Computers, Materials & Continua CMC, vol.63, no.1, pp.213-222, 2020

Efficient Heavy Hitters Identification over Speed Traffic Streams

Shuzhuang Zhang[1], Hao Luo[1], Zhigang Wu[1], Yanbin Sun[2, *], Yuhang Wang[2] and Tingting Yuan[3]

Abstract: With the rapid increase of link speed and network throughput in recent years, much more attention has been paid to the work of obtaining statistics over speed traffic streams. It is a challenging problem to identify heavy hitters in high-speed and dynamically changing data streams with less memory and computational overhead with high measurement accuracy. In this paper, we combine Bloom Filter with exponential histogram to query streams in the sliding window so as to identify heavy hitters. This method is called EBF sketches. Our sketch structure allows for effective summarization of streams over time-based sliding windows with guaranteed probabilistic accuracy. It can be employed to address problems such as maintaining frequency statistics and finding heavy hitters. Our experimental results validate our theoretical claims and verifies the effectiveness of our techniques.

Keywords: Traffic stream, heavy hitter, sliding window, frequency statistics.

1 Introduction

Detecting the most frequent items in large datasets and making accurate frequency estimations is a common query in big data and is of great significance for data monitoring [Zhou, Zhu, Liu et al. (2018); Tian, Su, Shi et al. (2019); Tian, Li, Qiu et al. (2019); Pan, Yang, Sheng et al. (2019); Pan, Yu, Yi et al. (2019); Pan, Qin, Yi et al. (2019)] and intrusion detection [Zhang, Yi, Wang et al. (2018)]. Abnormal traffic usually undergoes dramatic frequency changes during a certain period, and these changes are always sharp increases. This kind of data item is called a heavy hitter. The term "heavy hitter" is commonly used in data stream and network monitoring researches [Wang and Tao (2018); Galea, Moore, Antichi et al. (2018); Li, Sun, Jiang et al. (2018)]. A heavy hitter may correspond to a stream, a connection, or an aggregation of streams or connections. According to the definition above, it is obvious that many large-scale network attacks have the characteristics of heavy hitters, such as DDoS attacks and Eclipse attacks [Tan, Gao, Shi et al. (2018)].

[1] Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing, China.

[2] Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, China.

[3] Inria Diana Sophia Antipolis-Mediterranee, Sophia Antipolis, France.

[*] Corresponding Author: Yanbin Sun. Email: sunyanbin@gzhu.edu.cn.

Received: 28 May 2019; Accepted: 01 July 2019.

We need to work out algorithms to handle traffic streams in real time [Tian, Shi, Wang et al. (2019)], find heavy hitters, alert systems to them, and take measures to maintain network security. These algorithms must accurately reflect the heavy hitters (with guaranteed error) and ensure efficiency and feasibility. Traffic streams are born naturally over time. Each data item in a traffic stream carries a timestamp. We are more concerned about the behaviors of new data items that have recently arrived when we identity heavy hitters. We do not need to store the entire data stream and summarize the frequency of each data item, which just consumes huge space and is of no benefit. Instead, we should work out the frequency of data items over a recent period to determine whether there has been a sharp increase in frequency.

Various time-decay models for querying streaming data have been proposed in the literature [Edith and Martin (2003)], mostly concerning the relation of an item's weight with its age. The sliding window model [Datar, Gionis, Indyk et al. (2002); Ben, Einziger, Friedman et al. (2016)] is one of the most prominent and intuitive time-decay models that takes into account only the window of the most recent items seen in the stream so far. The window itself may be time-based or count-based. The sliding window model maintains the statistical properties of data, such as base counts and sums, but when the amount of data is too large it will take too much space. Our work is to identify heavy hitters in the traffic stream with guaranteed error and simultaneously use as little space as possible.

The rest of this paper is organized as follows: In Section 2, we discuss the related work on heavy hitter identification. In Section 3, we describe the EBF-sketches algorithm and analyze its complexity. In Section 4, we evaluate our experimental results. Finally, in Section 5, we conclude the paper.

2 Related work

In recent years, identifying heavy hitters in data streams has gained more and more attention. A large-scale data stream is characterized by continuous uninterrupted-arriving, fast and large-scaled number [Babcock, Babu, Datar et al. (2002)]. Since a data stream is infinite, it is obviously unrealistic to materialize the entire data stream, and since many applications require real-time performance, we need a one-shot scanning algorithm. At present, there are some commonly used methods to identify heavy hitters, such as data mining [Knorr and Ng (1998)], feature selection, and the neural network [Zhang, Qiu, and Li (2001)]. However, these methods require a lot of data training and high computational cost, and their real-time performance is very poor. Several algorithms have been proposed for maintaining different types of statistics over sliding window data streams, while requiring time and space that is significantly sub-linear [Gibbons and Tirthapura (2004); Qiao, Agrawal and Abbadi (2003); Tirthapura, Xu and Busch (2006)].

Most previous work on data stream processing has focused on developing space-efficient, one-pass algorithms for performing a wide range of centralized, one-shot computations on massive data streams; examples include computing quantiles [Greenwald and Khanna (2001)], estimating distinct values [Gibbons (2001)], counting frequent elements (i.e., "heavy hitters") [Charikar, Chen and Farach (2002); Cormode and Muthukrishnan (2005)], and estimating join sizes and stream norms [Alon, Matias and Szegedy (1996); Cormode and Muthukrishnan (1970)]. Out of these efforts, flexible, general-purpose

sketch summaries, such as the AMS [Alon, Matias, and Szegedy (1996)] and the Count-Min sketch [Cormode and Muthukrishnan (1970)], have found wide applicability in a broad range of stream-processing scenarios. However, these referenced works do not address the issues specific to the sliding window model.

Existing work on the sliding window model has focused on algorithms for maintaining simple statistics, such as basic counts and sums. Exponential histogram [Datar, Gionis, Indyk et al. (2002)] is a deterministic technique for maintaining ε-approximate counts and sums over sliding windows using $O\left(\frac{1}{\varepsilon}\log 2N\right)$ space. Deterministic waves [Gibbons and Tirthapura (2004)] solve the same basic counting/summation problem with the same space complexity as exponential histogram, but improve the worst-case update time complexity to $O(1)$; conversely, randomized waves [Gibbons and Tirthapura (2004)] rely on randomization through hashing to track duplicate-insensitive counts (i.e., Count-Distinct aggregates) over sliding windows.

Hung et al. [Hung and Ting (2008)] and Dimitropoulos et al. [Dimitropoulos, Stoecklin, Hurley et al. (2008)] proposed synopses based on Count-Min sketches for tracking heavy hitters and frequency counts over sliding windows; still, their techniques relied on keeping simple equi-width counters within the sketch, and thus, could not provide any meaningful error guarantees, especially for small query ranges. Similarly, the hybrid histograms of Qiao et al. [Qiao, Agrawal and Abbadi (2003)] combined exponential histogram with simplistic equi-width histograms for answering sliding window range queries; again, these structures could not give meaningful bounds on the approximation error. ECM sketches combine the functionalities of Count-Min sketches [Cormode and Muthukrishnan (1970)] and exponential histogram, and support both time-based and count-based sliding windows under the cash register model and give guaranteed error.

Our work combines the functionalities of Bloom Filter and exponential histogram. The combination of the two structures greatly reduces the need of space, and its real-time processing requires no complicated training in advance, which leads to high efficiency.

3 EBF sketches

3.1 Problem statement

In this paper, we format the data item in the traffic streams as follows: <*key*, *value*, *timestamp*>. We need to design an algorithm that can efficiently work on a collection of items with sharp changes during the recent periods.

As introduced before, an algorithm that supports dynamic, constantly updating, large-scale data queries are required to identify heavy hitters in traffic streams. Since traffic streams are constantly updating over time, queries should focus more on new data than on old data. Thus, we only work on the data within the sliding window that arrived recently. In simple terms, what we need to do is to calculate the frequency of data items that arrive during a certain period in real time and see whether it exceeds the threshold. We can define the problem as follows:

Problem 1. Given a threshold $\tau > 0$ and a data stream *S*, *S* contains a large number of timestamped data items, and each of them is composed of a key, a value, and a timestamp, i.e., <*key*, *value*, *timestamp*>. What we should do is determine whether a data item appearing with frequency f during a certain period is a heavy hitter, that is, whether $f > \tau$.

In this paper, we just compute approximately with guaranteed error to sacrifice a bit of accuracy in exchange for space savings. We show this in detail using Bloom Filter [Broder and Mitzenmacher (2002)] and exponential histogram [Datar, Gionis, Indyk et al. (2002)].

Now, we introduce the EBF-sketches algorithm. It is a structure that maintains statistics of data streams on sliding windows. The core of the EBF-sketches structure is a modified Bloom Filter. The standard Bloom Filter is just a bit array used to determine whether an element belongs to a collection. Without the counting device, it cannot answer the frequency of data items in a sliding window. We combine it with sliding window structures for the sake of efficiency. We maintain a counter for each bit in the Bloom Filter array so as to store information of every data item. Each counter is implemented as a time-based sliding window, which overlays the elements reached in the last N time units.

We have already discussed before how to implement the sliding window. To ensure that the Bloom Filter's error is guaranteed, the number of bits of the array is very large. We maintain a counter for each bit and hope it occupies as little memory as possible; otherwise, the space required for the entire dataset will not be acceptable. Therefore, we use the exponential histogram. Each bit of Bloom Filter points to an exponential histogram, and when computing on the sliding window with a time span of *N* units, the relative error will not exceed ε. For example, if a certain bit really counts as *x*, the returned counting statistics within the sliding window will be in the range of $(1 \pm \varepsilon)x$.

3.2 Working process

The pseudo-code of the main functions is shown in Fig. 1. When a data item arrives, as the first step, the *k* hash functions of EBF-sketches hash the key and get *k* results. The k hash functions can be derived from two initial hash functions, using the high and low 32 bits of a 64-bit hash function for cyclic shifting. In the second step, we use these k results as an index to find the corresponding counter in the Bloom Filter counter array, which is implemented as an exponential histogram, and increase it by one. As mentioned above, the counter here cannot simply use an accumulation counter because we never know the starting moment of an infinite data stream, and we never know when to start accumulating. Therefore, we use the sliding window model to implement the counters. Exponential histogram is a typical sliding window model that greatly reduces the storage space and computation time. In the third step, since we have already obtained the values of *k* counters that correspond to the data item, we choose the smallest one to reduce the error brought by hash collision. Then, we compare the smallest value with the threshold. If the threshold is exceeded, we report it as a heavy hitter. The entire process does not need complicated calculations. With only $O(N)$ time cost, the proposed method has high real-time performance.

```
Algorithm Arrival (key, value, time)
1       result[]=hash(key);
2       count=MAX_VALUE;
3       for i=0 to result.length-1do
4         temp= BF_Array[i].EH_Add(value, time);
5           If temp<count then
6         count=temp;
5           end if
6       end for
7       if count >τ then
8       report();
```

```
Algorithm EH_Add (value,time)
1       update expired time
2       if the last bucket is expired then
3         delete the last bucket and update related data;
4       end if
5       add the new coming data;
6       if merge is needed then
7         merge the buckets;
8       else
9         return;
10      end if
```

Figure 1: Pseudo-code of main functions

3.3 Complexity analysis

We use N to denote the number of data items within the sliding window, that is, the number of data items represented by each exponential histogram. A previous study [Datar, Gionis, Indyk et al. (2002)] stated that each bucket in the exponential histogram maintains its size and timestamp. The bucket size occupies at most $\log\log N$ bits, and the timestamp occupies at most $\log N$ bits; therefore, each bucket occupies at most $\log N + \log\log N$ bits. For an exponential histogram, there are at most $O\left(\log N / \varepsilon_{EH}\right)$ buckets, and thus the total space required for each exponential histogram is $O\left(1/\varepsilon_{EH} \log^2 N\right)$. Count queries can be processed in $O(1)$ time cost because the exponential histogram maintains two counts: the size of the last bucket and the sum of all the buckets. While inserting a new element, the insertion time cost is optimally $O(1)$, and the worst time cost can be $O\left(\log N\right)$ due to bucket merge.

The space complexity of EBF sketches is $O\left(\frac{1}{\varepsilon_{EH}} \ln\left(1/\varepsilon_{BF}\right) N \log^2 N\right)$, where ε_{BF} and ε_{EH} denote the error of the Bloom Filter and exponential histogram, respectively. In

order to guarantee the error, the length of the bit array is at least about $n\ln(1/\varepsilon_{BF})$. Each bit is implemented as an exponential histogram, which space required is $O\left(\frac{1}{\varepsilon_{EH}}\log^2 N\right)$.

In terms of time complexity, every time a data item is added, it needs k times of hashing, and needs to update k exponential histograms. Therefore, the average time complexity is $O(k)$ and the worst-case time complexity is $O(k\log N)$.

4 Experimental evaluation

Our EBF sketches are implemented in Java 1.7, and the machine is a 64-bit laptop. In terms of dataset, we use a real network dataset that we call *blog*. The *blog* dataset contains 15 million blog items.

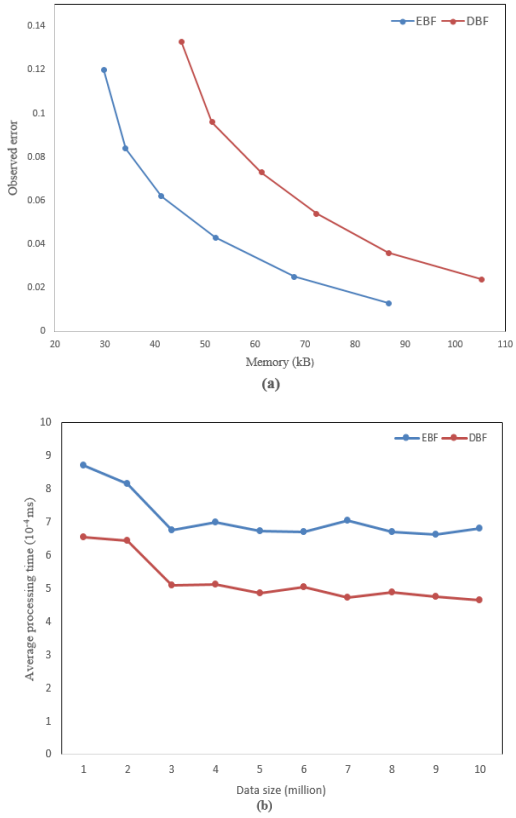

Figure 2: Comparison of EBF sketches and DBF sketches in terms of the observed error and average processing time

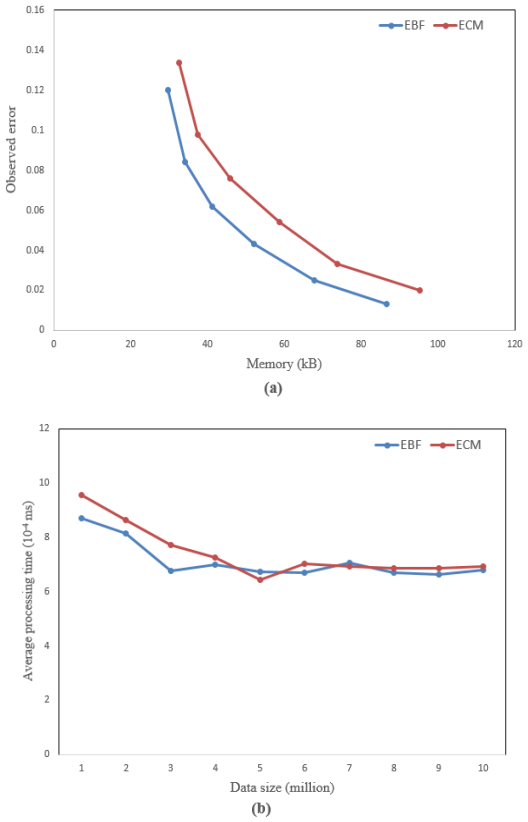

Figure 3: Comparison of EBF sketches and ECM sketches in terms of the observed error and average processing time

Fig. 2 plots the observed error and average processing time of the Bloom Filter combined with exponential histogram (the EBF sketches) and those of the deterministic waves (the DBF sketches). Fig. 2(a) shows the observed error in correlation with the required memory for the dataset. We can see that the EBF sketches require much less space than the DBF sketches for the same ε. The EBF sketches require about 90 kB to get a relative error of 0.01, but with identical memory the DBF sketches can only reach a relative error of 0.04. However, we can see from Fig. 2(b) that, concerning the average processing time, the DBF sketches are slightly faster than the EBF sketches, mainly due to its $O(k)$ worst-case complexity per update compared to the $O(k \log N)$ for the EBF sketches. However, considering the advantages of high accuracy and high memory efficiency, the slight disadvantage of processing time can be completely ignored.

Fig. 3 plots the observed error and average processing time of exponential histogram combined with the Bloom Filter (the EBF sketches) and the Count-Min sketch (the ECM sketches). Fig. 3(a) shows the observed error in correlation with the required memory for the dataset. We can see that the EBF sketches require less space than the ECM sketches for the same ε. We have explained above that a Count-Min sketch is implemented as a

two-dimensional array of counters, that is, an array for each hash function. However, the Bloom Filter uses only one counter array so that it requires less memory. The EBF sketches require about 90 kB to get a relative error of 0.01, but with identical memory the ECM sketches can only reach a relative error of 0.025. As seen in Fig. 3(b), concerning the average processing time, the two structures have very close performance. This is mainly because they both use exponential histogram to implement the counter. For these two algorithms, the time cost for every update mainly depends on the performance of the exponential histogram.

In summary, these results demonstrate the superiority of the EBF sketches compared to the DBF sketches and the ECM sketches, in terms of accuracy, memory efficiency, and computational performance.

5 Conclusions

In this work, we considered the problem of identifying heavy hitters in the network. Our proposal, the EBF sketches, utilizes the functionality of Bloom Filter, which answers existence problems with deterministic sliding window synopses. This structure provides probabilistic accuracy guarantees for the estimated quality and has good memory efficiency. We introduced how to combine Bloom Filter and exponential histogram to solve the considered problem and provide guaranteed accuracy. The EBF sketches were thoroughly evaluated with a set of experiments, and the results verified the high performance of the structure.

Acknowledgement: This study is supported by National key research and development program (2016YFB0801200).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Alon, N.; Matias, Y.; Szegedy, M. (1996): The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 20-29.

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; Widom, J. (2002): Models and issues in data streams system. *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 1-16.

Ben-Basat, R.; Einziger, G.; Friedman, R.; Kassner, Y. (2016): Heavy hitters in streams and sliding windows. *IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on Computer Communications*, pp. 1-9.

Broder, A.; Mitzenmacher, M. (2002): Network applications of bloom filters: a survey. Internet Mathematics. *Internet Mathematics*, vol. 1, no. 4, pp. 485-509.

Charikar, M.; Chen, K.; FarachColton, M. (2002): Finding frequent items in data streams. *Theoretical Computer Science*, vol. 312, no. 1, pp. 693-703.

Cormode, G; Muthukrishnan, S. (1970): An improved data stream summary: the count-min sketch and its applications. *LATIN 2004: Theoretical Informatics*, vol. 55, no. 1, pp. 29-38.

Cormode, G.; Muthukrishnan, S. (2005): What's hot and what's not: tracking most frequent items dynamically. *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 249-278.

Datar, M.; Gionis, A.; Indyk, P.; Motwani, R. (2002): Maintaining stream statistics over sliding windows. *Siam Journal on Computing*, vol. 31, no. 6, pp. 1794-1813.

Dimitropoulos, X.; Stoecklin, M.; Hurley, P.; Kind, A. (2008): The eternal sunshine of the sketch data structure. *Computer Networks the International Journal of Computer & Telecommunications Networking*, vol. 52, no. 17, pp. 3248-3257.

Edith, C.; Martin, J. S. (2003): Maintaining time-decaying stream aggregates. *Journal of Algorithms*, vol. 59, no. 1, pp. 223-233.

Galea, S.; Moore, A. W.; Antichi, G.; Bianchi, G.; Bifulco, R. (2018): Revealing hidden hierarchical heavy hitters in network traffic. *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, pp. 81-83.

Gibbons, P. B. (2001): Distinct sampling for highly-accurate answers to distinct values queries and event reports. *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 541-550.

Gibbons, P. B.; Tirthapura, S. (2004): Distributed streams algorithms for sliding windows. *Theory of Computing Systems*, vol. 37, no. 3, pp. 457-478.

Greenwald, M.; Khanna, S. (2001): Space-efficient online computation of quantile summaries. *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, vol. 30, no. 2, pp. 58-66.

Hung, R. Y. S.; Ting, H. F. (2008): Finding heavy hitters over the sliding window of a weighted data stream. *LATIN 2008: Theoretical Informatics. Springer Berlin Heidelberg*, vol. 4957, pp. 699-710.

Knorr, E. M.; Ng, R. T. (1998): Algorithms for mining distance-based outliers in large data sets. *Proceedings of 24th International Conference on Very Large Databases*, pp. 392-403.

Li, M.; Sun, Y.; Jiang, Y.; Tian, Z. (2018): Answering the min-cost quality-aware query on multi-sources in sensor-cloud systems. *Sensors*, vol. 18, no. 3, pp. 4486.

Pan, Z.; Qin, H.; Yi, X.; Zheng, Y.; Khan, A. (2019): Low complexity versatile video coding for traffic surveillance system. *International Journal of Sensor Networks*, vol. 30, no. 2, pp. 116-125.

Pan, Z.; Yang, C. N.; Sheng, V. S.; Xiong, N.; Meng, W. (2019): Machine learning for wireless multimedia data security. *Security and Communication Networks*, vol. 2019, no. 7682306, pp. 2.

Pan, Z.; Yu, W.; Yi, X.; Khan, A.; Yuan, F. et al. (2019): Recent progress on generative adversarial networks (GANs): a survey. *IEEE Access*, vol. 7, pp. 36322-36333.

Qiao, L.; Agrawal, D.; Abbadi, A. E. (2003): Supporting sliding window queries for continuous data streams. *Proceeding of the 15th International Conference on Scientific and Statistical Database Management*, pp. 85-94.

Tan, Q. F.; Gao, Y.; Shi, J. Q.; Wang, X. B.; Fang, B. X. et al. (2018): Towards a comprehensive insight into the eclipse attacks of tor hidden services. *IEEE Internet of Things Journal*, vol. 6, no. 2. pp. 1584-1593.

Tian, Z. H.; Shi, W.; Wang, Y. H.; Zhu, C. S.; Du, X. J. et al. (2019): Real time lateral movement detection based on evidence reasoning network for edge computing environment. *IEEE Transactions on Industrial Informatics*, pp. 1-9.

Tian, Z. H; Su, S.; Shi, W.; Du, X. J.; Mohsen, G. et al. (2019): A Data-driven model for future internet route decision modeling. *Future Generation Computer Systems*, vol. 95, pp. 212-220.

Tian, Z. H.; Li, Mohan; Qiu, M. K.; Sun, Y. B.; Su, S. (2019): Block-DES: a secure digital evidence system using blockchain. *Information Sciences*, vol. 491, pp. 151-165.

Tirthapura, S.; Xu, B.; Busch, C. (2006): Sketching asynchronous streams over a sliding window. *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing*, pp. 82-91.

Wang, S.; Tao, Y. (2018): Efficient algorithms for finding approximate heavy hitters in personalized pageranks. *Proceedings of the 2018 International Conference on Management of Data*, pp. 1113-1127.

Zhang, H. B.; Yi, Y. Z.; Wang, J. S.; Cao, N.; Duan, Q. (2018): Network security situation awareness framework based on threat intelligence. *Computers, Materials & Continua*, vol. 56, no. 3, pp. 381-399.

Zhang, G. J.; Qiu, J. J.; Li, J. H. (2001): Outlier identification and justification based on neural network. *Proceeding of the Chinese Society of Electrical Engineering*, vol. 21, no. 8, pp. 104-107.

Zhou, A. P.; Zhu, H. S.; Liu, L. J.; Zhu, C. G. (2018): Identification of heavy hitters for network data streams with probabilistic sketch. *IEEE 3rd International Conference on Cloud Computing and Big Data Analysis*, pp. 451-456.