Mnemonic Lossy Counting: An Efficient and Accurate Heavy-hitters Identification Algorithm

Rong Qiong, Guangxing Zhang, Gaogang Xie, Kavé Salamatian

Mnemonic Lossy Counting: An Efficient and Accurate Heavy-hitters Identification Algorithm

Qiong Rong, Guangxing Zhang, Gaogang Xie
Network Technology Research Center
Institute of Computing Technology, CAS
P.O.BOX 2704, Beijing, 100190, China
rongqiong, guangxing, xie@ict.ac.cn

Kavé Salamatian
Université de Savoie
LISTIC - Polytech Annecy-Chambéry, BP 80439,74944
Annecy le Vieux Cedex, France
kave.salamatian@univ-savoie.fr

Abstract—**Identifying heavy-hitter traffic flows efficiently and accurately is essential for Internet security, accounting and traffic engineering. However, finding all heavy-hitters might require large memory for storage of flows information that is incompatible with the usage of fast and small memory. Moreover, upcoming 100Gbps transmission rates make this recognition more challenging. How to improve the accuracy of heavy-hitters identification with limited memory space has become a critical issue. This paper presents a scalable algorithm named Mnemonic Lossy Counting (MLC) that improves the accuracy of heavy-hitters identification while having a reasonable time and space complexity. MLC algorithm holds potential candidate heavy-hitters in a historical information table. This table is used to obtain tighter error bounds on the estimated sizes of candidate heavy-hitters. We validate the MLC algorithm using real network traffic traces, and we compared its performance with two state-of-the-art algorithms, namely Lossy Counting (LC) and Probabilistic Lossy Counting (PLC). The results reveal that: 1) with same set of parameters and memory usage, MLC achieves between 31.5% and 6.67% fewer false positives than LC and PLC. 2) MLC and LC have a zero false negative ratio, whereas 38% of the cases PLC has a non-zero false negatives and PLC can miss up to 4.4% of heavy-hitters. 3) MLC has a slightly lower memory cost than LC during the first few windows and its memory usage decreases with time, when PLC memory usage declines sharply. 4) MLC has similar runtime than LC, and smaller time than PLC.**

Keywords-network traffic measurements; heavy-hitters; historical information; mnemonic

I. INTRODUCTION

Accurately measuring and monitoring the traffic is a fundamental requisite for security and traffic engineering in Internet. Many studies on network traffic found that network traffic pattern obeys heavy-tailed distribution, implying a small percentage of flows [3,4,5,9] consuming a large percentage of bandwidth, e.g., in [2], W.Fang et al. shows that 9% of the flows between AS pairs account for 90% of the byte traffic between all AS pairs. We call the flows with a huge amount of packets or bytes heavy-hitters. Many applications such as network resources provisioning and traffic engineering just need the heavy-hitters information and discard the mouse flows. Heavy-hitters identification is also essential in several applications, ranging from denial of service (DoS) attacks identification, load balancing, traffic monitoring, and heavy network users recognition.

Generally, a network traffic flow is identified using the 5-tuples in packet-header fields: source and destination IP addresses, source and destination port numbers, and protocol type. In general, we would like to be able to formulate queries that return in an increasing order flows with more packets or bytes than a given threshold k.

A naïve approach to identify heavy-hitters consists of storing for each flow identifier, a corresponding counter that monitors the number of observed packets of the flow. Although this approach seems straightforward, it is not applicable for high–speed links. In [6], Demaine et al. showed that in general, it is impossible to report all the true heavy-hitters in one pass using less than $O(N)$ memory place, where N is the current length of traffic flows (in number of packets). As N the number of packets in traffic flows increases, the occupied space for storing counters grows linearly. Nowadays, the number of flows is very large (For instance, over a typical OC-48 (2.5Gbps) link [22], The maximum, minimum and average count of concurrent flows in the order hundreds of thousands flows) and the packet inter-interval time on current backbone links is in the order of tenths of nanoseconds. These require a considerable memory space to implement the naïve approach. Furthermore, speeds are increasing and OC-768 (40Gbps) links with packet inter-interval less than 12ns [24] are being deployed. In such environments, fast access to memory that means storage in expensive SRAM, but keeping track of per-flow statistics needs large space, only fitting into DRAM. It is noteworthy than the DRAM speed cannot keep up with the link rate of OC-768.

This motivates the quest for heavy-hitters identification algorithms that can approximately determine all heavy-hitters with share exceeding a user-specified fraction s of the current traffic and use as less as possible fast memory. In this context, approximation may mean a number of things. This has motivated several papers in literature [6-8,10-13,15-16,18-21,23] that have tried to make a better usage of memory and processing power at the cost of approximate heavy-hitters recognition. The approximation happening through the introduction of false positives, the exclusion of false negatives and approximate counts of the heavy-hitters counters.

In this context, Lossy Counting [7] (LC) algorithm is one of the most efficient and well-known algorithms for identifying heavy-hitters. Using a limited space, LC guarantees that all true heavy-hitters consuming more than a give share are returned. However, LC overestimates the sizes of flows and can return some false positives. E.g., if a user-specified threshold for heavy-hitters is $s = 0.05\%$, the false positive ratio of LC can attain values as high as 58-73%.

Based on LC algorithm, Dimitropoulos and Hurley [23] proposed the Probabilistic Lossy Counting (PLC) algorithm. This probabilistic algorithm was based on the assumption that flow sizes follow a heavy-tailed distribution. This assumption is used in order to obtain tighter error bound on the estimated sizes of traffic flow. This modification reduces drastically the required memory and improves the accuracy of heavy-hitters identification. However, PLC algorithm needs to emulate heavy-tailed distribution at the end of each window, and this causes the computational complexity of PLC to be high. Therefore, although PLC algorithm reduces memory space, the computation complexity affects its usability in realistic contexts.

In this paper our goal is to present an efficient algorithm to identify heavy-hitters using fixed or bounded memory resources with high accuracy (very low false positives and no false negatives) and low complexity (space and time). To achieve this we need an algorithm that makes only one pass over the traffic stream. We present here a scalable algorithm based on LC, which is named Mnemonic Lossy Counting (MLC) that address the above described goal. MLC holds most likely candidate heavy-hitters flows in a fixed-size historical information table and use these stored flows to assigns tighter error bounds on the estimated sizes of heavy-hitters. These bounds are based on smoothing the weight over historical behavior of the flows that are much less complex that competitor's approach that has to emulate heavy-tailed distribution.

To validate our proposed algorithm, we have used real network traffic traces coming from an academic link in China and traces containing commercial traffic data from Japan made available by the WIDE project [25]. Over these traces we used five metrics to assess MLC performance: 1) false positive ratio 2) false negative ratio 3) detection ratio 4) memory cost and 5) computational complexity. We compare MLC algorithm with LC and PLC, and find that MLC has faster and higher identification accuracy.

The remainder of this paper is organized as follows: Section II formally reviews related previous works, Section III introduces MLC algorithm and presents complexity analysis. We also guide the selection of parameters in this section. Section IV describes experiment environment. We evaluate the performance of MLC and compare that to LC and PLC in section V. Section VI discusses identification on-line. Finally, we conclude the paper and describe future direction in Section VII.

II. Related Work

Heavy-hitters identification has received considerable attention during the past decade [6-8,10-13,15-16,18-21,23]. Heavy-hitters identifying algorithms can be roughly divided into three groups: sampling-based, hash-based and counter-based algorithms. Sampling-based algorithms [12,13,15,18,19] exploit cyclical sampling to reduce memory footprint and processing overhead, but their accuracy is limited by low sampling rate required to make the sampling operation affordable. Hash-based algorithms [8,10,16,21] can substantially reduce storage space for flow recording and accelerate processing speed. However, they need to find a balance between compression ratio and accuracy. Moreover, hash functions need to be chosen carefully in order to avoid collisions.

Counter-based algorithms [6,7,11,23] hold a fixed (or bounded) number of counters for tracking the size of heavy-hitters. In [6], Demaine et al. presents counting-only heavy-hitters identification algorithms, which use m counters and deterministically identifies all flows having a relative frequency above $1/(m+1)$. Nonetheless, they may return false positives and require a re-scan of the data stream to determine the exact set of heavy-hitters. Manku and Motwani [7] introduce another counting-based algorithm for identifying heavy-hitters, Lossy Counting. LC is a deterministic algorithm that computes frequency counts over a stream of single item transactions, using at most $log(\varepsilon N)/\varepsilon$ space, where N denotes the current length of the traffic stream in packets and ε is an error parameter such that $0 < \varepsilon \ll 1$. In LC, the incoming stream is divided into fixed-size window. The algorithm processes each window sequentially, and maintains a counter for each distinct element. It periodically deletes counters whose estimated frequencies and error bounds fall below a fixed threshold. The error bound are here to ensure that heavy-hitters are not missed by repeatedly deleting and re-starting counter. Querying heavy-hitters consists of selecting elements whose estimated frequencies and error bounds exceed a user-specified threshold. LC algorithm carries out simply, but because of the maximum error bound, the estimated sizes of heavy-hitters is less accurate and false positive ratio increases to 58-73%.

Based on LC algorithm, Dimitropoulos and Hurley [23] proposed PLC, which use a tighter error bound on the estimation sizes of traffic flows. PLC changes the error bound on the estimated size of an arbitrary traffic flow, relaxing the deterministic guarantees from LC. In PLC, network traffic pattern is considered to obey heavy-tailed distribution. This helps in tightening error bounds and making them less conservative in removing small flows. Given that flows of small size account for the majority of network traffic flows, PLC reduces the required memory for computing heavy-hitters. In addition, PLC generates fewer false positives than LC. Nonetheless, at the end of each window, PLC has to use existing traffic flows in table to emulate heavy-tailed distribution to calculate new error bounds. This last step needs more than $O(1)$. As a result, the computational complexity of PLC is not as good as that of LC.

III. Mnemonic Lossy Counting Algorithm

A. Motivation

An element is deleted in LC if the estimated frequency and the error bound falls below a user-specified threshold. Potential error on the frequency of an element is occasioned by a possible prior removal of an element from the table. In

LC, the error bound Δ of an item is set equal to $\Delta = index - 1$ (where $index$ is the id of the first window where the item has been seen) and increases linearly with the number of windows. This choice ensures that elements with a large error bound remain in the table longer. According to Little's law [1], when elements stay for a longer time in the table, the average size of the table increases. In order to reduce the memory cost, PLC makes the error bound substantially smaller than the deterministic error bound of LC by using heavy-tailed distribution, providing probabilistic guarantees. As a result, PLC improves the memory cost, whereas has higher computational complexity. In addition, the heavy-tailed distribution variable is strongly unstable, possibly leading to false negatives.

In this paper we are introducing a new approach named MLC (Mnemonic Lossy Counting), which improves LC. It achieves this by holding historical information about most likely candidate heavy-hitters. These information are used to compute appropriate error bounds on the estimated size of flows, which effectively reduces false positives and false negatives.

B. *Notations and Definitions*

In this section, we describe our notation, and define the metrics for assessing MLC performance.

MLC algorithm accepts four user-specified parameters: a support threshold $s \in (0,1)$, an error parameter $\varepsilon \in (0,1)$, chosen such that $\varepsilon \ll s$, a smoothing constant $q \in [0,1)$, and a size of the historical information table H_{size}. Let f_e denotes the true frequency of a flow, and Δ denotes the error bound on the estimated size of the flow, and N denotes the current length of the traffic flows, *i.e.*, the number of packets seen so far. At any given time, the answers returned by MLC include two components:

- all flows whose true frequencies f_e exceed sN
- flows with true frequencies validating the bound $sN - \Delta \le f_e \le sN$

We use the following five metrics to evaluate the performance of our algorithm: false positive and false negative ratios, detection ratio, memory cost, and computational complexity.

- False positive ratio is defined as the ratio of flows with true frequency smaller than sN that are returned by MLC over the total number of returned heavy-hitters.
- False negative ratio is defined as the ratio of flows with true frequency larger than sN that are not returned by MLC over the total number of returned heavy-hitters.
- Detection ratio is the number of true heavy-hitters returned by MLC over the total number of true heavy-hitters.
- Memory cost is measured as the number of table entries that need to be maintained.
- Computational complexity is derived as the runtime of the algorithm in the same environment.

C. *MLC Description*

MLC similar to LC splits the incoming stream into windows of $w = \lceil 1/\varepsilon \rceil$ elements each. Each Window is indexed with an integer $index (1 \le index \le \lceil N/w \rceil)$, where N denotes the length of the traffic flows (in number of packets). Two data structures are maintained: the samples table D and the historical information table H. Note that D is a set of entries of the form (e, f, Δ), where e is an element identifier, f is an estimated frequency, and Δ is an error bound. While H has a form $(e, index, \hat{c})$, where $\hat{c} = f + \Delta$, the sum of an estimated frequency f and its error bound Δ.

Algorithm: Initially, $D = H = \emptyset$. MLC processes windows sequentially. When a new element e arrives in the windows $index$, we first check whether it exists in D. If the element is already in D, the estimated frequency f of this element is incremented. Otherwise, we lookup H to see whether en entry for e exists or not. If H lookup succeeds, we delete the entry $(e, old\_index, \hat{c})$ from H and an entry $(e, 1, q^{index - old\_index}\hat{c})$ is inserted into D. If the element does not exist in both tables, we insert a new entry $(e, 1, \Delta)$ into D, where Δ has been calculated at the end of last window (to be seen further). Once an element is inserted into D, its Δ value remains unchanged. At the end of each window, MLC first garbage-collects D by deleting entries with $f + \Delta \le index$, and updates H by inserting elements with $f > 1$ into H. Secondly, MLC queries whether H is full or not by checking if it has became larger than H_{size}. If H is full, MLC stores most recently candidate heavy-hitters by deleting entries with small values of $q^{index - old\_index}\hat{c}$. Right after, MLC computes a new error bound $\Delta = max(q^{index - old\_index}\hat{c})$. Finally MLC returns all elements with $f + \Delta \ge sN$. We summarize MLC in Algorithm.

Algorithm: Pseudo-code of Mnemonic Lossy Counting

```
w = ⌈1/ε⌋ ;
D = H = { };
j = 0, index = 1, Δ = 0;
for all e do
    if e has an entry in D then
        D←(e, f + 1, Δ);
    else if e has an entry in H then
        D←(e, 1, $q^{index-old\_index}\hat{c}$);
        delete (e, old_index, ĉ) from H;
    else
        D←(e,1, Δ);
    end
    j++;
    if j % w == 0 then
        for all (e, f, Δ) in D do
            if f + Δ ≤ index then
                if  f > 1 then H←(e, index, f + Δ);
                delete (e, f, Δ) from D;
            end
        end for
        if H is full then
            delete entries with small values of $q^{index-old\_index}\hat{c}$;
            Δ = max($q^{index-old\_index}\hat{c}$);
        end
        index++;
    end
end for
```

Discussion: From Algorithm, it is clear that the MLC introduces a trade-off between storage space and accuracy as deleting an entry relative to item e from table H would cause an error on the estimated frequency, while removing a similar item from table D only reduces the accuracy of its estimated frequency. This trade-off is achieved by controlling the amount of historical information, which reflects network traffic dynamics [14, 20].

Lemma 1 *If the current window id is $i_{current}$, the error bound Δ of a new flow satisfies $\Delta \leq i_{current}$.*

Proof: At the end of last window, MLC has queried whether H was full or not: if H was full, MLC assigned a new error bound $\Delta = max(q^{i_{current}-1-old\_index}\hat{c})$, where old_index denotes the window id when a flow was removed from D, and $i_{current}$ denotes the current window id. Since flows that validated $\hat{c} = f + \Delta \leq old\_index$ were removed from D, we can be sure that in the current window we have:

$$\Delta = max(q^{i_{current}-1-old\_index}(f+\Delta)), \text{ and}$$
$$\Delta \leq max(q^{i_{current}-1-old\_index}old\_index)$$

Considering that $q \in [0,1)$, it results that:

$$\Delta \leq max(q * old\_index) \leq old\_index \leq i_{current}$$

Lemma 2 *the error bound of MLC becomes $\Delta \leq \varepsilon N$.*

Proof: MLC splits the traffic stream into windows of $w = \lceil 1/\varepsilon \rceil$ elements each. Each window is indexed with an integer $index(1 \leq index \leq \lceil N/w \rceil)$. Thus, $index \leq \lceil \varepsilon N \rceil$. From Lemma1, the error bound of a new flow validates $\Delta \leq i_{current}$. By combining the two last inequalities, the error bound of MLC becomes $\Delta \leq \varepsilon N$.

Theorem 1 *For each flow e in D, we have $f \leq f_e \leq f + \varepsilon N$.*

Proof: Flows stored in D belong to two components: 1) flows never removed from the samples table D, and 2) flows removed from D before, and re-inserted into D. If the flow e has never been removed from D, obviously the estimated frequency f is equal to the true frequency f_e, and we have obviously $f \leq f_e \leq f + \Delta$. Now, if the flow e was removed from D before, and was re-inserted, the situation becomes more complicated. When e was first removed from D, the estimated frequency f was equal to the true frequency f_e and $f + \Delta \leq old\_index$ (as it was removed), where old_index denotes the id of the window where the flow e was removed. When e was reinserted into D, obviously $f_e \geq f$. However MLC compensates the error bound Δ on the estimated frequency f, by considering the weighted historical information. This ensures that $f_e \leq f + \Delta$. So in all two cases, we have $f \leq f_e \leq f + \Delta$. Moreover, from lemma 2, the error was bounded by $\Delta \leq \varepsilon N$. We therefore conclude that $f \leq f_e \leq f + \varepsilon N$ for each flow e in D.

Heavy-hitters were defined as flows that have a size in byte or packet that exceeds a threshold s during some time period. When MLC is given a heavy-hitter query with a support threshold s, it returns flows in D where $f \geq sN - \Delta$. The answers therefore include: heavy-hitters with true frequencies f_e exceed sN and some false positives with true frequencies f_e between $sN - \Delta$ and sN.

D. Complexity analysis of MLC algorithm

Theorem 2 *MLC and Lossy Counting can share the same memory bounds by limiting the size of the historical information table H_{size}.*

Proof: MLC algorithm introduces the historical information table by holding the most likely candidate heavy-hitters. The candidate heavy-hitters are the flows with $f + \Delta \leq index$ and $f > 1$. We limit the size of the table H by H_{size}, *i.e.*, the maximum number of candidate heavy-hitters at the end of each window. It is expected that a small percentage of network traffic flows will account for a large percentage of traffic. In practice, we observed that in each window more than 90% of flows validated $f + \Delta \leq index$ were removed from D. In these flows no more than 50% of them had $f > 1$. Thus, the number of entry moved to the historical information table H at the end of a window is at most

$$1/\varepsilon * 90\% * 50\% \leq 1/2\varepsilon \quad (1)$$

Furthermore, following the same path as theorem 4.2 in [7], let's assumes that $|D| = \sum_{i=1}^{B} d_i$, where B is total number of windows, and d_i denote the number of entries remaining in D after the end of window i. Since $\sum_{i=1}^{j} d_i \leq \sum_{i=1}^{j}(w/i)$ and the maximum window id is εN, we get $|D| \leq \sum_{i=1}^{B}(w/i) \leq logB/\varepsilon \leq log(\varepsilon N)/\varepsilon$.

Therefore, Lossy Counting computes a ε-deficient synopsis using at most $log(\varepsilon N)/\varepsilon$ entries. Now the total memory cost of MLC is

$$|D| + |H| \leq \sum_{i=1}^{B} d_i + 1/2\varepsilon \leq \sum_{i=1}^{B}(w/\varepsilon) \quad (2)$$

By combining the two bounds (1) and (2) we get:

$$|D| + |H| \leq \frac{1}{\varepsilon}(\sum_{i=1}^{B} 1/i + 1/2) \leq logB/\varepsilon \leq log(\varepsilon N)/\varepsilon (3)$$

Proving that, MLC is using at most $log(\varepsilon N)/\varepsilon$ entries with limiting the number of entries in H to H_{size}.

Theorem 3 *The time complexity of MLC to update the error bound estimate is $O(m)$, where m denotes the length of the historical information table.*

Proof: At the end of each window, MLC queries whether the historical information table H is full or not. If H is full, MLC traverses H to delete relatively small entries, and assigns a new error bound $\Delta = max(q^{index-old\_index}\hat{c})$. For traversing H we need no more than m comparisons, where $m = |H|$. So the time complexity of updating the error bound of MLC is $O(m)$.

E. Parameters Discussion

Rough suggestion about how to choose the support threshold s and an error parameter ε are given in [23]. Here we describe how to choose the smoothing constant q, and the size of the historical information table H_{size}.

The smoothing constant q determines the weight of past flow behavior on the error bound value. Lower q gives less weight to past flow measurements and increases the importance of current behavior. Longer window results in more packets processed in each window, and the lower correlation between consecutive windows. Therefore the smoothing constant q must be selected smaller. Conversely, shorter windows size w needs larger smoothing constant q. Nonetheless the size of the window is related to the error parameter ε through the relation $w = \lceil 1/\varepsilon \rceil$. This means that the choice of window length w or equivalently the error parameter ε affects the selection of a smoothing constant q. We relate the average number of past windows N/w to the value of q according to the following equation.

$$q = \frac{N/w - 1}{N/w + 1} \tag{4}$$

There are two particular cases:

- If $w = N$, then $q = 0$, meaning that past information does not have an effect when a traffic flows is not divided.
- If $w \to 0$, then $q \to 1$, meaning that the algorithm splits traffic flows into arbitrary small fixed-size windows $0 \leq w \leq N$.

Therefore, we select q in $[0, 1)$.

To investigate the impact of parameter H_{size}, we present an empirical analysis. Obviously, the larger H_{size}, the more candidate heavy-hitters are held. However, when H is too large, it will hold candidate heavy-hitters, but also a large number of small flows. We can use two approaches to select H_{size}: an theoretical approach based on the upper bounds developed in this paper, and an empirical approach based on training data.

The first approach consists of using the available memory resource and the bounds obtained in (1) and (3) to assign D_{size} and H_{size}. In theorem 2 we saw that $H_{size} \leq 1/2\varepsilon$ and that $|D| + |H| \leq log(\varepsilon N)/\varepsilon$. Now let M denotes the available memory resource. If one sets $M = log(\varepsilon N)/\varepsilon$, he can observe that as N increases, he will have $M/2 = log(\varepsilon N)/\varepsilon \geq 1/2\varepsilon$. So we can assign half of the available memory resource to H_{size}, and the remaining to D_{size}. This approach is simple and provides strong guarantees on memory cost. However, memory resource cannot be fully utilized and we have an important memory waste.

Another approach consists of evaluating the memory cost of the algorithm in the target environment using training data. By submitting MLC to different flow traces, one can measure the number of flows moving from D to H for different values of w and use these values to set H_{size}. The limitation of this method is that it relies on training data for target environment.

The choice of one method from the two above depends on applicative requirements and availability of training data. The first approach does not need any prior data and can provide strong guarantees on memory cost, while the second one has a better memory usage but needs training data.

IV. Experiment Description

In this section, we will evaluate the performance of MLC on real network traces, and compare that to LC and PLC.

A. *Experiment Setup*

We collected in 2009 more than two hundred traces from the gateway of a campus network with 1500 users in China. We also used 15 traces from capture point-F (a trans-pacific link) of MAWI Repository [25]. We have analyzed all these traces and we found results typically similar. In this paper, because of space restriction we will only validate the results over two representative traces described in table I.

For each trace, we have run a brute-force algorithm to calculate the true heavy-hitters.

TABLE I. Statistics Of Traffic Traces

Dataset	***Trace I***	***Trace II***
Source	MAWI trace	Campus network
Date	2008-12-16 14:00-14:15	2009-08-24 16:20-16:35
Packets	23,602,516	49,999,860
Unique flows	1,534,211	4,136,226

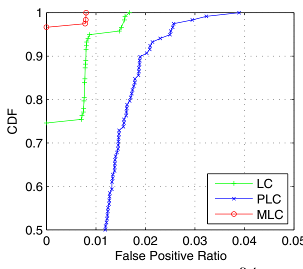
We also set the error at $\varepsilon = 0.001\%$ and experiment with three values of support threshold: $s = 1\%$, $s = 0.1\%$, and $s = 0.05\%$. The error value gives us a window length w of 100,000.

B. *Metrics*

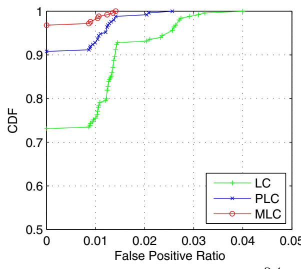
In section III, we describe our definition of five metrics. Here, we will exploit these metrics to evaluate MLC algorithm performance, and compare that to LC and PLC. It is noteworthy that the memory cost for LC and PLC algorithms is only the samples table D size, whereas for MLC it is the sum of samples table D size and the historical information table H size.

V. Validation

A. *False Positive Ratio*



(a) Trace I, $s = 0.1\%$



(b) Trace II, $s = 0.05\%$

Figure 1. CDF of FPR of MLC, LC, and PLC for the support $s = 0.05\%$ that results in the worst performance among all three algorithms in all the experiments.

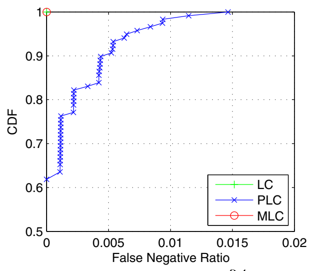
Among the different experiments performed using the two traces with $s \in \{1\%, 0.1\%, 0.05\%\}$, we are showing in Fig.1 the one where the three algorithms had the worst false positive ratio (FPR), namely $s = 0.05\%$. More precisely we run the three algorithms over 499 windows and calculated over two windows the FPR of heavy hitters.

False positives are returned heavy-hitters with true frequency between $sN - \Delta$ and sN. The difference in FPR of the three algorithms occurs because the three algorithms use a different error bound Δ.

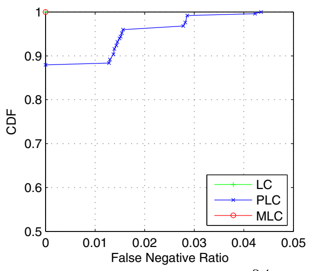
Figure 1 plots the empirical Cumulative Distribution Function (CDF) of the FPR of heavy-hitters larger than 0.05%. For the two traces, we observe that MLC exhibits a lower FPR than LC and PLC. For example in trace II, 96% of windows had a FPR equal to 0, where LC (resp. PLC) had no more than 73% (resp. 90%) of such windows.

The difference in performance between Fig. 1.a and 1.b is coming from the fact that distribution of traffic flow sizes in trace I is substantially different from a heavy-tailed one. This means that the estimate of error bound made by PLC that is based on a heavy-tailed assumption is wrong. This explains why PLC CDF is far from the two others. However, even in this case MLC have a very low FPR because of more appropriate error bounds coming from past history.

B. *False Negative Ratio*



(a) Trace I, $s = 0.1\%$



(b) Trace II, $s = 0.05\%$

Figure 2. CDF of FNR of MLC, LC, and PLC.

In this section we do a similar analysis to previous section, but on False Negative Ratio (FNR). We show in Fig.2 the FNR obtained with same values of support threshold s than in Fig.1. As expected LC and MLC have a zero false negative ratio (FNR), whereas PLC have an FNR that is coming from the probabilistic nature of PLC. We observed that in the worst case, 38% of windows had a non-zero FNR, and PLC can miss up to 4.4% of the heavy-hitters, as seen in Fig.2.

C. *Detection Ratio*

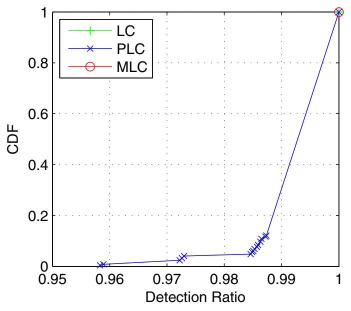


Figure 3. CDF of Detection Ratio of MLC, LC, and PLC.

Figure 3 compares the CDF of detection ratio of MLC, LC and PLC algorithms. Generally speaking, all the algorithms are able to track the true heavy-hitters with high detection ratio. Since LC and MLC algorithms guarantee to return all εN heavy-hitters, we expect 100% detection ratio, which is observed in the figure. PLC algorithm only provides probabilistic guarantees to return all εN heavy-hitters, and possibly returns false negatives. As expected PLC yields lower than 100% detection ratio.

D. *Memory Cost*

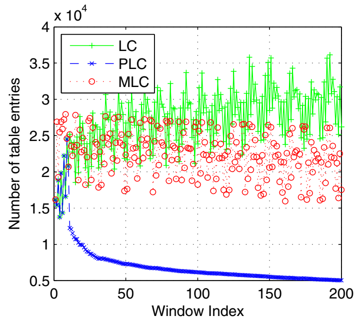


Figure 4. Memory cost comparison of MLC, LC and PLC. The plots depict numbers of table entries versus window index for both traces.

Figure 4 exhibits memory cost of the three algorithms in terms of the number of entries that need to be stored. Recall that each window size in our experiments is 100,000 packets. Actually, 90% (resp. 98%) of the flows in trace I (resp. trace II) are deleted from the table D at the end of each window. In Fig.4, we note that memory cost of LC increases with time (as expected because Δ increases with time) and the maximum number of table entries in table is 36,000. Moreover, PLC and MLC achieve similar maximum memory cost, resp. 25,000 and 27,000 entries. Nevertheless, PLC reaches its maximum memory cost during the first few windows and then the number of table entries decreases steadily and fast. MLC also shows a decline in memory cost but with a slow decline. This is because PLC reduces drastically the required memory for computing heavy-hitters, where MLC have to store most likely candidate heavy-hitters in the historical information table, and this cost memory. However, MLC have almost the same maximal memory usage that can be controlled by limiting H_{size}. Moreover, MLC achieves a very low FPR and a zero FNR where PLC had poor performance in this respect. By running MLC we can even achieve better memory performance as the memory usage will continue to decrease with time.

E. *Computation Complexity*

TABLE II. RUNTIME OF MLC, LC AND PLC PER PACKETS

Algorithm	Runtime (µsec)	
	$s = 0.05\%$	$s = 0.1\%$
Lossy Counting	6.57	6.03
Mnemonic Lossy Counting	7.83	7.82
Probabilistic Lossy Counting	19.11	19.07

Table II compares runtime of MLC, LC, and PLC per packets over trace I (with around 23.6 Millions packets) over a computer with an Intel Core2 Duo E8400 processor (3.00GHz) with 2GB of RAM. Trace I was captured over a link with capacity of 150 Mbps and a throughput of 130 Mbps. Over this trace the mean arrival time of packets is 17.8 µsec.

It can be observed that MLC has similar runtime with LC, and much shorter runtime than PLC. This results from the fact that PLC has to estimate Δ using heavy-tailed distribution. At the end of each window, PLC has to emulate heavy-tailed distribution with a computational complexity of about $O(n^2)$ where n denotes the current number of entries of table D. For MLC theorem 3 proved that we need $O(m)$ update times to estimate the error bound, where m was the length of the historical information table in entries. When n goes to infinity, the computational complexity of PLC becomes in the order of $O(1)$. Finally for LC the computational complexity is about $O(1)$, as it estimates the error bound as $\Delta = index - 1$.

In summary, our evaluation shows that MLC has greatly improved over LC and PLC algorithm in term of FPR. LC and MLC algorithms have 100% detection ratio, whereas PLC algorithm has lower detection ratio because of FNR. Nonetheless, PLC has a smaller memory cost compared with LC and MLC, but the three algorithms have similar maximum memory utilization. The computational complexity of MLC and LC are significantly better than PLC.

VI. Discussion Of Identification On-line

In section IV, we said that we have collected hundreds of traces from network links with commercial and academic traffic, and used them off-line treatment to assess the performance of the MLC algorithm. However, the main application of the algorithm is for online identification. We will discuss here some issues with online identification.

First it is noteworthy from table II that even our non-optimized code make MLC and LC suitable for an online usage over the link used for trace I as we need a processing time in the order 6 to 8 μsec where the arrival delay of packets was 17.8 μsec; on the same link PLC will not be usable online as its per packet processing time is higher than the arrival delay.

Indeed, as the bandwidth of backbone links increases (almost doubling yearly according to [9]) the performances given in table II will not be enough and we have to face with harder memory and real-time constraints. As proven in Theorem 2, the space requirement of MLC grows logarithmically with N, N being the current length of traffic in packets. However, this theorem bounds the worst-case that corresponds to a rather pathological traffic that would almost never occur in practice. As shown in Fig.4, the maximal memory cost of MLC increases sharply during the first few windows and remains almost stable after that. In practice, the maximal memory cost of MLC remains far from the worst-case bounds and the maximum memory cost does not increase with N. We moreover showed in this paper that the total memory cost of MLC (being the sum of the samples table D and the historical information table H) is closely related to user-specified parameters, namely error parameter ε, and size of the historical information table H_{size}.

Another important factor affecting online identification is the network packet processing efficiency. For MLC, this efficiency relies on the update operation in D and H.

In summary, the two main questions to solve in order to use MLC for online heavy-hitters identification in practice are:

- How to select ε and H_{size} in order to optimize the memory usage?
- How to speed up updating operation in D and H for MLC, while we have already fixed D_{size} andH_{size}?

The first issue was discussed in section III.E where we presented an empirical approach to guide the selection of H_{size}, which leads to the choice of ε.

For addressing the second issue we have first to evaluate the complexity of updating operation of D and H. Let m denotes the length of H. By reviewing MLC, we observed that upon reception of a new element e, D and H are checked, to update the estimated frequency of the flow matching e. Since the checking operation needs to traverse D and H, at most $\lceil 1/\varepsilon + m \rceil$ comparisons are needed. At the end of each window, MLC algorithm garbage-collects D to remove entries with $f + \Delta \leq index$, where entries with $f > 1$ are inserted into H. The pruning operation need at most m comparisons. Thereafter, MLC checks whether H is full or not. If H is full, based on the number of new candidate heavy-hitters, MLC deletes entries with relative small values of $q^{index-old\_index}\hat{c}$. This step needs at most $2m$ comparisons. In summary, the main operations over D are inserting and deleting operations, whereas for H these are inserting, comparing and deleting.

Two class of storage can be used to implement the data structure of D: first solution that is hardware based consists of using content addressable memory (CAM) in order to distinguish the flow object by accessing the storage only one time. A second solution that is software based only uses a circular linked list and SRAM with fast reading speed; D can be implement using a hash table adopting the chain address method to resolve hash collision. Both of these means reduce the time complexity of the inserting operation to $O(1)$. However, the deleting still has to traverse D. In order to reduce the time complexity of the deleting operation, we can modify inserting, by adding an extra 1 bit besides the inserting or updating counter to indicate whether the flow will have to be deleted. Thus, the time complexity of deleting operation will fall to $O(1)$.

For the table H the most complex operation is also the deleting, but in situation when H is completely full, and needs $2m$ comparisons to select entries that should be deleted. The historical table H can be implemented using both the all sequencing and partly-sequencing, *i.e.*, a partly sequencing bi-directional linked list or binary sort tree. This last implementation leads to an equal balancing between the branches of the tree, and reduces the time complexity of deleting operation to $O(1)$, but at the cost of an increasing time of the inserting operation of H. Generally, the inserting operation over H is applied in batch. In the worst cases, this operation can cost m comparisons to find the inserting location. However by using a sort list binary tree for the data structure of H, the comparison time in the worst cases reduces to $m/2$. Therefore by using a sort list binary tree as the historical information table's data structure, and since $m \ll 1/\varepsilon$, the processing time of each flow in MLC can be reduced to $O(1)$ by applying the equal balancing idea presented before.

VII. Conclusion

In this paper, we introduced and motivated the need for a fast and low memory usage heavy-hitters identification algorithm. We explained that LC algorithm has more false positives, and therefore it is not suitable for accurate identification in high-speed network. We presented a new algorithm based on the LC algorithm, the Mnemonic Lossy Counting that efficiently identifies heavy-hitters in high-speed network. Our scheme introduces historical information for reinforcing and pre-protecting "mnemon", *i.e.*, candidate heavy-hitters. We derived the error bound of MLC and showed that it can be appropriately tailored on the estimated size of traffic flows.

We also provided an evaluation of the proposed algorithm over real network traces, and compared its cost in terms of memory, CPU usage, and accuracy with the state-of-the-art Probabilistic Lossy Counting (PLC). MLC shows comparable accuracy with significant lower computational complexity, whereas PLC shows lower memory cost. Both MLC and PLC have their own strengths and weakness, which can be selected according to applicative requirements.

In summary, MLC exhibits excellent performance for false negative ratio and detection ratio at relatively low memory cost, and low computational complexity. Moreover MLC exhibits much less false positives than LC and PLC algorithms using similar memory resources. MLC seems therefore to be valuable for heavy-hitters identification in context where both accuracy and bounded memory resources is targeted.

Future work includes a study of how to optimize the implementation of MLC along the direction given in section IV to be able to execute it online for up to OC-768 links with speed of 40 Gbps. This challenge will need at least a division by 100 of the running time of the algorithm.

Acknowledgment

This work was supported by China's Basic Research Program under grant No.2007CB310702, by the NSF of China under grant No. 60873242 and No. 60903208 and by the Major Research Equipment Development Plan of Chinese Academy of Sciences (YZ200824).

References

[1] J.D.C.Little. "A proof for the queueing formula: $L = \lambda W$." Operations Research, vol.9, pp.383-387, May 1961.

[2] W.Fang and L.Peterson. "Inter-AS traffic patterns and their implications". In Proc. IEEE Globecom, pp.1859-1868, Dec.1999,

[3] A.Feldmann, A.Greenberg, C.Lund, N.Reingold, J.Rexford, and F.True. "Deriving traffic demands for operational IP networks: Methodology and experience". In IEEE/ ACM Transactions on Networking, June 2001.

[4] L.A.Adamic and B.A.Huberman. "Zipf's law and the Internet". Glottometrics, vol.3, pp.143-150,2002.

[5] Y.Zhang, L.Breslau , V.Paxson, and S.Shenker. "On the characteristics and origins of Internet flow rates". In Proceedings of ACM SIGCOMM, , Pittsburgh, PA, August 2002.

[6] E.D.Demaine, J.I.Munro, and A.Lopez-Ortiz. "Frequency estimation of internet packet streams with limited space". In European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, Springer-Verlag, pages 348-360, Heidelberg, Germany. Sept.2002.

[7] G.S.Manku, and R.Motwani. "Approximate frequency counts over data streams". In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 346-357, August 2002.

[8] M.Charikar, K.Chen, and M.Farach-Colton. "Finding frequent items in data streams". In Proceedings of International Colloquium on Automata, Languages and Programming(ICALP), 2002.

[9] C.Estan, G.Varghese. "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice". ACM Transactions on Computer Systems, Vol 21, (3), 2003

[10] B.Krishnamurthy, S.sen, Y.Zhang, and Y.chen. "Sketch-based change detection: Methods, evaluation, and applications". In Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference, 2003.

[11] R.M.Karp, S.Shenker, and C.H.Paradimitriou. "A Simple algorithm for finding frequent elements in streams and bags". ACM Transactions on Database Systems, vol.28, no.1, pp.51-55, 2003

[12] Kodialam.M, Lakshman.T.V, Monhanty.S "Runs bAsed Traffic Estimator(RATE): A Simple,Memory Efficient Scheme for Per-Flow Rate Estimator". Proc. of IEEE INFOCOM, Hong Kong,China, 2004.

[13] N.Duffeld, C.Lund, and M.Thorup. " Flow sampling under hard resource constraints". In SIGMETRICS '04/Performance'04: Proceedings of the joint international conference on Measurement and Modeling of Computer Systems, ACM Press, pages85-96, New York, NY, USA, 2004.

[14] K.Papagiannaki, N.Taft, and C.Diot. "Impact of flow dynamics on traffic engineering design principles". In Proceedings of IEEE INFOCOM, 2004.

[15] C.Barakat, G.Iannaccone, and C.Diot. "Ranking flows from sampled traffic". In Proceeding of CoNext, 2005.

[16] G.Cormode, and S.Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". J.Algorithms, 55(1): 58-75, 2005.

[17] A.Metwally, D.Agrawal, and A.E.Abbadi. "An integrated efficient solution for computing frequent and top-k elements in data streams". ACM trans. Database Syst./ 31(3):1095-1133, 2006

[18] R.Stanojevic. "Scalable heavy-hitter identification". Preprint,2006.

[19] N.Kamiyama, and T.Mori. "Simple and accurate identification of high-rate flows by packet sampling". In Proceedings of IEEE INFOCOM, 2006

[20] J.Wallerich and A.Feldmann. "Capturing the Variability of Internet Flows Across Time". In 25th IEEE International Conference on Computer,Communications(INFOCOM-2006),Apr.2006.

[21] S.Bhattacharyya, A.Madeira, S.Muthukrishnan, and T.Ye. "How to scalably and accurately skip past streams". In Proceedings of IEEE 23rd International Conference on Data Engineering, pages 654–663, 2007.

[22] G.Xie, G.Zhang, J.Yang, Y.Min, V.Issarny, and A.Conte. "Survey on traffic of metro area network with measurement on-line". In Proceedings of the 20th International Teletraffic Congress(ITC'07), vol.4516 of Lecture Notes in Computer Science, pp.666-677, Ottawa, Canada, June 2007

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

[24] P.Lieven, B.Scheuermann. "High-Speed Per-Flow Traffic Measurement with Probabilistic Multiplicity Counting". INFOCOM 2010, san Diego, CA, USA, March 2010

[25] http://mawi.nezu.wide.ad.jp/mawi/