Priority-Aware Per-flow Measurement using Cuckoo Sketch

Yibo Yan[1], Cheng Chen[2], Huiping Lin[2], Olivier Ruas[2], Tengjiao Wang[2], and Tong Yang[2]

[1]School of Electronics and Computer Engineering (SECE), Peking University, Shenzhen, P.R. China
[2] Peking University, Beijing, P.R. China
{yanyibo, chen.cheng, phoenixrain}@pku.edu.cn, olivier.ruas@gmail.com,
tjwang@pku.edu.cn, yangtongemail@gmail.com

***Abstract*—Flow size estimation is a key task in per-flow measurement. In real scenarios, among millions of flows, people are often particularly interested in a small subset of given flows. These flows with high priorities are called *important flows*. In this poster, we propose the Cuckoo sketch, which provides higher accuracy for important flows. The key idea of the Cuckoo sketch is to separate the important flows from the unimportant ones and store them in different structures: the important flows are stored in a Cuckoo hash table where the exact size of the flow is stored while the unimportant flows are stored in a Count-Min sketch. The reallocation mechanism of the Cuckoo hashing is used to make some space in the Cuckoo hash table for the important flow by evicting the ones that are less important, which leads to a slight loss in throughput. Experimental results show that Cuckoo sketch provides better accuracy on important flows, reducing the average relative error by up to 69%, with only suffering from a negligible loss in accuracy for unimportant flows.**

I. Introduction

Per-flow measurement plays an important role in network management. It provides usage accounting [1] with fundamental traffic statistics and finds heavy hitters and heavy for intrusion detection [2], [3]. Being able to identify and quantify the different flows is important because it has an important impact on how the packets will be handled. For example, ISPs ensure high Quality of Service (QoS) for premium users. To meet this end, they need to achieve more accurate flow size estimation on the network traffic flows from or to the premium users than other flows for traffic accounting and billing.

There exist a few studies in network measurement that make a distinction between the flows. Cuckoo sampling [4] provides a sampling method that guarantees enough memory for incoming flows by discarding information when memory is not enough. However, sampling-based methods are not friendly to mice flows. Besides, Cuckoo Counter [5] proposes a method that utilizes the Cuckoo hashing to evict mice flows from entries with large sizes for elephant flows, which provides higher accuracy for elephant flows. Each entry of the Cuckoo Counter records the fingerprint and the number of packets of a flow. Because a flow whose the number of packets is not exact when different flows are stored in its entry, the fingerprint of the flow is useless for querying, which causes the waste of memory. This method is quite similar to our method. The main difference between them is that our method achieves higher space utilization and is more flexible.

In this poster, we divide flows into two kinds: *(i)* ***important flows***, small in numbers but carrying the most valuable information, and **(ii)** ***unimportant flows*** representing the majority of the packets and whose value is of little importance. To provide important flows the highest accuracy as possible and guarantee that approximations for the estimations of unimportant flows are acceptable, we propose a two-layer sketch named Cuckoo sketch. Sketch is a kind of memory-efficient probabilistic data structures that provide both fast insertions and queries while producing high quality estimations. *The key idea of Cuckoo sketch is to separate important flows from unimportant flows and to store them into two different layers, one with high accuracy for important flows and one more memory efficient for unimportant flows*. Besides, Cuckoo hashing is employed to improve the load factor of the sketch.

II. Our solution: the Cuckoo sketch

In this section, we present the Cuckoo sketch and the two operations it supports: the insertion of a packet and the query of the estimation of the size of a given flow.

A. Data Structure

The Cuckoo sketch, represented in Fig. 1, is composed of two layers L_1 and L_2 and a priority information table (PIT).

The first layer L_1 is a hash table using Cuckoo hashing, and is associated with two hash functions h_1 and h_2. There are w_1 buckets in the layer L_1, and each bucket contains m cells. Let

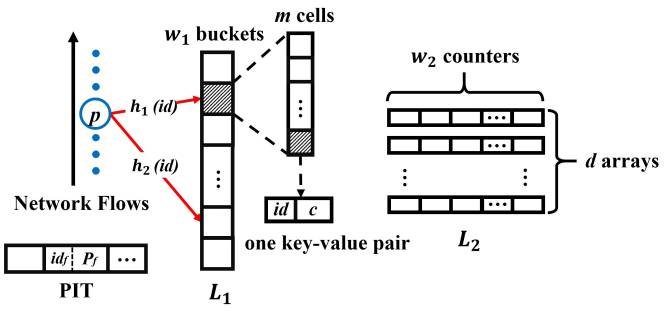


Fig. 1. Data structure of the Cuckoo sketch.

Annex to ISBN 978-3-903176-28-7© 2020 IFIP

id_f denote the ID of the flow f and c_f denote the estimated size of the flow f. Each cell stores one key-value pair $\langle id, c\rangle$ whose key is the ID of a flow and whose value is the estimated size of the flow. The ID of a flow could be any combination of its source IP address, source port, protocol, destination IP address and destination port. The j^{th} cell of the i^{th} bucket in the layer L_1 is denoted as $L_1[i][j]$ $(i \in [1, w_1], j \in [1, m])$. We mainly use this layer to store important flows. The second layer L_2 of the Cuckoo sketch is a CM sketch [6]. The CM sketch consists of d arrays, and each of the arrays contains w_2 counters. Arrays $(A_i)_{i\in[1,d]}$ are associated with d hash functions $(g_i)_{i\in[1,d]}$, respectively. The j^{th} counter of the i^{th} array in the layer L_2 is denoted as $L_2[i][j]$ $(i \in [1, d], j \in [1, w_2])$. Note that these d hash functions $(g_i)_{i\in[1,d]}$ need to be pairwise independent. The second layer is used to store the unimportant flows, resulting in a large gain in memory while guaranteeing good accuracy.

To make a distinction between important and unimportant flows, we assume that each flow has a priority. The priority can be determined according to applications' demands. The higher the priority is, the more important the flow is. The priorities of flows are maintained in a table denoted as PIT (priority information table). Each entry of the table records a flow's ID and priority. The priority of a flow f is denoted as P_f. Besides, we introduce virtual priorities to flows. The virtual priority of a flow is an arbitrary value given by a function $k(.)$. Given the ID of a flow f, the virtual priority of the flow is $k(id_f)$. Note that the virtual priority is independent of the applications, and it can be implemented as a classic hash function: the only requirement is the existence of a total order on the hash values.

B. Insertion

Initially, all the key-value pairs stored in the layer L_1 are initialized to $\langle null, 0\rangle$, and all the counters in the layer L_2 are initialized to 0. For each incoming packet p belonging to flow f, the Cuckoo sketch constructs a key-value pair $kv = \langle id_f, 1\rangle$ which is first tried to be inserted into L_1. If the insertion fails, the key-value pair kv is then inserted into the layer L_2.

a) Insertion in the first layer L_1: To store the key-value pair kv in the layer L_1, the two hashes $h_1(id_f) \in [1, w_1]$ and $h_2(id_f) \in [1, w_1]$ are computed to map kv to two corresponding buckets $L_1[h_1(id_f)]$ and $L_1[h_2(id_f)]$. The two buckets are checked at the same time. There are three cases:

- Case 1: If there is already a pair $\langle id_f, c_f\rangle$ in either $L_1[h_1(id_f)]$ or $L_1[h_2(id_f)]$, then the associated estimated flow size c_f is incremented by 1. Then insertion ends.
- Case 2: If there is no key-value pair whose key is id_f in either $L_1[h_1(id_f)]$ nor $L_1[h_2(id_f)]$ but there is at least one empty cell in one of the two buckets, the pair kv is stored in any one empty cell. Then insertion ends.
- Case 3: If there is no key-value pair whose key is equal to id_f in either $L_1[h_1(id_f)]$ nor $L_1[h_2(id_f)]$ and there is no empty cell, the pair whose corresponding flow has the lowest priority in both buckets is tried to be evicted to make some space for kv. More precisely, let f_{min} be the flow with the lowest priority and kv_{min} be its associated key-value pair. In the case there are several flows sharing the same lowest priority, one flow whose virtual priority is the lowest is chosen. If there are many flows whose virtual priorities are the lowest among the flows whose priorities are the lowest, one of them will be randomly chosen. If f_{min} has a higher priority than f, i.e. $P_{f_{min}} > P_f$, kv will be stored in the second layer L_2. If f has the same priority as f_{min} and the virtual priority of f is lower than or equal to that of f_{min}, kv will also be stored in the second layer L_2. Otherwise, kv_{min} is evicted, and f_{min} becomes an evicted flow. Then, kv is stored at its place, and kv_{min} is reallocated using the same insertion mechanism.

This process is done recursively until the evicted flow finds an empty cell or is stored in the second layer. To prevent prohibitive computation time, the process stops after a predefined number of T recursive calls. After T recursive calls, the current evicted flow is directly stored into the second layer.

b) Insertion in the second layer L_2: A key-value pair $kv = \langle id, c\rangle$ is inserted into the second layer L_2 by computing the d hash functions $(g_i(id))_{i\in[1,d]}$ to map the corresponding flow to d counters $(L_2[i][g_i(id)])_{i\in[1,d]}$. Then, the Cuckoo sketch increments all the d counters by c:

$$\forall i \in [1, d], L_2[i][g_i(id)] = L_2[i][g_i(id)] + c$$

C. Query

Similarly to the insertion, the query is first performed in the first layer, and if the queried flow is not found, it is then queried in the second layer. To query a flow f in the first layer, the two hashes $h_1(id_f)$ and $h_2(id_f)$ are computed and all the cells of the corresponding buckets $L_1[h_1(id_f)]$ and $L_1[h_2(id_f)]$ are checked. If a key-value pair whose key is id_f is found, its value c is returned as the estimated size of f. If no such key-value pair is found, then we estimate the size of the flow by using the second layer. For a CM sketch, the estimation is the minimum value of all the counters $(L_2[i][g_i(id_f)])_{i\in[1,d]}$:

$$\min\{L_2[i][g_i(id_f)], i \in [1, d]\}$$

Motivation: One might wonder why the Cuckoo sketch relies on such an insertion scheme when a simpler one could have worked: if a flow is important, then the flow is stored in L_1, else it is stored directly in L_2. While it may have been a good solution when the important flows fit perfectly in the Cuckoo hash table, it fails to provide an efficient solution for an unknown stream. Indeed, if the important flows are few in numbers, the Cuckoo hash table will be wasting space, and the resulting estimation for the unimportant flows will be lower than with the Cuckoo sketch. On the other hand, if there are too many important flows, the Cuckoo hash table becomes full and unable to store anymore flows, while the Cuckoo sketch makes space for the most important flows in the first layer by reallocating the least important one into the second layer.

III. Performance Evaluation

A. Experimental Setup

Dataset: We use seven real IP traces from CAIDA [7] to evaluate the performance of the Cuckoo sketch. In our experiments, the priority of each flow is chosen at random. Priorities are integers going from 0 to 7 and the number of flows for each priority is following the Zipf distribution [8], a distribution widely used for modeling network traffic [9]. The probability mass function of Zipf distribution is defined as $Pmf_{\alpha,n}(k) = \frac{k^{-\alpha}}{\sum_{i\in\{1,\ldots,n\}} i^{-\alpha}}$ where α is the skewness parameter of the Zipf distribution and n is a positive integer. We set n to 8 and α to 2.

Implementation: We compare our Cuckoo sketch with three typical sketches: the CM sketch, the CU sketch [10], and the Count sketch [11]. All these sketches are implemented in C++, and we have released the source code at GitHub [12]. For these typical sketches including the second layer of the Cuckoo sketch, each of them has three arrays, and the size of their counters is 4-byte. Each cell in L_1 is 8-byte, and 4 bytes is for storing the ID of a flow. In addition, T is set to 10.

Evaluation metrics: We evaluate the performance of the Cuckoo sketch through ARE and RMSE.

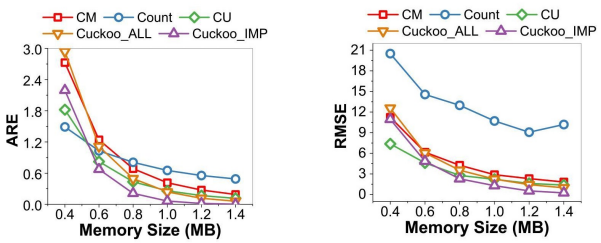
- *Average Relative Error (ARE)*: $\frac{1}{n}\sum_{i=1}^{n}\frac{|f_i-\hat{f}_i|}{f_i}$, where n is the number of flows, f_i is the actual flow size, and $\hat{f}_i$ is the estimated flow size. It is used to evaluate the accuracy of flow size estimation, and large ARE indicates low accuracy. We are interested in two distinct ARE values: the ARE for all flows, denoted as $Cuckoo\_ALL$, and the ARE for important flows only, denoted as $Cuckoo\_IMP$.
- *Root Mean Square Error (RMSE)*: $\sqrt{\frac{\sum_{i=1}^{n}(\hat{f}_i-f_i)^2}{n}}$, where the n, the $\hat{f}_i$, and the f_i are the same as defined in the ARE.

B. Experimental Results

We evaluate the performance of the Cuckoo sketch by varying the memory size from 0.4MB to 1.4MB.

ARE vs. memory size (Fig. 2(a)): The AREs of all the sketches decreases as the memory size increases, and compared with the CM sketch, CU sketch, and the Count sketch, the Cuckoo sketch benefits more from enlarging memory size. When the memory size is 0.4MB, both of the Count sketch and the CU sketch are more accurate than the Cuckoo sketch. As the memory size increases to 1MB, the Cuckoo sketch becomes the most accurate sketch among those sketches. The AREs of the Cuckoo_ALL and the Cuckoo_IMP are 0.24 and 0.06, respectively, while the AREs of the Count sketch and the CU sketch is 0.65 and 0.26.

RMSE vs. memory size (Fig. 2(b)): As expected, the change of memory size has a similar impact on RMSE. When the memory size is 0.8MB, the RMSE of the Cuckoo_IMP is the smallest. As the the memory size creases to 1.2MB, the Cuckoo sketch becomes the most accurate sketch and its RMSE is 1.6, 6.28, and 1.12 times lower than that of the CM sketch, the Count sketch, and the CU sketch, respectively.



(a) ARE vs. Memory size (b) RMSE vs. Memory size

Fig. 2. Experiments on memory sizes.

IV. Conclusion

In this poster, we propose the Cuckoo sketch, a priority-aware sketch that provides a better accuracy for important flows. The Cuckoo sketch improves the accuracy of the important flows by separating them from the unimportant ones: a Cuckoo hash table is used to stored the important flows with their exact size while the unimportant flows are stored in a Count-Min sketch, which stores only an approximation of the sizes. Our experimental results show that the Cuckoo sketch outperforms its competitors, providing a better size approximation for important flows.

References

[1] CRISTIAN ESTAN and GEORGE VARGHESE. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *Acm Transactions on Computer Systems*, 21(3):p.270–313, 2003.

[2] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings 2003 VLDB Conference*, pages 464–475. Elsevier, 2003.

[3] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A. Dinda, Ming Yang Kao, and Gokhan Memik. Reversible sketches: Enabling monitoring and analysis over high-speed data streams. *Networking IEEE/ACM Transactions on*, 15(5):1059–1072, 2007.

[4] Josep Sanjuaas-Cuxart, Pere Barlet-Ros, Nick Duffield, and Ramana Kompella. Cuckoo sampling: Robust collection of flow aggregates under a fixed memory budget. *Proceedings IEEE Infocom*, pages 2751–2755, 2012.

[5] Jiuhua Qi, Wenjun Li, Tong Yang, Dagang Li, and Hui Li. Cuckoo counter: A novel framework for accurate per-flow frequency estimation in network measurement. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 1–7. IEEE, 2019.

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

[7] The CAIDA Anonymized Internet Traces. http://www.caida.org/data/overview/.

[8] David MW Powers. Applications and explanations of zipf's law. In *Proceedings of the joint conferences on new methods in language processing and computational natural language learning*, pages 151–160. Association for Computational Linguistics, 1998.

[9] Lada A Adamic and Bernardo A Huberman. Zipf's law and the internet. *Glottometrics*, 3(1):143–150, 2002.

[10] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[11] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[12] The source codes of our and other related algorithms. https://github.com/Cuckoo-Sketch/Cuckoo-Sketch.