The Variable-Increment Counting Bloom Filter

Ori Rottenstreich, Yossi Kanizo and Isaac Keslassy

***Abstract*—Counting Bloom Filters (CBFs) are widely used in networking device algorithms. They implement fast set representations to support membership queries with limited error, and support element deletions unlike Bloom Filters. However, they consume significant amounts of memory.**

In this paper we introduce a new general method based on *variable increments* to improve the efficiency of CBFs and their variants. Unlike CBFs, at each element insertion, the hashed counters are incremented by a hashed variable increment instead of a unit increment. Then, to query an element, the exact value of a counter is considered and not just its positiveness. We present two simple schemes based on this method. We demonstrate that this method can always achieve a lower false positive rate and a lower overflow probability bound than CBF in practical systems. We also show how it can be easily implemented in hardware, with limited added complexity and memory overhead. We further explain how this method can extend many variants of CBF that have been published in the literature. We then suggest possible improvements of the presented schemes, and provide lower bounds on their memory consumption. Last, using simulations with real-life traces and hash functions, we show how it can significantly improve the false positive rate of CBFs given the same amount of memory.

***Index Terms*—Counting Bloom Filter, Variable Increments, False Positive Rate.**

I. Introduction

A. Motivation

CBF (Counting Bloom Filter) variants are increasingly used in networking device algorithms, in fields as diverse as accounting, monitoring, load-balancing, caching, policy enforcement, routing, filtering, security, and differentiated services. For instance, for a given flow, they can determine whether it has at least one packet currently queued (set membership), how many of its packets are queued (counter representation), whether it is in a given state (state representation), or where to forward its packets (IP lookups) [1]–[7].

CBFs (and the simpler Bloom Filters) are reportedly used in such well-known products as Mellanox's IB Switch System [8], Facebook's distributed storage system Cassandra [9], Google's web browser Chrome [10], Google's database system BigTable [11], the network storage system Venti [12] or the Web Proxy Cache Squid [13].

CBFs are often used in networking devices because they can be easily implemented in hardware. In particular, element insertions, deletions and queries can be implemented in CBFs using a *constant complexity* that is essentially independent of the number of elements for a given bits-per-element ratio. However, CBFs also consume *significant amounts of memory*. For instance, using four bits per entry and ten entries per element yields a needed memory space in bits that is 40 times larger than the number of inserted elements.

This paper is about a general method to improve the memory efficiency of CBFs in networking devices, with limited added hardware complexity. We introduce a novel method based on variable increments to reduce the amount of memory used by CBFs for a given false positive rate. This method can also implement element insertions, deletions and queries using a constant complexity per element. It has a low hardware implementation overhead when compared to CBFs, and can replace CBFs as a sub-module in any networking device implementation without required outside changes.

B. Intuition for Variable Increments

We now provide some intuition for variable increments by comparing Bloom Filters (BFs), Counting Bloom Filters (CBFs) and Variable-Increment Counting Bloom Filters (VI-CBFs).

A Bloom Filter (BF) is a well-known simple data structure used to represent a set of n elements $S = \{x_1, \ldots, x_n\}$ elements from a universe U using an array of m bits [14]. However, it is not designed to support deletions of elements, which are often needed in networking device algorithms.

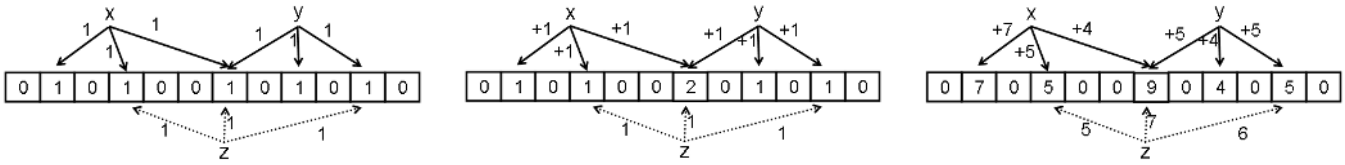
As illustrated in Figure 1(a), BF uses k uniformly-distributed hash functions over the range $\{1, \ldots, m\}$ of its m-bit filter. For each element $x \in S$, k hash entries are calculated using the hash functions and the corresponding bits are set to one. For instance, in the figure, the bits of x and y are set to one. In order to check whether an element z is in S, we check whether all of its k corresponding bit locations $h_i(z)$ are set to one. If this is not the case, we know that z is not in S. If all of them are set, as in Figure 1(a), we state that $z \in S$, although this might be a *false positive* error. For each $z \notin S$, the *false positive rate*, i.e. the probability of a false positive error, is $(1 - p_0)^k$, where $p_0 = (1 - 1/m)^{nk}$ is the probability that a specific bit is still zero after the insertion of n elements. Since BF does not support deletions of elements, it cannot for instance be used to represent the current set of packets of a flow in a router where flows might dynamically change.

The Counting Bloom Filter (CBF) suggested by Fan et al. [1] is a generalization of BF, in which each hash entry contains a counter with a fixed size of b bits, instead of a single bit in BF. Unfortunately, while supporting deletions, CBF also needs large amounts of memory space (i.e. b times the memory space consumed by BF), which is often valuable in networking devices.

As shown in Figure 1(b), to insert an element, all the corresponding hashed counters are incremented by one. Likewise,

O. Rottenstreich and I. Keslassy are with the Department of Electrical Engineering, Technion, Haifa 32000, Israel (e-mails: or@tx.technion.ac.il, isaac@ee.technion.ac.il).

Y. Kanizo is with the Department of Computer Science, Technion, Haifa 32000, Israel (e-mail: ykanizo@cs.technion.ac.il).



(a) BF (Bloom Filter) (b) CBF (Counting Bloom Filter) (c) $VI-CBF$ (Variable-Increment CBF)

Fig. 1. Comparison of the concepts behind BF, CBF and our proposed $VI-CBF$, using $S=\{x,y\}$ and a query of element $z \notin S$. In this example, BF and CBF yield false positives while $VI-CBF$ does not.

to delete it, all of its hashed counters are decremented. To determine if an element $z \in S$, we check if all of its hashed entries are positive. For instance, in Figure 1(b), we state that $z \in S$, which might be a false positive as in BF. Given only insertions, the false positive rate of CBF is the same as for BF with the mentioned increase in memory space. CBF might also suffer from counter overflows with a probability that depends on its counter size b, although $b=4$ is sufficient to practically obtain a negligible overflow probability [1].

We now want to introduce the use of variable increments. We notice that CBF does not store much information in its counter values. When querying an element, it does not distinguish between any counter values greater than zero, and only considers their positiveness. In this paper, we use the *specific value* of a counter in order to give a more complete answer to the query.

The Variable-Increment Counting Bloom Filter, denoted as $VI-CBF$, is a generalization of CBF that uses variable increments to update each entry. We first define a set of possible variable increments D. Then, for each counter update by an element, we hash the element into a value of D and use it to increment the counter. Likewise, to delete an element, we decrement by its hashed value in D. Last, to determine if an element $z \in S$, we check in each of its counters if its hashed value in D could be part of the sum. If this is not the case in at least one counter, then necessarily $z \notin S$. Otherwise, as for BF and CBF, we state that $z \in S$, which might be a false positive.

Figure 1(c) illustrates $VI-CBF$ with $D=\{4,5,6,7\}$. First, x and y increment their corresponding counters by their corresponding hashed values in D. For instance, x increments its first counter value by $7 \in D$. Consider now a query of whether z is in S. The second hashed entry of z has counter value 9, while for this entry z hashes to increment $7 \in D$. Since $9-7=2$ cannot be presented as a sum of elements of D, the increment 7 cannot be part of the sum 9, and we deduce that necessarily $z \notin S$, avoiding the false positive that occurred in CBF (Figure 1(b)). Note that we could have known that as well from the third hashed entry of z (since $6>5$), but not from its first entry (because x and z hash to the same increment, so the sum of 5 can be composed of the increment 5).

C. Related Work: Applications of the Variable-Increment Method

The variable-increment method is a *generic* approach that can actually be implemented to improve or extend *most variants of CBF* in networking devices. In this paper, we detail and evaluate its application to both CBF and *ML-HCBF*.

- *CBF* [1]: As presented above.
- *ML-HCBF* (MultiLayer Hashed CBF) [4]: This algorithm uses a hierarchical compression of CBF filters to achieve better performance. We explain in Section V how the same hierarchical compression idea could be combined with $VI-CBF$, and show in Section VIII how this combination can achieve even better results.
- *VL-CBF* (Variable Length CBF) [15]: This algorithm uses a variable-length coding, such as the Huffman coding, to represent counters with a variable number of bits. A similar coding can be used for efficient representation of counters in $VI-CBF$. Unfortunately, even using index tables, a lookup in *VL-CBF* might be 100 times slower than in the standard BF. Thus, it cannot be implemented at line rate.
- *SBF* (Stateful BF) *ACSM* (Approximate Concurrent State Machine) [2]: This scheme enables the representation of dynamically-changing states of flows. Using variable increments, it is possible to represent the set of two states hashing into the same entry as the sum of their hashed values, instead of simply storing a DK (Don't Know) value as currently done.
- *Counter Braids* [6]: This efficient counter architecture incrementally compresses flow counters as it uses a hierarchy of counters braided via a random graph. It later uses an iterative reconstruction scheme for the recovery of flow counters. The decoding of all flow counters is required to obtain any single flow counter. By using variable increments in each of its stages, it would provide more information to its reconstruction scheme, which may help ensure better guarantees on the termination of the decoding scheme.
- *Fingerprint-based Schemes* [2], [3]: Fingerprint-based schemes typically use multiple-choice hashing schemes (e.g. d-left) to obtain balanced allocations of elements into buckets, and then use hashed fingerprints within each bucket to store information associated with each flow. While fingerprint-based schemes belong to a different family of algorithms than CBF, they may also be complemented using the variable-increment idea. For instance, a fingerprint-based scheme may store up to h states for each flow, or allow to store up to h flow mini-fingerprints together with each main fingerprint. By summing these state values, a variable-increment idea may decrease the

number of bits required to store these values. In addition, it may behave more gracefully when there are more than h states, by temporarily losing some information, but being able to recover it with high probability upon deletion (i.e. decrement) of some of the states.

- And many additional schemes like Selective CBF [16], compressed CBF [17] and Access-Efficient Balanced BF [18].

However, there are also a few variants to CBF that the generic variable-increment idea does not necessarily improve. For instance, [19] uses counters to estimate item multiplicities and the suggested counter updating schemes might make it impossible to obtain the exact sum of increments in a counter.

Finally, there are many works on B_h sequences [20]–[22], yet none have been used in network applications. For instance, [22] suggests dense constructions of such sequences.

D. Contributions

This paper presents an improved Counting Bloom Filter technique based on variable increments. This technique can also implement element insertions, deletions and queries in networking devices using a constant complexity per element. We suggest two schemes based on CBFs with variable increments.

We first present the $B_h - CBF$ scheme. This scheme is based on B_h *sequences*. To the best of our knowledge, *this is the first time that B_h sequences are used in network applications*. Intuitively, a B_h sequence is a set of integers with the property that for any $h' \leq h$, all the sums of h' elements from the set are distinct. Therefore, given a sum of h' elements, we can determine whether an element of the B_h *sequence* is a part of the sum. In the $B_h - CBF$ scheme we have in each hash entry a pair of counters: one with fixed increments, and another one with variable increments that are selected from the B_h sequence. We illustrate the $B_h - CBF$ scheme and compute its false positive rate.

Then, we present the $VI - CBF$ scheme. In this scheme, each hash entry only contains a single counter, as illustrated above. We analytically show that the $VI - CBF$ scheme can always achieve a lower false positive rate and a lower overflow probability bound than CBF in practical systems.

We also provide detailed implementation considerations for these schemes in networking devices. We discuss the complexity and throughput of the schemes, and show that their complexity overhead is lower than would be expected, especially for the $VI - CBF$ scheme, which can avoid using any lookup table.

Further, although in each operation both schemes require calculating $2k$ hash functions instead of k in CBF, we show that the relative increase in the number of required random bits is very small. For instance, the $VI - CBF$ scheme typically needs $k \cdot (\lceil \log_2(m) \rceil + 1)$ bits instead of $k \cdot (\lceil \log_2(m) \rceil)$ in CBF.

Next, we show that $VI - CBF$ can be combined with the MultiLayer Hashed CBF scheme [4] in networking device implementations to further decrease its memory requirements for a requested false positive rate. Later, we suggest an improved query scheme for the $B_h - CBF$ that further reduces its false positive rate.

Then, we provide lower bounds on the memory requirements of the $B_h - CBF$ and the $VI - CBF$ schemes.

Last, we evaluate the efficiency of all schemes in networking devices, using simulations with real-life traces and hash functions. We show that this method can reduce the false positive rate of the original CBF by up to an order of magnitude, or alternatively reduce the memory requirements for a requested false positive rate by 33%.

II. The $B_h - CBF$ Scheme

A. B_h Sequences

In this section we introduce the $B_h - CBF$ scheme, a *variable-increment* CBF (Counting Bloom Filter) based on B_h sequences [20]. We start with the formal definition of B_h sequences. B_2 sequences are also called Sidon sequences [21].

Definition 1 (B_h Sequence): Let $D = \{v_1, v_2, ..., v_\ell\} \subseteq \mathbb{N}^*$ be a sequence of positive integers. Then D is a B_h *sequence* iff all the sums $v_{i_1} + v_{i_2} + \cdots + v_{i_h}$ with $1 \leq i_1 \leq \cdots \leq i_h \leq \ell$ are distinct.

Example 1: Let $D = \{v_1, v_2, v_3, v_4\} = \{1, 4, 8, 13\} \subseteq \mathbb{N}^*$. We can see that all the 20 sums of 3 elements of D are distinct: $1+1+1=3, 1+1+4=6, 1+1+8=10, 1+1+13=15, 1+4+4=9, 1+4+8=13, 1+4+13=18, 1+8+8=17, 1+8+13=22, 1+13+13=27, 4+4+4=12, 4+4+8=16, 4+4+13=21, 4+8+8=20, 4+8+13=25, 4+13+13=30, 8+8+8=24, 8+8+13=29, 8+13+13=34, 13+13+13=39$. Therefore, D is a B_3 sequence. However, $4+4+4+4=16=1+1+1+13$, therefore D is not a B_4 sequence.

We can observe that for any $h' \in [1, h]$ the sums of exactly h' elements of the B_h sequence are also distinct.

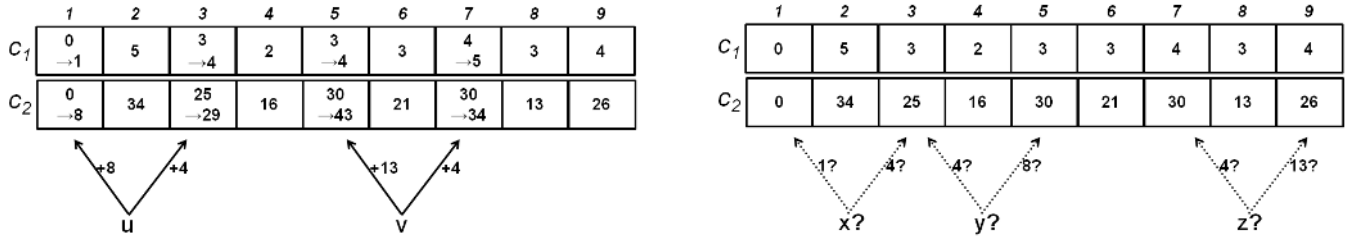
Observation 1: If $D = \{v_1, v_2, ..., v_\ell\}$ is a B_h sequence and $h' \in [1, h]$ then D is also a $B_{h'}$ sequence.

Proof: The case of $h' = h$ is trivial. If for some $h' \in [1, h-1]$ we have the same sum of h' elements with two different sets of elements of D, we can simply add to each of the sums the same arbitrary $h - h'$ new elements. We obtain two non-distinct sums of exactly h elements from the B_h sequence. Contradiction to Definition 1. ■

B. Scheme Principles

We now introduce the $B_h - CBF$ scheme to represent $|S| = n$ elements using m entries. While in CBF, each entry contains a single counter with fixed increments of one, in $B_h - CBF$, each entry contains *a pair of counters*: The first counter, with fixed increments of one, counts the number of elements hashed into this entry (as in CBF). The second counter, with variable increments, provides a weighted sum of these elements. Its variable increments are selected from a pre-determined B_h sequence $D = \{v_1, v_2, ..., v_\ell\}$.

Figure 2 illustrates how these counters are used and updated given element insertions and queries. Note that in Section IV we further show how to implement these operations efficiently in hardware.



(a) Insertion (see Example 2)

(b) Query (see Example 3)

Fig. 2. Examples of insertion and query in $B_h - CBF$ with the B_3 sequence $D = \{1, 4, 8, 13\}$.

$B_h - CBF$ uses two sets of k hash functions. The first set $H = \{h_1, \ldots, h_k\}$ uses k hash functions with range $\{1, \ldots, m\}$, i.e. it points to the set of entries. The second set $G = \{g_1, \ldots, g_k\}$ uses k functions with range $\{1, \ldots, \ell\}$, i.e. it points to the set D.

Insertion — Upon insertion, an element x is hashed into the k hash entries pointed by $\{h_1, \ldots, h_k\}$. At each entry $h_i(x)$, $B_h - CBF$ updates the pair of counters as follows. The first counter, with fixed increments, is incremented by one. The second counter, with variable increments, is incremented by the element $v_{g_i(x)}$ of the B_h sequence D, where g_i is the corresponding hash function of G.

Upon deletion, the counters are decremented similarly.

Example 2: Figure 2(a) illustrates the insertion of two elements u and v. It uses the B_3 sequence $D = \{v_1, v_2, v_3, v_4\}$ $= \{1, 4, 8, 13\}$ from Example 1, and each element is hashed into $k = 2$ entries. In this example, $\{h_1(u), h_2(u)\} = \{1, 3\}$ and $\{g_1(u), g_2(u)\} = \{3, 2\}$. The packet belonging to flow u is hashed into entries $1, 3$ using hash functions h_1, h_2. It increments the first counter of each entry by one. It also increments the second counters of these entries by $(v_{g_1(u)}, v_{g_2(u)}) = (v_3, v_2)$ = $(8, 4)$, respectively. Similarly, a packet of flow v is hashed into entries 5 and 7 and increments their variable-increment counters by 13 and 4, respectively.

Query — To query whether an element y is in S, $B_h - CBF$ uses the unique properties of the B_h sequence. For $i \in [1, k]$, let $c_1(i)$ and $c_2(i)$ be the values of the fixed-increment and variable-increment counters in entry $h_i(y)$, respectively. $B_h - CBF$ *asks whether y might have been inserted in this entry in the past*. Namely, if y is hashed into value $v_{g_i(y)}$, it asks whether $c_2(i)$ can be a sum of $c_1(i)$ elements including $v_{g_i(y)}$. Specifically, in each entry $h_i(y)$, it considers several cases depending on the value of $c_1(i)$:

• *If* $c_1(i) = 0$, then as in CBF, $B_h - CBF$ determines that $y \notin S$.

• *If* $c_1(i) \in [1, h]$, $B_h - CBF$ considers the exact values of both the counters. In this case, no more than h elements were hashed into the hash entry $h_i(y)$. Therefore, $c_2(i)$ is a sum of $c_1(i) \leq h$ elements of D. Since D is a B_h sequence, $B_h - CBF$ can deduce the elements of D used in the sum $c_2(i)$ (i.e. in the insertions of these elements into this hash entry). In particular, $B_h - CBF$ can determine whether $v_{g_i(y)}$ is part of the sum $c_2(i)$. Otherwise, necessarily $y \notin S$.

• *If* $c_1(i) > h$, $B_h - CBF$ considers the entry as useless. The value of $c_2(i)$ is not used and we cannot determine that $y \notin S$ based on the current hash entry.

If $B_h - CBF$ cannot determine that $y \notin S$ based on any of the k hash functions, it determines that $y \in S$. With some probability, $B_h - CBF$ might be wrong and yield a false positive. More precisely, in case $y \notin S$, the false positive occurs if for each of the k hash functions, either there are more than h elements hashed into the corresponding hash entry, or the corresponding element of the B_h sequence was used by another element hashed into the same entry.

Example 3: As illustrated in Figure 2(b), we look at three queries for $B_h - CBF$ introduced in Example 2. In this figure, we assume that for x, y, z the hash entries selected by the hash functions h_1 and h_2 are provided by their left and right outgoing arrows, respectively. For each element $a \in \{x, y, z\}$, we denote the values of the two counters in the hash entry $h_i(a)$ by $c_1^a(i), c_2^a(i)$.

First, in order to determine whether $x \in S$, we start by looking at the hash entry $h_1(x)$. Since $c_1^x(1) = 0$, $B_h - CBF$ can determine immediately that $x \notin S$. We now consider y. In its first hashed entry, $h_1(y)$, the number of elements is $c_1^y(1) = 3$. Since we use a B_h sequence D with $h = 3$, then $B_h - CBF$ can determine the components of the weighted sum by definition of the B_h sequence. In this case, $B_h - CBF$ can deduce that $c_2^y(1) = 25$ is comprised of $4 + 8 + 13 = 25$. Since y increments this counter by $v_{g_1(y)} = 4$, $B_h - CBF$ cannot exclude $y \in S$ based on this hash entry. However, since $c_1^y(2) = 3 \leq h$ as well, the variable-increment counter $c_2^y(2) = 30 = 4{+}13{+}13$ is not comprised of $v_{g_2(y)} = 8$. Thus, $B_h - CBF$ can also determine that $y \notin S$. Finally, applying the above method to z, $B_h - CBF$ cannot deduce that $z \notin S$ based on the two hash entries since $c_1^z(1), c_1^z(2) = 4 > h$. Therefore, $B_h - CBF$ determines that $z \in S$. This might of course yield a false positive if $z \notin S$. (In fact, using a more careful examination, it is possible to determine that indeed $z \notin S$, as detailed below with the *improved $B_h - CBF$ scheme*.)

C. False Positive Rate

We now provide the false positive rate of $B_h - CBF$. As shown in the literature, in real-world systems, practical hash functions usually work as if they were fully random [23]. We assume in our proofs that the hash functions map items to random numbers uniformly distributed over their given range, and that the inserted elements and the elements in the query are independent. We will show that the false positive rate of $B_h - CBF$ depends only on the parameters h and ℓ of the

B_h sequence D and is not affected by the exact values of $v_1, v_2, \cdots, v_\ell$.

Theorem 1: The false positive rate of $B_h - CBF$ is given by:

$$FPR = \left(1 - \sum_{j=0}^{h} \binom{nk}{j} \left(\frac{\ell-1}{\ell m}\right)^j \left(1 - \frac{1}{m}\right)^{nk-j}\right)^k. \quad (1)$$

Proof: Let X denote the number of elements hashed into an arbitrary entry. The probability of the event $X = j$ is given by

$$\Pr(X = j) = \binom{nk}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{nk-j}. \quad (2)$$

When exactly $X = j$ elements are hashed into an entry, a specific value of the B_h sequence is not used by any of them with probability $\left(1 - \frac{1}{\ell}\right)^j$. Therefore, the false positive rate FPR, i.e. the probability that for an element $y \notin S$ we cannot deduce from each of the k hash entries that $y \notin S$, is

$$FPR = \left(1 - \sum_{j=0}^{h} \Pr(X = j)\left(1 - \frac{1}{\ell}\right)^j\right)^k, \quad (3)$$

yielding the result. ∎

Note that by adopting a more complex query scheme, we can further decrease the false positive rate. This *improved* $B_h - CBF$ *scheme* analyzes the hash entry even when there are more than h hashed elements. In such a case, the B_h sequence definition does not directly help anymore, because the elements in the sum $c_2(i)$ are not necessarily unique when there are more than h elements. However, by examining all the possible variable increments that can lead to the sum $c_2(i)$, we can still sometimes conclude that an element cannot have been inserted into this entry. Unlike the $B_h - CBF$ scheme, the false positive rate of the *improved* $B_h - CBF$ *scheme* is influenced by the exact values of the elements $v_1, v_2, \cdots, v_\ell$ composing the B_h sequence D. More details and analysis can be found in Section VI.

III. The $VI - CBF$ Scheme

A. Scheme Description

The $B_h - CBF$ scheme suggested above uses two counters per entry instead of a single counter. This nearly doubles the needed number of bits (neglecting the differences in counter sizes). Consequently, we introduce $VI - CBF$ (Variable-Increment Counting Bloom Filter), which also uses variable increments but only relies on a single variable-increment counter per entry, without the additional counter that indicates the number of hashed elements.

Specifically, as previously illustrated in Figure 1(c), we use again an array of m entries to represent $|S| = n$ elements. In each array entry, the single variable-increment counter is updated exactly like the second counter in the $B_h - CBF$ scheme, using variable increments selected from a set $D = \{v_1, v_2, ..., v_\ell\}$. We use again two sets of k hash functions, $H = \{h_1, \ldots, h_k\}$ and $G = \{g_1, \ldots, g_k\}$. Upon insertion, at each corresponding array position $h_i(x)$, the counter is incremented by the element $v_{g_i(x)}$ of the set D. Likewise, upon deletion, counter $h_i(x)$ is decremented by $v_{g_i(x)} \in D$.

We now want to find an appropriate set D for the $VI - CBF$ scheme. A problem in the $VI - CBF$ scheme is that *it cannot directly use* B_h *sequences anymore*. This is because the B_h sequence definition requires to know the number of elements in a sum. However, unlike the $B_h - CBF$ scheme, the $VI - CBF$ scheme cannot obtain it because it does not have a small counter that provides the number of elements hashed into a given entry.

B. A First Option for D: $\tilde{B}_h$ Sequences

According to the last observation, we now suggest to use more complicated sequences, and call them $\tilde{B}_h$ *sequences*. Informally, the $\tilde{B}_h$ sequences are the sequences that can still help distinguish the elements in a sum *even when the exact number of elements is unknown*. As long as there are at most h elements in this sum, we can know it from the sum and determine the elements.

Definition 2 ($\tilde{B}_h$ Sequence): Let $D = \{v_1, v_2, ..., v_\ell\} \subseteq \mathbb{N}^*$ be a sequence of positive integers. Then D is a $\tilde{B}_h$ *sequence* iff all the sums $v_{i_1} + v_{i_2} + \cdots + v_{i_{h'}}$ with $1 \leq h' \leq h$ and $1 \leq i_1 \leq \cdots \leq i_{h'} \leq \ell$ are distinct and differ from all the sums of more than h elements of D.

Example 4: Let $D = \{v_1, v_2\} = \{3, 4\} \subseteq \mathbb{N}^*$. Then all the sums of up to two elements are distinct, and also different from all the sums of at least three elements: the sums of at most two elements are $3, 4, 3+3 = 6, 3+4 = 7, 4+4 = 8$, while the sums of at least three elements are at least 9. Therefore, D is a $\tilde{B}_2$ sequence.

Exactly as in $B_h - CBF$, for an element $y \notin S$ we can deduce that indeed $y \notin S$ based on one of the counters if no more than h elements are hashed into this counter and any one of them don't have the same variable increment from D as y. We can now deduce the following lemma.

Lemma 1: While considering hash entries with at most h hashed elements, the false positive rate of $VI - CBF$ with a $\tilde{B}_h$ sequence $\tilde{D}$ equals the false positive rate of $B_h - CBF$ with a B_h sequence D (with the same other parameters m, n, k, ℓ) and is given by Equation (1).

C. A Simple Option for D: $D_L = [L, 2L-1]$

In the general case, to query whether an element is hashed into an entry, the implementation of the $B_h - CBF$ and $VI - CBF$ schemes requires the use of a predetermined two-dimensional binary table based on the set D (see Section IV). However, we will now present a set $D = D_L$ that *does not need such a lookup table*. Therefore, D_L is easier to implement in hardware.

In the next subsections, we first analyze the $VI - CBF$ scheme given $D = D_L$. In Section IV we show that for this case, no additional memory is required. We then provide an exact calculation of the false positive rate of this detailed scheme. We also show that it *always* improves the false positive rate of CBF given a number $m \geq 10$ of memory entries and a number n of inserted elements.

Let $L \geq 2$ be a positive integer of the form $L = 2^i$. We define the set D_L of size L as $D_L = [L, 2L-1] = \{L, L+1, ..., 2L-1\}$.

We now want to compute the false positive rate of the $VI-CBF$ scheme. First, if an element $y \notin S$ hashes into an entry counter $h_i(y)$ of value c, we want to determine the probability that we will be able to tell that $y \notin S$ given c. Note that the entry counter value c is defined as a sum of elements of D_L. To do so, we distinguish different values of c using the following lemma.

Lemma 2: Let y be an element whose i-th hash function $h_i(y)$ hashes into an entry of value c. If $\left(c - v_{g_i(y)}\right) \in (-\infty, -1] \cup [1, L-1]$ then $y \notin S$.

Proof: Intuitively, we want to determine when the counter value c can be written as a sum of increments in D that includes the increment $v_{g_i(y)} \in D$. It is only true when $c - v_{g_i(y)}$ is also a sum of increments from D. Therefore, We distinguish different values of c:

• If $c = 0$, then the number of elements in the sum is zero, and therefore $y \notin S$.

• If $c \in [L, 2L-1]$, we can deduce that c is composed of a single element of D_L, because the minimal value of a sum of two or more elements is $L + L = 2L$. Further, this element is of course c. Therefore if $v_{g_i(y)} \neq c$, then $y \notin S$.

• If $c \in [2L, ..., 3L-1]$, we must have that c is a sum of two elements, because the maximal value of one element is $2L-1$ and the minimal value of three elements is $3L$. For instance, $c = L + (c-L)$, or $c = (L+1) + (c-L-1)$, etc. Therefore, c can be comprised of any of the elements $\{L, L+1, ..., c-L\}$, but not of any $x \in \{c-L+1, ..., 2L-1\}$, since in such a case $(c-x) < L$. So if $v_{g_i(y)} \notin \{L, L+1, ..., c-L\}$ (i.e. $(c - v_{g_i(y)}) < L$), then $y \notin S$.

• If $c \geq 3L$, c can be comprised of any of the elements in D_L, since $(c - v_{g_i(y)}) > L$ for any $v_{g_i(y)} \in [L, 2L-1]$.

Summarizing the cases above, we cannot exclude that $y \in S$ if $c = v_{g_i(y)}$ or $(c - v_{g_i(y)}) \geq L$, hence the result. ∎

In the next theorem we present the false positive rate of this case.

Theorem 2: The false positive rate of the $VI-CBF$ scheme using $D = D_L$ is given by:

$$FPR = \left(1 - \left(1 - \frac{1}{m}\right)^{nk} - \frac{L-1}{L}\binom{nk}{1}\frac{1}{m}\left(1 - \frac{1}{m}\right)^{nk-1} - \frac{(L-1)(L+1)}{6L^2}\binom{nk}{2}\left(\frac{1}{m}\right)^2\left(1 - \frac{1}{m}\right)^{nk-2}\right)^k. \tag{4}$$

Proof: Let $y \notin S$ be an input to a query. As usual, since there are k hashed entries, the false positive rate FPR is given by

$$(1-p)^k, \tag{5}$$

where p is the probability that by considering one of the k hash entries used by hash function h_i, we can determine that $y \notin S$. Let X denote again the number of elements hashed into this entry and let c be the resulting counter value, i.e. the weighted sum of X elements of D_L. We distinguish several cases based on X.

As explained earlier, if $X = 0$ then clearly $y \notin S$.

If $X = 1$, then c has one of the L values of D_L, each with the same probability of $\frac{1}{L}$. Therefore c has one of the $L-1$ values that differ from $v_{g_i(y)}$ w.p. $\frac{L-1}{L} \cdot \Pr(X=1)$, in which case we can deduce that $y \notin S$.

If $X = 2$, there are L^2 possible ordered pairs of increments. Further, there are L^3 combinations of the values of the two increments and the corresponding increment $v_{g_i(y)}$ of the examined element, each having an equal probability of $1/L^3$. We now distinguish depending on the value of c. First, as explained in the proof of Lemma 2, if $c \in [2L, ..., 3L-1]$, there are exactly $c - 2L + 1$ options to obtain a sum of c with two addends of D_L: The possibilities are $\{(L) + (c-L), (L+1) + (c-L-1), ..., (c-L) + (L)\}$. Also, the sum of c cannot be comprised of $(3L - c - 1)$ of the values in D_L. Therefore, out of the L^3 combinations above, in

$$\sum_{c=2L}^{3L-1}(c-2L+1)(3L-c-1) = \sum_{i=0}^{L-1}(i+1)(L-i-1) = \sum_{i=1}^{L-1} i(L-i) \tag{6}$$

of them, we can determine that $y \notin S$. In addition, if $c \geq 3L$, then the value of X is not necessarily known and c can be comprised of any of the L values of D_L. Thus, the current entry is not used to determine that $y \notin S$.

If $X \geq 3$ then $c \geq 3L$ and the current entry is not used again to determine that $y \notin S$.

Combining all cases for X, we obtain the following formula for the probability p that we can determine $y \notin S$ using c.

$$p = \Pr(X=0) + \frac{L-1}{L}\Pr(X=1) + \frac{1}{L^3}\Pr(X=2)\sum_{i=1}^{L-1} i(L-i). \tag{7}$$

We use the formula $\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$ and get

$$\sum_{i=1}^{L-1} i(L-i) = L\sum_{i=1}^{L-1} i - \sum_{i=1}^{L-1} i^2 = \frac{1}{6}(L-1)L(L+1). \tag{8}$$

Simplifying p, using Equations (2) and (5), we obtain the result. ∎

D. Improving the False Positive Rate of CBF

We now demonstrate that $VI-CBF$ can always improve the false positive rate of CBF as the system is scaled. For a fair comparison, we assume that the two schemes use the same amount of memory for the same number of inserted elements. In addition, since this false positive rate computation does not take into account the counter overflow probability, we also show that *the $VI-CBF$ scheme always obtains a lower overflow probability bound than CBF*.

Assume that CBF uses four bits per counter (a common assumption, initially suggested by Fan et al. [1]). Let α denote the memory *bit-per-element ratio*, so that for every

number of elements n, the memory size is αn bits with $m = \lceil \frac{\alpha n}{4} \rceil$ counters. Then the following theorem compares the performances of $VI-CBF$ and CBF as n is scaled and *both schemes use the same memory size*.

Theorem 3: While keeping the same bit-per-element ratio $\alpha > 0$ (and, as a consequence, also the same total memory size), $VI-CBF$ satisfies the following properties when compared to CBF:
(i) $VI-CBF$ obtains a lower false positive rate than CBF with m counters for any $m \geq m_0 = 10$.
(ii) When α goes to infinity, i.e. the two systems are made increasingly efficient, the ratio of their false positive rates goes to 0.
(iii) $VI-CBF$ obtains a lower counter overflow probability bound than the classical bound for CBF from [1].

Proof: We first consider a CBF with m counters, n elements and k hash functions. Its false positive rate $FPR(CBF)$ is optimized when $k = \frac{m}{n}\ln(2)$ [1] and equals

$$FPR(CBF) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{\ln(2)m}\right)^k > \left(\frac{1}{2}\right)^k. \quad (9)$$

Incidentally, when k is not integer, it is possible to let each flow use either $k_1 = \lfloor k \rfloor$ hash functions with probability $p_k = \lceil k \rceil - k$ (i.e., as determined by an additional hash function), or $k_2 = \lceil k \rceil$ w.p. $1 - p_k$. The probability that an entry bit is equal to zero remains unchanged, but the resulting false positive rate is a weighted sum of those for k_1 and k_2. The proof then stays unchanged, simply using the weighted sums instead. In the remainder, we assume that k is integer.

As mentioned in [1], using four bits per counter in CBF suffices to obtain a negligible counter overflow probability of

$$\Pr\left(\max_i c(i) \geq 16\right) \leq m\left(\frac{enk}{16m}\right)^{16} \approx 1.37 \cdot 10^{-15} \cdot m. \quad (10)$$

Let's now consider a $VI-CBF$ with m' counters, n elements and k hash functions. Note that we use the same k as in CBF scheme for simplicity, but might further improve the false positive rate with a different k. We choose the parameter L of this scheme to be $L = 4$, and therefore $D_L = \{L, L+1, ..., 2L-1\} = \{4, 5, 6, 7\}$. The maximal counter increment is $\max(D_L) = 2L-1 = 7$. Instead of using four bits per counter as in CBF, we suggest to use seven bits per counter. For simplicity, we also assume that m satisfies that $\frac{4m}{7}$ is an integer. Therefore, given the same total number of memory bits, we use $m' = \lceil \frac{4m}{7} \rceil = \frac{4m}{7}$ counters, i.e. $\frac{4}{7}$ of the number of counters since the counters are larger in a factor of $\frac{7}{4}$. Let $r = 4/7$ denote the ratio in the number of counters, such that $m' = mr$. Likewise, let p denote again the probability that by considering one of the k hash entries used by hash function h_i, we can determine that $y \notin S$ and let X denote again the number of elements hashed into this entry.

We compare the false positive rate of $VI-CBF$ from Theorem 2 to the false positive rate of CBF as presented in Equation (9). In order to demonstrate that the false positive rate of $VI-CBF$ is lower than the false positive rate of CBF, we just need to prove that $p > \frac{1}{2}$, where

$$p = \left(1 - \frac{1}{m'}\right)^{nk} + \frac{L-1}{L}\binom{nk}{1}\frac{1}{m'}\left(1 - \frac{1}{m'}\right)^{nk-1} + \frac{(L-1)(L+1)}{6L^2}\binom{nk}{2}\left(\frac{1}{m'}\right)^2\left(1 - \frac{1}{m'}\right)^{nk-2} = \Pr(X=0) + \frac{L-1}{L}\Pr(X=1) + \frac{(L-1)(L+1)}{6L^2}\Pr(X=2). \quad (11)$$

Distinguishing the elements in the formula of p, we get

$$\Pr(X=0) = \left(1 - \frac{1}{m'}\right)^{nk} = \left(1 - \frac{1}{mr}\right)^{m\ln(2)} \geq \left(1 - \frac{1}{m_0 r}\right)^{m_0\ln(2)}, \quad (12)$$

where the last inequality is satisfied for $m \geq m_0$. Likewise,

$$\begin{aligned}\Pr(X=1) &= \binom{nk}{1}\frac{1}{m'}\left(1 - \frac{1}{m'}\right)^{nk-1} \quad (13)\\ &= \frac{m\ln(2)}{mr-1} \cdot \left(1 - \frac{1}{mr}\right)^{m\ln(2)}\\ &> \frac{\ln(2)}{r} \cdot \left(1 - \frac{1}{mr}\right)^{m\ln(2)}\\ &\geq \frac{\ln(2)}{r} \cdot \left(1 - \frac{1}{m_0 r}\right)^{m_0\ln(2)},\end{aligned}$$

where again the last inequality is again satisfied for $m \geq m_0$. Furthermore, for $m \geq m_0$,

$$\begin{aligned}\Pr(X=2) &= \binom{nk}{2}\left(\frac{1}{m'}\right)^2\left(1 - \frac{1}{m'}\right)^{nk-2} \quad (14)\\ &= \frac{m\ln(2)(m\ln(2)-1)}{2(mr-1)^2} \cdot \left(1 - \frac{1}{mr}\right)^{m\ln(2)}\\ &\geq \frac{m\ln(2)(m\ln(2)-1)}{2(mr)^2} \cdot \left(1 - \frac{1}{mr}\right)^{m\ln(2)}\\ &\geq \frac{\ln(2)(\ln(2) - \frac{1}{m_0})}{2r^2} \cdot \left(1 - \frac{1}{m_0 r}\right)^{m_0\ln(2)}.\end{aligned}$$

Therefore, combining Equations (11-14), we get

$$p \geq p_0 \triangleq \left(1 - \frac{1}{m_0 r}\right)^{m_0\ln(2)} \cdot \left(1 + \frac{L-1}{L} \cdot \frac{\ln(2)}{r} + \frac{(L-1)(L+1)}{6L^2} \cdot \frac{\ln(2)(\ln(2) - \frac{1}{m_0})}{2r^2}\right). \quad (15)$$

p_0 is a lower bound on p. For $L = 4, r = 4/7, m_0 = 10$, we get $p_0 > 0.5$ and the false positive rate is indeed improved. The analysis so far assumed CBFs with four bits per counter. The comparison of the false positive rate can be easily generalized to CBFs with d bits per counter for a general d. Then, for $L = 4$ and a maximal variable increment of $2L-1 = 7$, $VI-CBF$ can use counters of $d + \lceil \log_2(7) \rceil = d+3$ bits. Thus the ratio in the number of counters is $r(d) = d/(d+3)$. For larger d, the ratio $r(d)$ is larger and the relative reduction in the number of counters is smaller. Accordingly, the improvement in the false positive rate is more significant. We can see that for $d \geq 5$, the inequality $p_0 > 0.5$ is satisfied even for $m_0 = 5$. When $d = 3$, the improvement is smaller and $p_0 > 0.5$ is guaranteed for $m_0 = 17$.

Next, we can see that (for $p_0 > 0.5$) when α goes to infinity, the optimal number of hash functions for CBF also goes to infinity. Thus, we have

$$\frac{FPR(VI-CBF)}{FPR(CBF)} = \frac{(1-p)^k}{\left(1-\left(1-\frac{1}{m}\right)^{\ln(2)m}\right)^k} \leq \left(\frac{1-p_0}{1/2}\right)^k \xrightarrow{k\to\infty} 0. \tag{16}$$

To complete the proof, for CBFs of four bits per counter, we look at the counter overflow probability of $VI-CBF$, using seven bits per counter. For $i \in [1, m']$, let $\gamma(i)$ be the number of addends in the sum $c(i)$ of the corresponding counter. Clearly, the maximal value of a counter that does not yield to an overflow is 127.

We denote by $\gamma(i,j)$ (for $i \in [1,m'], j \in [1, \ell = 4]$) the number of addends in the sum $c(i)$ that equal v_j, s.t. $\gamma(i)$, the number of addends in the sum $c(i)$, satisfies $\gamma(i) = \sum_{j=1}^{\ell} \gamma(i,j)$. Since the maximal increment is $v_\ell = 2L-1 = 7$, we observe that if the i-th counter has encountered an overflow (i.e. $c(i) \geq 128$), then $\gamma(i) \geq \lceil \frac{128}{v_\ell} \rceil = \lceil \frac{128}{7} \rceil = 19$. Further, if $c(i) \geq 128$ and $\gamma(i) = 19$, then $\gamma(i,4) \geq 14$, i.e. at least 14 among the 19 addends in the sum $c(i)$ are $v_4 = 7$. Since all the elements of D are uniformly selected and the four events, $\{(\gamma(i,1) \geq 14, \gamma(i) = 19), (\gamma(i,2) \geq 14, \gamma(i) = 19), (\gamma(i,3) \geq 14, \gamma(i) = 19), (\gamma(i,4) \geq 14, \gamma(i) = 19)\}$ are disjoint, we must have that $\Pr(\gamma(i,4) \geq 14 | \gamma(i) = 19) \leq \frac{1}{\ell} = 0.25$.

We finally have (for $r = 4/7$) that

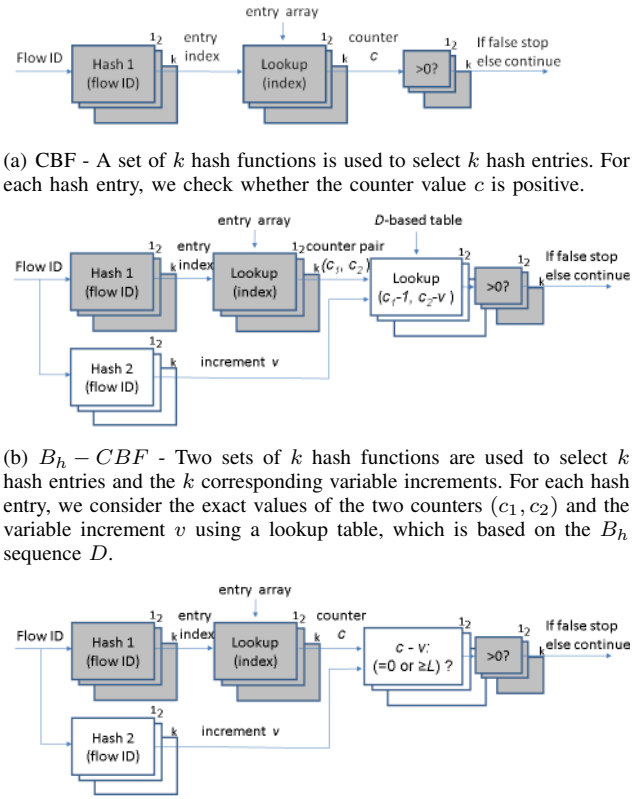
$$\begin{aligned}
\Pr\left(\max_i c(i) \geq 128\right) &\leq m' \Pr\left(c(i) \geq 128\right) \qquad (17)\\
&\leq m' \cdot \Big(\Pr\left(\gamma(i,4) \geq 14, \gamma(i) = 19\right) + \Pr(\gamma(i) \geq 20)\Big)\\
&\leq m' \cdot \Big(0.25 \cdot \Pr\left(\gamma(i) = 19\right) + \Pr(\gamma(i) \geq 20)\Big)\\
&\leq m' \cdot \Big(0.25 \cdot \Pr\left(\gamma(i) \geq 19\right) + \Pr(\gamma(i) \geq 20)\Big)\\
&\leq m' \cdot \left(0.25 \cdot \left(\frac{enk}{19m'}\right)^{19} + \left(\frac{enk}{20m'}\right)^{20}\right)\\
&= mr \cdot \left(0.25 \cdot \left(\frac{em\ln(2)}{19mr}\right)^{19} + \left(\frac{em\ln(2)}{20mr}\right)^{20}\right)\\
&= mr \cdot \left(0.25 \cdot \left(\frac{e\ln(2)}{19r}\right)^{19} + \left(\frac{e\ln(2)}{20r}\right)^{20}\right)\\
&\approx mr \cdot 1.104 \cdot 10^{-15} \approx 6.31 \cdot 10^{-16} \cdot m.
\end{aligned}$$

Comparing to the classical overflow probability bound of CBF (from Equation (10)), this upper bound of the overflow probability of $VI-CBF$ (with $r = 4/7$) is strictly lower. ■

IV. Implementation Considerations

In this section we discuss the implementation of the $B_h - CBF$ and $VI-CBF$ schemes in comparison with CBF. We consider several issues such as computational complexity, logic complexity, memory throughput and hashing complexity.

Computational Complexity — We consider the computational complexity of a query in the two suggested schemes.



(a) CBF - A set of k hash functions is used to select k hash entries. For each hash entry, we check whether the counter value c is positive.

(b) $B_h - CBF$ - Two sets of k hash functions are used to select k hash entries and the k corresponding variable increments. For each hash entry, we consider the exact values of the two counters (c_1, c_2) and the variable increment v using a lookup table, which is based on the B_h sequence D.

(c) $VI-CBF$ - Two sets of k hash functions are used to select k hash entries and the k corresponding variable increments. For each hash entry, we check the difference $(c-v)$ of the counter value c and the variable increment v.

Fig. 3. Logical View of Hardware Implementation. Components that also appear in CBF are presented in gray, while white ones are new.

In the $B_h - CBF$, let (c_1, c_2) be the values of the two counters and let $v \in D$ be the current variable increment. In order to efficiently determine whether the weighted sum c_2 can be comprised of v using exactly c_1 addends, we suggest using a predetermined two-dimensional binary table A that is based on D. The bit $A[i][j]$ is set if there is a sum of i addends from D that equals j. We notice that the value c_2 can be comprised of v using c_1 addends for $c_1 \geq 1$ only if there is a sum of exactly $c_1 - 1$ addends that equals the value $c_2 - v$, i.e. if the bit value $A[c_1-1][c_2-v]$ is set. The cases out of the table boundary ($c_1 = 0$ and $c_1 > h$) can easily be defined in the lookup procedure. In summary, if $c_1 > h$ or if the relevant bit value is set, $B_h - CBF$ continues to the next hash entry. Otherwise, it determines that the element is not in the set.

Note that $B_h - CBF$ enters table A only for cases where $c_1 \in [1,h]$. Therefore the first dimension index is $c_1 - 1 \leq h-1$ and the maximal possible value of the weighted sum is at most $(c_1-1)\cdot\max(D) \leq (h-1)\cdot v_\ell$. Thus, the memory size of the table A is at most $h\cdot((h-1)\cdot v_\ell + 1) \leq h^2 \cdot v_\ell$ bits. Note also that the set D is fixed, and therefore the binary table is also fixed and is calculated only once and not for each query. To calculate the table, we can simply go over all the multisets of up to h elements from D. As explained in Section VII, there are $\binom{h+\ell}{h}$ such multisets. Practically, both numbers are relatively small. For instance, for the B_h sequence $D = \{1, 4, 8, 13\}$ with $h =$

$3, \ell = 4$, we can calculate once the table by simply going over the $\binom{3+4}{3} = 35$ multisets. For these parameters, the size of the table is limited by $h \cdot ((h-1) \cdot v_\ell + 1) = 3 \cdot ((3-1) \cdot 13 + 1) = 81$ bits.

Next, we consider the complexity of the $VI-CBF$ scheme given the set $D = D_L$. As explained in Section III-C, a lookup table is not required in this case. Instead of the lookup operation, for a counter value c, we just check whether $c - v = 0$ or $c - v \geq L$ (Lemma 2). If so, $VI-CBF$ continues to the next hash entry. Otherwise, it determines that the element is not in the set.

To summarize, upon a query in both schemes we simply calculate the difference of a counter value and the variable increment. Then, we either compare it a to fixed bound or use it as an index to a lookup table. Heavy computations that include going over all possible sums or other complicated algorithms for judging whether an element can be a part of a given sum are not required upon query.

Pipeline Complexity — The insertion, deletion and query of each element can be organized in a pipeline manner, so that each operation is implemented at line rate. We assume for simplicity that there are no conflicts between elements at different steps of the pipeline.

Figure 3 illustrates the logical pipeline implemented to go through a query of packet x. It focuses on one of k parallel pipelines, shown on k parallel planes, and corresponding to the k hash entries.

Figure 3(a) presents the implementation of CBF in which the flow ID of the packet is hashed into one of the CBF array entries. The corresponding counter value c in considered. If it equals zero, CBF determines that $x \notin S$. Otherwise, CBF continues to check the next hash entry.

Figure 3(b) illustrates the implementation of $B_h - CBF$. Components that also appear in CBF are presented in gray, and the additional components in white. $B_h - CBF$ uses two hash functions instead of one. The first points to an entry in the $B_h - CBF$ array with the pair of counters (c_1, c_2). The second points to an increment from the set D denoted by v. These three values are then considered as described above.

Next, we consider the implementation of the $VI-CBF$ scheme given the set $D = D_L$ presented in Figure 3(c). As in the previous scheme, two hash functions are calculated. The first points to an entry in the $VI-CBF$ array with a counter c and the second points again to an increment $v \in D$. These two values are then considered without a lookup table as explained earlier.

Memory Throughput — In order to increase the memory throughput, we can implement the two schemes using the ideas implemented in the Blocked Bloom Filter [24]. For each element, all hash functions are mapped into a single block in the memory, i.e. a single memory word. Although this technique suffers from a higher false positive rate, it is clearly energy-efficient and improves the memory throughput, because there is a single memory word access instead of up to k.

Hashing Complexity — In each operation of insertion, deletion or query, the two suggested schemes require the computation of $2k$ hash functions instead of k in CBF. However, the total required number of random bits is much less than twice the number in CBF. In order to point to a random element in D, only $\lceil \log_2(|D|) \rceil = \lceil \log_2(\ell) \rceil$ random bits are required. Since the selection of D with small cardinality (such as $\ell = 4$) is enough to have an improved false positive rate, as explained in the proof of Theorem 3, we can reduce to two the number of random bits generated by each of $G = \{g_1, \ldots, g_k\}$ hash functions. In addition, given the same overall memory size, the number of counters in $VI-CBF$ is reduced by a factor of two, because they are twice larger, therefore they need one less bit for each counter selection. Thus, $VI-CBF$ typically needs $k \cdot (2 + (\lceil \log_2(m) \rceil - 1)) = k \cdot (\lceil \log_2(m) \rceil + 1)$ bits instead of $k \cdot (\lceil \log_2(m) \rceil)$ in CBF.

V. Combining with the *ML-HCBF*

In this section we show that $VI-CBF$ can be combined with the MultiLayer Hashed CBF scheme (*ML-HCBF*) [4] to further decrease its memory requirements for a requested false positive rate.

Instead of using one level with a constant number of bits per counter, *ML-HCBF* uses a hierarchical structure of several layers with narrower counters such that the number of counters is a decreasing function of the level number. The number of counters in the first level is a baseline number of counters, and their number in each additional level is based on the overflow probability of counters in the previous level. During insertion, a counter in the first level is examined. If it is not saturated, it is simply incremented. If it is saturated, the counter position is hashed to obtain an address of a counter in the next levels. If it is also saturated, the procedure continues in the next levels until a non-saturated counter is found.

A similar concept can be implemented using variable increments, as in $VI-CBF$. We denote it *ML-VI-HCBF*. We increment a counter in the first level by a variable increment till it reaches its maximal value. If it was saturated, we continue and increment another counter in the next level by the rest of the increment, such that the sum of their increments equals the requested one. By using perfect hashing as in [4], the actual value of a counter can be calculated as the sum of the corresponding counters in one or more levels. For instance, instead of using counters of 8 bits as for $VI-CBF$, we can have a hierarchical structure of 4 levels with $(4, 3, 3, 3)$ bits per counter, i.e. only four bits per counter in the first level and three bits per counter in all the other levels.

VI. Improvements of the $B_h - CBF$ Scheme

In this section we consider a possible improvement of the $B_h - CBF$ Scheme. We show that we can further decrease its false positive rate by considering also hash entries with more than h hashed elements.

Consider a hash entry $h_i(y)$ with more than h hashed elements. In such a case, the B_h sequence definition does not help anymore, because the elements in the sum $c_2(i)$ are not necessarily unique. However, we can still obtain some information in certain cases. By examining all the possible variable increments from the B_h sequence $D = \{v_1, v_2, ..., v_\ell\}$ that can lead to the sum $c_2(i)$, we can conclude that $y \notin S$ if $v_{g_i(y)}$ does not belong to any of these possible variable

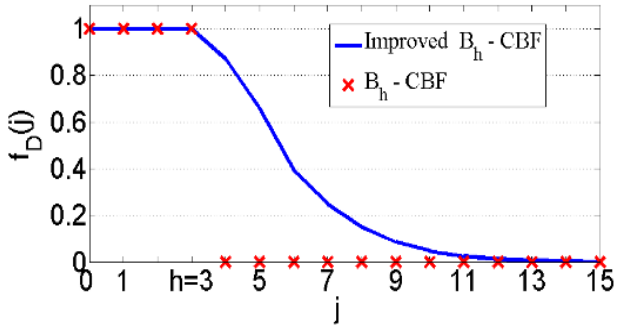


Fig. 4. The detection probability function $f_D(j)$ for the B_3 sequence $D = \{1, 4, 8, 13\}$. For the $B_h - CBF$ scheme, $f_D(j) = 0$ for $j > h$. For the improved $B_h - CBF$ scheme, $f_D(j) > 0$ for some $j > h$.

increments. For a given number of bits per hash entry, this improvement enables to reduce the false positive rate while keeping the same counter overflow probability. To implement this improvement using the suggested two-dimensional binary table, a slightly larger table is required. The additional entries of the table stand for sums of more then h addends.

Let v_i be a uniformly-distributed element of $D = \{v_1, v_2, ..., v_\ell\}$. Let c be a sum of j uniformly-distributed elements of $D \setminus \{v_i\}$. With some probability, we can determine that c is not comprised of v_i. Given a B_h sequence D, we denote by $f_D(j)$ the probability that we can indicate this fact based on the values c and j. The function $f_D(j)$ is called the *detection probability* function. Of course, since D is a B_h sequence, for any $j \in [0, h]$, there is only one sum of j elements that equals c. Thus, this sum must have the property that its elements are all in $D \setminus \{v_i\}$. Therefore, we must have that $f_D(j) = 1$ for $j \in [0, h]$. In the previously presented $B_h - CBF$ scheme, $f_D(j) = 0$ for $j > h$, since we cannot leverage the properties of the B_h sequence in such cases. In the improved scheme, $f_D(j)$ equals the probability that all the sums of j elements of D which equal c are not comprised of v_i. In all such cases, we can deduce that c is not comprised of v_i.

Using the last definition, we can now present the false positive rate of the improved scheme.

$$\left(1 - \sum_{j=0}^{\infty} \Pr(X = j)\left(1 - \frac{1}{\ell}\right)^j f_D(j)\right)^k \tag{18}$$
$$= \left(1 - \sum_{j=0}^{h} \binom{nk}{j} \left(\frac{\ell - 1}{\ell m}\right)^j \left(1 - \frac{1}{m}\right)^{nk-j} - \sum_{j=h+1}^{\infty} \binom{nk}{j} \left(\frac{\ell - 1}{\ell m}\right)^j \left(1 - \frac{1}{m}\right)^{nk-j} f_D(j)\right)^k.$$

From the last statement, it is clear that this scheme has an improved false positive rate in comparison with the basic $B_h - CBF$ scheme. The false positive rate of the basic $B_h - CBF$ scheme presented in Equation (1), can be achieved from Equation (18) using $f_D(j) = 1$ for $j \in [0, h]$ and $f_D(j) = 0$ for $j > h$ since with this scheme we do not consider any hash entry with more than h hashed elements.

Example 5: We look again at the B_3 sequence $D = \{1, 4, 8, 13\}$ from Example 1. Figure 4 presents $f_D(j)$ for $j \in [0, 15]$. As explained earlier, for the basic $B_h - CBF$ scheme, we have that $f_D(j) = 1$ for $j \in [0, h] = [0, 3]$ and $f_D(j) = 0$ for $j > h$. We now try to calculate $f_D(j)$ for our improved $B_h - CBF$ scheme. As explained earlier, $f_D(j) = 1$ for $j \in [0, h] = [0, 3]$. To calculate $f_D(4)$, for instance, we consider the $4^4 = 256$ possible combinations of four ordered elements of D. Among them there are, for each of the four values of D, 3^4 (ordered) combinations in which it does not appear. There is a total number of $4 \cdot (3^4) = 324$ cases of four addends $v_{i_1}, v_{i_2}, v_{i_3}, v_{i_4} \in D \setminus \{v_t\}$ for some $t \in [1, \ell]$. We denote by c the sum $v_{i_1} + v_{i_2} + v_{i_3} + v_{i_4}$. Among these cases, there are 282 cases in which any sum of four addends that equals c is not comprised of v_t. Therefore, $f_D(4) = \frac{282}{324} \approx 0.87$. Likewise, we can obtain that $f_D(8) = \frac{3916}{26244} \approx 0.15$. We can now use these results to examine a query of an element $y \notin S$. Consider a hash entry $h_i(y)$ with counters values $c_1(i)$ and $c_2(i)$. If $c_2(i)$ is not comprised of $v_{g_i(y)}$ and $c_1(i) = h + 1 = 4$, we can determine that $y \notin S$ based on the current hash entry with probability of about 0.87.

Example 6: Consider the query of the element z from Example 3, as illustrated in Figure 2(b). Although $c_1^z(1) = 4 > h$, we can notice that $c_2^z(1) = 30$, as a sum of $c_1^z(1) = 4$ elements from D, cannot be comprised of $v_{g_1(z)} = 4$. This is because $c_2^z(1) - v_{g_1(z)} = 30 - 4 = 26$ cannot be comprised of exactly $c_1^z(1) - 1 = 4 - 1 = 3$ addends from the B_3 sequence D, as illustrated in Example 1. Thus, we can determine that $z \notin S$, avoiding a false positive.

VII. Lower Bounds on the Coding Schemes

We now consider the $B_h - CBF$ scheme that uses a B_h sequence D and the $VI - CBF$ scheme with the option of a B_h sequence D. A fundamental view of these algorithms is that in each entry, using either one or two counters, they represent a multiset of up to h elements taken from a space of size $|D| = \ell$. This can provide us a lower bound on the needed number of bits for the representation of a hash entry.

As an example, consider again the $B_h - CBF$ scheme with the B_3 sequence $D = \{v_1, v_2, v_3, v_4\} = \{1, 4, 8, 13\}$ with $|D| = \ell = 4$. There are 20 multisets of exactly three elements from D (as illustrated in Example 1), 10 multisets of two elements and of course ℓ multisets of size one. The empty set is also a subset of D. Thus there are 35 multisets of up to three elements from D. Therefore at least $\lceil \log_2(35) \rceil = 6$ bits are required to represent the information in such cases.

More generally, in order to represent differently all these multisets, we must have a number of bits that equals at least the base-2 logarithm of the number of possible multisets. In particular, since the number of multisets of up to h elements from a set of size ℓ equals $\phi(h, l) = \binom{h+\ell}{h}$, we obtain a lower bound of

$$\lceil \log_2 \phi(h, l) \rceil = \left\lceil \log_2 \binom{h + \ell}{h} \right\rceil \tag{19}$$

on the number of bits per entry. For instance, as we explained with $h = 3$ and $\ell = 4$, at least $\lceil \log_2 \phi(h, l) \rceil = \lceil \log_2 \binom{3+4}{3} \rceil =$

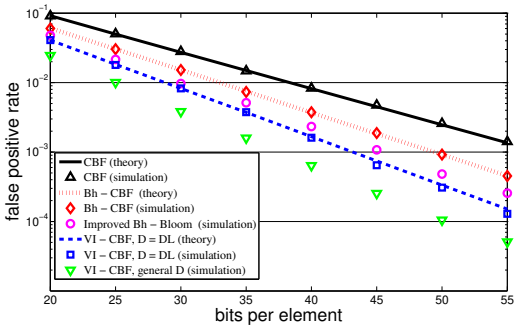


Fig. 5. Comparison of the false positive rates of the CBF, $VI-CBF$ and B_h-CBF schemes and the theoretical models from Theorem 1 and Theorem 2.

$\lceil \log_2 \binom{7}{3} \rceil = \lceil \log_2 35 \rceil = 6$ bits are required for each hash entry.

However, while these provide attractive lower bounds, they might lack additional properties of the B_h-CBF and the $VI-CBF$ schemes that are essential for fast hardware implementation: in particular, during insertions and deletions, they first need to decode the value before re-encoding the next value, which is not needed in the schemes presented in this paper. Of course, special codes that can be incremented iteratively may be suggested. Further, by allocating more bits per entry, these two schemes have a relatively low counter overflow probability, even when a hash entry represents more than h elements.

VIII. Experimental Results

A. Trace-Driven Simulations

We conduct experiments using real-life traces recorded on a single direction of an OC192 backbone link [25], and rely on a 64-bit mix hash function [26] of the IP 5-tuple to implement the requested hash functions.

Figure 5 plots the false positive rate of various schemes as a function of the memory size (in bits per element). It also compares the experimental values with the values obtained by theory.

For B_h-CBF, we use a B_h sequence $D=\{v_1,...,v_\ell\}$ with $h=3$ and $\ell=4$. We compare the false positive rate of the basic scheme with Theorem 1. We remind that in the (basic) B_h-CBF scheme, the false positive rate does not depend on the exact values of $v_1,\cdots,v_\ell$ but only on the parameters h,ℓ. We also present the results of the improved B_h-CBF scheme that we introduced after Theorem 1. For this scheme we use a specific B_h sequence with the above parameters $D=\{1,4,13,15\}$, which was found to yield a low false positive rate on synthetic inputs.

For $VI-CBF$, we first use $D=D_L$ with $L=4$, where the false positive rate is given by Theorem 2, and later also use the more general set $D=\{8,12,14,15\}$ (that was again found to be a good set) to compare the two sets. We classically assume four bits per counter for CBF, and for the two B_h-CBF schemes, twelve bits per hash entry (four and eight bits for the fixed-increment and variable-increment counters, respectively). For the $VI-CBF$ scheme, we use seven bits per counter when $D=D_4$ and eight bits per counter for the general D.

First, the simulation results confirm the theory from Theorem 1 and Theorem 2. Further, all the suggested schemes improve upon CBF. In addition, the two variants of $VI-CBF$ outperform the B_h-CBF scheme and the improved B_h-CBF scheme. $VI-CBF$ with the general set D yields the best performance, and the improvement is especially significant for larger numbers of bits per element. For instance, for 30 bits per element, the false positive rate for CBF, B_h-CBF, improved B_h-CBF, $VI-CBF$ with $D=D_L$ and $VI-CBF$ with a general D are 0.02803, 0.01521, 0.00970, 0.00825 and 0.00383, respectively, thus obtaining an improvement by a factor of 7. Likewise, for 50 bits we have 0.00258, 0.00092, 0.00048, 0.00031 and 0.00011 respectively, thus improving the result by over an order of magnitude. Of course, as mentioned, $VI-CBF$ with $D=D_L$ is easier to implement in hardware than $VI-CBF$ with a general D, hence there is a clear tradeoff between efficiency and complexity.

Alternatively, for the same false positive rate of CBF with 50 bits per element, $VI-CBF$ with a general D requires approximately 32 bits per element, hence a reduction of about a third in the memory requirement.

B. Comparison with State-of-the-Art Algorithms

We adopt the same settings as in [4] to examine the memory requirements of our suggested schemes in comparison with several well-known schemes such as CBF [1], Spectral BF [19], dlCBF [3] and *ML-HCBF* [4]. Specifically, we compare the total memory size required to obtain a false positive rate of 10^{-3} when $|S|=n=2000$ elements are represented.

The performance results of the previous schemes are taken from [4]. We present the results of six new schemes: B_h-CBF, the improved B_h-CBF scheme, $VI-CBF$ with $D=D_L$, $VI-CBF$ with a general D and each of these two versions of *ML-VI-HCBF*, i.e. $VI-CBF$ combined with *ML-HCBF*. For B_h-CBF and $VI-CBF$, we use the same set D and the same number of bits per counter as in the previous simulation. Table I summarizes the results.

For the new schemes, we consider the additional size of the lookup table, when required. For instance, as explained in Section IV, B_h-CBF requires a table of size $h^2\cdot v_\ell = 3^3\cdot 15 = 135$ bits ≈ 0.02 KB. For the improved B_h-CBF scheme we base, in this simulation, the query decision on counters with up to $2\cdot h=6$. Thus, the table size is $(2h)^2\cdot v_\ell \approx 0.07$ KB. For the $VI-CBF$ scheme with the set $D=D_L$, a lookup table is not required. For the general set D, the size is at most $15\cdot 15=225$ bits ≈ 0.03 KB.

In B_h-CBF, the required memory size is 12.09 KB, and it drops to 11.38 KB in the improved B_h-CBF scheme. $VI-CBF$ with $D=D_4$ requires 12842 counters of seven bits, while with the general D, 9500 counters of eight bits are used in addition to the lookup table. This yields a total memory size of 10.97 KB and 9.31 KB, respectively, i.e. improvements of 22.2% and 34.0% in comparison with CBF.

	CBF	Spectral BF	dlCBF	*ML-HCBF*	B_h-*CBF*	Improved B_h-*CBF*	*VI-HCBF,* $D = D_L$	*ML-VI-CBF,* $D = D_L$	*VI-CBF,* general D	*ML-VI-HCBF,* general D
Main Structure (KB)	14.1	8.12	5.2	7.14	12.07	11.31	10.97	6.27	9.28	5.80
Secondary Structure (KB)	-	-	-	0.41	-	-	-	0.33	-	0.82
Additional Tables (KB)	-	4	-	-	0.02	0.07	-	-	0.03	0.03
Total Size (KB)	14.1	12.12	5.2	7.55	12.09	11.38	10.97	6.60	9.31	6.65
False Positive Rate	10^{-3}	10^{-3}	$1.5 \cdot 10^{-3}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}

TABLE I

MEMORY REQUIREMENTS COMPARISON. THE RESULTS OF FOUR STATE-OF-THE-ART ALGORITHMS ARE PRESENTED ON THE LEFT SIDE OF THE TABLE, AS IN [4], AND THE RESULTS OF SIX SUGGESTED SCHEMES APPEAR ON THE RIGHT SIDE.

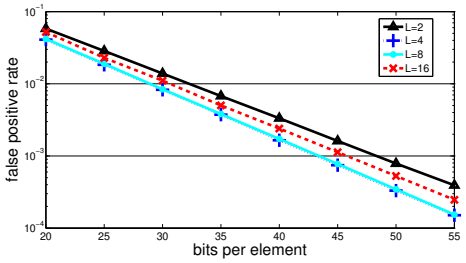


Fig. 6. False positive rate of $VI-CBF$ with $D = D_L$ for various L values.

All these schemes present lower memory requirements than CBF and the Spectral BF, but higher than those of the dlCBF. However, dlCBF may have additional issues, like overflows and complexity [4]. As mentioned in the Introduction, improving dlCBF in the same way by using variable increments is left to future work.

For *ML-VI-HCBF* with $D = D_4$, we use four layers with 12842 counters in the first level (as in the original scheme), 802 in the second, 100 in the third and 6 in the fourth levels. In these four levels, $4, 3, 3$ and 3 bits per counter were used, respectively. *ML-VI-HCBF* uses a total memory size of 6.60 KB, an improvement of 40% in comparison with $VI-CBF$, and a 2x improvement (precisely, 53%) in comparison with CBF. Likewise, when the set $D = \{8, 12, 14, 15\}$ is considered, we have four levels of $5, 5, 5, 4$ bits with an almost similar memory consumption of 6.65 KB. Both of these hierarchical schemes also perform better than *ML-HCBF*.

C. Optimizing the $VI-CBF$ Parameters

We want to examine the effect of the parameter L for the $VI-CBF$ scheme with $D = D_L$ on the false positive rate. On the one hand, increasing L makes it easier to exclude membership of a non-member element based on one counter, while on the other hand it requires more bits per counter and thus reduces their number.

We now assume a set with $n = 1024$ elements and variable numbers of bits per elements. We examine the values $2, 4, 8, 16$ for L and for each value we use the optimal number k of hashed functions. Further, for each value of L, we use $4 + \lceil \log_2(v_\ell) \rceil = 4 + \lceil \log_2(2L-1) \rceil$ bits per counter. For instance, for $L = 4$, we have $4 + 3 = 7$ bits. The false positive rates of the optimal values of k are presented in Figure 6.

We can see that the performances of D_L with $L = 4$ and $L = 8$ are similar and are much better than the cases of $L = 2$ and $L = 16$. For example, for 30 bits per element we have false positive rates of $0.01388, 0.00825, 0.00841, 0.01105$ for $L = 2, 4, 8, 16$, respectively.

D. Optimizing the $B_h - CBF$ parameters

In this section we provide several simulation results in order to examine the $B_h - CBF$ scheme. For instance, we show how (a) we can pick an optimal B_h sequence among all such sequences that satisfy the given constraints; (b) we can decode the information provided by the variable-increment counter in case there are more than h elements; (c) we can trade off the false positive rate against the overflow probability by considering different entry sizes; and (d) we can choose an optimized coding of the counters by coding their difference with their expected values.

We first try to find a B_h sequence that minimizes the false positive rate of the $B_h - CBF$ scheme. Based on Theorem 1, for any values of k, m, n, the false positive rate is always decreased when h, ℓ are increased. Therefore, given k, m, n, the false positive rate is minimized when $h, \ell \to \infty$. Unfortunately, we encounter some constraints on their values. We also observe that according to this theorem the false positive rate of the $B_h - CBF$ scheme depends only on the parameters h, ℓ of the B_h sequence and not on the specific values of $\{v_1, v_2, ..., v_\ell\}$. As explained later, this is not the case for the improved $B_h - CBF$ scheme. Upon insertion of an element x, the counter with variable increments is incremented by the element $v_{g_i(x)} \in D$, thus the increment is at most v_ℓ. Since each of the counters has a fixed size we would like to bound this increment. For example, we can bound the maximal increment to be at most $2^4 - 1 = 15$, i.e. require that $v_\ell \leq 15$. In such a case, we can deduce that the variable-increment counter needs approximately four more bits than the typical CBF counter size of four bits, i.e. it requires a total of about eight bits. Now, with such a constraint, we have a tradeoff: we can choose a B_h sequence with large h and small ℓ or vice versa. In our case, there are four different possibilities: First, we can have a B_h sequence with $\ell = 15, h = 1$, i.e. a B_1 sequence of all the 15 elements of $[1, 15]$. Second, we can use a B_2 sequence, which has a maximal size of $\ell = 5$. Similarly, there is a maximal size of $\ell = 4$ for $h = 3$. We can also find a B_{12} sequence of size $\ell = 3$. We assume the following parameters: $n = 1024$ and the number of bits per elements is

TABLE II
B_h VARIATIONS WITH THE CONSTRAINT $v_\ell \leq 15$

h, ℓ	B_h Example	Optimal k	FPR (for 30 bits per element)
$h = 1, \ell = 15$	[1,15]	2	0.04630
$h = 2, \ell = 5$	{1,2,4,8,13}	4	0.01963
$h = 3, \ell = 4$	{1,2,5,14}	5	0.01521
$h = 12, \ell = 3$	{1,2,15}	5	0.02902

$bpe = 30$, i.e. the total memory size is $n \cdot bpe$ bits. For each of these four possibilities of the parameters of the B_h sequences, a minimal false positive rate is achieved for some value of k.

The theoretical results, based on Theorem 1, are summarized in Table II where an example of a B_h of each of the four cases is also given. We can see that the minimal false positive rate is achieved for $h = 3, \ell = 4$ and $k = 5$.

To further reduce the false positive rate, we can also exploit the information provided by the variable-increment counter in case there are more than h elements, as presented in Section VI. To implement this generalization, we suggest again using a predetermined two-dimensional binary table with more entries than the original table described earlier in Section IV.

With this improvement, the false positive rate is affected by the exact values of $\{v_1, v_2, ..., v_\ell\}$. Thus, we want to examine the possible B_h sequences with $h = 3, \ell = 4, v_\ell \leq 15$. Without loss of generality, we assume that $v_1 = 1$. Otherwise, given a general B_h sequence $D = \{v_1, v_2, ..., v_\ell\}$, we can just have the same false positive rate using the B_h sequence $\{v_1 - (v_1 - 1), v_2 - (v_1 - 1), ..., v_\ell - (v_1 - 1)\}$. There are 48 such B_h sequences. For each of them, the false positive rate is smaller than the result of Theorem 1 that appears in Table II. For the parameters above, the minimal simulated false positive rate is achieved for the B_h sequence $D = \{1, 4, 13, 15\}$ and equals 0.00970, which is lower by 36% than the unoptimized value of 0.01521.

There exists a tradeoff between the false positive rate and the overflow probability. In fact, decreasing the number of bits per variable-increment counter enables to store more counters, thus decreasing the false positive rate, but also running into overflow more frequently. To illustrate the tradeoff, we consider again the parameters above, $n = 1024$ with 30 bits per element and $k = 5$. Using four bits per fixed-increment counter and six bits per variable-increment counter (i.e. $4 + 6 = 10$ bits for each hash entry), we have a false positive rate of 0.00475 and a counter overflow probability of 0.00234. With three and five bits (respectively), there are more counters and the false positive rate falls to 0.00262, at the expense of a greater overflow probability of 0.05726. We notice that the overflow probability can be larger than the false positive rate, since it is calculated as the ratio of counters with overflow. However, a false answer to a query can be returned based on one of the counters, even if some of the other counters have encountered an overflow.

E. Lower Bounds on the False Positive Rate

We now consider again the $B_h - CBF$ scheme with a B_h sequence D and the $VI - CBF$ scheme with the option of a $\tilde{B}_h$ sequence D. We want to compare their false positive rates

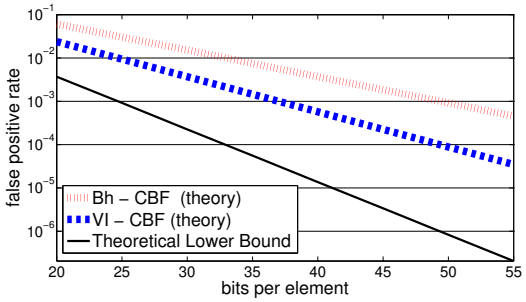


Fig. 7. Comparison of the false positive rates of $B_h - CBF$, $VI - CBF$ with the option of a $\tilde{B}_h$ sequence D and the theoretical lower bound on their false positive rate based on Section VII.

to the false positive rate achieved when hash entries that satisfy the theoretical lower bound on their sizes from Section VII are used. For the $B_h - CBF$ scheme, we use again the B_h sequence $D = \{1, 2, 5, 14\}$ that satisfy $h = 3, \ell = 4$ that was previously found in Section VIII-D to achieve the minimal false positive rate for $v_\ell \leq 15$. As in similar cases before, twelve bits per hash entry (four and eight bits for the fixed-increment and variable-increment counters, respectively) are required for the $B_h - CBF$ scheme in this case.

For the $VI - CBF$ scheme with the option of a $\tilde{B}_h$ sequence D, we would like to use less bits per hash entry since each hash entry has only a single variable-increment counter. We found out that there exist $\tilde{B}_h$ sequences that satisfy $h = 3, \ell = 4$ and $v_\ell \leq 15$. An example of such a $\tilde{B}_h$ sequence is the set $D = \{30, 35, 37, 43\}$. In addition to the four elements of D themselves, the 10 sums of 2 elements from D are $30+30 = 60, 30+35 = 65, 30+37 = 67, 30+43 = 73, 35+35 = 70, 35+37 = 72, 35+43 = 78, 37+37 = 74, 37+43 = 80, 43+43 = 86$. Likewise, the 20 sums of 3 elements are $90, 95, 97, 100, 102, 103, 104, 105, 107, 108, 109, 110, 111, 113, 115, 116, 117, 121, 123$ and 129. We can see that all these values are distinct. Further, we can also see that any of these values cannot be presented as a sum of four or more elements from D. A sum of exactly four elements is at least 120 but cannot equal $121, 123$ or 129 and a sum of five or more elements is at least 150. Thus $D = \{30, 35, 37, 43\}$ is a $\tilde{B}_3$ sequence. Since $43 \leq 63$, we use in this scheme ten bits per hash entry, six more bits than the typical CBF counter size of four bits.

We remind that from Section VIII-D, the theoretical lower bound on the entry size of these two schemes is $\lceil \log_2 \phi(h, l) \rceil = \lceil \log_2 \binom{3+4}{3} \rceil = \lceil \log_2 35 \rceil = 6$ bits.

Since the false positive rate of these two schemes is given by Equation (1), a lower bound on the false positive rate is achieved when the number of hash entries is calculated based on the lower bound of the size in bits of a hash entry. Figure 7 compares the false positive rate of these two schemes with their theoretical lower bound as a function of the memory size (in bits per element). For $B_h - CBF$, the false positive rate is the same as in Figure 5. The false positive rate of $VI - CBF$ with a $\tilde{B}_h$ sequence is slightly better. However, it is still larger by more than an order of magnitude than the theoretical lower bound. For instance, for 30 bits per elements

the respective false positive rates are 0.01521 and 0.00369 while the lower bound on the probability is 0.00022.

IX. Conclusion

In this paper we presented a novel method based on *variable increments* to improve the efficiency of CBFs and their variants in networking devices. We showed that it can be efficiently implemented in hardware with limited added complexity. We also demonstrated that this method can always achieve a lower false positive rate and a lower overflow probability bound than CBF in practical systems. More generally, we explained how this method can extend many variants of CBF published in the literature.

To our knowledge, this is the first time that B_h sequences are used in network applications. We believe that this is a first step towards a more general use, because they seem to fit increasingly-complex coding needs in networking applications. These applications often require both a scalability in the number of states to encode, yet also low hardware complexity—and indeed, as explained in this paper, B_h sequences efficiently compress sets of h states, yet can also be readily encoded and decoded at line rates using fixed translation tables.

X. Acknowledgment

We would like to thank David Hay for his helpful suggestions. This work was partly supported by the European Research Council Starting Grant No. 210389, by the Intel ICRI-CI Center, by the Hasso Plattner Center for Scalable Computing, by the Japan Technion Society and Greenberg (Ottawa) Research Funds, by the Google Europe Fellowship in Computer Networking, by the Andrew Viterbi graduate fellowship, and by the Israel Ministry of Science and Technology.

Ori Rottenstreich received the B.S. degree in computer engineering (summa cum laude) from the electrical engineering department of the Technion, Haifa, Israel in 2008. He is now pursuing a Ph.D. degree in the same department. He is mainly interested in computer networks as well as in algorithm design and analysis. Ori Rottenstreich is a recipient of the Google Europe Fellowship in Computer Networking, the Andrew Viterbi graduate fellowship, the Jacobs-Qualcomm fellowship, the Intel graduate fellowship and the Gutwirth Memorial fellowship. He is a co-recipient of the Best Paper Runner Up Award at the IEEE Infocom 2013 conference.

Yossi Kanizo received the B.S. degree in computer engineering from the computer science department of the Technion, Haifa, Israel in 2006. He is now pursuing a Ph.D. degree in the same department. He is mainly interested in computer networks, hash-based data-structures, and switch architectures.

Isaac Keslassy (M'02, SM'11) received his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 2000 and 2004, respectively.

He is currently an associate professor in the Electrical Engineering department of the Technion, Israel. His recent research interests include the design and analysis of high-performance routers and multi-core architectures. The recipient of the European Research Council Starting Grant, the Alon Fellowship, the Mani Teaching Award and the Yanai Teaching Award, he is an associate editor for the IEEE/ACM Transactions on Networking.

References

[1] L. Fan *et al.*, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. on Networking*, vol. 8, no. 3, 2000.
[2] F. Bonomi *et al.*, "Beyond Bloom filters: from approximate membership checks to approximate state machines," in *ACM SIGCOMM*, 2006.
[3] ——, "An improved construction for counting Bloom filters," in *ESA*, 2006.
[4] D. Ficara *et al.*, "Enhancing counting Bloom filters through Huffman-coded multilayer structures," *IEEE/ACM Trans. on Networking*, vol. 18, no. 6, 2010.
[5] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using Bloom filters," *IEEE/ACM Trans. on Networking*, vol. 14, no. 2, 2006.
[6] Y. Lu *et al.*, "Counter braids: a novel counter architecture for per-flow measurement," in *ACM SIGMETRICS*, 2008.
[7] H. Song *et al.*, "IPv6 lookups using distributed and load balanced Bloom filters for 100Gbps core router line cards," in *IEEE Infocom*, 2009.
[8] "Mellanox IB QDR 324P Switch System - Overview." [Online]. Available: http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?lang=en&cc=us&taskId=120&prodSeriesId=5033638&prodTypeId=12883&objectID=c01775272
[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM Operating Systems Review*, vol. 44, no. 2, 2010.
[10] "Google Chrome safe browsing." [Online]. Available: http://src.chromium.org/viewvc/chrome/trunk/src/chrome/browser/safe_browsing/
[11] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.
[12] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *FAST*, 2002.
[13] "Squid Web Proxy Cache." [Online]. Available: http://www.squid-cache.org/
[14] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, 1970.
[15] L. Li, B. Wang, and J. Lan, "A variable length counting Bloom filter," in *2nd Int'l Conf. on Computer Engineering and Technology*, 2010.
[16] O. Rottenstreich and I. Keslassy, "The Bloom paradox: When not to use a Bloom filter?" in *IEEE Infocom*, 2012.
[17] M. Mitzenmacher, "Compressed Bloom filters," *IEEE/ACM Trans. on Networking*, vol. 10, no. 5, 2002.
[18] Y. Kanizo, D. Hay, and I. Keslassy, "Access-efficient balanced Bloom filters," in *IEEE ICC*, 2012.
[19] S. Cohen and Y. Matias, "Spectral Bloom filters," in *ACM SIGMOD*, 2003.
[20] S. Graham, "B_h sequences," *Analytic Number Theory*, vol. 1 (Allerton Park, IL, 1995).
[21] K. OBryant, "A complete annotated bibliography of work related to Sidon sequences," *Electron. J. Combin Dynamic Survey 11*, 2004.
[22] "Bh[g]-sequences with large upper density," *Journal of Number Theory*, vol. 56, no. 2, pp. 298 – 308, 1996.
[23] A. Kirsch, M. Mitzenmacher, and G. Varghese, "Hash-based techniques for high-speed packet processing," in *Algorithms for Next Generation Networks, Springer-Verlag*, 2009.
[24] F. Putze, P. Sanders, and J. Singler, "Cache-, hash- and space-efficient Bloom filters," in *Workshop on Experimental Algorithms*, 2007.
[25] C. Shannon *et al.*, "CAIDA anonymized 2008 Internet trace equinix-chicago 2008-03-19 19:00-20:00 UTC (DITL) (collection)," http://imdc.datcat.org/collection/.
[26] T. Wang, "Integer hash function," http://www.concentric.net/~Ttwang/tech/inthash.htm.