Estimating the number of Active Flows in a Data Stream over a Sliding Window

Éric Fusy * Frédéric Giroire *

Abstract

A new algorithm is introduced to estimate the number of distinct flows (or connections) in a data stream. The algorithm maintains an accurate estimate of the number of distinct flows over a sliding window. It is simple to implement, parallelizes optimally, and has a very good trade-off between auxiliary memory and accuracy of the estimate: a relative accuracy of order $1/\sqrt{m}$ requires essentially a memory of order $m \ln(n/m)$ words, where n is an upper bound on the number of flows to be seen over the sliding window. For instance, a memory of only $64kB$ is sufficient to maintain an estimate with accuracy of order 4 percents for a stream with several million flows. The algorithm has been validated both by simulations and experimentations on real traffic. It proves very efficient to monitor traffic and detect attacks.

1 Introduction

A multiset is a set where each element can appear several times. The *cardinality* n of the multiset is the number of distinct elements, while the *size* N of the multiset is the total number of elements, counting the repetitions. An important issue in computer science is to estimate the cardinality of a multiset having a very large size. This problem has arisen in the 1980's, motivated by optimisation of classical algorithmic operations on data bases (union, intersection, sorting,...). As the data sets to be measured have mostly a very large size N, far beyond the RAM capacities, a natural requirement is to treat the data in one pass using a simple loop, and with a small auxiliary memory (constant or logarithmic in N). The crucial point, first developed by Flajolet and Martin [FM83] in their algorithm Probabilistic Counting, is to relax the contraint of giving the exact number of distinct values in the multiset and to build only a *probabilistic estimate* of n. This algorithm as well as the recent LogLog Counting algorithm [DF03] build the estimate by maintaining a *bitmap pattern*. Even more recently, a whole family of estimators —based on the minimal hashed value over the elements of the multiset— has been introduced by the second author [Gir05]. All these algorithms perform a single pass on the data, use a *constant* auxiliary memory, and return an *unbiased* estimate of n (i.e., the expectation of the estimate is n). In addition, they have a similar trade-off between accuracy and memory: a memory of m words is required to have an estimate with a relative accuracy of order $1/\sqrt{m}$. Another type of algorithm, based on the principle of *Adaptative Bitmap*, has been introduced by Estan and Varghese [EVF03]. It is very efficient in practice, but the analysis requires some assumptions on the regularity of the traffic, in contrast to the algorithms [DF03, FM83, Gir05].

In the past decade, the problem of counting distinct elements in a multiset has appeared as a crucial algorithmic operation in the context of *data streams*. In contrast to a data base, the elements of a data stream have a *time stamp* indicating their time of arrival. Typically, the elements are *packets*, each packet belonging to a *flow* (also called connection) identified by a source-adress and a destination-adress. In this framework, the packets are the elements of the multiset and the flows are the distinct elements of the multiset. Estimating the number of distinct flows in a data stream has many applications in network monitoring and network security, see the detailed survey of Estan and Varghese [EVF03]. For instance, the system FlowScan [Plo00] counts distinct flows on a traffic (using an unoptimized algorithm) to detect Denial Of Service attacks, where abnormally many distinct connections are opened in a short period of time. In the context of data streams, the multiset to be treated is not fixed (as it is the case in applications of data bases) but evolves with the time, getting a new element at each packet arrival and losing elements that become outdated. Hence, the problem is to be solved over a *sliding window* of size W (W is the lapse of time after which a packet is outdated). Precisely, one has to be able to answer at any time t a request of the form "What is the approximate number of distinct flows on the traffic over the last w units of time ($w \leq W$)". Notice that the previous "static" algorithms [FM83, DF03, EVF03, Gir05] can not answer such requests: they should be launched at time $t - w$

*Projet Algo, INRIA Rocquencourt, B. P. 105, 78153 Le Chesnay Cedex, France

and be run until time t, i.e., they should know *a priori* at time $t-w$ that there will be a request w units of time later.

In this article, we develop an efficient algorithm, called Sliding MinCount, to solve the problem of estimating the cardinality of a multiset over a sliding window. Our algorithm builds upon the "static" algorithm —called MinCount— of [Gir05], where the estimate is based on the minimal hashed value of the elements. The idea is to maintain a list of values that may become a minimum over a future window of time (including the current one). In particular, the current minimum over $[t-W,t]$ is always in the list at time t, so that the estimate can be calculated *a posteriori*. The list, called LFPM (for List of Future Possible Minima) is very simple to maintain and surprisingly short: the precise analysis over the whole execution of the algorithm (Theorem 4.1) ensures that its size is logarithmic in the maximal number of distinct flows to be seen over a window of time. The idea of using the minimal hashed value is fruitful, being also exploited by Datar and Muthukrishnan in [DM02] to measure a so-called *rarity parameter*. Let us mention that it is also possible to make the probabilistic algorithms based on bitmap patterns [DF03, FM83] work with a sliding window. The process is briefly explained in the seminal article of Datar et al [DGIM02]. They obtain the same order of complexity as our algorithm, though no detailed analysis nor experimental results are provided. In contrast, we give a thorough analysis of the required memory. This includes the analysis of the maximal required memory over a long period of time, a relevant question to get sure that there is no chance of overflooding the memory.

In addition, we provide various experimental validations of our algorithm. Simulations with ideal traffic show good agreement between the algorithm behaviour and what the analysis predicts. Using only $64kB$ of memory, Sliding MinCount succeeds in estimating the number of distinct flows in a data stream of 5 million elements for a sliding window of one million elements within a precision of 4%. Observations of real traffic reveal that Sliding MinCount would readily monitor in real time at the rate of several millions of packets per second. Over traces of one day (containing 150 millions of packets), the algorithm detects peaks of the number of connections that are invisible by just looking at the number of packets. This is of first interest to detect Denial of Service attacks as they generate lots of connections with few packets. An automatic detection of unusual traffic can be easily set up by triggering an alarm when the estimate of Sliding Mincount exceeds a given threshold.

2 Preliminaries

In this section, we present the algorithm MinCount [Gir05] that estimates the cardinality of a fixed multiset. As our algorithm Sliding MinCount —to be presented in Section 3— operates on data streams, we will from now on use the terminology of packets (elements of the multiset) and flows (distinct elements of the multiset), to be defined next.

Definitions. A *flow* is identified by a set of header fields. Such an identifier, shortly called FlowID, is typically a pair <source IP, destination IP>, but can also be of the form <source IP, destination IP, destination Port> if the port information is taken into account (e.g. to detect port scans). A packet consists of a *header*, containing header fields, and a *body*, containing the information transported. Two packets are said to belong to the same flow if their FlowID is the same. From now on, we assume to have at our disposal a *hash function* h mapping a FlowID to a real value that "looks like" uniformly distributed in the interval $[0,1]$ (see the detailed study of Knuth [Knu98]).

Let $S=(p_1,\ldots,p_N)$ be a set of packets and let n be the number of distinct flows in S. Under the assumption on the hashed function h, and without making any assumption on the nature of the traffic, the set $(h(p_1),\ldots,h(p_N))$ of hashed packets can be considered as built from n real values taken independently uniformly at random in $[0,1]$, and then replicated and permuted in an arbitrary way. Such a set of uniform random values in $[0,1]$ with arbitrary replications and order of appearance is called an *ideal multiset*. Thus, *Estimating the number of distinct flows in a set of packets without making any assumptions on the traffic amounts to estimating the cardinality of an ideal multiset.*

The MinCount algorithm. We present a recently introduced probabilistic algorithm [Gir05] that estimates the number of distinct values in an ideal multiset with a very good trade-off between memory and accuracy. The idea is that the minimum of the values of an ideal multiset does not depend on the replication structure of the data nor on their order of appearance, and gives an indication on the number n of distinct values of the multiset (basically, the minimum of n independent uniform values on $[0,1]$ has more chances of being small if n is large). The principle is then to build an observable based on the minimum, and to combine it with a *stochastic averaging process*, as introduced in [FM83], in order to have an accurate estimate of n. Stochastic averaging consists in simulating the effect of $m=2^b$ experiments on the multiset and then averaging an observable over the m experiments. The first b bits of a value x of the multiset are used to direct x to one of m *buckets* (the one whose index i, written in binary base,

Algorithm MinCount (with $m = 2^b$ buckets)
Input: $(p_1, \ldots, p_N)$ a set of packets;
Initialize $M^{(1)}, \ldots, M^{(m)}$ to 0;
for i from 1 to N
  $u_i \leftarrow h(p_i)$; [hash p_i to a real value in $[0, 1]$]
  $j \leftarrow$ integer corresponding to the first b bits of u_i;
  $\widetilde{u}_i \leftarrow 2^b u_i - \lfloor 2^b u_i \rfloor$; [$\widetilde{u}_i$ is u_i truncated of its first b bits]
  $M^{(j)} \leftarrow \min(M^{(j)}, \widetilde{u}_i)$;
return $\xi := m\Gamma(2 - 1/m)^{-m} \exp\left(-\frac{1}{m}\sum_{j=1}^{m} \ln(M^{(j)})\right)$
  as estimate of the number of distinct flows

Figure 1: Pseudo-code of the algorithm MinCount.

corresponds to the b bits), and the value directed to the bucket is the original value x truncated of its b first bits (thus this value is also uniform in $[0, 1]$). The algorithm then builds an accurate estimate of the number of distinct values from the minima of the buckets, see Figure 1 for a summary given in pseudo-code.

Theorem 2.1. (Giroire [Gir05]) *Let $\mathcal{I}$ be an ideal multiset and let n be the cardinality of $\mathcal{I}$. Let $m = 2^b$ be the number of buckets to be used by the estimate. For $1 \le j \le m$, let $M^{(j)}$ be the minimum of the values that are directed to the bucket of index j. Then, writing Γ for Euler's Gamma function, the quantity*
(2.1)
$$\xi := m\Gamma(2-1/m)^{-m} \exp\left(-\frac{1}{m}(\ln(M^{(1)}) + \ldots + \ln(M^{(m)})\right)$$
can be computed with a memory of m words and is an accurate estimate of the cardinality of $\mathcal{I}$ in the sense that:

- *It is asymptotically unbiased, i.e., $\mathbb{E}(\xi) \underset{n\to\infty}{\sim} n$.*
- *Its relative accuracy, defined as $\sqrt{\mathbb{V}(\xi)}/\mathbb{E}(\xi)$, is about $1.3/\sqrt{m}$.*

3 Presentation of the algorithm Sliding MinCount

We have seen in the last section that the number of distinct flows in a fixed set of packets can be accurately estimated with a small auxiliary memory using an estimate that is computed from the *minima* of the hashed values in each of m buckets. In this section, we build upon this idea to estimate the number of distinct flows over a *sliding window.* What we consider here is not a fixed data set as in Section 2, but a *data stream.* A data stream is an infinite sequence of packets. In addition, each packet has a *timestamp* indicating its time of arrival (timestamps are for instance given in traces of routers). Thus, using a hash function h as in Section 2, a data stream can be formalized as an infinite sequence $< t_k, H_k >_{k\ge1}$, where t_k is the timestamp and H_k is the hashed value of the kth arrived packed (thus t_k is increasing). We identify the kth packet with the pair $< t_k, H_k >$. To maintain an estimate over the sliding window, we also use the stochastic averaging process: the first b bits of H_k are used to direct the kth packet into one of $m = 2^b$ buckets, and then the packet directed into the bucket is $< t_k, H'_k >$ where H'_k is obtained from H_k by truncating the first b bits. This process partitions the datastream $< t_k, H_k >$ into m datastreams, one for each bucket. If we are able to maintain the minimum in each bucket, we will also be able to compute the estimate of Theorem 2.1 over a sliding window.

3.1 Maintaining the minimum over a sliding window. The problem of maintaining the minimum of values of a datastream $< t_k, H_k >$ over a sliding window is formulated as follows: given a *maximal-window* parameter W, find a process such that the minimum of values of the stream in a window $[t - w, t]$ can be found at any time t and for any $w \le W$.

The solution we propose is to maintain a list of packets that *may become a minimum* in a future window. After completing a first draft of the paper, we have seen in the article of Datar *et al* [DGIM02] that a similar solution is briefly indicated, but without any analysis. We provide a more detailed presentation of the process, with a specific terminology, and analyze the required memory in Section 4.

At time t, we consider the set of packets of the stream that are in the window $[t - W, t]$. We define the *minimum packet* of the window as the latest arrived packet among the set of packets whose hashed value realizes the minimum over $[t - W, t]$. A *future possible minimum*, shortly called FPM, in the window $[t - W, t]$ is a packet that *may* become a minimum packet over a future window of time, without making any assumption on the future traffic. The list containing all future possible minima in the window $[t - W, t]$ is shortly called the LFPM. In particular, the minimum-packet of the window $[t - W, t]$ is in the LFPM Observe that, for two packets $< H_k, t_k >$ and $< H_{k'}, t_{k'} >$ over $[t - W, t]$, the packet $< H_k, t_k >$ has no chance of becoming a minimum-packet in the future if $t_k < t_{k'}$ and $H_k \ge H_{k'}$. Indeed, in the future, any window containting $< H_k, t_k >$ will also contain $< H_{k'}, t_{k'} >$, so that $< H_{k'}, t_{k'} >$ will prevent $< H_k, t_k >$ from being the minimum packet. Thus an FPM $< H_k, t_k >$ has to be such that no later arrived packet $< t_{k'}, H_{k'} >$ verifies $H_{k'} \le H_k$; i.e., an FPM *has to be* a strict minimum record of the list of packets of the window taken in reverse chronological order. Conversely, if a packet $< H_k, t_k >$ is such a reverse-time strict minimum record over $[t - W, t]$ and if the traffic is such that no packet

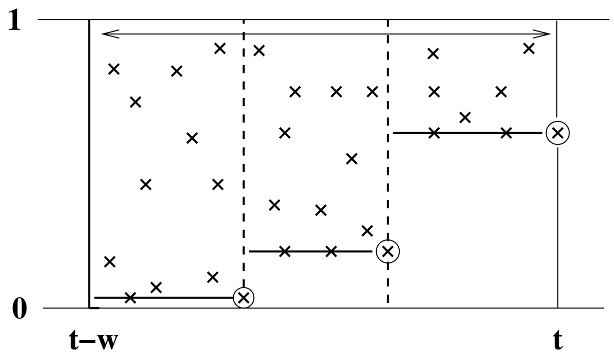

Figure 2: LFPM and minimum records.

```
while(true) do
  1) Read packet (H_k, t_k).
  2) Delete the packets (H_i, t_i) of L
     with t_i < t_k − W.
  3) Delete the packets (H_i, t_i) of L
     with H_i ≥ H_k.
  4) Add (H_k, t_k) at the end of L.
end do
```

Figure 3: Loop maintaining the LFPM.

arrives anymore after time t, then the packet $< H_k, t_k >$ will be the minimum packet at time $t_k + W$. As a consequence, we have the following characterization of the LFPM, see Figure 2:

Fact 3.1. *The LFPM contains the strict minimum records of the list of packets over* $[t - W, t]$ *taken in reverse chronological order.*

In particular, the elements of the LFPM are strictly increasing from left to right, see Figure 2.

Fact 3.2. *If the LFPM can be maintained for a sliding window of length* W*, then at any time* t *and for any* $w \leq W$*, the minimum packet over the window* $[t - w, t]$ *can be extracted from the LFPM: it is the oldest packet of the LFPM whose timestamp is larger than* $t - w$*.*

Fact 3.1 and Fact 3.2 make it possible to obtain a simple internal loop to maintain the LFPM over a sliding window of length W. The internal loop is given in Figure 3: when a packet arrives, the packets that are outdated are deleted in Step 2, and the packets that cease to be future possible minima (due to the arrival of the new packet) are deleted in Step 3. As a consequence, it is easily seen that the following invariant is maintained at each packet arrival:

Fact 3.3. *The list maintained by the internal loop of Figure 3 is the LFPM.*

Finally, Fact 3.2 and Fact 3.3 ensure that our internal loop provides a solution to the problem of maintaining the minimum over a sliding window, as formulated a the beginning of Section 3.1. In addition, our solution is optimal in the sense that, to maintain the minimum without making any assumption on the future traffic, one has to have all the FPM's of $[t-W, t]$ stored at any time t. Indeed, if one of the FPM $< H_k, t_k >$ is missing at a time t and if the traffic is empty after time t, then $< H_k, t_k >$ will be the unique packet realizing the minimum shortly before time $t_k + W$.

The very nice result which we will prove in Section 4 is that the size of the list is surprisingly short: at any time t, it is of logarithmic order in the number of distinct values over $[t - W, t]$. In addition, the length of the LFPM does not fluctuate much over a long period of time.

3.2 The algorithm Sliding MinCount. The algorithm Sliding MinCount, given in Figure 4, simply combines the method to maintain the minimum of a data stream over a sliding window with the stochastic averaging process used by MinCount. This means that we use $m = 2^b$ buckets and split the datastream into m datastreams, depending on the first b bits of the hashed value of a packet. For each bucket, we keep the minimum over a sliding window of length W by maintaining the LFPM associated to the bucket, thanks to the internal loop of Figure 3. Then, when there is a request at a time t asking for an estimate of the number of distinct flows in the last w units of time (with $w \leq W$), we extract the minimum over $[t - w, t]$ from the LFPM of each bucket. We do it in the way indicated by Fact 3.2, i.e., by taking the oldest packet whose timestamp is larger than $t - w$ in the LFPM of each bucket. From these minima, the same estimate as the one of MinCount is computed. Thus, the estimate returned by Sliding MinCount is *exactly the same* as the one that would have been obtained by running *a priori* the MinCount algorithm from $t-w$ to t. Hence it has exactly the same accuracy as MinCount, as is recapitulated in Theorem 3.1.

Theorem 3.1. (Accuracy of Sliding MinCount) *For any request asking for the number of distinct flows over the last* w *units of time (with* $w \leq W$*), the estimate* ξ *returned by* Sliding MinCount *has a relative accuracy* $\sqrt{\mathbb{V}(\xi)}/\mathbb{E}(\xi)$ *about*

$$\approx \frac{1.3}{\sqrt{m}}.$$

INTERNAL LOOP:
$L_1 \leftarrow \emptyset \ldots L_{2^b} \leftarrow \emptyset$ $[m = 2^b]$
while(true) do
1) Read packet (H_k, t_k).
2) $j \leftarrow$ first b bits of H_k
3) Truncate H_k of its first b bits
4) Delete the packets (H_i, t_i) of L_j with $t_i < t_k - W$.
5) Delete the packets (H_i, t_i) of L_j with $H_i \geq H_k$.
6) Add (H_k, t_k) at the end of L_j.
end do

REQUEST:
Input: "what is approximately the number of distinct flows in the last w units of time ?"
Answer:
for j from 1 to m do
$M^{(j)} \leftarrow$ leftmost hashed value of L_j with timestamp larger than $t - w$
end for
$\xi \leftarrow m\Gamma(2 - \frac{1}{m})^{-m} \exp\left(-\frac{1}{m}\sum_{j=1}^{m} \ln(M^{(j)})\right)$
return ξ

Figure 4: The algorithm SLIDING MINCOUNT

For instance, taking $m = 2^{10}$ will provide estimates with a precision of order 4 percents.

3.3 Distributivity. An important feature of SLIDING MINCOUNT (as well as the previous "static" algorithms [FM83, DF03, Gir05]) is to be very suitable for parallelization. If the data stream to be measured is split among several routers, each router maintains its own LFPMs from the packets it receives. Then, to answer a request on the number of distinct flows in the whole traffic, each router extracts the minimal hashed values from its own LFPMs and sends them to a central calculator that computes the estimate. Such a situation is encountered very often in practice. For instance, in Backbone Networks, the quantity of traffic is too huge to be concentrated on a single machine, so that the packets arriving to a Point of Presence are distributed among several routers.

3.4 Counting the number of packets with a sliding window. Giving the number of packets in a fixed set is straightforward, using a simple counter. However, answering the same request over a sliding window is more complex, as one should keep in memory the time of arrival of the packets. With only a *slight change* of SLIDING MINCOUNT, we can estimate the number of packets of a data stream over a sliding window: instead of hashing the FlowIDs, we hash the timestamps of the packets. As each timestamp is unique, the modified algorithm now estimates the number of packets over a sliding window.

4 Analysis of the algorithm

As the accuracy of SLIDING MINCOUNT is the same as MINCOUNT, the main point that remains to be analyzed is the memory. The memory used by SLIDING MINCOUNT consists of the m LFPMs (one for each bucket). We first study the probability distribution of the memory at a fixed time t. Then we analyze the evolution of the size of the LFPM over the time. Indeed, it comes out handy for the implementation to store the LFPMs in arrays instead of lists. To decide which size of array is sufficient to avoid overflooding, it is thus crucial to know the distribution of the maximum of the size of the LFPM over a long period (e.g. one week or even one year). The proofs are given in the appendix.

Study of the memory at a fixed time. We start the analysis by investigating the size of a single LFPM.

LEMMA 4.1. (THE REPETITIONS HAVE NO EFFECT) *Over a given window $[t - W, t]$, the LFPM is the same as the one built by applying the internal loop of Figure 3 only on the latest arrived packet of each flow.*

Proof. All the packets of a flow are hashed to the same value. Thus, a packet that is not the latest in his connection can not be a strict minimum record for the list of packets taken in reverse chronological.

LEMMA 4.2. (FORMULATION ON PERMUTATIONS) *At a given time t, let n be the number of distinct flows over the current window $[t-W, t]$. Then, the probability distribution of the size of the LFPM is the same as the distribution of the number of minimum records in a random permutation of size n.*

Proof. It follows from corollary 4.1 and from the perfect randomness assumed for the hash function that the LFPM is the same as the one built with n i.i.d. random variables uniform over $[0, 1]$.

LEMMA 4.3. (SIZE OF A SINGLE LFPM) *Let L_n be the random variable giving the size of a single LFPM at a time t, knowing that the number of distinct flows over $[t - W, t]$ is equal to n. Then the distribution of L_n satisfies*

$$\mathbb{E}[L_n] = H_n, \text{ with } H_n = \sum_{k=1}^{n} \frac{1}{k} \underset{n\to\infty}{\sim} \ln(n). \tag{4.2}$$

$$\mathbb{V}[L_n] = H_n - \sum_{k=1}^{n} \frac{1}{k^2} \underset{n\to\infty}{\sim} \ln(n), \tag{4.3}$$

Asymptotically in n, L_n is a gaussian law of expectation $\ln n$ *and variance* $\ln n$*, i.e.,*

$$L_n \underset{n\to\infty}{=} \ln(n) + \sqrt{\ln(n)}Z, \text{ with } Z \in \mathcal{N}(0,1). \tag{4.4}$$

Proof. Lemma 4.2 ensures that the distribution of L_n is the same as the distribution of the number X_n of minimum records in a random permutation of size n. The analysis of X_n is a classical problem in combinatorics, first studied by Goncharov, see [FS, II.6,IX.5]: a concise solution is obtained from the closed form of the characteristic function $f(u) = \sum_k \mathbb{P}(X_n = k)u^k$ of X_n, $f(u) = \frac{1}{n!}u(u+1)\dots(u+n-1)$. Then the expectation and variance are extracted using $\mathbb{E}(X_n) = f'(1)$ and $\mathbb{V}(X_n) = f''(1) + f'(1) - f'(1)^2$. The asymptotic gaussian distribution is obtained by rescaling X_n around its mean and contracted by a factor $\sqrt{\mathbb{V}(X_n)}$, and proving that the rescaled characteristic function converges to the characteristic function of a standard gaussian law.

From the study of the size of a single LFPM, we can analyze the memory required by Sliding MinCount, which is the sum of the sizes of the LFPMs of the m buckets.

Lemma 4.4. (Total size of the m LFPMs) *Let L_n^{tot} be the random variable giving, at time t, the sums of the sizes of the m LFPMss (one for each bucket), knowing that the number of distinct flows over $[t-W,t]$ is equal to n. Then the distribution of L_n^{tot} satisfies*

$$\mathbb{E}[L_n^{\text{tot}}] \underset{n\to\infty}{\sim} mH_{\lfloor n/m \rfloor} \underset{n\to\infty}{\sim} m\ln(n/m), \tag{4.5}$$

$$\mathbb{V}[L_n^{\text{tot}}] \underset{n\to\infty}{\sim} m\ln(n/m), \tag{4.6}$$

Asymptotically in n, L_n^{tot} is a gaussian law of expectation $m\ln(n/m)$ and variance $m\ln(n/m)$, i.e.,

$$L_n^{\text{tot}} \underset{n\to\infty}{=} m\ln(n/m) + \sqrt{m\ln(n/m)}Z, \text{ with } Z \in \mathcal{N}(0,1). \tag{4.7}$$

Proof. Asymptotically in n, the number of flows falling in each of the m buckets is equal to n/m, up to fluctuations of order $\sqrt{n/m}$. Thus, the calculations are still valid (but much more simple) by stating that, at the first order, the number of distinct flows falling in each bucket is equal to $\lfloor n/m \rfloor$. In this equirepartition calculation model, L_n^{tot} is the sum of m i.i.d. random variables that are each the size of an LFPM built from $\lfloor n/m \rfloor$ distinct values. According to Lemma 4.3, the distribution of the size of such an LFPM is, asymptotically in n, equal to $\ln(n/m) + \sqrt{\ln(n/m)}Z$, where Z is a Gaussian normal law. Thus the distribution of L^{tot} is, asymptotically in n, equal to $m\ln(n/m) + \sqrt{\ln(n/m)}\sum_{i=1}^{m} Z_i$, where the Z_i are m i.i.d. normal gaussian laws. Finally, it is well known that the sum of m i.i.d. normal gaussian laws has the same distribution as $\sqrt{m}Z$, where Z is a normal gaussian law.

Proposition 4.1. (Memory at a fixed time) *Let $n_{\max}$ be an upper bound on the number of distinct flows to be observed over a window of length W. Then, at time t, the memory used by* Sliding MinCount *is of order $m\ln(n/m)\,(\ln_2(n_{\max}/m) + \ln_2(W))$, where $m = 2^b$ is the number of buckets used by* Sliding MinCount *and n is the number of distinct flows in the window $[t-W,t]$.*

Proof. First, Lemma 4.4 ensures that the sum of the lengths of the LFPMs is of order $m\ln(n/m)$. Each element of an LFPM is a packet consisting of a hashed value truncated of its first b bits. As we have seen in Section 2, the hash function has to map a FlowID to a bit string of length a little more than $\ln_2(n_{\max})$ in order to avoid collisions. Moreover, as the packets are outdated after W units of time, the timestamps can be stored on $\ln_2(W)$ bits.

Study of the maximal required memory over a long running time.

In this section, we investigate on the *maximum* of the total size L^{tot} of the LFPMs over a long running time of Sliding MinCount. Indeed, even if L^{tot} is small at a fixed time —of order $m\ln(n/m)$— it could happen that, over a very long running time, L^{tot} has a high peak at some time. As we prove next, this situation does not arise. What happens is that L^{tot} fluctuates rapidly from one packet to a few packets later, but these fluctuations are small and *remain small* even over a very long running time of Sliding MinCount.

Theorem 4.1. (Memory over whole execution) *Let $n_{\max}$ be an upper bound on the number of distinct flows to be observed over a window of length W. Consider a long but finite datastream, with S packets ($S >> 1$), on which the internal loop of* Sliding MinCount *is applied. Then the maximal value taken by the sum L^{tot} of the sizes of the m LFPMs is of order at most*

$$m\ln(n_{\max}/m) + \sqrt{m\ln(n_{\max}/m)2\ln(S)},$$

i.e., with very high probability, it will never be larger than this value up to a few units. As a consequence, the *maximal memory used by* Sliding MinCount *over its execution is of order at most*

$$\left(m\ln(n_m) + \sqrt{m\ln(n_m)2\ln(S)}\right)(\ln_2(n_m) + \ln_2(W)),$$

where $n_m := n_{\max}/m$.

Proof. Proposition 4.4 ensures that, at each arrival of a packet $< H_k, t_k >$, the distribution of the sum of the sizes of the m LFPMs is close to the distribution of $\ln(n/m) + \sqrt{m\ln(n/m)}Z$, where n is the number of distinct flows in $[t_k - W, t_k]$ and Z is a normal law. As there are S packets in the data streams, bounding the maximum of L^{tot} over the data stream amounts to bounding the maximum of S normal laws. This last task is readily carried out, based on the fact that the tail of a normal law decays very fast: for $x \geq 1$, $\mathbb{P}(Z \geq x) \leq e^{-x^2/2}$. Thus, for S normal laws $N_1, \ldots, N_S$, we have the simple formula

$$\begin{aligned}\mathbb{P}(\max(Z_1,\ldots,Z_S) \geq x) &= \mathbb{P}((Z_1 \geq x) \cup \ldots \cup (Z_S \geq x)) \\ &\leq S \cdot \mathbb{P}(Z \geq x) \leq Se^{-x^2/2}.\end{aligned}$$

As a consequence, the probability that the maximum is larger than x decays very fast toward 0 when x gets larger than $\sqrt{2\ln(S)}$.

As we see, the maximal fluctuation over a whole data stream with S packets is only $\sqrt{2\ln S}$ times the typical fluctuation at fixed time, of order $\sqrt{m\ln(n/m)}$. As the fonction $S \to \sqrt{2\ln S}$ grows *extremely slowly*, the maximal fluctuation of the total size of the LFPMs will be small even for *very long running times*. For instance, a running time of several years instead of one hour will have little effect on the maximal auxiliary memory used by Sliding MinCount.

5 Validation and Experimentations on real traffic

Validation by simulations. At first, we validate Sliding MinCount with *ideal traffic*: the packets arrive at a constant rate and all belong to distinct connections. We simulate a data stream of 5 millions packets and ask the algorithm, every 100 read packets, for an estimate of the number of distinct flows seen in a window of 1 M packets. Figure 5 (left) shows these estimates and Figure 5 (right) the memory used by the algorithm (corresponding to the number of elements in all LFPMs). The algorithm estimates the number of connections with no bias and the experimental standard error is within the expected value of 4% (for $m = 2^{10}$) with few excursions to 6%. Same remark for the size of the LFPMs. Observe that the evolution of the memory is burstier than the evolution of the estimate. Indeed, the LFPMs change at each packet arrival while the estimate is modified only when the minimal hashed value changes. Results of the tables of Figure 6 correspond to simulations for different values of m (the number of buckets) with two different windows (100,000 packets for the left figure and 1M packets for the right figure). They show good agreement between the algorithm behaviour and what is predicted by the analysis. L_{tot}, the average of the total size of the LFPMs over the execution, is very close to its expected value $mH_{\lfloor n/m \rfloor}$. L_{max},the maximal value of the total size of the LFPMs over the execution, is close to the mean size, validating the fact that a long running time has little effect on the maximal auxiliary memory used by Sliding MinCount. We also see that the experimental standard error (Stderr) is close to its expected value ($\approx 1.3/\sqrt{m}$). It decreases by a factor of 2 when we use 4 times more memory, as predicted by the analysis. The algorithm succeeds in estimating the number of connections within 4% percents using only $64kB$ (the LFPMs have less than 8,000 elements with a 32 bit timestamp and a 32 bit hashed value).

Traffic Analysis. We captured traces of packets arriving at a gate router of the campus of INRIA in Rocquencourt. This traffic is an aggregate of the activity of 400 computers and has a typical rate of 150 millions packets per day. The analysis of two traces is presented here: Trace A corresponds to the traffic of monday April 3^{rd} and Trace B to the one of sunday April 2^{nd}. The algorithm is very fast: the one-day traces are treated in three minutes with a CPU of 1 GHz ($1M$ packets per second) even with a non optimized implementation. Hence, it would face no problem for real-time monitoring of such traffics. Figure 7 shows the estimates given every second by Sliding MinCount for two different windows: one hour (bottom) and one minute (top). With the sliding window of one hour, we see that the traffic of monday (left) has a shape very different from the one of sunday (right). We are able to distinguish peaks of activity corresponding to backups of data that take place during the night (at 2 am and 6:30 am for Trace A) as well as the working hours. With the sliding window of one minute, we see that, in fact, these evolutions are mostly caused by very sharp peaks, in the scale of the minute.

Figure 8 gives the comparison between the number of distinct connections (Figure 8 left) and the number of packets (Figure 8 right) over a sliding window of one hour using Trace A, see Section 3.4 for the methodology to estimate the number of packets. The number of packets for one hour ranges between 3 millions and 12 millions and in average a connection has 30 packets. Note that the number of connections may increase a lot even when the number of packets remains almost constant (for example, observe the traffic between 20,000s

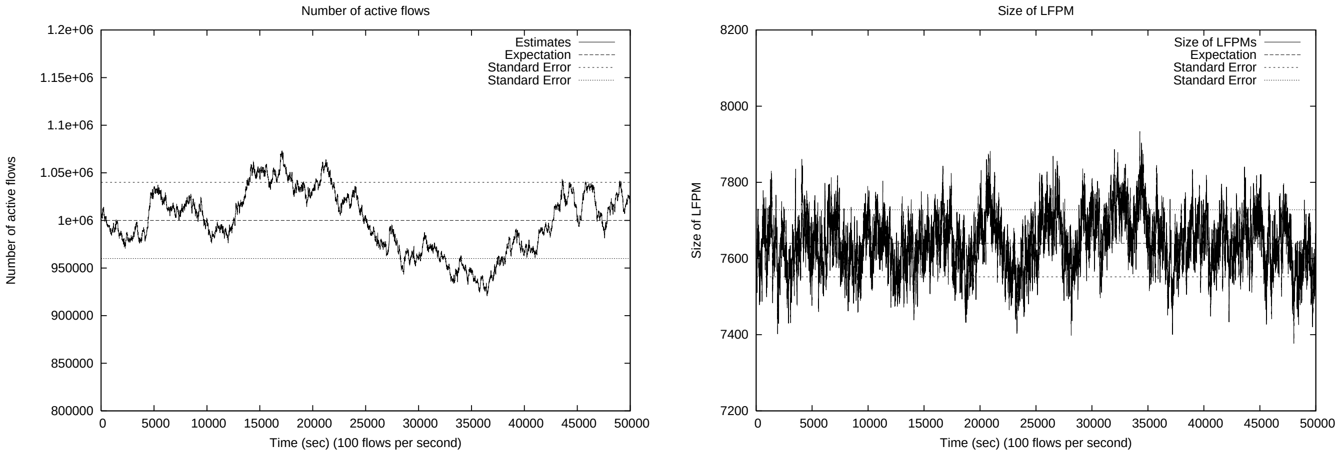

Figure 5: Behaviour of SLIDING MINCOUNT during a simulation on an ideal traffic of 5 million elements, with a window of one million elements. The left figure shows the estimates given every second by SLIDING MINCOUNT. The right figure gives the memory used by the algorithm during the execution ($m = 2^{10}$).

$\log_2 m$	4	6	8	10
$mH_{\lfloor n/m \rfloor}$	149	508	1676	5288
$Ltot$	148.9	507	1676	5293
$Lmax$	199	605	1815	5503
$1.3/\sqrt{m}$	0.32	0.16	0.08	0.04
$StdErr$	0.32	0.14	0.09	0.03

$\log_2 m$	4	6	8	10
$mH_{\lfloor n/m \rfloor}$	186	655	2265	7641
$Ltot$	185.9	653.5	2264	7648
$Lmax$	245	752	2430	7933
$1.3/\sqrt{m}$	0.35	0.16	0.08	0.04
$StdErr$	0.35	0.15	0.07	0.04

Figure 6: The average memory and precision of SLIDING MINCOUNT for a data stream of 5 millions packets of ideal traffic, with a window of 100,000 connections (right) and a window of one million connections (left) for different values of the number m of buckets.

and 28,000s). This confirms the fact that counting the number of distinct connections gives more informations than the number of packets. As stated in the introduction, this has important applications in network monitoring (e.g., detection of Denial of Service attacks).

ACKNOWLEDGEMENTS The authors would like to thank Philippe Flajolet for interesting and helpful discussions, as well as Denis Joiret and André Balsa for their precious help to obtain traffic traces.

References

[DF03] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *Proceedings of the European Symposium on Algorithms*, 2003.

[DGIM02] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

[DM02] Mayur Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 323–334, London, UK, 2002. Springer-Verlag.

[EVF03] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. In *Technical Report CS2003-0738*, UCSE, Mars 2003.

[FM83] P. Flajolet and P. N. Martin. Probabilistic counting. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 76–82. IEEE Computer Society Press, 1983.

[FS] P. Flajolet and R. Sedgewick. Analytic combinatorics. Available electronically. Preliminary version of the forthcoming book.

[Gir05] F. Giroire. Order Statistics and Estimating Cardinalities of Massive Datasets. In *Proceedings of Discrete Mathematics and Theoretical Computer Science*, 2005.

[Knu98] D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching, 2nd edition*, volume 3. Addison-Wesley, 1998.

[Plo00] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *USENIX LISA*, pages 305–317, 2000.

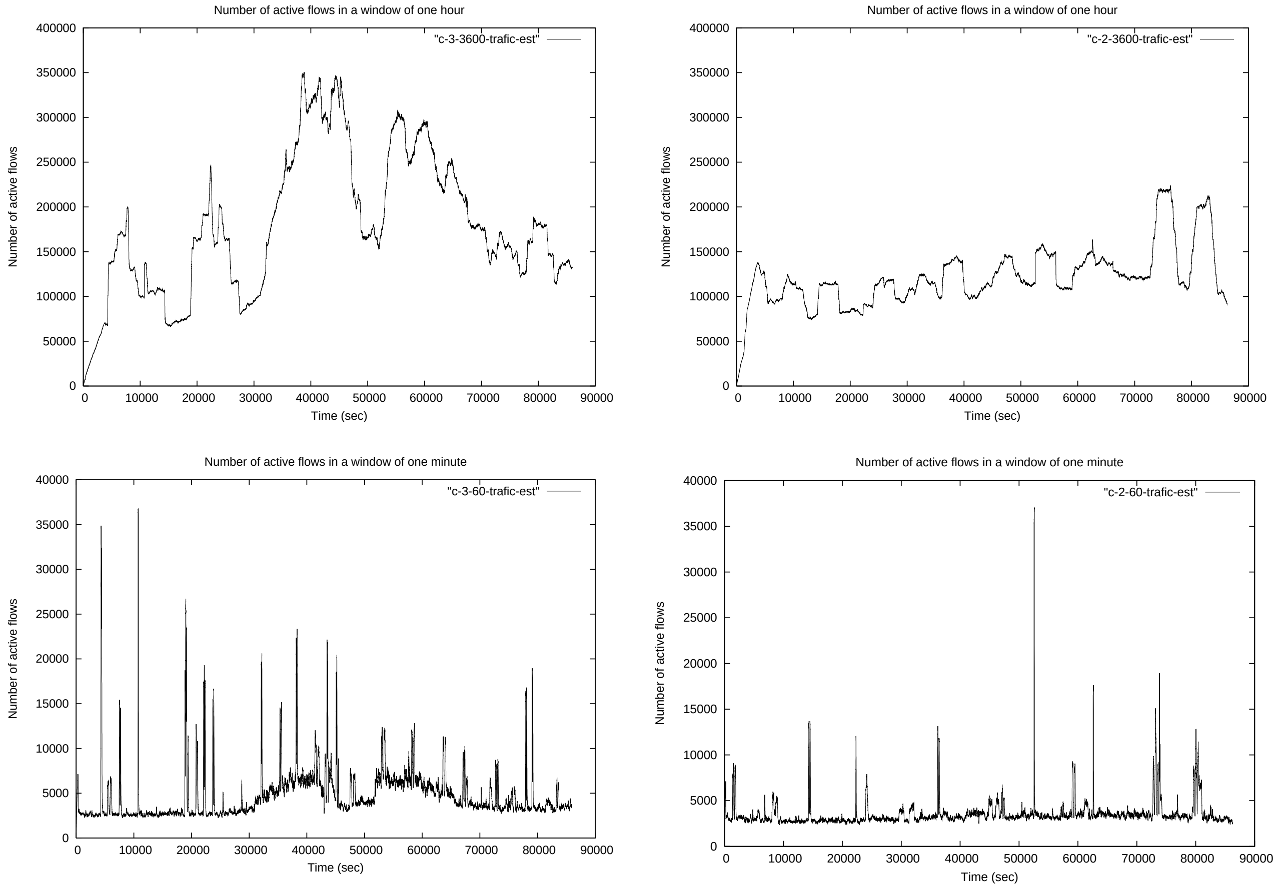

Figure 7: Estimates of the number of distinct flows given every second by SLIDING MINCOUNT for Trace A (Monday, April 3^{rd}) (Left) and for Trace B (Sunday, April 2^{nd}) (Right) for a sliding window of one hour (top) and a sliding window of one minute (bottom).

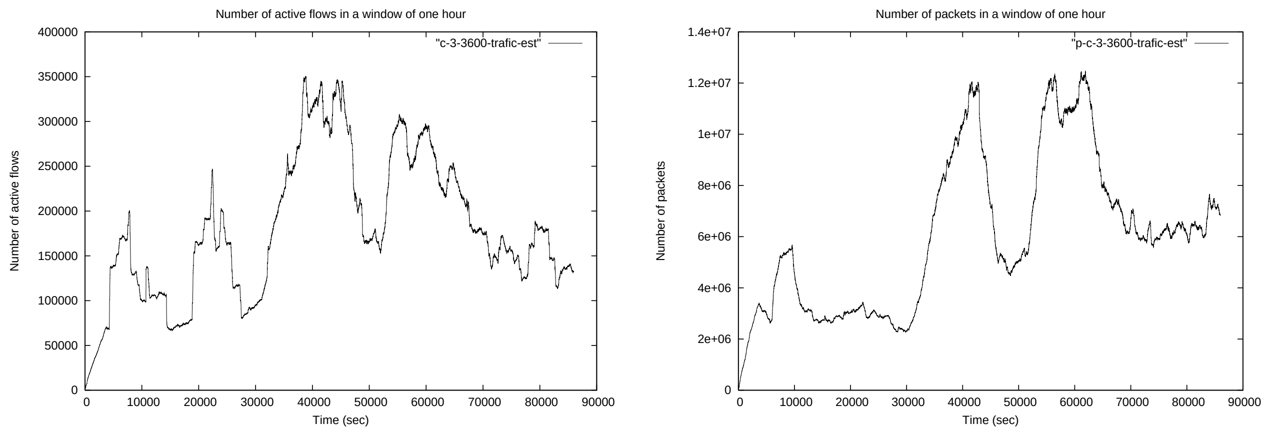

Figure 8: Comparison between the number of distinct connections (Left) and the number of packets (Right) for Trace A, with a window of one hour.