# A Cost-effective and Lightweight Membership Assessment for Large-scale Data Stream

Yann Busnel, Noël Gillet

# A Cost-effective and Lightweight Membership Assessment for Large-scale Data Stream

Yann Busnel
*IMT Atlantique, IRISA, UMR CNRS 6074*
F-35700 Rennes, France
yann.busnel@imt-atlantique.fr

Noël Gillet
*IMT Atlantique, IRISA, UMR CNRS 6074*
F-35700 Rennes, France
noel.gillet@imt-atlantique.fr

***Abstract*—The membership primitive is a classic and fundamental problem in many use cases. In networking for instance, it is useful to check if a given IP address belongs to a black list or not, in order to allow access to a given server. This has also become a key issue in very large-scale distributed systems, or in massive databases. Formally, from a subset belonging to a very large universe, the problem consists in answering the question "Given any element of the universe, does it belong to a given subset?". Since the access of a perfect oracle answering the question is commonly admitted to be very costly, it is necessary to provide efficient and inexpensive techniques in the context where the elements arrive continuously in a data stream (for example, in network metrology, log analysis, continuous queries in massive databases, *etc.*). In this paper, we propose a simple but efficient solution to answer membership queries based on a couple of Bloom filters. In a nutshell, the idea is to contact the oracle only if an item is seen for the first time. We use a classical bloom filter to remember an item occurrence. For the next occurrences, we answer the membership query using a second bloom filter, which is dynamically populated only when the database is queried. We provide theoretical bounds on the false positive and negative probabilities and we illustrate through extensive simulations the efficiency of our solution, in comparison with standard solutions such as a classic bloom filter.**

***Index Terms*—data stream, membership query, bloom filter**

## I. Introduction

A basic but fundamental task in computer science is to decide efficiently if a given element belongs to a set or not. In network monitoring for instance, the set can represent a blacklist of IP addresses that are forbidden to access a given server. In real systems, it is required to be able to test the authorization with the lowest delay. In medical informatics, the detection of unknown adverse drug effects (ADE) from electronic health records can be done by checking if a given ADE belong to an existing database of known ADEs. The dynamic aspect of the health data production requires the use of efficient monitoring solutions.

More generally, we want to be able to test the membership to a given set $\mathcal{B}$ of items received as a stream. We consider that we have access to an *oracle* $O_{\mathcal{B}}$, which is able to answer any membership query in an exact manner. An easy solution consists in querying $O_{\mathcal{B}}$ for each item of the stream. However, querying $O_{\mathcal{B}}$ is typically very costly and resource consuming, because it is stored remotely or it is a very large database, implying a high latency.

To reduce the querying cost, a classic static solution is to summarize this set beforehand, and to query the perfect oracle only in cases where the summary is not precise enough. The use of a Bloom filter [1] is a well-known solution to compactly represent a set of items. Without going into details, the idea of a bloom filter is to map every item of $\mathcal{B}$ to several locations in a bit array using hash functions. Deciding if a given item belongs to $\mathcal{B}$ is done by checking if the corresponding bits in the array are set to 1. A recent survey on bloom filters and their many variations can be found in [2]. Despite a negative answer gives the guarantee that $u_i \notin \mathcal{B}$, there is a non-zero probability that the Bloom filter answers positively for $u_i \notin \mathcal{B}$, leading to a *false positive*. If we want to avoid such false positive, we can query $O_{\mathcal{B}}$ only if a positive answer occurs, instead of accessing it for each item.

However, such simple approach presents several drawbacks: (1) the communication cost remains important (2) we must load the whole database beforehand, even if only a subset of $\mathcal{B}$ appears in the stream. This could lead to a waste of memory in case of a large database.

A lot of efforts has been done to propose solutions that minimize the *false positive probability*, that is the probability that an item $u \notin \mathcal{B}$ is considered as belonging to $\mathcal{B}$. For example, several works propose to introduce false negative to reduce the false positive probability [3], [4], but they also require to load $\mathcal{B}$ entirely. Existing works propose filters that deal with dynamic sets [5] and can be use to represent only the items in $\mathcal{B}$ seen so far in the stream. However such a naive solution leads to a high communication cost since we have to check for each received item which is not in the filter if it belongs to $\mathcal{B}$ or not.

*a) Contributions:* In this paper, we propose an *approximate oracle* which guarantees a very low false positive probability while keeping a low memory consumption. The idea is to *dynamically* add discovered items in $\mathcal{B}$ to our oracle, avoiding the need to load the whole set. This is done by querying the oracle only if an item is seen for the first time. This idea is especially interesting when the number of distinct items is low.

Unhappily, our solution as in [3], [4] may produce false negatives, that is items that belong to $\mathcal{B}$ for which the oracle

This work was partially funded by the French ANR project INSHARE (ANR-15-CE19-0024).

answers negatively. The false negative probability can be parametrized as a trade-off with the false positive probability. However, having no false negative is not a strong requirement in many use cases. In ADE detection use case for instance, a false negative item will imply the monitoring of an already known ADE and thus a small waste of resources. However a false positive item corresponds to an unknown ADE, with a potential sanitary risk, and we definitively want to avoid such error. Other applications in which reducing the number of false positives while introducing false negatives is beneficial are presented in [3], such that file distribution in peer-to-peer systems or resource routing.

*b) Road map:* After having introduced some backgrounds (Section Section II) and discussed about related works (Section III), we present in Section IV our solution called *Dynamic Appending Bloom Filter*. In Section V, we give theoretical bounds on the false positive and negative probabilities. In Section VI, we discuss time and communication cost. Finally, we present in Section VII extensive simulations to evaluate our proposal on different distributions. We also evaluate a variation of our solution inspired from the sliding window streaming model.

## II. Preliminaries

### A. Model

The universe $\mathcal{U}$ contains $U$ items and we denote by $B$ the size of the set $\mathcal{B} \subseteq \mathcal{U}$. We consider that a stream $\mathcal{S}$ of $m$ items is received. Answering a *membership query* consists, for any item $u$ of $\mathcal{S}$, to determine if $u$ belongs to $\mathcal{B}$.

Let $\mathcal{S}_t = u_1, \ldots, u_t$ be the items received at time $t$, thus $\mathcal{S}_m = \mathcal{S}$. The set of distinct items in $\mathcal{S}_t$ is denoted by $D_t$ and we denote by $d_t$ the cardinality of $D_t$. Similarly, we define the set $\hat{D}_t = \mathcal{S}_t \cap \mathcal{B}$ of cardinality $\hat{d}_t$ as the set of distinct items belonging to $\mathcal{B}$ in $\mathcal{S}_t$.

### B. Membership oracle

A membership oracle $O$ is a compact representation of a set $\mathcal{B} \subseteq \mathcal{U}$ with access to a primitive $O.lookup(u)$, returning a boolean value for any item $u \in \mathcal{U}$. If the oracle is *perfect*, then $O.lookup(u)$ return $true$ if and only if $u \in \mathcal{B}$, for every $u \in \mathcal{U}$. Otherwise the oracle is *approximate*. A false positive is any item $u \in \mathcal{U}$ such that $u \notin \mathcal{B}$ but $O.lookup(u)$ is true. A false negative is similarly defined as an item $u \in \mathcal{U}$ such that $O.lookup(u)$ is false for item $u \in \mathcal{B}$. Hence the false positive (resp. negative) probability of $O$ is the probability that an item is classified as a false positive (resp. negative).

An important notion is also the *false positive rate* with is defined as the ratio of the number of items which are false positive, to the total number of items of $\mathcal{S}$ which do not belong to $\mathcal{B}$. Intuitively, if we have a false positive probability of $f$, then on average $fm$ items will be wrongly detected as positives. However, if an item $u$ is a false positive and occurs many times in $\mathcal{S}$, then the false positive rate can drop significantly. Hence having a strong guarantee on the false positive rate is desirable. Similarly, we introduce the notion of *false negative rate*, which corresponds to the ratio of the number of false negative to the number of item belonging to $\mathcal{B}$ in $\mathcal{S}$.

### C. Bloom Filter

A bloom filter $BF$ [1] is an approximate oracle using a bit array $A_{BF}$ of size $n$ and $k$ independent random hash functions $\{h_1, \ldots, h_k\}$, mapping items from $\mathcal{U}$ to an index in $[0, n-1]$ chosen uniformly at random. For each inserted item $u \in \mathcal{B}$ and every hash function $h_i$, $A_{BF}[h_i(u)]$ is set to 1. The membership primitive $BF.lookup(u)$ is easily implemented by returning $true$ if $A_{BF}[h_i(u)] = 1$ for every $i \in [1, k]$ and $false$ otherwise.

For the purpose of the theoretical proofs, we introduce the notion of *masks*. The *mask* $H_{BF}(u)$ of item $u$ is defined as a binary array of size $n$ such that $H_{BF}(u)[h_i(u)] = 1$ for every $i$, 0 otherwise. The membership query can be solved by doing a bitwise AND operation between $H_{BF}(u)$ and $A_{BF}$ and by checking if the resulting array $H_{BF}(u) \wedge A_{BF} = H_{BF}(u)$. Let $H_{BF}(X) = \bigvee_{x \in X} H_{BF}(x)$ be the result of a bitwise OR operation over all items in $X$. With this in mind, we remark that $u \notin \mathcal{B}$ is a false positive if there exists a set $X \subseteq \mathcal{B}$ such that $H_{BF}(u) \wedge H_{BF}(X) = H_{BF}(u)$. We say that $u$ *collides* with $X$ and we denote this event by $u \sim_{BF} X$.

It is clear that the false negative probability is equal to 0, since for any item $u \in S(O)$ the bits are set to 1 implying that $BF.lookup(u)$ returns true. However the false positive probability is strictly positive if at least one item has been inserted in the filter. Let $p(n, k, d)$ be the false positive probability of such a bloom filter when $d \geq 1$ items have been inserted. Since the hash functions are independent and uniform, we can easily deduce the following:

$$p(n, k, d) = \left(1 - \frac{n_0(d)}{n}\right)^k \tag{1}$$

where $n_0(d)$ denotes the number of 0-bits after the insertions of $d$ items. We have $n_0(0) = n$ and clearly $n_0(d)$ is a decreasing function. Hence $p(n, k, d)$ is an increasing function in $d$, for a given $n$ and $k$. Suppose now that $d$ and $k$ are given. It has been proven in [6] that $p(n, k, d) \leq (1 - (1 - 1/n)^{dk + (k-1)/2})^k$, which confirms the intuition that the false positive probability increases when $n$ decreases.

## III. Related works

### A. Trade-off between false positives and negatives

Solutions already provide a tradeoff between false positive and false negative probabilities [3], [4]. The retouched bloom filter proposed in [3] is a classical bloom filter with the additional feature that some bits can be *cleared*, that is set to 0. By doing this, the false positive probability naturally decreases but at the cost of introducing false negative. To select the bits to clear, the authors suppose that the set of encountered false positive items is known beforehand and they focus the clearing on these items. This assumption is too strong in the context of data streams, where the item distribution is not known a priori. The generalized bloom filter [4] is also very similar to the classical bloom filter. There are $k_1$ hash

functions $h_1 \dots h_{k_1}$ and $k_2$ hash functions $g_1 \dots g_{k_2}$. When item $u$ is inserted, every bit at position $h_i(u)$ is set to 1 and every bit at position $g_i(u)$ is set to 0. If $h_i(u) = g_j(u)$ then the bit is set to 0. This solution introduces false negative but the authors show that the false positive probability depends only on the number of hash functions. However applying this solution in our context would imply the load of the whole database, implying a uselessly large memory consumption.

### B. Dynamic oracles

The dynamic bloom filter DBF [5] was introduced to represent dynamic sets. At the begin, there is only one BF into which items are inserted. When the occupancy of the BF overreaches a predefined threshold, then a new BF is created in order to receive new items, and so on. This solution is really interesting when ones do not have any idea about the size of the set. However, the use of a simple bloom filter presents better performances if the size of the set is known. Moreover, it has been shown in [7] that the false positive probability grows linearly with the size of the dynamic set, for a given amount of memory.

In [7], the authors present an oracle to represent a dynamic set with a bounded false positive probability $f$ given as input, independently of the size of the set. This impressive result comes with an important memory cost in memory. Indeed, the total number of bits can theoretically reach $O\left(-\frac{kU}{\ln(1-f^{1/k})}\right)$ if the item distribution is uniform. However extensive simulations (not shown in the extended abstract) tends to demonstrate that in practice the solution consumes only slightly more space than DBF, since the distribution in real world data set are more skewed.

### C. Adaptivity

Most of the bloom filters variations focus on minimizing the false positive probability or finding a good trade-off between false positive and false negative. However, few works deal with the minimization of the false positive rate. In [8], the authors present the adaptive cuckoo filter, which learns from the answers of the perfect oracle and removes false positive items from the data-structure. Simulations over real datasets show a reduction on the false positive rate in practice. In [9], the authors use similar techniques to build an adaptive oracle called the *broom filters* with strong theoretical guarantees. More precisely, they prove that the probability that an item is a false positive (even if $\mathcal{S}$ is chosen by an adversary learning from perfect oracle answers) is bounded by a constant $\epsilon < 1$. However, these works suppose that the perfect oracle is queried after each positive answers from the filters, which can not fit reasonably in our context.

## IV. Dynamic appending

### A. Intuition

Depending on the skewness of item distributions and the length of $\mathcal{S}$, the number of items that belong to $\mathcal{B}$ and appear in $\mathcal{S}$ can be very low. In such case, using a simple solution such a bloom filter containing the whole set $\mathcal{B}$ seems not optimal. Instead, a natural idea is to append a newly received item $u$ to the filter if $u$ belongs to $\mathcal{B}$. For the same amount of memory, this dynamic version of the bloom filter will naturally provide a better precision for the first items of $\mathcal{S}$, since the number of bits at zero is greater than the one with the naive solution. The counterpart is that this solution implies to query the perfect oracle $O_{\mathcal{B}}$ after each item reception, even if $u$ occurs several times in $\mathcal{S}$. To reduce the communication cost, a simple solution could be to remember which item has been seen so far and to query $O_{\mathcal{B}}$ only if a new distinct item is received. The communication cost is then reduced from $m$ to $d$ (in the worst case). The communication saving is even more important when the item distribution is skewed. However, remembering exactly which item has been seen could lead to slow solutions, with high memory cost.

We propose a membership oracle called the *Dynamic Appending Bloom Filter* (DABF). The idea of DABF is based on the use of a pair of Bloom filters, denoted by $\beta$ and $\beta'$, significantly smaller in size than the classic solution using a unique static bloom filter. The first bloom filter is used to memorize if an element has already been seen in the past, by gradually populating it online. The second bloom filter is used to determine if an already received element belongs to the considered subset. Thus, the memory cost becomes proportional to the number of distinct items in the stream that belong to the subset. The communication cost is proportional to the number of distinct items in the stream.

### B. Algorithm description

We denote by $n$ and $n'$ the sizes of the bit arrays of $\beta$ and $\beta'$ respectively and we use $S$ and $S'$ to denote their representative sets. In the following, we present the implementation of the primitive $DABF.lookup(u_i)$ when an item $u_i$ is received:

- If $\beta.lookup(u_i)$ return false, add $u_i$ to $\beta$ and send a membership query to $O_{\mathcal{B}}$.
  - If $O_{\mathcal{B}}.lookup(u_i)$, add $u_i$ to $\beta'$.
  - In any case, return $O_{\mathcal{B}}.lookup(u_i)$.
- If $\beta.lookup(u_i)$, return $\beta'.lookup(u_i)$

Since the filters are populated dynamically, we use notations $\beta_t$ and $\beta'$ to refer to their states after the reception and processing of item $u_t$. $S_t$ and $S'_t$ represent the set of items added respectively to $\beta$ and $\beta'$ at time $t$. Initially, $S_0$ and $S'_0$ are both empty. We define $DABF_t$ as the combination of $\beta_t$ and $\beta'_t$.

We do not suppose that both filters use the same set of hash functions but we suppose that every hash function is perfectly random (it draws an index uniformly at random from $[1, n]$), which is a classic assumption in the literature. We can also use 2-universal random hash functions instead, which are easily implementable and provide strong theoretical guarantees. We denote by $k$ and $k'$ the number of hash functions for $\beta$ and $\beta'$ respectively.

Moreover, the *false positive* (resp. *negative*) probability of DABF at time $t$ is denoted by $p_t$ (resp. $q_t$).

### C. Extension to the streaming model

We can naturally extend this algorithm to a streaming model in the following manner. Let us denotes by $F_0 = \langle \beta, \beta' \rangle$ and $F_1 = \langle \beta, \beta' \rangle$ two identical DABF's, initially empty. The idea is to fill both filters and to alternately flush them at the end of the window. Let $w$ be the number of items in a window and suppose, for presentation purpose, that $m$ is a multiple of $w$. Let $W_i$, with $i \in [0, m/w]$, be the window including items $u_{iw+1} \dots u_{(i+1)w}$. We proceed as follows. For the first $w$ items, we only fill $F_0$. Then for any window $W_i$ we fill both filters and we use filter $F_{(i-1 \mod 2)}$ to answer the membership queries. At the end of the window, we flush filter $F_{(i-1 \mod 2)}$. By doing this, we are able to answer the queries in foreground while maintaining a more up-to-date structure in the background, reducing the impact of the flushing overhead.

## V. Impact of the dynamic

In the case of DABF, the false positive and negative probabilities evolve over time. Indeed the two bloom filters that compose our structure are initially empty and are filled on the fly when items are received. In this section, we provide bounds on $p_t$ and $q_t$ for an item $u$ received at time $t$, taking into account that $u_t$ is the first occurrence of $u$ or not.

### A. First occurrence

In the following, we consider the first occurrence of each distinct item $u \in D_t$.

The special case $t = 1$ is easily treated as follows: since both bloom filters are empty, we get $\neg\beta.lookup(u_1)$ so $O_{\mathcal{B}}$ is queried implying that $u_1$ is correctly classified. In the following, we suppose that $t > 1$.

Before presenting our main theorem, we show that false negatives can arise when using our oracle.

**Proposition 1.** *Let $u \in \mathcal{B}$ and suppose that there is a set of items $X$ such that $v \notin \mathcal{B}$ for every $v \in X$ and $H_{DABF}(u) \wedge H_{DABF}(X) = H_{DABF}(u)$. Then there is a stream and a time $t$ such that at least one false negative occurs a time $t' > t$.*

*Proof.* Let $x = |X|$. Consider a stream $\mathcal{S}$ such that the $x$ first items correspond to items in $X$ and suppose that $u_{x+1} = u$. We deduce that $\beta.lookup(u_{x+1})$ return $true$, implying that $O_{\mathcal{B}}$ is not queried and $u$ is not added to $\beta'$. Since $X \cap \mathcal{B} = \emptyset$, $\beta'$ is empty, so $\beta'.lookup(u_{x+1})$ returns necessarily $false$, implying that $DABF.lookup(u_{x+1})$ returns $false$. □

Our results are summarized by the following theorem.

**Theorem 1.** *Let $u$ be an item received at time $t$ such that no previous occurrence of $u$ has been received.*

- *If $u \in \mathcal{B}$, then $q_t = p(n, k, |S_{t-1}|)(1 - p(n', k', |S'_{t-1}|))$.*
- *If $u \notin \mathcal{B}$, then $p_t = p(n, k, |S_{t-1}|)p(n', k', |S'_{t-1}|)$.*

*Proof.* If $\neg\beta.lookup(u)$, then $O_{\mathcal{B}}$ is queried so the false positive or negative probability is 0 in any case.

Let $\mathcal{E}_\beta$ denotes the event "$\beta_{t-1}.lookup(u)$ returns true", $\mathcal{E}_{\neg\beta'}$ (resp. $\mathcal{E}_{\beta'}$) denotes the event "$\beta'_{t-1}.lookup(u)$ returns false (resp. true)" and let $\mathcal{E}_{\neg DABF}$ (resp. $\mathcal{E}_{DABF}$) denotes the event "$DABF_{t-1}.lookup(u)$ returns false (resp. true)".

An item $u \in \mathcal{B}$ is a false negative if $\mathcal{E}_\beta \wedge \mathcal{E}_{\neg\beta'}$ occurs. We have $\Pr(\mathcal{E}_\beta \wedge \mathcal{E}_{\neg\beta'}) = \Pr(\mathcal{E}_\beta) \cdot \Pr(\mathcal{E}_{\neg\beta'} \mid \mathcal{E}_\beta) = p(n, k, |S_{t-1}|)(1 - p(n', k', |S'_{t-1}|))$.

An item $u \notin \mathcal{B}$ is a false positive if $\mathcal{E}_\beta \wedge \mathcal{E}_{\beta'}$ occurs. We have $\Pr(\mathcal{E}_\beta \wedge \mathcal{E}_{\beta'}) = \Pr(\mathcal{E}_\beta) \cdot \Pr(\mathcal{E}_{\beta'} \mid \mathcal{E}_\beta) = p(n, k, |S_{t-1}|)p(n', k', |S'_{t-1}|)$. This concludes the proof. □

The following propositions aim at showing the link between the precision of $\beta$ and the precision of $\beta'$. The idea is that if $\beta$ is more accurate, then more items will be added to $\beta'$, degrading its precision.

**Proposition 2.** *The number $X'_t$ of false positive from $\beta$ at time $t$ is lower bounded by $\sum_{i=1}^{\hat{d}_t}(1 - (1 - 1/n)^{ki})^k$.*

*Proof.* Let $X_i$ be an indicator random variable with value 1 if item $u_i \in \mathcal{B}$ is a false positive for $\beta$, and value 0 otherwise. The false positive probability is given by $(1 - n_0(i)/n)^k$, where $n_0(i)$ holds for the number of 0 cells. Hence $E[X_i] = (1 - n_0(i)/n)^k$.

Let $Y_i$ be the indicator variable set to 1 if cell $i$ is empty. The probability that cell $i$ is empty is given by $(1 - 1/n)^{ki}$, so in expectation we get $E[Y_i] = (1 - 1/n)^{ki}$. Since $n_0(i) = \sum_{j=1}^{n} Y_j$, we deduce from the linearity of expectation that $E[n_0(i)] = n(1 - 1/n)^{ki}$.

Using Jensen inequality, we get that $(1 - E[n_0(i)]/n)^k \leq E[(1 - n_0(i)/n)^k] = E[E[X_i]] = E[X_i] = (1 - n_0(i)/n)^k$. Let $X = \sum_{i=1}^{\hat{d}_t} X_i$ be the number of false positive from $\beta$ that belong to $\mathcal{B}$ in $\mathcal{S}_t$. Then we get $X \geq \sum_{i=1}^{\hat{d}_t}(1 - E[n_0(l_i)]/n)^k \geq \sum_{i=1}^{\hat{d}_t}(1 - (1 - 1/n)^{ki})^k$. □

**Proposition 3.** *The false positive probability $p_t$ is bounded by $p(n', k', \hat{d}_t(1 - (1 - (1 - 1/n)^k)^k)$.*

*Proof.* From Proposition 2, we know that the number of false positive from $\beta$ is at least $\sum_{i=1}^{\hat{d}_t}(1-(1-1/n)^{ki})^k > \sum_{i=1}^{\hat{d}_t}(1-(1-1/n)^k)^k > \hat{d}_t(1 - (1 - 1/n)^k)^k$. It implies that $|S'_t| = \hat{d}_t - X'_t \leq \hat{d}_t - \hat{d}_t(1 - (1 - 1/n)^k)^k$, which concludes the proof. □

### B. Item repetition

In the previous analysis, we considered the first occurrence of distinct items. However, it is highly probable that an item is seen several times in $\mathcal{S}$, especially if the distribution is heterogeneous. The probabilities are varying from an item to an other, depending on the stream received so far. In the following, we use the notation $t_u$ to denote the time at which the first occurrence of $u$ has been seen in $\mathcal{S}$.

**Lemma 1.** *Let $FN_t \subseteq \mathcal{B}$ be the set of items such that $DABF.lookup(u)$ is false for every $u \in FN_t$. Then we have, for any $t \geq t_u$, for any $u \in \mathcal{B}$, $u \in FN_t \Longrightarrow u \in FN_{t_u}$.*

*Proof.* The proof is by contraposition. Suppose that $u \in \mathcal{B}$ is received at time $t_u$ for the first time and $DABF_{t_u}.lookup(u)$ is true, implying that $u \notin FN_{t_u}$. We have two cases:

1) If $\neg\beta_{t_u}.lookup(u)$ then $O_{\mathcal{B}}$ is queried and $u$ is added to $\beta'$. Then $u \notin FN_{t_u}$ and $u \notin FN_t$, for any $t > t_u$, since $\beta'_t.lookup(u)$ will always answers true by definition.
2) If $\beta_{t_u}.lookup(u)$ (*i.e.*, $u$ is a false positive of $\beta_{t_u}$), then, as $u \notin FN_{t_u}$, it implies that $\beta'_{t_u}.lookup(u)$ is true. Since the bits are never cleared, we deduce that any further occurrence of $u$ will also be correctly classified.

We deduce that $u \notin FN_{t_u}$ implies that $u \notin FN_t$, for any $t \geq t_u$. By contraposition, we deduce that $u \in FN_t$ implies that $u \in FN_{t_u}$, for any $t \geq t_u$, which concludes the proof. □

**Theorem 2.** *Let $u$ be an item received at time $t$ such that $t_u \neq t$.*

- *If $u \in \mathcal{B}$, then $q_t \leq q_{t_u}$.*
- *If $u \notin \mathcal{B}$, then $p_t = p(n', k', S'_{t-1})$.*

*Proof.* Consider that $u \in \mathcal{B}$ is received at time $t$ and the first occurrence of $u$ is received at time $t_u$. Thanks to Lemma 1, we know that any false negative at time $t$ was also a false negative when its first occurrence has occurred. In other words, we have $FN_t \subseteq FN_{t_u}$. This implies that $\Pr(u \in FN_t) \leq \Pr(u \in FN_{t_u})$, leading to the statement.

Consider now that $u \notin \mathcal{B}$ is received at time $t$. First, it is obvious that the reception of item $u$ implies no modification on $\beta'$. Moreover, since $u$ has already been seen so far, $\beta.lookup(u)$ will necessarily return true, implying that the false positive probability of $DABF$ depends only on the false positive probability of $\beta'$, which is $p(n', k', |S_{t-1}|)$. □

## VI. Time and communication complexity

The reception of each item implies to update our structure and to potentially communicate with the perfect oracle $O_{\mathcal{B}}$. The time complexity only depends on the time required to probe the $k$ bits of each bloom filters bit array. If we suppose that the probing operation is done in constant time, we deduce that the time complexity is $O(k)$.

The communication complexity, measured as the number of time $O_{\mathcal{B}}$ is queried, depends on $d_m$, that is the number of distinct items in the stream. However, we remark that any false positive item for $\beta$ implies that $O_{\mathcal{B}}$ is not queried. We deduce that the exact number of communications is $C = d_m - N_m$, where $N_m$ is the number of false positive produced by $\beta$.

We prove the following lower bound on $C$.

**Proposition 4.** *Let $d_m$ be the number of distinct items in $\mathcal{S}$. The number of communications with $O_{\mathcal{B}}$ is at least*

$$d_m \left(1 - \left(\frac{kd_m}{n}\right)^k\right).$$

*Proof.* Let $X_i$ the binary random variable with value 1 if item $u_i \in \mathcal{B}$ is a false positive for $\beta$, and value 0 otherwise and let $X = \sum_{j=1}^{d_i} X_j$.

$$E[X_i] = \left(\frac{\sum_{j=1}^{i} z(j)}{n}\right)^k$$

where $z(j)$ holds for the number of bits switched from 0 to 1 at time $j$. From this, we deduce that

$$\begin{aligned} E[X] = \sum_{j=1}^{d_i} \left(\frac{\sum_{l=1}^{j} z(l)}{n}\right)^k &\leq \left(\frac{1}{n}\right)^k \sum_{j=1}^{d_i} \left(\sum_{l=1}^{j} z(l)\right)^k \\ &\leq \left(\frac{1}{n}\right)^k \sum_{j=1}^{d_i} \left(\sum_{l=1}^{j} k\right)^k \\ &\leq \left(\frac{1}{n}\right)^k \sum_{j=1}^{d_i} (jk)^k \\ &\leq (d_i)^{k+1} \left(\frac{k}{n}\right)^k \end{aligned}$$

We deduce that $d_m - X_m \geq d_m - (d_m)^{k+1} \left(\frac{k}{n}\right)^k$, which concludes the proof. □

## VII. Performance evaluation

*c) Performance measures:* In our simulations, we measure the precision and the recall of our solution. Formally, let $P_t \subseteq D_t$ be the set of distinct items in $\mathcal{S}_t$ for which DABF answers positively. The *precision* is given by $\frac{|P_t \cap \mathcal{B}|}{|P_t|}$ and the *recall* is given by $\frac{|P_t \cap \mathcal{B}|}{|\hat{D}_t|}$. It is possible that either $P_t$ or $\hat{D}_t$ are empty, implying a division by zero. In such case, the chosen convention is to set the corresponding measure at 1. This can be justified by the fact that if $P_t$ (or $\hat{D}_t$) is empty, then it means that we can not misclassified an item. Moreover, we want to specify that an item $u$ is added to $P_t$ if DABF answers positively for at least one of its occurrence in $\mathcal{S}_t$.

We also present in some simulations the number of communications initiated with $O_{\mathcal{B}}$. Finally we are interested in the false positive and negative rates $FPR_t$ and $FNR_t$. Let $\bar{N}_t$ be the multiset of all items which appear in $\mathcal{S}_t$ items (with potential repetitions) which do not belong to $\mathcal{B}$. We similarly define $\bar{P}_t$ for items belonging to $\mathcal{B}$. We define $\bar{FP}_t$ and $\bar{FN}_t$ as the multisets of items in $\mathcal{S}_t$ which are respectively false positives and false negatives. We define the false positive rate as follows:

$$FPR = \begin{cases} |\bar{FP}_t|/|\bar{N}_t|, & |\bar{N}_t| > 0 \\ 0, & |\bar{N}_t| = 0 \end{cases}$$

We define the false negative rate as follows:

$$FNR = \begin{cases} |\bar{FN}_t|/|\bar{P}_t|, & |\bar{P}_t| > 0 \\ 0, & |\bar{P}_t| = 0 \end{cases}$$

*d) General simulation set up:* We consider a universe $\mathcal{U}$ of size $U = 10^4$ which consists of integers in range $[0, U-1]$. The size $B = |\mathcal{B}|$ is chosen among the set $\{10, 100, 1000\}$ and items in $\mathcal{B}$ are sampled uniformly at random among $\mathcal{U}$ at the beginning of each simulation. In each simulation, we generate 500 streams following a Zipf's law of parameter $\alpha \in \{0.5, 1.0, 2.0, 3.0\}$. In the simulation descriptions that follow, the item distribution is denoted by `zipf-`$\alpha$. The results correspond to the average precision or recall among the 500 runs.

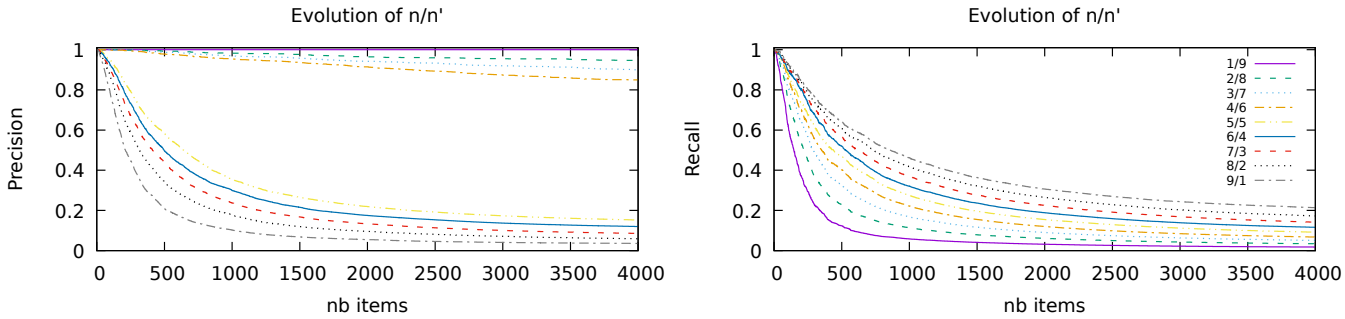


Fig. 1. Evolution of precision and recall with the variation of ratio $n/n'$. ($B = 100$, $N = 500$, $\alpha = 0.5$)

*e) Bloom filter implementation:* The corner stone of our solution is to use classic static bloom filters. To implement these data structures, we have made the following choices.

First, we know from [10] that the false positive probability of a classic static bloom filter of size $n$ that receives $d$ distinct items is minimized when the number of random hash functions is $k = \frac{n}{d} \ln 2$. Since $\frac{n}{d} \ln 2$ is generally not an integer, a typical choice is to take $\lfloor \frac{n}{d} \ln 2 \rfloor$ as value for $k$, in order to minimize the computational cost of the hash functions. The number of items added to $\beta$ corresponds to the number of distinct items $d$ in $\mathcal{S}$. Since we do not make any assumptions on the items distributions, we must assume that $d = U = |\mathcal{U}|$ if the stream is large enough. This leads to $k = 1$ for $n < 2U/\ln 2$, which is not optimal at early times when the real number of items in $\beta$ is really low. As a trade-off, we have chosen to set $k = \lfloor \frac{n}{10^3} \ln 2 \rfloor$ for $\beta$, whatever is the value of $n$. This setting provides better results for the first items of $\mathcal{S}$. For bloom filter $\beta'$, a natural upper bound is $B$ so we have set $k' = \left\lfloor \frac{n'}{B} \ln 2 \right\rfloor$.

For every item $u$, we use a pseudo-random number generator with a seed set at $u$. Then we generate $k$ random values in range $[0, n-1]$, where $n$ is the size of the corresponding bloom.

As a baseline, we implement a simple solution called $1bf$ which simply consists in a single bloom filter populated with the whole set $\mathcal{B}$. For the $1bf$ solution with $N$ bits of memory, we use $\lfloor \frac{N}{B} \ln 2 \rfloor$ hash functions and we also use a pseudo-random number generator for the probes.

### A. Single window

In this section, we present simulations with the basic version of DABF, that is without using a sliding window. The generated streams are of size $m = 4000$ in all simulation.

*f) Memory partitioning:* In this first simulation, we aim at determining the best way to partition a given amount of memory among our two bloom filters. Let $N$ be the total amount of memory we can afford, then we set $n = N \cdot 0.1 \cdot i$ (for $i \in [1, 9]$) and $n' = N - n$. In Figure 1, we present the results for a `zipf-0.5` distribution, $B = 100$ and $N = 500$. The results are similar for others values of $B$ or $N$.

After only few dozens of items insertions, the precision becomes proportional to the memory given to $\beta$, while the opposite can be notice for the recall. This confirms our theoretical analysis on the false positive and negative probability.

*g) Impact of the memory:* In this simulation, we choose the ratio $\theta = n/n' = 1/9$ for the memory partitioning, in order to optimize the precision of our solution. As we argue above, maximizing the precision is fundamental in many use cases. However, this heuristic can be set differently, as a user parameter. We vary the skewness parameter of the Zipf's law and the overall memory $N$ provided to the structure.

We focus on the case $B = 100$ and we vary the memory $N$ from 1000 to 2000 bits, by step of 200. The results are summarized in Figure 2.

Let first analyse the case $\alpha = 0.5$. For the precision, DABF is optimal with only 1000 bits of memory. However the recall drops quickly due to the high number of distinct items. The first bloom filter $\beta$ is indeed saturated with only few decades of items, implying a high number of false negative, which degrades the recall. On the other hand, $1bf$ is optimal in term of precision with 2000 bits and always produces an optimal recall (as bloom filters avoid false negatives).

In the case $\alpha = 2.0$ however, the performances of DABF are greatly improved. The precision of our solution is already optimal with $N = 1000$ and the recall decreases slowly. If we consider the first 1000 items, the recall never goes below 0.9, even with 1000 bits. Please notice that it is possible to opt for a trade-off by choosing a ratio $\theta = 0.5$. This leads to a precision greater than 0.9 while slowing down the recall decrease (cf. Figure 1).

Consider now the case $B = 1000$, with $\theta = 1/9$ as for case $B = 100$. The results are summarized in Figure 3. We can notice that the precision is nearly optimal with only $N = 4000$ bits using $DABF$, even for `zipf-0.5` distribution. As a comparison, the $1bf$ solution provides a precision of about 0.4 for the same value of $N$. However, as for the case $B = 100$, the recall with $DABF$ is quickly decreasing when the distribution is `zipf-0.5`. Even for $N = 10000$, the recall drops below 0.7 after less than 1000 insertions. In comparison, $1bf$ provides a precision above 0.9 for $N \geq 10000$ while keeping a perfect recall. In the case $\alpha = 2.0$ however, our solution clearly outperforms $1bf$, with a near perfect precision and a recall of more than 0.89 with only 4000 bits of memory.

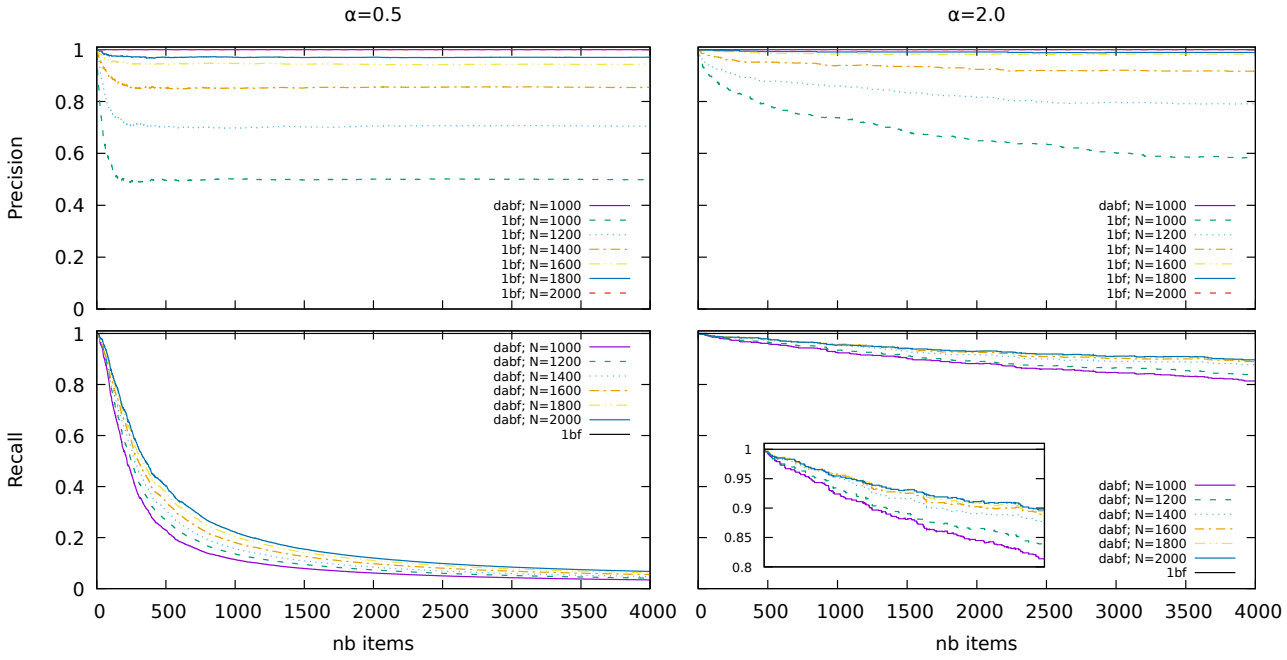


Fig. 2. Impact of the memory with a ratio $\theta = 1/9$ and $B = 100$.

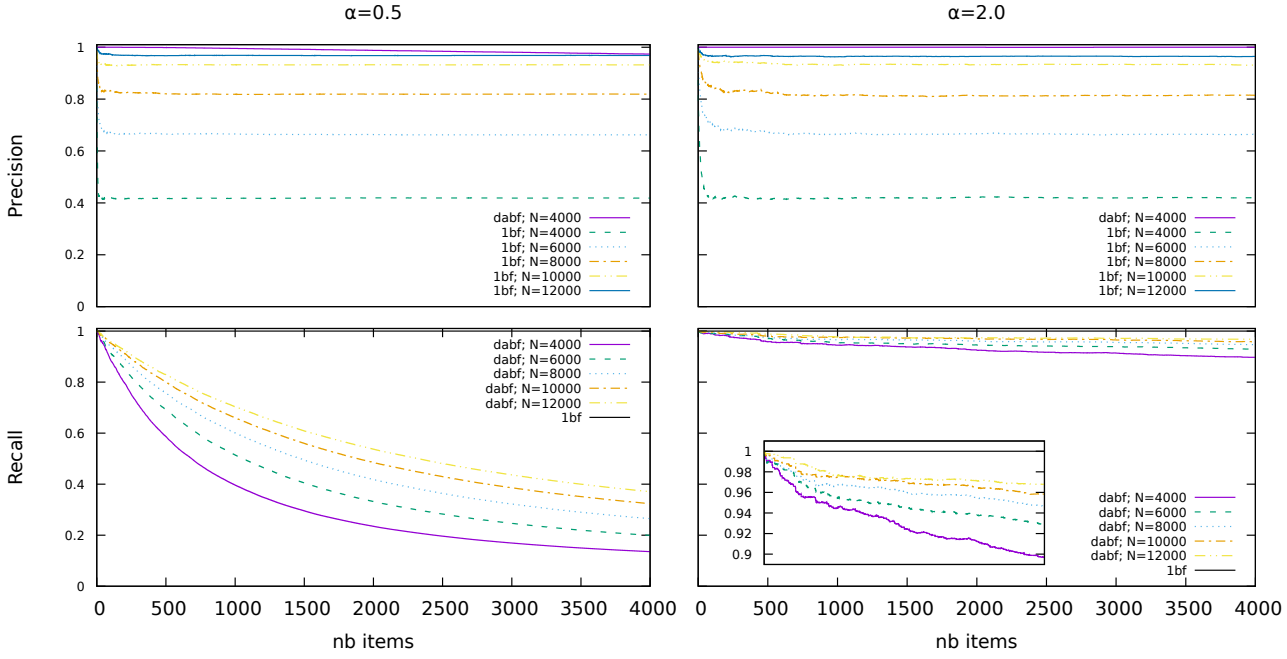


Fig. 3. Impact of the memory with a ratio $\theta = 1/9$ and $B = 1000$.

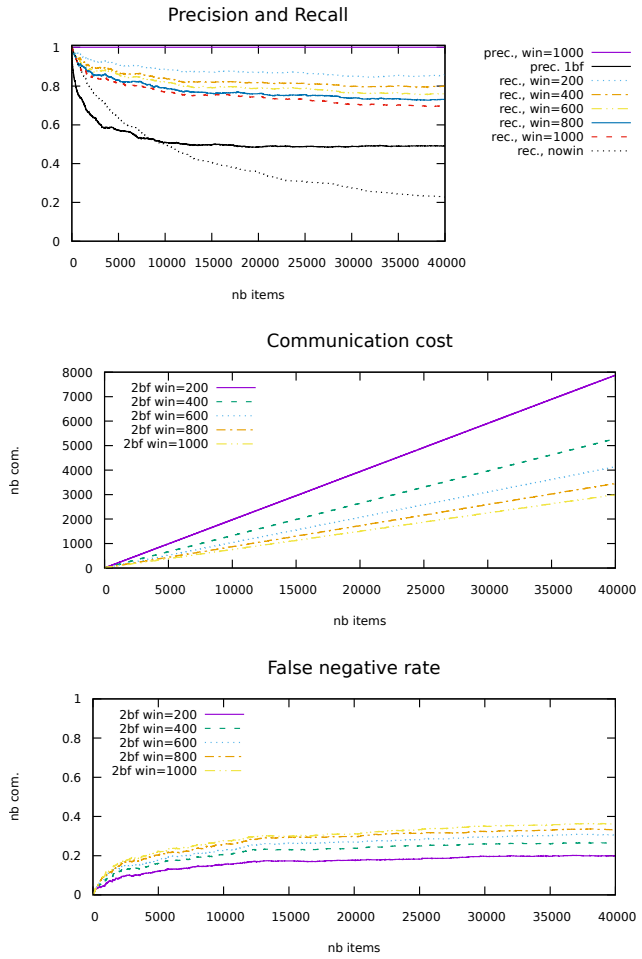


Fig. 4. Results with the sliding window version ($\alpha = 2, N = 1000, B = 100, m = 40000$)

### B. Sliding window

In order to maintain an up-to-date state of our data structure and to avoid the recall drop, we study the sliding window version. We vary the size of the window from 100 to 1000 by step of 100. We focus our study on the `zipf-2.0` distribution, $N = 1000$ and $B = 100$. The size of the streams is $m = 40000$. Here $N$ corresponds to the overall amount of memory, including the two DABF of the sliding window version. The results are summarized in Figure 4.

The top figure presents the precision and recall for different values of the windows size. As baselines, we also present the precision of 1bf and the recall of DABF without using the sliding window mode (nowin). The middle figure represents the communication cost, measured in term of number of communications initiated with $O_{\mathcal{B}}$. Finally the bottom figure represents the false negative rate, depending on the window size, with DABF. The false positive rate is 0 even with a window of size 1000. On the top figure, we observe that the recall is inversely proportional to the size of the window, which makes sense since flushing more often the bloom filters improves their accuracies. This impacts directly the communication cost, as illustrated by the middle figure, which is inversely proportional to the window size. We highlight here that the communications can be easily reduced by querying the oracle if at least one of the filters need to contact him and making the answer available for both filters. The precision with DABF is close to 1 even with a sliding window of 1000 items. As a comparison, 1bf converges to approximately 0.5.

Finally the bottom figure shows that the false negative rate is really stable, whatever is the size of the window. With a window of 1000 bits, the false negative rate is close to 0.2, meaning that 20% of negative answers corresponds to misclassified items that belong to $\mathcal{B}$. But this represent a very small number of distinct items for skewed distributions.

## VIII. Conclusion and further works

We have presented a simple but efficient solution to evaluate the membership of items in a stream to a given set $\mathcal{B}$. The main idea is to summarize only the subset of items in $\mathcal{B}$ present in the stream instead of representing the whole set. This leads to a notable space saving in comparison with basic solutions such as a bloom filter. We keep a good precision and recall, especially for skewed distributions.

In the current version of the solution, we are using two bloom filters: one to remember which item has already been seen and a second to summarize the subset of items in $\mathcal{B}$ which have been already seen. Especially, as we make no assumption on the items distribution, we must use a large enough amount of memory to handle any scenario. An interesting perspective is to use solutions for representing dynamic sets such as [5] or [7] and evaluate the interest on the memory consumption in comparison with the use of simple bloom filters. This should be especially promising for the first bloom filter $\beta$ of $DABF$.

## References


[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[2] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[3] B. Donnet, B. Baynat, and T. Friedman, "Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives," in *CoNEXT*. ACM, 2006, p. 13.

[4] R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte, "A generalized bloom filter to secure distributed network applications," *Computer Networks*, vol. 55, no. 8, pp. 1804–1819, 2011.

[5] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 1, pp. 120–133, 2010.

[6] A. Goel and P. Gupta, "Small subset queries and bloom filters using ternary associative memories, with applications," in *SIGMETRICS*. ACM, 2010, pp. 143–154.

[7] S. Negi, A. Dubey, A. Bagchi, M. Yadav, N. Yadav, and J. Raj, "Dynamic partition bloom filters: A bounded false positive solution for dynamic set membership (extended abstract)," *CoRR*, vol. abs/1901.06493, 2019.

[8] M. Mitzenmacher, S. Pontarelli, and P. Reviriego, "Adaptive cuckoo filters," in *ALENEX*. SIAM, 2018, pp. 36–47.

[9] M. A. Bender, M. Farach-Colton, M. Goswami, R. Johnson, S. McCauley, and S. Singh, "Bloom filters, adaptivity, and the dictionary problem," in *FOCS*. IEEE Computer Society, 2018, pp. 182–193.

[10] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.