Efficient Identification of TOP-*K* Heavy Hitters over Sliding Windows

Haina Tang[1], Yulei Wu[2], Tong Li[3], Chunjing Han[3], Jingguo Ge[3], Xiangpeng Zhao[1]

Abstract. Due to the increasing volume of network traffic and growing complexity of network environment, rapid identification of heavy hitters is quite challenging. To deal with the massive data streams in real-time, accurate and scalable solution is required. The traditional method to keep an individual counter for each host in the whole data streams is very resource-consuming. This paper presents a new data structure called *FCM* and its associated algorithms. *FCM* combines the count-min sketch with the stream-summary structure simultaneously for efficient TOP-*K* heavy hitter identification in one pass. The key point of this algorithm is that it introduces a novel *filter-and-jump* mechanism. Given that the Internet traffic has the property of being heavy-tailed and hosts of low frequencies account for the majority of the IP addresses, *FCM* periodically filters the mice from input streams to efficiently improve the accuracy of TOP-*K* heavy hitter identification. On the other hand, considering that abnormal events are always time sensitive, our algorithm works by adjusting its measurement window to the newly arrived elements in the data streams automatically. Our experimental results demonstrate that the performance of *FCM* is superior to the previous related algorithm. Additionally this solution has a good prospect of application in advanced network environment.

Keywords heavy hitters · count-min sketch · Space Saving · sliding window

Haina Tang[1]
hntang@ucas.ac.cn

Yulei Wu[2]
y.l.wu@exeter.ac.uk

Tong Li[3]
litong@iie.ac.cn

Chunjing Han[3]
hanchunjing@iie.ac.cn

Jingguo Ge[3]
gejingguo@iie.ac.cn

Xiangpeng Zhao[1]
zhaoxiangpeng16@mails.ucas.ac.cn

[1] School of Engineering Science, University of Chinese Academy of Sciences, Beijing, 100049, China

[2] School of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, EX4 4QF, UK

[3] Institute of Information Engineering, Chinese Academy of Science, Beijing, 100195, China

1 Introduction

Currently, with the rapid growth of Internet traffic, characterizing network anomalies in real-time becomes progressively more challenging. An important indicator for detecting abnormal network events is the substantial increase in the number of flows [1-2]. For example, when a compromised host wants to infect a large number of other machines, it may carry out a wide range scan to find possible victims which will result in an unexpected large number of network connections. Moreover, during distributed denial-of-service (DDoS) attacks, the targeted machine typically receives tremendous requests from various sources which may overload the system and prevent it from providing normal service to its users.

The heavy hitter generally refers to an element with high occurrence in data streams. When defined in network

environment, it often represents a host which connects with large number of sources (or destinations) during a definite time interval. If we take the incoming element as NetFlow 5-tuple records, the problem of heavy hitter identification can be formulated as follows: given a stream $S_N = s_1,s_2 \ldots s_N$ with the format of <*sourceIP*, *sourcePort*, *destinationIP*, *destionPort*, *protocol*>, find *sourceIP* (or *destinationIP*) that is paired with a large number of *destinationIP* (or *sourceIP*) which is above a predefined threshold. Identification of heavy hitters is helpful for detecting the on-going malicious events such as DDoS attacks, worm propagation and port scans.

However, nowadays the exponentially increasing traffic and emerging network environment bring new challenge to the solution of this issue. According to Akamai latest Internet security report [3], the size, complexity and frequency of Internet malicious activities have greatly increased in the past decades. On the other side, considering the crucial importance of Smart Grids, malicious attacks are more damaging which may disrupt the sensitive information and critical operations on the complex infrastructure [4]. So given the strict requirement of Smart Grids in terms of robustness and performance [5-6], high demand for the accuracy and scalability of the recognizing algorithms is put forward accordingly.

In this paper, we work on the following problems which are ignored in most of previous studies. First, abnormal events are generally time-sensitive which require to be detected using the latest arrived items in data streams over sliding windows. Dealing with datasets in statically predefined interval may miss the targets which happen across the border of two adjacent measurement windows. Second, when the network behaves abnormally, the number of flows in definite interval may increase sharply. The problem is much more severe for high speed environment. Considering that Internet traffic has the property of being heavy-tailed, hosts of low frequencies make up a large proportion in data streams. Dealing with those hosts increases the memory consumption together with more possibilities of hash collisions for both counter-based and sketch-based solutions. Last, since currently network operators are accustomed to monitor the top lists of suspected events, the algorithm should be able to provide solution for both TOP-*K* and heavy hitter identification over sliding windows.

Targeting at those challenges, we propose a new algorithm called *FCM* for identifying heavy hitters. To summarize, the main contributions of this paper are as follows:

(1) The proposed *FCM* algorithm provides an efficient *filter-and-jump* scheme to detect Top-*K* heavy hitters over sliding windows. Count-min sketches are constructed to calculate the number of connections for each host in previous 2*s* seconds, and a dynamic pointer is used to continuously adjust the measurement window in real-time and trigger the update of the structure.

(2) Due to the limited computational resources of advanced network environments like Smart Grids, hosts with little possibility of heavy hitter should be deleted in time to make space for new data. *FCM* periodically filters hosts of low frequencies with size below certain threshold. Therefore, the accuracy of the algorithm can be greatly improved.

(3) A revised and enhanced version of the Space Saving algorithm is designed to get the TOP-*K* list of heavy hitters with arbitrary ranking. Although space saving has provided a lightweight and effective solution for TOP-*K* elements detection in data streams, papers also found that its over-estimate-error is relatively high [7]. Instead of incrementing its counter every time a new element arrives, our algorithm redesigns the space saving algorithm to update the corresponding counter only when needed.

(4) Extensive experiments are conducted to validate the accuracy and evaluate the performance of the proposed algorithm. The results show that the *FCM* outperforms the existing related algorithm in terms of false positive rate, false negative rate, ordering deviation rate and average estimate error rate.

The remainder of the paper is organized as follows. After introducing the related work of TOP-*K* heavy hitter identification in Section 2, we present the relevant

definitions to be used in subsequent sections in Section 3. Section 4 elaborates the design of the proposed *FCM* algorithm. Then a performance study of this algorithm is reported in Section 5. Section 6 evaluates the performance of the algorithm via extensive experiments, and finally Section 7 concludes this work.

2 Related Work

In this section, we summarize existing solutions for identifying heavy hitter elements.

Currently, extensive studies have been conducted and various algorithms have been proposed on identifying heavy hitters. For example, both the snort intrusion detection system [8] and Flowscan [9] detected port scans by maintaining a counter for every host which records the number of distinct IP addresses it connects with in the past several minutes. This solution is not feasible when carried out in high speed networks. Paper [10] proposed a family of bitmap algorithms to address this problem, in which a bitmap is allocated to estimate the number of contacts for every IP addresses, and another index structure is used for mapping the IP address to this bitmap. Obviously, maintaining 4 bytes bitmap for each IP address is space consuming. The authors in [11] presented a virtual connection degree sketch for estimating the connection of hosts in high speed environments, together with a filtered bitmap to reduce the noise information caused by the bit-sharing mechanism. Paper [12] proposed a hash-based algorithm to find heavy hitters which implements one-level/two-level filtering to sample the input streams. This algorithm demonstrates better scalability since it does not need to keep state for each host. However, paper [13] also pointed out that the hashing and sampling mechanism results in high estimate error, and the bucket-chaining and sorting process also brings overheads.

Another similar problem is the identification of frequent items which generally includes two kinds of solutions: counter-based algorithms such as Lossy Counting [14], Frequent [15], Space-Saving [16], and sketch-based methods such as Count Sketch [17], Count Min-Sketch (CM) [18], and LD-Sketch [19].

Sketch-based methods use techniques such as hashing to map items to a reduced set of counters, and maintain approximate summaries of all elements in the stream. Among those solutions, count-min sketch is predominant than others for summarizing large scale datasets with strong accuracy guarantees [20]. The algorithm works in a fixed amount of space to store counting information which is independent with the size of the input streams. However, due to its sub-linear space for the counters, papers [21-22] pointed out that the frequency-estimation error may increase as more and more items come, and the items with low frequency may be incorrectly recognized as high-frequency ones. Moreover, sketches are not reversible to provide the corresponding keys since they only store the counters, therefore they are not suitable to be used for TOP-*K* solution directly.

Counter-based algorithms maintain a small fixed number of items from the inputs, and monitor counts associated with them. Space-Saving is the best counter-based algorithm which can provide both frequent items and TOP-*K* solutions. It takes limited space and delivers better performance over other algorithms. Several papers [7][23] have pointed out the shortcoming of Space Saving and various solutions have been implemented on how to improve it efficiently. For example, considering the inaccuracy problem that elements which appear in the end of the data stream with low frequencies are more likely to be misclassified as TOP-*K* elements, Nuno Homem et al. [7] provided solutions which combines hash table and space saving method to eliminate the over-estimate-error. Parallel versions of Space Saving [24-25] have also been presented to accelerate its performance. Most recently Ran Ben-Basat et al. [26] redesigned Space Saving with statically allocated memory and demonstrated better performance for the space requirement.

For detection of heavy hitters over sliding windows, the authors in [27] presented an FSW algorithm (Filtered Space-Saving with Sliding Window) which implements additional histograms to track the expiration of the monitored elements. Paper [23] used divided frames to approximately address this issue. Zhen et al. [28] designed BF_LRU which adopts LRU to remove the

mice and bloom filters to represent elephants. Although LRU is efficiently applied to record the most recent elements with time locality property, it does not provide fixed size of measurement window. Recently Massimo et al. [29] presented a new sketch based algorithm for mining frequent items in data streams in the time fading model, which introduces an estimated decaying factor to fade older items.

3 Definitions of heavy hitter

3.1 Definition 1

Cdegree(H) — the connection degree of Host *H*: Given a host *H* with IP address IP_H, the connection degree of *H* is the number of destinations (sources) it connects with as source (destination) within certain measurement period.

3.2 Definition 2

TOP-$K(S_N)$ — TOP-*K* heavy hitters in data stream S_N: Assuming a data stream can be expressed as $S_N = s_1,s_2 \ldots s_N$ (*N* is the length of data stream), and each s_i can be expressed as (src_i, dst_i) pairs, find every IP address IP_i whose connection degree is among the TOP-*K* list in data streams S_N within certain measurement period, and $Cdegree(IP_i) \geq \varphi N$.

4 FCM: Algorithm for TOP-*K* heavy hitter identification over sliding windows

In this section, we describe our solution to solve the problem of both TOP-*K* and heavy hitter identification in data streams over sliding windows. We start by describing the Space-Saving algorithm. Then we introduce our proposed algorithm of *FCM*.

4.1 Space Saving

Space-Saving [16][30] is a counter-based algorithm for finding both the frequent items and the TOP-*K* elements which has the highest frequencies in a data stream. In *Space-Saving*, *m* (item, count) elements are stored to maintain necessary information of the data streams which is sorted by their estimated frequencies in the associated *Stream-Summary* data structure. When a newly arrived item exists in the *m-length* monitoring list, its count is incremented. If it does not match a monitored item, the (item, count) pair in the tail of the list has its item replaced with the new item, and the count incremented. Assuming no specific data distribution, Space-Saving uses a number of $1/\varepsilon$ elements to find all frequent items with error ε.

Although Space Saving appears conclusively better than other counter-based algorithms, it still suffers from the deficiency issue. For example, since space saving executes the increment operation every time a new element arrives, the over-estimate-error will be accumulated. Considering the compact stream-summary structure is much smaller than the total size of data streams, heavy hitter which appears ahead may be replaced by host of low frequencies in the end of the data streams.

Below is the pseudo code for Space Saving:

Algorithm 1 for Space Saving:

```
1:  Initialization(T)
2:  function additem( Item x) {
3:      if x ∈ T then
4:              i = Query(x)
5:              c_i := c_i +1
6:              return
7:      endif
8:      if |T|< k then
9:              T:= T ∪ {x}
10:             i:= |T|+1
11:             c_i:=1
12:             return
13:     endif
14:     j:=arg min_{j∈T} c_j
15:     c_j:= c_j +1
16:     T:=T∪ {i}\{j}
17: }
```

4.2 *FCM*

We first present the TOP-*K* Heavy Hitter algorithm —*FCM*—in landmark window over data streams.

To deal with the inaccuracy problem of Space Saving caused by its tail-replacement mechanism, a complementary count-min sketch is added to calculate the connection degree of each host which is provided as the input of the stream-summary structure. So *FCM* adapts two levels of computation: The first level is processed using count-min sketch to filter those flows which do not belong to heavy hitters, and the second is executed by the stream-summary structure to decide whether the host in the arriving flow has been currently in the TOP-*K* list. In this way, the stream-summary structure can be provided with more accurate input.

Since the length of the arriving data streams in definite time interval (for example, 5 minutes) is uncertain, the possibility of hash collision will arise and the sketch may become full with large data streams, resulting in weakened estimation accuracy. Considering that the input data streams have the nature with heavy-tailed distribution, *FCM* introduces an additional small memory to save the hosts with low connection degree and flushes this structure periodically (every *t* seconds). Therefore, *FCM* can greatly decrease hashing collisions of the sketch and enhances the accuracy and efficiency of Space Saving.

As shown in Figure 1, our proposed algorithm —*FCM*— consists of two kinds of data structures. The first part —D^2*_cms*— is made up of two count-min sketches [18] which are very compact data structures to estimate the connection degree of items with high accuracy, and the second part —*topSS*— is the revised Stream-Summary structures of space saving algorithm.

topSS is a list utilized to keep the TOP-*K* elements sorted by their estimated connection degrees. The list is initially empty. It can be denoted as an array with *k* length, and each entry in this array maintains a 2-tuple (*ip*,*c*), where *ip* represents the IP address, and *c* denotes the counter of its connection degree. If a new arrived element in data streams exists in the TOP-*K* list of *topSS*, the corresponding *c* will be directly processed with increment operation. Otherwise, the connection degree of each host in data streams is calculated using D^2*_cms*.

The two count-min sketches used by D^2*_cms* are denoted by *sum_cms*, *filter_cms*, respectively. *sum_cms* is the first count-min sketch for representing those elements with connection degree above certain threshold α. And *d* hash functions (h_1, h_2…h_d) is chosen for membership query. Whenever a new element *e* arrives, *FCM* will probe *d* entries of *sum_cms and* estimate its connection degree *Cdegree(e)* as the minimum value of *sum_cms[j, h_j(e)]* ($1 \le j \le d$). If the estimated value is bigger than α, the *d* associated counters will be incremented by 1. If *Cdegree(e)*+1 is bigger than the minimum *c* value of *topSS,* the element will be removed from *sum_cms* and added to the TOP-*K* summary-structure.

Otherwise it is clear that the element either appears for the first time or its connection degree is below α. *filter_cms* is the second count-min sketch used to periodically filter those hosts with low connection degree. Then the *d* corresponding entries in *filter_cms* is added by 1. If its connection degree is now bigger than α, it will be removed to *sum_cms*. Every *t* seconds, the flush operation of *filter_cms* is triggered.

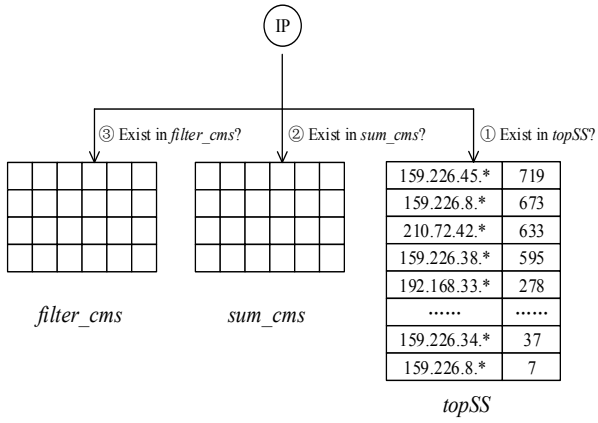

Fig. 1: The data structure of *FCM*

Update procedure

Let $S=s_1\ s_2\ \ldots\ s_N$ be the input data stream that arrives

sequentially, where each element s_i is a flow which can be briefly expressed as (src_i, dst_i). We use a $k \times 2$ bit array, $topSS[i][j]$ ($0 \le i < k$, $0 \le j < 2$), to store the TOP-K heavy hitters. Each item in the list contains 2 elements: The IP address of the host, the corresponding connection degree for this host. In another word, each entry in *topSS* can be expressed as <*ip*, *c*>. And *FCM* keeps a value *minfq* to be the minimum *c* value in *topSS*.

For the incoming $s_i = (src_i, dst_i)$, if src_i matches the xth monitored item in *topSS*, $topSS[x][1]$ will be updated by adding 1. And the connection degree of src_i can be expressed as following:

$$Cdegree(src_i) = topSS[x][1] \quad (1)$$

Otherwise, if src_i exists in *sum_cms*, *FCM* will calculate the $Cdegree(src_i)$ and compare it with *minfq*.

$$sum\_cms[j, h_j(src_i)] \mathrel{+}= 1,\ 1 \le j \le d \quad (2)$$

$$Cdegree(src_i) = \min(sum\_cms[j, h_j(src_i)]) \quad 1 \le j \le d \quad (3)$$

If $Cdegree(src_i)$ is greater than *minfq*, the entry with the minimum *c* value of *topSS* will be replaced with (src_i, $Cdegree(src_i)$).

The last case, if none of the above happens which indicates that src_i represents a host of low connection degree. Then *FCM* will calculate its connection degree in *filter_cms* as below:

$$filter\_cms[j, h_j(src_i)] \mathrel{+}= 1,\ 1 \le j \le d \quad (4)$$

$$Cdegree(src_i) = \min(filter\_cms[j, h_j(src_i)]) \quad 1 \le j \le d \quad (5)$$

If $Cdegree(src_i)$ is above the threshold α, then src_i will be removed from *filter_cms*, and inserted into *sum_cms*.

4.3 *FCM over sliding windows*

In this section, we introduce the extended version of *FCM* with sliding window capability, and the corresponding operations over it. The size of the sliding window in *FCM* is defined in terms of time units as 2s seconds. And each window is split into two equal sub_windows. So every t seconds, the *filter_cms* is flushed just the same as in Section 4.2. And every s seconds, all data in the data structure of the old sub_window are cleared which means the elements monitored in this period have decayed, and a new one is added to the head. Then *FCM* will jump and continue to work on the next 2s seconds. Using the proposed scheme as in Figure 2, *FCM* can work on dynamic datasets efficiently.

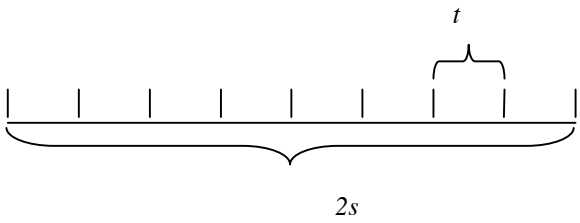

Fig. 2: The *filter-and-jump* mechanism of *FCM*.

So the sliding scheme of *FCM* is implemented using an additional *count-min sketch* and *stream-summary* structure to distinguish between historical flows and current flows. The datasets for the latest s minutes are mapped to *active_topSS* and *active_cms*, whereas *topSS* and *sum_cms* represent datasets for the latest 2s minutes. To remove the decaying elements in time, whenever a new item comes, both *topSS* and *active_topSS* will be updated. Every s seconds, *topSS* is outputted as the TOP-K measurement result. Then *topSS* and *sum_cms* will be flushed and *FCM* will switch to *active_topSS* and *active_cms* for data processing.

Update procedure

When using *FCM* to detect TOP-K heavy hitters over sliding windows, the initial values of all counters are set to zero. And a pointer p is defined to denote the current measurement windows *as* ($(p-1)\times s+1$, $p\times s$) which is equal to (1,s) initially. For each newly arrived element flow s_i denoted by (src_i, dst_i) in the stream, we execute the following several operations in order:

1) First, the three data structures—*topSS*, *sum_cms* and *filter_cms*— is accessed and updated in the same way as described in Section 4.2. Based on it, the element is mapped to *topSS*, *sum_cms* or *filter_cms* if it matches

the corresponding measurement scope.

2) For *active_topSS* and *active_cms*, the operation is just the same as being executed with *topSS* and *active_cms*. If src_i has been included in *active_topSS* with index *x'*, then:

$$active\_topSS[x'][1] += 1 \quad (6)$$

Otherwise, if src_i is a member of *active_cms*, *FCM* will calculate the $Cdegree_{active}(src_i)$ from *active_cms* and compare it with $minfq_{active}$. if $Cdegree_{active}(src_i)$ is greater than $minfq_{active}$, the item with the counter equal to $minfq_{active}$ is removed from *active_topSS* to *active_cms*, and (src_i, $Cdegree_{active}$ (src_i)) is inserted into *active_topSS*.

3) At the end of each *s* minutes in data streams, *FCM* will output the TOP-*K heavy hitters* from *topSS*. Then, the following operation is executed to ensure that *topSS* and *sum_cms* continues to save elements of latest 2s minutes, and the elements in the latest s minutes are stored into *active_topSS* and *active_cms*. And *p* is incremented by 1 to move to the next *s* minutes for measurement.

$$topSS = active\_topSS; \quad (7)$$

$$sum\_cms = active\_cms; \quad (8)$$

$$Initialize(active\_topSS);$$
$$Initialize(active\_cms);Initialize(filter\_cms); \quad (9)$$

5 Complexity Analysis

FCM is proposed to deal with the issue of heavy hitters identification in high speed networks with accurate and scalable solutions, so time complex is considered here. From the description of the *FCM* algorithm, whenever a new element *e* is observed, the update procedure is carried out by at most two kinds of operations: searching in the list of *topSS* and *active_topSS* to check whether *e* has existed in it, and update the corresponding counter if it matches; performing *d* hashing operations to calculate the connection degree of *e* in *filter_cms*, *sum_cms* and *active_cms*, and increment the same *d* cells by adding 1 if the membership is verified.

Since the above procedure can be fulfilled simultaneously, the time complexity of *FCM* can be measured by the operation on single count-min sketch and the stream-summary structure. According to [30], there are two versions of implementation for the Space Saving algorithm: *SSH* (Space Saving with a heap) which requires $O(logk)$ time per update, and *SSL* (Space Saving with linked lists) which takes $O(1)$ time cost. For *FCM,* point queries and updates need *d* hash and *d* update operations, together with a constant look up for certain elements in the stream-summary data structure implemented using linked lists. So the time required to insert an new item in *FCM* is $O(1)$.

6 Performance Evaluation and Experiments

In this section, we present the experimental evaluation of the *FCM* algorithm described in the previous section. To verify the advantages of *FCM*, we choose the most relevant algorithm — Space Saving (*SS*) — for comparison which is prominently applied for both TOP-*K* and heavy hitters identification. We implement the *FCM* algorithm based on the *SSL* source code downloaded from MassDAL [31] and select *SSL* as the benchmark to compare with. The dataset we use to evaluate our proposal is generated using the network traffic of CSTNET (China Science & Technology Network), a small Internet service provider, where 288 NetFlow trace files are collected from a backbone link of 10Gbps on 17th July, 2016. In all the experiments we use the flow five-tuple which includes source and destination IP addresses, source and destination ports, protocol. So the trace file consists of five fields.

We summarize the overall statistics of the data traces as shown in Table 1.

Table 1: Statistics of the data traces

Dataset	Statistics
Date	2016-7-17
Duration	24 hours
Flow Records	65715486
Distinct Flow Records	15068363
Distinct Source IPs	3327335
Distinct Destination IPs	3366161

6.1 Traffic statistics of experimental datasets

For implementing the *filter-and-jump* mechanism of *FCM*, we start by exploring the pattern of Internet traffic with detailed statistics of our experimental datasets. In our experiment, each unique five-tuple flow represents a distinct data-stream element. Figures 3 and 4 show the dynamics of flow records and unique flow records every 5 minutes. Figure 3 illustrates the time-series graphs of the 288 traffic traces, in which the number of total flows and distinct flows is observed to change over time just like the typical model of Internet traffic. Meanwhile the number of distinct flows is about half of total flows, as illustrated in Figure 4. Figure 5 plots the number of distinct source IPs and distinct destination IPs for each trace file which is quite similar with almost equal size.

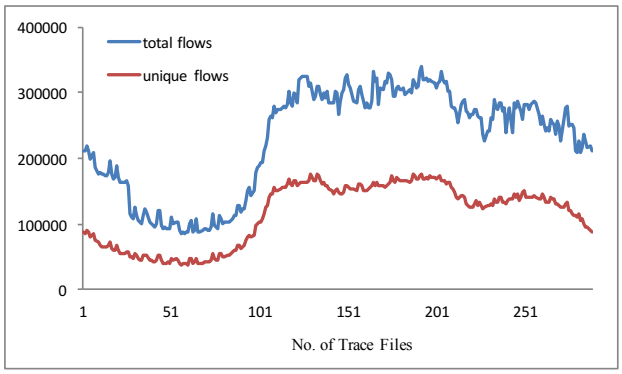

Fig. 3: The number of total flow records and distinct flow records for each trace file.

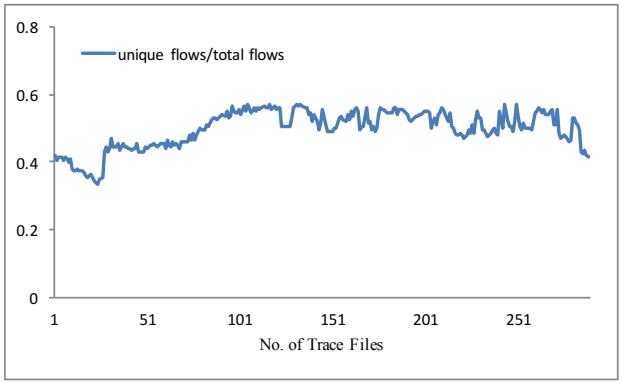

Fig. 4: The radio of distinct flows to total flows for each trace file.

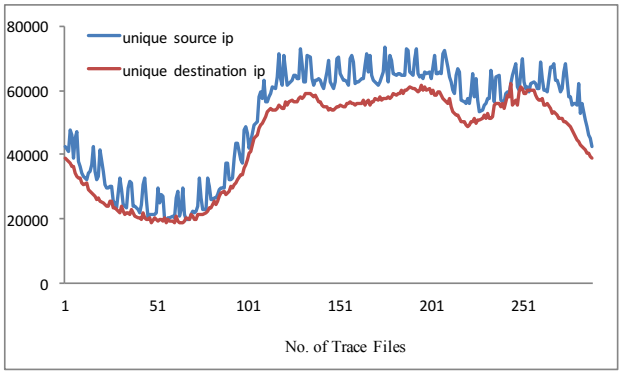

Fig. 5: The number of distinct source IPs and distinct destination IPs for each trace file.

On this basis, we provide further statistics for the hosts with different number of total connections and distinct connections. Figures 6 and 7 plot the cumulative distribution of those two functions which exhibits obvious regularity. In addition, Figure 7 draws the distribution of host's distinct connections which highlights that there is a large number hosts with low frequencies in network traffic traces. For example, the number of hosts with single connection occupies around 82% in 5 minutes. The number of hosts with less than 7 connections occupies around 96%. In turn, it demonstrates the correctness of our design to filter those hosts with low frequencies in time so as to improve the performance of TOP-*K* heavy hitter detection.

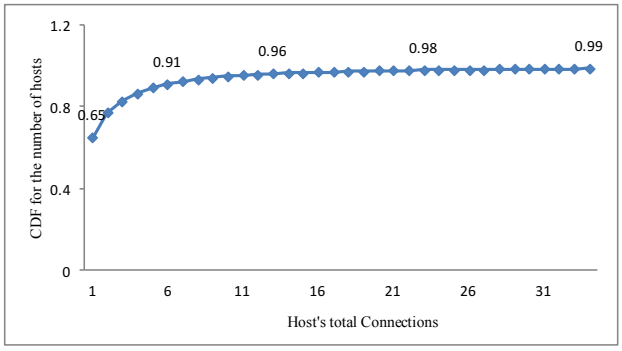

Fig. 6: CDF for the number of total connections of source IP in 5 minutes.

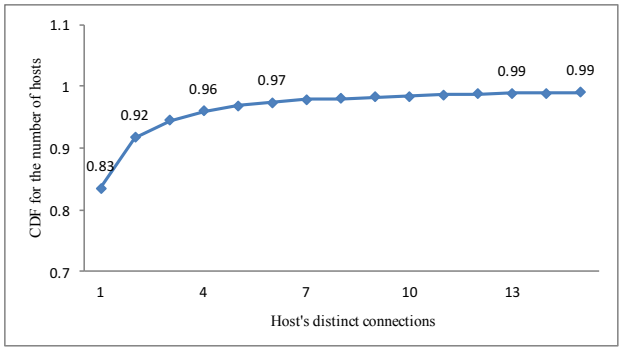

Fig. 7: CDF for the number of distinct connections of source IP in 5 minutes.

6.2 Performance evaluation of *FCM*

To evaluate the performance of the identification algorithms, most of the previous studies mainly focus on the analysis of false positive rate and false negative rate. The false positives is measured by the part of items in the final list identified by the algorithm which actually are not in the TOP_*K* heavy hitters, and the false negatives refer to the part of items in the actual TOP_*K* heavy hitters which could not be identified by the algorithm. However, using only those two metrics cannot give a comprehensive overview about the efficiency of the algorithm. More specifically, previous approaches mainly focus on how to find the correct set while ignoring the accuracy of corresponding distinct values. Therefore, we define another two metrics to evaluate the performance of our algorithm for TOP-*K* identification as following.

1. Ordering deviation rate: For each element *e* in the TOP-*K* list identified by the algorithm, the ordering deviation means the absolute value of the difference between its actual rank number *seq(e)* and its estimated rank number *seq'(e)*. The ordering deviation rate of the identification algorithm is expressed as dividing the sum of ordering deviation by k^2, which is equal to $\sum_{i=0}^{k} | seq(topSS[i][0]) - seq'(topSS[i][0]) | / k^2$.

2. Average estimate error rate: For each element *e* in the final TOP-*K* list identified by the algorithm, suppose its actual connection degree is *Cdegree*(*e*), and estimated connection degree is *Cdegree'*(*e*), then its estimate error rate is equal to |*Cdegree*(*e*) – *Cdegree'*(*e*)|/ *Cdegree*(*e*), and the average estimate error rate of the identification algorithm is expressed as dividing the sum of the estimate error rate by *k*, which is equal to $\sum_{i=0}^{k} | Cdegree(topSS[i][0]) - Cdegree'(topSS[i][0]) | / (Cdegree(topSS[i][0]) \times k)$.

We evaluate the performance of *FCM* based on our datasets and compare it with *SS* algorithm. In all experiments we define the host's connection degree as the number of destinations it connects with as source IP address.

In our first experiment, the measurement epoch is set to be 10 minutes. Considering the various requirements of security analysis in high speed network environment, we set *k* to be 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100, separately. Moreover, from the definition it is obviously that false negative rate is equal to false positive rate, so we neglect its computation below. Figure 8 plots the difference of *FCM* and *SS* in capability to identify the right set of TOP-*K* heavy hitters. The false positive rate of *FCM* gives a tight bound of 2% with slightly instability, while the calculated result of *SS* increases sharply with the growth of *k*. Obviously, *FCM* is significantly superior to *SS* for identifying the TOP_*K* heavy hitters in high speed networks with apparent accuracy and scalability.

Once an anomaly happens, in order to quantify the severity of the situation, how to accurately know the right list of contacts for each TOP_*K* host is quite important. So to demonstrate the advantage of our algorithm to identify TOP-*K* heavy hitters in correct order and with precise value, we evaluate the ordering deviation rate (Figure 9) and average estimate error (Figure 10) of *FCM* algorithm compared with *SS*. The ordering deviation rate of *SS* reaches around 12% when *k*=100, while for *FCM* it is always below 2.8%. The case is almost the same for the measurement of average estimate error rate, when the value of *SS* grows substantially with the increase of *k,* while for *FCM* it keeps almost very stable.

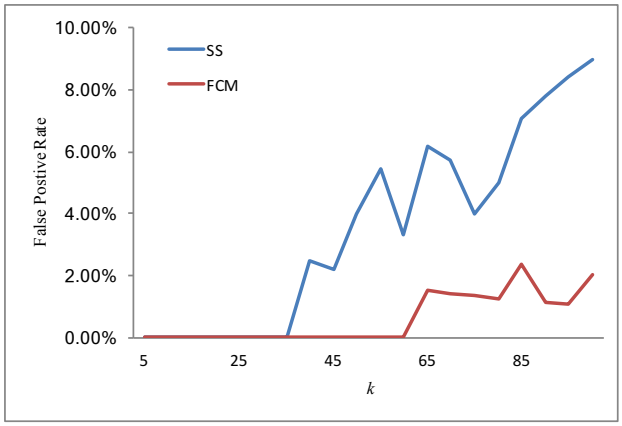

Fig. 8: False Positive Rate of *FCM* and *SS* with different *K*.

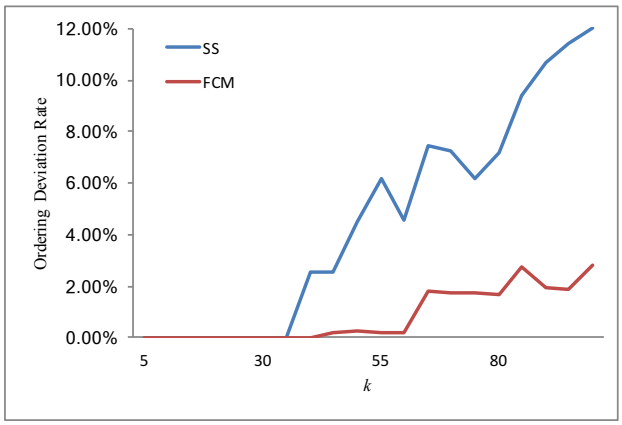

Fig. 9: Order Deviation Rate of *FCM* and *SS* with different *K*.

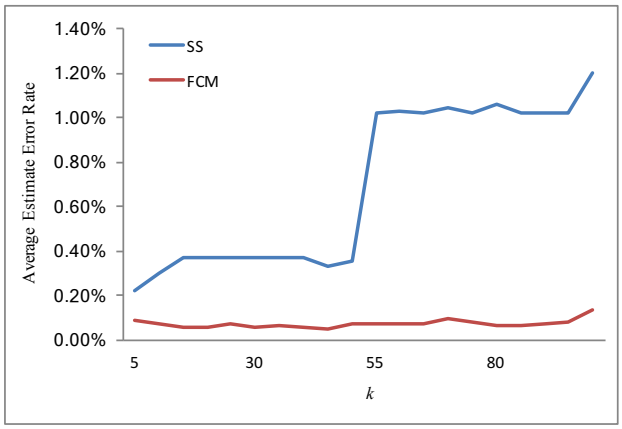

Fig. 10: Average Estimate Error of *FCM* and *SS* with different *K*.

As shown in Figure 2, the number of flow records in certain time interval varies significantly with the change of time every day. In actual environment, the network operator may define the measurement interval to be 1, 5 or 10 minutes according to different requirements. Furthermore, the volume of network traffic may increase dramatically when abnormal event happens, for example, the slammer worm may cause many infected hosts to send up to ten thousand scans a second, which happens only occasionally. Considering the above situations, the scalability of the identifying algorithm is seriously required. To compare the scalability between *FCM* and *SS*, we executed the experiment using various value of *s* which is assigned to 5, 10, 15, 20, 25, till 60 minutes dynamically. Since *FCM* supports sliding window mechanism, it is convenient to carry out this test. For *SS*, we add an additional program to provide the same input datasets as *FCM* with the corresponding time range. For each input, we implement the experiment with k=10, 50 and 100, respectively. Figures 11-13 plot the comparison result. When k=10, both *FCM* and *SS* show good performance. But when the value of *s* or *k* increases, the accuracy of *SS* drops markedly. This is because *SS* needs to predefine the length of its linked lists, which is in inverse proportion to ε. Therefore, when the length of the input file grows, the probability for replacement also increases, and the over-estimate-error especially for the elements in the tail of the linked lists increases too. In contrast, *FCM* is much more stable since the filtering mechanism decreases the possibility of collisions. The first count-min sketch in *FCM* is refreshed regularly, which is executed per minute in our experiment, and is not affected by the variability of *k* or *s*.

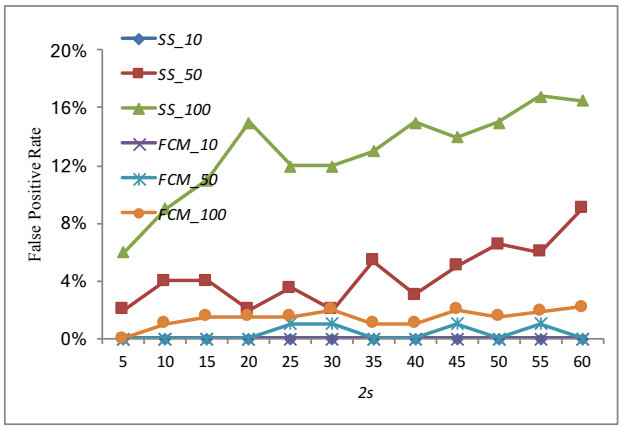

Fig. 11: False Positive Rate of *FCM* and *SS* with different *s*.

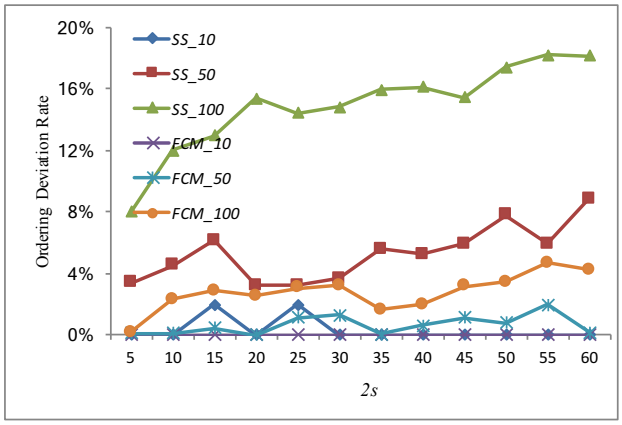

Fig. 12: Ordering Deviation Rate of *FCM* and *SS* with different *s*.

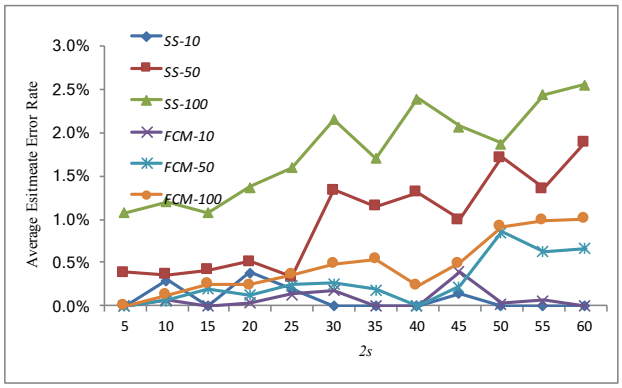

Fig. 13: Average Estimate Error Rate of *FCM* and *SS* with different *s*.

7 Conclusions

Heavy hitters identification in high speed networks is an important yet challenging issue in many security analysis scenarios such as the detection of port scans and DDoS attacks, and tracking worm propagations. In this paper we have addressed this problem based on a new insight, i.e., a large proportion of the total traffic is occupied by hosts with low frequencies and removing those elements in time can further decrease the over-estimate-error caused by hash collision. Based on it, we have developed a novel algorithm called *FCM* to identify TOP-*K* heavy hitters over sliding windows. The algorithm consists of two level of data process. The first level is a multistage filter implemented with count-min sketches — hosts with low connection degree are filtered periodically and only those with connection degree above the threshold are passed to the next level for process. At the second level, a stream-summary structure is implemented and enhanced to calculate the final TOP-*K* heavy hitter list. A jumping scheme has also proposed to support dynamic datasets in a sliding way.

We have compared our proposal with the best counter-based algorithm in terms of the metrics such as the false positive rate and estimate error rate with real data traces collected from an Internet service provider. Extensive experimental results have been conducted to demonstrate the effectiveness of our approach, and the results have shown that this method can identify the TOP-*K* heavy hitters precisely and efficiently. Although our current scheme is mainly carried on the improvement of Space Saving, this method is also suitable for many data processing tasks with heavy-tailed traffic and can be applied to other algorithms to improve the efficiency. In addition, this work sheds new light on exploring network anomalies, and it can be extended to deal with many other practical problems such as detection of heavy distinct hitters.

Acknowledgements

This work was supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010306 and the National Natural Science Foundation of China under Grant No. 61303241. Furthermore, this work is done also with the support of Chinese Academy of Sciences project under Grant No. CXJJ-16M119.

References

[1] Q. Zhao, A. Kumar, and J. Xu, "Joint data streaming and sampling techniques for detection of super sources and destinations", in Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, pp. 7-7, October 2005.

[2] R. Kompella, S. Singh, and G. Varghese, "On scalable attack detection in the network", in Proceedings of the 4th ACM SIGCOMM

conference on Internet Measurement, pp. 187-200, October 2004.

[3] Akamai, "Akamai Q1 2016 State of the Internet Security Report", [Online]. https://content.akamai.com/PG6292-SOTI-Security.html. Accessed October 2016.

[4] S. Shapsough, F. Qatan, R. Aburukba, F. Aloul, and A. Ali, "Smart grid cyber security: Challenges and solutions", in International Conference on Smart Grid and Clean Energy Technologies (ICSGCE), pp. 170-175, October 2015.

[5] Y. Yao, S. Xiong, H. Qi, Y. Liu, L. Tolbert, and Q. Cao, "Efficient Histogram Estimation for Smart Grid Data Processing With the Loglog-Bloom-Filter", IEEE Transactions on Smart Grid, vol. 6, no. 1, pp. 199-208, August 2014.

[6] A. Procopiou, and N. Komninos, "Current and Future Threats Framework in Smart Grid Domain", in IEEE International Conference on Cyber Technology in Automation Control and Intelligence Systems(CYBER), pp. 1852-1857, June 2015.

[7] N. Homem, and J. Carvalho, "Finding top- k elements in data streams", Information Sciences, vol. 180, no. 24, pp. 4958-4974, December 2010.

[8] M. Roesch, "Snort–lightweight intrusion detection for networks", in Proceedings of the USENIX LISA 1999, pp. 229-238, November 1999.

[9] D. Plonka, "FlowScan: A Network Traffic Flow Reporting and Visualization Tool", in Proceedings of USENIX LISA 2000, pp. 305–317, December 2000.

[10] C. Estan, G. Varghese, and M. Fiskin, "Bitmap algorithms for counting active flows on high speed links", in Proceedings of the 3rd ACM SIGCOMM conference on Internet Measurement, pp. 153-166, October 2003.

[11] P. Wang, X. Guan, W. Gong, and D. Towsley, "A new virtual indexing method for measuring host connection degrees", in INFOCOM 2011, pp. 156-160, April 2011.

[12] S. Venkataraman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders", in Proceedings of Network and Distributed System Security Symposium (NDSS), pp. 149-166, January 2005.

[13] N. Bandi, D. Agrawal, and A. El, "Fast Algorithms for Heavy Distinct Hitters using Associative Memories", in International Conference on Distributed Computing Systems, pp. 6-6, June 2007.

[14] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: an efficient algorithm for finding heavy hitters", ACM SIGCOMM Computer Communication Review, vol. 38, no. 1, pp. 5-5, January 2008.

[15] R. Karp, S. Shenker and C. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags", ACM Transactions on Database Systems (TODS), vol. 28, no. 1, pp. 51-55, March 2003.

[16] A. Metwally, D. Agrawal, and A. El, "Efficient computation of frequent and Top-k elements in data streams", in International Conference on Database Theory. Springer Berlin Heidelberg, pp. 398-412, January 2005.

[17] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams", in International Colloquium on Automata, Languages, and Programming. Springer Berlin Heidelberg, pp. 693-703, July 2002.

[18] G. Cormode, "Count-Min Sketch", Encyclopedia of Algorithms. Springer US, vol. 29, no. 1, pp. 1-6, November 2014.

[19] Q. Huang, and P. Lee, "LD-Sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams", in international conference on computer communications, pp. 1420-1428, April 2014.

[20] E. Anceaume, Y. Busnel, and N. Rivetti "Estimating the Frequency of Data Items in Massive Distributed Streams", in IEEE Symposium on Network Cloud Computing and Applications (NCCA), pp. 59-66, June 2015.

[21] P. Roy, A. Khan, and G. Alonso, "Augmented Sketch: Faster and More Accurate Stream Processing", in Proceedings of the 2016

International Conference on Management of Data, pp. 1449-1463, June 2016.

[22] G. Pitel, and G. Fouquier, "Count-Min-Log sketch: Approximately counting with approximate counters", in 1st International Symposium on Web Algorithms, June 2015.

[23] R. Ben-Basat, G. Einziger, R. Friedman, and Y.Kassner, "Heavy hitters in streams and sliding windows", in IEEE INFOCOM 2016, pp. 1-9, April 2016.

[24] P. Roy, J. Teubner, and G. Alonso, "Efficient frequent item counting in multi-core hardware", in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 1451-1459, August 2012.

[25] S. Das, S. Antony, D. Agrawal, and A. El, "Thread Cooperation in Multicore Architectures for Frequency Counting over Multiple Data Streams", Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 217-228, August 2009.

[26] G. Einziger, R. Friedman, "Counting with TinyTable: every bit counts!", in Proceedings of International Conference on Distributed Computing and Networking (ICDCN 2016), Article No. 27, 2016.

[27] N. Homem, and J. Carvalho, "Finding top-k elements in a time-sliding window", Evolving Systems, vol. 2, no. 1, pp. 51-70, March 2011.

[28] Z. Zhang, B. Wang, and J. Lan, "Identifying Elephant Flows in Internet Backbone Traffic with Bloom Filters and LRU", Computer Communications, vol. 61, pp. 70-78, May 2015.

[29] M. Cafaro, M. Pulimeno, I. Epicoco, and G. Aloisio, "Mining frequent items in the time fading model", Information Sciences, vol. 370, pp. 221-238, November 2016.

[30] G. Cormode, and M. Hadjieleftheriou, "Methods for finding frequent items in data streams", The VLDB Journal, vol. 19, no. 1, pp. 3-20, February 2010.

[31] G. Cormode, and M. Hadjieleftheriou, "Finding frequent items in data streams", Proceedings of the VLDB Endowment, vol. 1, no. 2, pp. 1530-1541, August 2008.