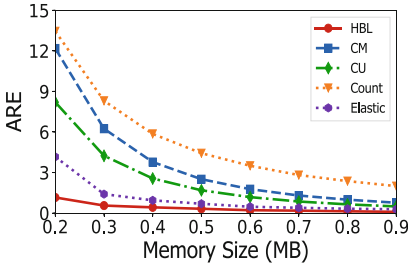


Fig. 2. ARE vs. memory sizes for flow size estimation.

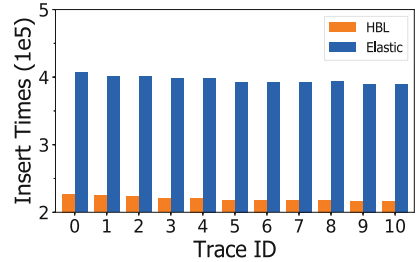


Fig. 3. Average insertion times for flow size estimation.

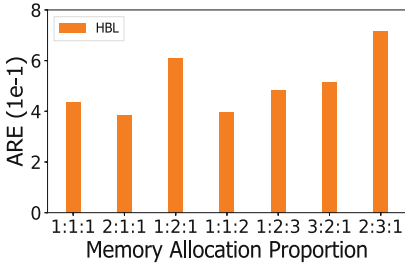


Fig. 4. ARE for flow size estimation under different memory allocations.

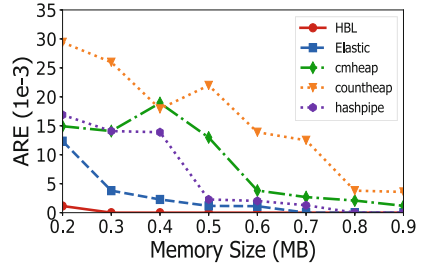


Fig. 5. ARE vs. memory sizes for elephant flow detection.

4.2 Experimental Results

Accuracy Comparison

Flow Size Estimation: We compare five methods: Elastic, CM, CU, Count, and our HBL sketch.

Figure 2 plots the AREs of flow size estimation for different methods on different memory sizes increasing from 0.2 MB to 0.9 MB with a step of 0.1 MB.

Our experimental results show that (1) As the total size of memory increases, the error rate decreases since the possibility of hash collision is getting smaller. (2) HBL Sketch outperforms other methods and reduces the error rate by 65%–93% on average. (3) Compared to state-of-the-art method Elastic Sketch, the error rate of HBL Sketch is 77% less than that in the extreme case.

Figure 3 plots the difference between HBL Sketch and Elastic Sketch with no buffer layer in terms of average insertion times (to the light part) when estimating flow sizes in different traces. It clearly shows that our solution with the buffer layer and the mechanism defined in Sect. 3 bring about 44% less average insertion times to the light part. This is in line with our motivation and also validates effectiveness of the buffer layer to significantly reduce the number of accesses to the light part, so as to avoid the heavy part wrongly passing the elephant flow record to the light part. The accuracy of measurement is thus greatly guaranteed.

Figure 4 plots the ARE of flow size estimation in the HBL sketch, when the total memory is 0.6 MB and the memory allocation ratios between heavy part, buffer layer and light part are 1:1:1, 2:1:1, 1:2:1, 1:1:2, 1:2:3, 3:2:1 and 2:3:1 respectively. Our experiment results show that the ARE is different with different allocation proportions between the heavy part, the buffer layer and the light part. It can be seen that the proportion of memory allocation is an influential parameter against accuracy of measurement. While we argue that network operators can employ a suitable proportion by some approaches of exploration and exploitation [10], nevertheless, the details of building such a parameter selection module are outside the scope of this paper. In the future, we thus aim to extend our study to this problem.

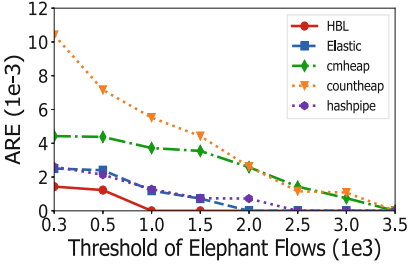


Fig. 6. ARE vs. threshold of elephant flows.

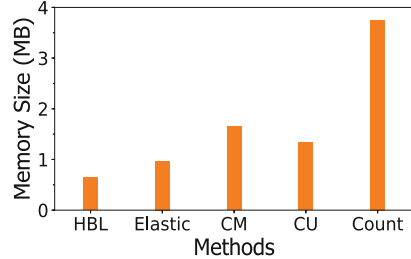


Fig. 7. Memory usage for flow size estimation along with fixed ARE = 0.2.

Elephant Flow Detection: We compare five methods: Elastic, cmheap, countheap, hashpipe, and our HBL sketch.

Figure 5 depicts the ARE of different methods for elephant flow detection, where the size of memory increasing from 0.2 MB to 0.9 MB with a step of 0.1 MB. It can be seen that HBL Sketch always achieves a more accurate detection against elephant flows. While some methods like cmheap, countheap and

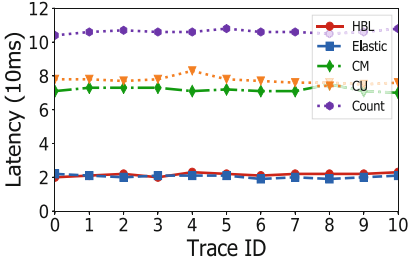


Fig. 8. Average delay for flow size estimation under different traces.

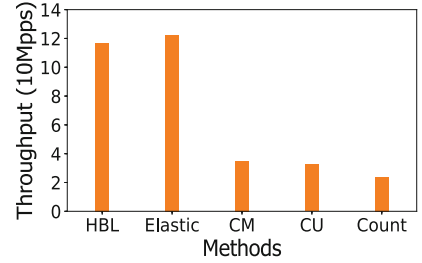


Fig. 9. Throughput of different methods for flow size estimation.

hashpipe are highly sensitive to the variations of memory size. In Fig. 5, we find that HBL Sketch reduces the error rate by 94%–99% on average.

Figure 6 depicts the ARE of different methods for elephant flow detection, where the thresholds of elephant flows are set to 300, 500, 1000, ..., 3500, respectively. Our experiment results show that HBL Sketch always achieves a more accurate detection against elephant flows. And it reduces the error rate by 61%–92% on average compared with other methods.

Memory Usage and Speed. We compare five methods: Elastic, CM, CU, Count, and our HBL sketch.

To show the memory usage among different methods at the same level of error rate, we plot Fig. 7 when the ARE for flow size estimation is 0.2. It clearly shows that HBL Sketch performs the best memory utilization. It has about 1/3 less memory usage compared with Elastic Sketch.

To quantify the impact of the buffer layer on the performance of processing speed, we compare HBL Sketch with other methods. Figures 8 and 9 plot the difference in terms of average latency and throughput. For latency, while HBL Sketch brings about 6% more overhead on average compared to state-of-the-art method Elastic Sketch, it overwhelms the rest of methods and decreases the latency by 70%–80% on average. This can also be observed for throughput. Our solution outperforms the methods except Elastic Sketch by 2.3–3.9 \times . This validates effectiveness of HBL Sketch to trade much accuracy improvement using very little loss in processing speed.

5 Conclusion

Sketch is one kind of useful data structure especially in the field of network measurement [4, 23]. In this paper, we proposed a new sketch, namely the HBL sketch, which can improve the measurement accuracy without increasing memory. The key idea behind our proposed HBL sketch is to add a buffer layer and maintain the sketch composed of heavy part, buffer layer and light part to improve measurement accuracy. Our experimental results show that our HBL

sketch significantly outperforms the-state-of-the-art in terms of accuracy and memory usage. In terms of speed, our HBL sketch is basically the same as the latest work Elastic sketch and obviously better than other methods.

Acknowledgments. This work was supported by the State Key Program of National Natural Science of China (Grant No. 61432002), NSFC Grant Nos. 61772112, U1836214, U1701263, 61672379, and 61751203, and the Science Innovation Foundation of Dalian under Grant 2019J12GX037.

References

1. The caida anonymized internet traces. <http://www.caida.org/data/overview/>
2. Cisco netflow. <http://www.cisco.com>
3. AlGhadhban, A., Shihada, B.: Flight: a fast and lightweight elephant-flow detection mechanism. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1537–1538. IEEE (2018)
4. Alipourfard, O., Moshref, M., Zhou, Y., Yang, T., Yu, M.: A comparison of performance and accuracy of measurement algorithms in software. In: Proceedings of the Symposium on SDN Research, p. 18. ACM (2018)
5. Ben Basat, R., Einziger, G., Friedman, R., Luizelli, M.C., Waisbard, E.: Constant time updates in hierarchical heavy hitters. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 127–140. ACM (2017)
6. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
7. Brauckhoff, D., Tellenbach, B., Wagner, A., May, M., Lakhina, A.: Impact of packet sampling on anomaly detection metrics. In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, pp. 159–164. ACM (2006)
8. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45465-9_59
9. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
10. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput. Surv. **45**(3), 35–68 (2013)
11. Estan, C., Varghese, G.: New directions in traffic measurement and accounting. ACM SIGCOMM Comput. Commun. Rev. **32**, 323–336 (2002)
12. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. **31**(2), 182–209 (1985)
13. Gong, J., et al.: HeavyKeeper: an accurate algorithm for finding top-k elephant flows. In: 2018 USENIX Annual Technical Conference (USENIX ATC 2018), pp. 909–921 (2018)
14. Huang, Q., et al.: SketchVisor: robust network measurement for software packet processing. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 113–126. ACM (2017)
15. Li, Y., Miao, R., Kim, C., Yu, M.: FlowRadar: a better NetFlow for data centers. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2016), pp. 311–324 (2016)

16. Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., Braverman, V.: One sketch to rule them all: rethinking network flow monitoring with univmon. In: Proceedings of the 2016 ACM SIGCOMM Conference, pp. 101–114. ACM (2016)
17. Liu, Z., Gao, D., Liu, Y., Zhang, H., Foh, C.H.: An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network. *Int. J. Network Manage.* **27**(6), e1987 (2017)
18. Mai, J., Chuah, C.N., Sridharan, A., Ye, T., Zang, H.: Is sampled data sufficient for anomaly detection? In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, pp. 165–176. ACM (2006)
19. Poupart, P., et al.: Online flow size prediction for improved network routing. In: 2016 IEEE 24th International Conference on Network Protocols (ICNP), pp. 1–6. IEEE (2016)
20. Przybylski, S., Horowitz, M., Hennessy, J.: Characteristics of performance-optimal multi-level cache hierarchies. In: The 16th Annual International Symposium on Computer Architecture, pp. 114–121. IEEE (1989)
21. Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., Rexford, J.: Heavy-hitter detection entirely in the data plane. In: Proceedings of the Symposium on SDN Research, pp. 164–176. ACM (2017)
22. Wang, M., Li, B., Li, Z.: sFlow: towards resource-efficient and agile service federation in service overlay networks. In: Proceedings of the 24th International Conference on Distributed Computing Systems, pp. 628–635. IEEE (2004)
23. Wellem, T., Lai, Y.K., Chung, W.Y.: A software defined sketch system for traffic monitoring. In: Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp. 197–198. IEEE Computer Society (2015)
24. Wellem, T., Lai, Y.K., Huang, C.Y., Chung, W.Y.: A hardware-accelerated infrastructure for flexible sketch-based network traffic monitoring. In: 2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR), pp. 162–167. IEEE (2016)
25. Yang, T., et al.: Elastic sketch: adaptive and fast network-wide measurements. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pp. 561–575. ACM (2018)
26. Yang, T., et al.: Sf-sketch: a fast, accurate, and memory efficient data structure to store frequencies of data items. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 103–106. IEEE (2017)
27. Yang, T., et al.: Empowering sketches with machine learning for network measurements. In: Proceedings of the 2018 Workshop on Network Meets AI & ML, pp. 15–20. ACM (2018)
28. Zhou, A., Zhu, H., Liu, L., Zhu, C.: Identification of heavy hitters for network data streams with probabilistic sketch. In: 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 451–456. IEEE (2018)
29. Zhou, Y., Jin, H., Liu, P., Zhang, H., Yang, T., Li, X.: Accurate per-flow measurement with bloom sketch. In: IEEE INFOCOM 2018–IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1–2. IEEE (2018)
30. Zhou, Y., Liu, P., Jin, H., Yang, T., Dang, S., Li, X.: One memory access sketch: a more accurate and faster sketch for per-flow measurement. In: GLOBECOM 2017–2017 IEEE Global Communications Conference, pp. 1–6. IEEE (2017)