Mining Approximate Frequent Closed Flows over Packet Streams

Imen Brahmi[1], Sadok Ben Yahia[1], and Pascal Poncelet[2]

1 Faculty of Sciences of Tunis, Tunisia
imen.brahmi@gmail.com, sadok.benyahia@fst.rnu.tn
2 LIRMM UMR CNRS 5506,161 Rue Ada, 34392 Montpellier Cedex 5, France
poncelet@lirmm.fr

Abstract. Due to the varying and dynamic characteristics of network traffic, the analysis of traffic flows is of paramount importance for network security, accounting and traffic engineering. The problem of extracting knowledge from the traffic flows is known as the *heavy-hitter* issue. In this context, the main challenge consists in mining the traffic flows with high accuracy and limited memory consumption. In the aim of improving the accuracy of heavy-hitters identification while having a reasonable memory usage, we introduce a novel algorithm called ACL-Stream. The latter mines the approximate closed frequent patterns over a stream of packets. Carried out experiments showed that our proposed algorithm presents better performances compared to those of the pioneer known algorithms for heavy-hitters extraction over real network traffic traces.

Keywords: Network Traffic Analysis; Heavy-Hitters; Traffic Flow; Approximate.

1 Introduction

Recently, data streams possess interesting computational characteristics, such as unknown or unbounded length, possibly very fast arrival rate, inability to backtrack over previously arrived items (only one sequential pass over the data is allowed), and a lack of system control over the order in which the data arrive [1]. In this context, the analysis of network traffic has been one of the primary applications of data streams. Generally, the main objectives in network monitoring can be summarized under two aspects as follows: (*i*) understanding traffic features specially the most frequent ones; and (*ii*) detecting outburst network anomalies [11].

For example, given a large-scale campus network or enterprise network, there always exist a huge amount of network flows which have similar traffic features. Flows are usually considered to be sequences of packets with a five-tuple of common values (*i.e.*, protocol, source and destination of IP addresses and port numbers), and ending after a fixed timeout interval when no packets are observed. In this case, many research works considered that the network traffic

pattern obey a heavy-tailed distribution [7, 14, 15], implying a small percentage of flows consuming a large percentage of bandwidth. The flows which are responsible for a huge amount of packets are baptized as *heavy-hitters* [7, 15]. The latter have been shown useful for many applications, such as detecting Denial of Service (DoS) attacks, warning heavy network users, monitoring traffic trends and balancing traffic load [15], to cite but a few.

A straightforward approach to detect the heavy-hitters consists of mining the frequent flows with their corresponding frequency count. Nevertheless, this simple approach is not applicable for high-speed traffic streams. In fact, the traffic streams have often a very large number of distinct network packets, which results in overwhelming and unpredictable memory requirements for flow mining. As an example, we consider the case of a small enterprise network and a NetFlow collector that computes the generated traffic flows.The number of flows over a period of a month is close to 100 million, which corresponds to 2.5 GBytes of memory for storing 136-bit flow identifiers and 64-bit counters. Consequently, the large memory requirements hampers the computation of heavy-hitters over packet streams. In addition, the use of a disk to store a subset of the flow identifiers and counters severely impacts performances and is unsuitable when fast processing is required.

In this paper, we investigate another way of computing the heavy-hitters using limited memory resources. Thus, we introduce a single-pass algorithm, called ACL-Stream (*Approximate CLosed frequent pattern mining over packet Streams*) that provides a condensed representation of heavy-hitters. In this respect, ACL-Stream allows the incremental maintenance of frequent closed patterns as well as the estimation of their frequency counts over a packet stream. Clearly, it has been shown in [7, 14] that, it is unfeasible to find the exact frequency of heavy-hitters using memory resources sub-linear to the number of distinct traffic patterns. Consequently, memory-efficient algorithms approximate the heavy-hitters over a packet stream. Through extensive carried out experiments on a real network traffic traces, we show the effectiveness of our proposal on accuracy, detection ability and memory usage performances.

The remainder of the paper is organized as follows. We scrutinize, in Section 2, the related work. We define the background used to propose our approach in Section 3. In Section 4, we introduce the ACL-Stream algorithm. We also report the encouraging results of the carried out experiments in Section 5. Finally, we conclude by resuming the strengths of our contribution and sketching future research issues.

2 Related work

Due to its high practical relevance, the topic of flow mining has grasped a lot of attention in recent years. Generally, within literature, the algorithms, aiming at the identification of heavy-hitters, are the algorithms of frequent pattern extraction over a stream of packets [15]. Indeed, these algorithms can be roughly

divided into three categories: sampling-based, hash-based and counter-based algorithms.

Sampling-based algorithms [2, 8–10] exploit cyclical sampling to reduce memory footprint and processing overhead, but their accuracy is limited by a low sampling rate required to make the sampling operation affordable.

Hash-based algorithms [3–5, 16] can substantially reduce the 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 carefully chosen in order to avoid collisions.

Counter-based algorithms [6, 7, 12, 14] hold a fixed (or bounded) number of counters for tracking the size of heavy-hitters. In this context, one of the well known examples is the LossyCounting (LC) algorithm [12]. In this respect, it divides the incoming traffic stream into a fixed-size window $w = [1/\epsilon]$, where ϵ is an error parameter such that $\epsilon \in (0, ms)$ and ms is a user-specified minimum support threshold such that $ms \in [0, 1]$. Querying heavy-hitters consists of mining patterns whose estimated frequencies exceed $(ms - \epsilon)$ over each window. Moreover, LC guarantees that the obtained results does not include false negatives[1]. Although, the smaller value of ϵ is the more accurate approximation is. Thus, it leads to requiring both more memory space and more CPU processing power. In addition, if ϵ approaches ms, then more false positives[2] will be outputted.

Aiming at improving the LC algorithm in computing network traffic heavy-hitters, Dimitropoulos et *al.* [7] proposed the ProbabilisticLossyCounting (PLC) algorithm. PLC uses a tighter error bound on the estimated sizes of traffic flows. Consequently, it drastically reduces the required memory and improves the accuracy of heavy-hitters identification. However, PLC needs to emulate heavy-tailed distribution at the end of each window, causing a high computational complexity.

Recently, Zhang et *al.* introduced an algorithm called WeightedLossy-Counting (WLC) [15]. WLC is able to identify heavy hitters in a high-speed weighted data stream with constant update time. Moreover, it employs an ordered data structure which is able to provide a fast per-item update speed while keeping the memory cost relatively low.

Due to its usability and importance, reducing the memory space of frequent flows still present a thriving and a compelling issue. In this respect, the main thrust of this paper is to propose a new algorithm, called ACL-Stream, to mine approximate closed frequent patterns from flows, which can be seen as an extension of a concise representation of flows to the heavy-hitters identification search space. The main idea behind our approach comes from the conclusion drawn from the Data Mining community that focused on the closed frequent pattern mining over a data stream. In fact, the extraction of the latter requires

[1] The false negatives are the patterns considered as frequent on the entire traffic data and infrequent in window.

[2] The false positives are the patterns considered as frequent in the window and infrequent on the entire traffic data

less memory. Thus, this fact has been shown to be much suitable for the mining stream, since it presents the best compactness rates.

3 Approximate closed patterns

One of the most known condensed representation of patterns is based on the concept of closure [13].

Definition 1. *A pattern X is a closed pattern if there exists no pattern X' such that: (i) X' is a proper superset of X; and (ii) every packet in a network traffic*[3] *containing X also contains X'. The closure of a the maximal superset of X having the same support value as that of X.*

Table 1. A snapshot of network traffic data.

Packet_ID	Packets
p_1	src_IP1,protocol
p_2	src_port,dst_port
p_3	src_port,dst_port,src_IP1
p_4	src_port,dst_port,src_IP1
p_5	src_port,src_IP1,protocol

Example 1. Let Table 1 sketching a set of packets. The set of closed patterns with their corresponding frequency counts (*i.e.*, supports) is as follows: { (src_port: 4); (src_IP1: 4); (src_port dst_port: 3); (src_port src_IP1: 3); (src_IP1 protocol: 2); (src_port dst_port src_IP1: 2); (src_port src_IP1 protocol: 1)}.

Due to the dynamically characteristic of traffic stream, a pattern may be infrequent at some point in a stream but becomes frequent later. Since there are exponentially many infrequent patterns at any point in a stream, it is infeasible to keep all infrequent patterns. Suppose we have a pattern X which becomes frequent after time t. Since X is infrequent before t, its support in the stream before t is lost. In this respect, to estimate X's support before t, the counter-based algorithm for heavy-hitters mining [7, 12, 14, 15] uses an error parameter, ϵ. X is maintained in the window as long as its support is at least $\epsilon \times N$, where N is the number of packets within the current window. Thus, if X is kept only after t, then its support before t is at most $\epsilon \times N$. However, the use of small ϵ results in a large number of patterns to be processed and maintained. Consequently, this fact drastically increases the memory consumption and severely degrades the processing efficiency.

To palliate this drawback, we consider ϵ as *a relaxed minimum support threshold* and propose to progressively increase the value of ϵ for a pattern as it is retained longer in the window.

Definition 2. *The relaxed minimum support threshold is equal to* $r \times ms$*, where* $r(0 \leq r \leq 1)$ *is the relaxation rate.*

[3] Network traffic is data in network.

Since all patterns whose support is lower than $r \times ms$ are discarded, we define the approximate support of a pattern as follows.

Definition 3. *The approximate support of a pattern X over a time unit t is defined as*

$$\widetilde{SUP(X,t)} = \begin{cases} 0, & \text{if } sup(X,\ t) < r \times ms; \\ sup(X,t), & \text{otherwise.} \end{cases}$$

4 The ACL-Stream algorithm

To effectively mine the closed frequent patterns within a packet stream environment, we propose a novel algorithm, called ACL-Stream, for maintaining the frequent closed patterns. The main idea behind their extraction is to ensure an efficient computation of heavy-hitters that reduces the memory requirements.

With the consideration of time and space limitation, the proposed algorithm uses two in-memory data structures which are called CITable (*Closed Incremental Table*) and CIList (*Closed Identifier List*) respectively. In addition, it employs a *hash table*, called $Temp_{New}$, to put the patterns that have to be updated whenever a new packet arrives. In fact, the rationales behind such in-memory data structures are: (*i*) saving storage space; and (*ii*) reducing the cost of the incremental maintenance of patterns.

Table 2. Example of CITable.

Cid	Clos	Count
0	{0}	0
1	{src_IP1 protocol}	2
2	{src_port dst_port}	3
3	{src_port dst_port src_IP1}	2
4	{src_IP1}	4
5	{src_port src_IP1 protocol}	1
6	{src_port}	4
7	{src_port src_IP1}	3

Table 3. Example of the CIList.

Item	cidset
src_port	{2, 3, 5, 6, 7}
dst_port	{2, 3}
src_IP1	{1, 3, 4, 5, 7}
protocol	{1, 5}

The CITable is used to keep track of the evolution of closed patterns. Each record of the CITable represents the information of a closed pattern. It consists of three fields: *Cid*, *Clos* and *Count*. Each closed pattern was assigned a unique closed identifier, called Cid. The Cid field is used to identify closed patterns. Given a Cid, the ACL-Stream algorithm gets corresponding closed patterns in the Clos field. The support counts are stores in the Count field.

Example 2. According to the database shown by Table 1, the CITable is sketched by Table 2.

The CIList is used to maintain the items and their cidsets. It consists of two fields: the Item field and the cidset field. The cidset of an item X, denoted as cidset(X), is a set which contains all cids of X's super closed patterns.

Algorithm 1: The ACL-Stream algorithm

Input: $\mathcal{T}$, r, ms
Output: Updated CITable
1 **Begin**
2 $w := [1/r \times ms]$;
3 //Phase 1
4 $Temp_{New} := (p_{New}, 0)$;
5 SET($\{p_{New}\}$) = cidset(i_1) ∪...∪ cidset(i_k);
6 **Foreach** *Cid(i)* ∈ $SET(\{p_{New}\})$ **do**
7 $\mathcal{IR}$:= Null;
8 $\mathcal{IR} := p_{New} \cap Clos[i]$;
9 **If** $\mathcal{IR} \in Temp_{New}$ **then**
10 **If** $\widetilde{SUP(Clos}[i]) > \widetilde{SUP(Clos}[z])$ **then**
11 replace ($\mathcal{IR}$, i) with ($\mathcal{IR}$, z) in $Temp_{New}$
12 **Else**
13 $Temp_{New} := Temp_{New} \cup (\mathcal{IR}, i)$
14 //Phase 2
15 **Foreach** *(X, c)* ∈ $Temp_{New}$ **do**
16 **If** $X == Clos[c]$ **then**
17 $\widetilde{SUP(Clos}[c]) := \widetilde{SUP(Clos}[c]) + 1$;
18 **Else**
19 j := j+1;
20 CITable := CITable ∪(j, X, $\widetilde{SUP(Clos}[c]) + 1$);
21 **Foreach** $i \in p_{New}$ **do**
22 cidset(i) := cidset(i) ∪ j
23 **End**

Example 3. According to Table 1 and the CITable shown by Table 2, {src_port dst_port} is closed and its Cid is equal to 2. Thus, 2 will be added into cidset(src_port) and cidset(dst_port) respectively. Table 3 illustrates a CIList. It maintains the items and their corresponding superset cids shown by Table 2.

The pseudo-code of the ACL-Stream algorithm is shown by Algorithm 1. In this respect, the ACL-Stream algorithm attempts to mine a concise representation of heavy-hitters that delivers approximate closed frequent flows. Indeed, the algorithm takes on input a network traffic trace $\mathcal{T}$, a minimum support threshold ms and a relaxation rate r. It starts by reading a fixed window of packets, w, such as $w = [1/r \times ms]$ (line 2). The window facilitates the continuous monitoring of changes in the stream. Moreover, it can be used to palliate the drawback of unbounded memory over the packet streams. In addition, whenever a new packet p_{New} arrives, the ACL-Stream algorithm consists of two phases. During the first one, the algorithm finds all patterns that need to be updated with their closures, and puts them into $Temp_{New}$ (lines 4−13). Within the second phase, the ACL-Stream algorithm updates their supports, CITable and CIList (lines

15−22). Consequently, the updated closed patterns can be obtained without multiple scans of whole search spaces, *i.e.*, by scanning the CITable once.

4.1 Incremental maintenance over packet streams

We assume that p_{New} denotes a new incoming packet. $\mathcal{T}_{orig}$ the original network traffic, *i.e.*, the network traffic before adding p_{New}. $\mathcal{T}_{up} = \mathcal{T}_{orig} \cup p_{New}$ is the updated network traffic after adding p_{New}. $\text{Clos}_{Torig}(X)$ and $\text{Clos}_{Tup}(X)$ represent the closure of X within $\mathcal{T}_{orig}$ and $\mathcal{T}_{up}$ respectively.

Property 1. Whenever p_{New} arrives to $\mathcal{T}_{orig}$, then the patterns of $p_{New} \in \text{Clos}_{Torig}$.

Property 2. Whenever p_{New} arrives to $\mathcal{T}_{orig}$, if a pattern Y is not a subset of p_{New}, then the status of Y will not be changed, *i.e.*, $\widetilde{SUP(Y)}$ remains such as it is and $\text{Clos}_{Torig}(Y) = \text{Clos}_{Tup}(Y)$.

Property 3. Suppose a pattern $\mathcal{IR} = p_{New} \cap X$, $\mathcal{IR} \in \mathcal{T}_{orig}$. If $\mathcal{IR} \neq \emptyset$, then $\mathcal{IR}$ is a closed pattern in $\mathcal{T}_{up}$.

In the following, we thoroughly discuss the two phases of the ACL-Stream algorithm according to the pseudo-code shown by Algorithm 1.

Phase 1: According to Property 1, whenever a p_{New} arrives it is considered as closed in $\mathcal{T}_{orig}$. The ACL-Stream algorithm puts p_{New} into $Temp_{New}$. The table $Temp_{New}$ takes the TI field as a key, and the Closure_Id field as value. Initially, ACL-Stream sets the Closure_Id of p_{New} to 0, since its closure is unknown (line 4). Besides, the ACL-Stream algorithm intersects p_{New} with its associated closed patterns. The set of cids of associated closed patterns is defined as $\text{SET}(\{p_{New}\}) = \text{cidset}(i_1) \cup ... \cup \text{cidset}(i_k)$. Consequently, the algorithm finds the patterns of p_{New} that need to be updated (line 5). According to Property 3, the results of the intersection are closed patterns within the updated network traffic $\mathcal{T}_{up}$. Suppose $\mathcal{IR}$ is the intersection result of p_{New} and a closed pattern C having a Cid $i \in \text{SET}(\{p_{New}\})$ (lines 7−8). If $\mathcal{IR}$ is not in $Temp_{New}$, then ACL-Stream puts ($\mathcal{IR}$, i) into $Temp_{New}$ (lines 12−13). Otherwise, if $\mathcal{IR}$ is already in $Temp_{New}$ with its current Closure_Id t, then ACL-Stream compares $\widetilde{SUP(C)}$ and $\widetilde{SUP(Q)}$ such that Q is a closed pattern having a Cid z in the CITable, *i.e.*, $\text{Clos}[z] = Q$ (line 9−10). If $\widetilde{SUP(C)}$ is greater than $\widetilde{SUP(Q)}$, then ACL-Stream replaces ($\mathcal{IR}$, z), already in $Temp_{New}$ with ($\mathcal{IR}$, i) (line 11). The reason is that the closure of $\mathcal{IR}$ has a support greater than any of its superset's support (Properties 2 and 3). The intersections of p_{New} with C iterates till all cids in $\text{SET}(\{p_{New}\})$ are processed (line 6). Consequently, the phase 1 allows the identification of patterns that need to be updated and finds their closure before the new incoming packet arrives.

Phase 2: The ACL-Stream algorithm gets patterns X with their Closure_Id c from $Temp_{New}$, and checks that whether X is already in the CITable. If X

is already in the CITABLE with Cid c, then X is originally a closed in $\mathcal{T}_{orig}$. In this case, ACL-STREAM directly increases $\widetilde{SUP(X)}$ by 1. Otherwise, X is a new closed pattern after the $Temp_{New}$ arrival. In this case, $\widetilde{SUP(X)}$ is equal to the support of its closure increased by 1 (line 16−17). At the same time, ACL-STREAM assigns to X a new Cid n, puts X into the CITABLE, and updates the CILIST (lines 19−20). The phase 2 is repeated till all records in $Temp_{New}$ are processed (line 15). Finally, ACL-STREAM comes to end and outputs the updated CITABLE. The obtained CITABLE captures all the information enclosed in a packet stream.

Example 4. According to Table 1, before that p_1 arrives, we have $\mathcal{T}_{orig} = \emptyset$. The first record of the CITABLE is set to (0, 0, 0). Each cidsets in the CILIST is set to $\emptyset$. As p_1 = {src_IP1 protocol} arrives, $\mathcal{T}_{up} = \mathcal{T}_{orig} \cup p_1$. The ACL-STREAM algorithm puts {src_IP1 protocol} into $Temp_{New}$ and sets its Closure_Id to 0. Then, ACL-STREAM merges cidset(src_IP1) and cidset(protocol) to get SET({src_IP1 protocol}), *i.e.*, SET({src_IP1 protocol}) = cidset(src_IP1) ∩ cidset(protocol) = $\emptyset$. Since SET({src_IP1 protocol}) is empty, p_1 does not need to intersect with any closed patterns. Therefore, the phase 1 was completed. ACL-STREAM goes to phase 2. Within phase 2, ACL-STREAM updates patterns within $Temp_{New}$ by their Closure_Id. Only ({src_IP1 protocol}, 0) in $Temp_{New}$. ACL-STREAM finds a closed pattern whose Cid is 0 from the CITABLE, Clos[0] ={0}. Since {0} is not equal to {src_IP1 protocol}, then {src_IP1 protocol} is a new closed pattern after the p_1 arrival. Hence, ACL-STREAM assigns {src_IP1 protocol} a new Cid equal to 1. Then, ACL-STREAM determines $SUP(\{\widetilde{src\_IP1\ protocol}\})$, which is equal to $\widetilde{SUP(Clos}[0])$ increased by 1. Therefore, $SUP(\{\widetilde{src\_IP1\ protocol}\})$ is 1. Finally, ACL-STREAM updates the CITABLE and the CILIST respectively. Thus, it inserts (1, {src_IP1 protocol}, 1) into the CITABLE and inserts 1 into the CILIST. Then, it handles p_2, p_3, p_4 and p_5 in the same manner. After the insertion of the packets shown by Table 1, the obtained CITABLE and CILIST are shown by Table 2 and Table 3, respectively.

Table 4. Example of $Temp_{New}$ whenever p_6 arrives.

TI	Closure_id
{src_IP1}	{4}
{dst_port}	{2}
{dst_port src_IP1}	{3}

Assume that a new packet p_6 = {dst_port src_IP1} arrives, then ACL-STREAM puts {dst_port src_IP1} into $Temp_{New}$, and sets its Closure_Id to 0. Moreover, SET({dst_port src_IP1}) = cidset(dst_port) ∩ cidset(src_IP1) = {2, 3} ∩ {1, 3, 4, 5, 7} = {1, 2, 3, 4, 5, 7}, according to Table 3. Thus, ACL-STREAM intersects p_6 with the closed patterns whose cids belongs to SET({dst_port src_IP1}). Clearly, the first is Clos[1] = {src_IP1 protocol} and {dst_port src_IP1} ∪ {src_IP1 protocol} = {src_IP1}. Hence, it puts {src_IP1} into $Temp_{New}$ and sets its Closure_Id to 1. Then, it deals with 2, Clos[2] = {src_port dst_port} and {dst_port src_IP1} ∪ {src_port dst_port} = {dst_port}. Consequently, ACL-

STREAM puts ({dst_port}, 2) into $Temp_{New}$. It deals with Clos[3] = {src_port dst_port src_IP1}, *i.e.*, {dst_port src_IP1} ∪ {src_port dst_port src_IP1}= {dst_port src_IP1}. However, {dst_port src_IP1} is already in $Temp_{New}$ and its current Closure_Id is 0. Additionally, $\widetilde{SUP(Clos}[3])$ is greater than $\widetilde{SUP(Clos}[0])$. Therefore, ACL-STREAM replaces ({dst_port src_IP1}, 0) with ({dst_port src_IP1}, 3). After dealing with the remaining closed patterns with the same processing steps, the result of $Temp_{New}$ is shown by Table 4.

5 Experiments

To evaluate the effectiveness and efficiency of our algorithm ACL-STREAM, we carried out extensive experiments. Indeed, we compare our approach with the pioneering algorithms falling within the detection of heavy-hitters trend, namely, LC [12], PLC [7] and WLC [15]. All experiments were carried out on a PC equipped with a 3GHz Pentium IV and 4GB of main memory running under Linux Fedora Core 6.

During the carried out experiments, we used a real network traffic trace. The latter is collected from the gateway of a campus network with 1500 users in China[4]. Table 5 sketches dataset characteristics used during our experiments.

Table 5. The considered real traffic traces at a glance.

Traffic traces	
Source	Campus network
Date	2009-08-24 16:20-16:35
Packets	49,999,860
Unique flows	4,136,226

During the experiments, we set $r = 0.1$ and vary ms from 0.1 to 1 such that $\epsilon = 0.1 \times ms$ within LC, PLC and WLC.

5.1 Accuracy assessment

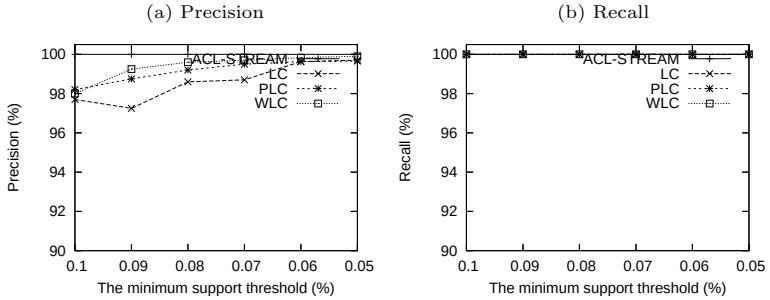

Fig. 1. Precision and Recall of ACL-STREAM *vs.* LC, PLC and WLC.

The accuracy of mining results are measured by the use of two metrics, *Precision* and *Recall*.

[4] We thank Mr. Q. Rong [14] for providing us with the real network traffic trace.

Since the approximate algorithms, falling within the detection of heavy-hitters trend, possibly return false positives, the precision indicates the number of false positive results [15]. In fact, Figure 1(a) illustrates the precision of ACL-Stream *vs.* those respectively of LC, PLC and WLC. We remark that the ACL-Stream algorithm achieves 100% precision. Although, there is no clear difference between the three algorithms LC, PLC and WLC, such that $\epsilon = 0.1 \times ms$. They attain high precision whenever ms is small. Whereas, their precision drops linearly to be less than 95% as far as ms increases (*i.e.*, the increase in the error parameter ϵ). Thus, the increase of ϵ, associated to ms, results in worsening the precision for the three algorithms. On the contrary, Figure 1(b) shows that whenever ms increases, all algorithms (*i.e.*, ACL-Stream, LC, PLC and WLC) have 100% recall. This is can be explained by the fact that all algorithms guarantee to retrieval of all frequent patterns over the packet stream.

The experimental results reveal that the estimation mechanism of the LC, PLC and WLC algorithms relies on the error parameter ϵ to control the accuracy. Compared with these three algorithms, ACL-Stream is much less sensitive to ϵ and is able to significantly achieves high accurate approximation results by increasing ϵ.

5.2 The detection ability

Generally, to evaluate the algorithm's ability to detect the heavy-hitters, two interesting metrics are usually of use [14]: the *False Positive Ratio* (FPR) and the *False Negative Ratio* (FPR). Figures 2(a) and 2(b) respectively plot the FPR and FNR against the minimum support threshold ms, for ACL-Stream, LC, PLC and WLC.

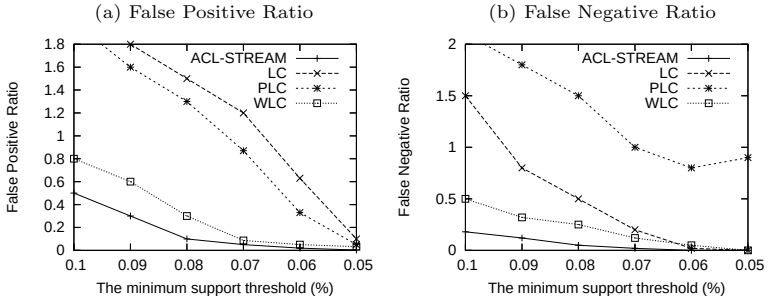

Fig. 2. Detection ability of ACL-Stream *vs.* LC, PLC and WLC.

On the one hand, we remark that both PLC and LC generate many false positives along with the decrease of ms. Therefore, they are not suitable for accurate mining of heavy-hitters within high-speed network. For example, if the minimum support threshold is equal to 0.09%, the FPR of LC can reach values as high as 1.8. On the other hand, among the four investigated algorithms, PLC is the most generator of false negatives. Whenever ACL-Stream, LC and WLC have 0 FNR, PLC has 0.9 FNR. This is due to the probabilistic nature of PLC. Generally, ACL-Stream exhibits a lower FNR and a lower FPR than the other algorithms. Thus, we conclude that our algorithm is able to correctly detect heavy-hitters with few FPR and FNR.

5.3 Memory consumption and Throughput

Figure 3(a) shows *the throughput*, in a logarithmic scale, of ACL-Stream *vs.* LC, PLC and WLC. The throughput is measured by the number of packets processed per second by the algorithms. Indeed, we remark that the result points out the ability of ACL-Stream to handle high-speed packet streams as it can process up to 45,000 packets per second. For all minimum support thresholds, the throughput of ACL-Stream is over two orders of magnitude higher than that of both LC and WLC and three orders of magnitude higher than that of PLC.

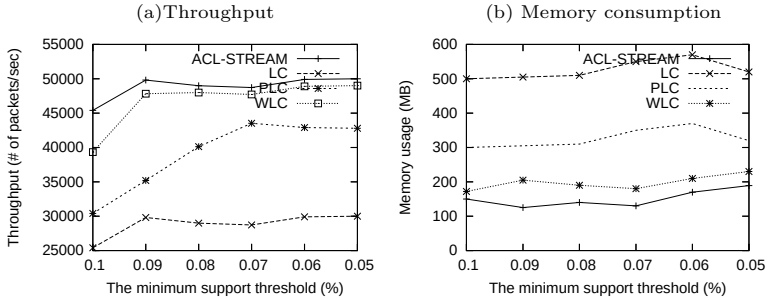

Fig. 3. Throughput and Memory consumption of ACL-Stream *vs.* LC, PLC and WLC.

Moreover, Figure 3(b) shows that ACL-Stream achieves a roughly constant memory consumption of no more than 150MB. Considering the four algorithms, the memory consumption of ACL-Stream is considerably lower than that of WLC and substantially lower than that of both LC and PLC.

6 Conclusion

In this paper, we focused on the condensed representation of heavy-hitters algorithms to tackle the mentioned above challenges, *i.e.*, large memory requirement for heavy-hitters computation. Thus, we introduced a novel stream mining algorithm called ACL-Stream. The carried out experimental results showed the effectiveness of the introduced algorithm and highlighted that the ACL-Stream presents better performance as well as a good detection ability than the pioneering algorithm in heavy-hitters identification. The approximate but high-quality online results, provided by ACL-Stream, are well-suited to detect heavy-hitters, where the main goal is to identify generic, interesting or unexpected patterns.

Future issues for the present work mainly concern: (*i*) The consideration of the intrusion detection over on-line packet streams and the mining of closed frequent patterns from flows for network monitoring; (*ii*) Study of the extraction of "generic streaming" association rules based on the ACL-Stream algorithm for heavy-hitters analysis.

References

1. C.C Aggrawal. D*ata S*t*reams:* M*odels and* A*lgorithms*. Springer, 2007.
2. C. Barakat, G. Iannaccone, and C. Diot. Ranking Flows From Sampled Traffic. In *Proceedings of the 2005 ACM conference on Emerging network experiment and technology, Toulouse, France*, pages 188–199, 2005.
3. S. Bhattacharyya, A. Madeira, S. Muthukrishnan, and T. Ye. How To Scalably and Accurately Skip Past Streams. In *Proceedings of the IEEE 23rd International Conference on Data Engineeringg, Istanbul, Turkey*, pages 654–663, 2007.
4. T. Bu, J. Cao, A. Chen, and P.P.C. Lee. Sequential Hashing: A Flexible Approach for Unveiling Significant Patterns in High Speed Networks. *Computer Network*, 54(18):3309–3326, 2010.
5. G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.
6. E.D. Demaine, A. López-Ortiz, and J.I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th Annual European Symposium on Algorithms, Rome, Italy*, pages 348–360, 2002.
7. X. Dimitropoulos, P. Hurley, and A. Kind. Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy-hitters. *ACM SIGCOMM Computer Communications Review*, 38(1):5–5, 2008.
8. N. Duffeld, C. Lund, and M. Thorup. Flow Sampling Under Hard Resource Constraints. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, ACM Press, New York, NY, USA*, pages 85–96, 2004.
9. N. Kamiyama and T. Mori. Simple and Accurate Identification of High-rate Flows by Packet Sampling. In *Proceedings of the 25th IEEE International Conference on Computer Communications, Barcelona, Spain*, pages 2836–2848, 2006.
10. M. Kodialam, T.V. Lakshman, and S. Monhanty. Runs Based Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China*, pages 1808–1818, 2004.
11. X. Li and Z.H. Deng. Mining Frequent Patterns from Network Flows for Monitoring Network. *Expert System with Applications*, 37(12):8850–8860, 2010.
12. G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China*, pages 346–357, 2002.
13. N. Pasquier, Y. Bastide, R. Touil, and L. Lakhal. Discovering Frequent Closed Itemsets. In *Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel*, pages 398–416, 1999.
14. Q. Rong, G. Zhang, G. Xie, and K. Salamatian. Mnemonic Lossy Counting: An Efficient and Accurate Heavy-hitters Identification Algorithm. In *Proceedings of the 29th IEEE International Performance Computing and Communications Conference, Albuquerque, United States*, 2010.
15. Y. Zhang, B.X. Fang, and Y.Z. Zhang. Identifying Heavy-hitters in High-speed Network Monitoring. *Science China*, 53(3):659–676, 2010.
16. Z. Zhang, B. Wang, S. Chen, and K. Zhu. Mining Frequent Flows Based on Adaptive Threshold with a Sliding Window over Online Packet Stream. In *Proceedings of the International Conference on Communication Software and Networks, Macau, China*, pages 210–214, 2009.