Avoiding Flow Size Overestimation in Count-Min Sketch With Bloom Filter Constructions

Ori Rottenstreich, *Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE*, Ely Porat, and S. Muthukrishnan

***Abstract*—The Count-Min sketch is the most popular data structure for flow size estimation, a basic measurement task required in many networks. Typically the number of potential flows is large, eliminating the possibility to maintain a counter per flow within memory of high access rate. The Count-Min sketch is probabilistic and relies on mapping each flow to multiple counters through hashing. This implies potential estimation error such that the size of a flow is overestimated when all flow counters are shared with other flows with observed traffic. Although the error in the estimation can be probabilistically bounded, many applications can benefit from accurate flow size estimation and the guarantee to completely avoid overestimation. We describe a design of the Count-Min sketch with accurate estimations whenever the number of flows with observed traffic follows a known bound, regardless of the identity of these particular flows. We make use of a concept of Bloom filters that avoid false positives and indicate the limitations of existing Bloom filter designs towards accurate size estimation. We suggest new Bloom filter constructions that allow scalability with the support for a larger number of flows and explain how these can imply the unique guarantee of accurate flow size estimation in the well known Count-Min sketch.**

***Index Terms*—Network algorithms, measurement, bloom filter, Count-Min sketch.**

Manuscript received October 6, 2020; revised January 8, 2021; accepted March 16, 2021. Date of publication March 25, 2021; date of current version September 9, 2021. Ori Rottenstreich was partially supported by the German-Israeli Foundation for Scientific Research and Development (GIF), by the Gordon Fund for System Engineering as well as by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel National Cyber Directorate. Pedro Reviriego would like to acknowledge the support of the ACHILLES project PID2019-104207RB-I00 and the Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Science and Innovation and of the Madrid Community research project TAPIR-CM grant no. P2018/TCS-4496. A preliminary version of this paper appeared at *ACM Symposium on SDN Research* (*SOSR*) '20, San Jose, California, March 2020 [1]. The associate editor coordinating the review of this article and approving it for publication was S. Kanhere. *(Corresponding author: Ori Rottenstreich.)*

Ori Rottenstreich is with the Taub Department of Computer Science, Technion, Haifa 3200003, Israel, and also with the Viterbi Department of Electrical Engineering, Technion, Haifa 3200003, Israel (e-mail: or@technion.ac.il).

Pedro Reviriego is with the Departamento de Ingenería Telemática, Universidad Carlos III de Madrid (Leganés), 28911 Madrid, Spain (e-mail: revirieg@it.uc3m.es).

Ely Porat is with the Department of Computer Science, Bar Ilan University, Ramat Gan 5290002, Israel (e-mail: porately@cs.biu.ac.il).

S. Muthukrishnan is with the Department of Computer Science, Rutgers University, New Brunswick, NJ 08854, USA (e-mail: muthu@cs.rutgers.edu).

Digital Object Identifier 10.1109/TNSM.2021.3068604

I. Introduction

A. Background

ESTIMATING flow size is a required capability in many networking applications, in fields as diverse as accounting, monitoring, load balancing and filtering, and in other streaming applications beyond networking. Counting the exact size for every flow is often challenging due to a typically large number of potential flows, making it difficult to maintain a counter-per-flow within a memory accessible at the line rate. There can be two types of errors in flow size estimation: Overestimation and underestimation. The state-of-the-art data structure for flow size estimation is the Count-Min sketch (CM) suggested by Cormode and Muthukrishnan in 2005 [2].

The CM relies on a two-dimensional array with w columns and k rows of counters initialized to values of 0s. A set of k hash functions are used to map a flow to k counters, one in each of the rows. Upon a flow arrival, each of these counters is incremented by the length of the flow sequence. To estimate the size of a flow, its k selected counters are considered and the size is estimated as the minimum among the k counters. Since multiple flows can contribute to the same counter, the computed value can be larger than the correct one in case other flows contributed to all k counters implying an overestimation. CM completely avoids underestimation. A tradeoff exists between the level of accuracy and the allocated memory such that more memory reduces collisions among flows. Equivalently, reducing the flow number improves accuracy.

The CM is illustrated in Fig. 1. Flows I, II of size 4 and 5, respectively, are recorded in the sketch (shown on the left side). Each flow increases the value of $k = 3$ counters by its own size. The size of Flow III (right side) is estimated by querying the CM. Its size is estimated as the minimal among the k counters it is mapped to.

The estimation derived from the Count-Min sketch is imperfect and can be an overestimation. With accuracy probability denoted by $1-\delta$, its overestimation can be bounded as part ϵ of the number of packets in the measured stream N. Formally, upon using a CM with depth k and width w (as illustrated in Fig. 1), the estimation $\hat{f}$ of flow f satisfies with probability $1-\delta$

$$\hat{f} \leq f + \epsilon N.$$

To achieve that the CM depth should be $k = \lceil \ln \frac{1}{\delta} \rceil$ and the CM width should be $w = \lceil \frac{e}{\epsilon} \rceil$ for Euler's number e.

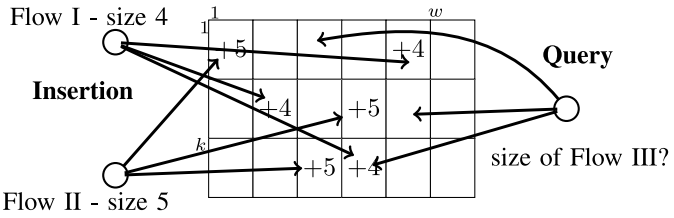


Fig. 1. The Count-Min sketch (CM) [2], allowing flow size estimation. A flow size is estimated as the minimum among the counters it is mapped to by a set of hash functions.

The property bounds the overestimation error, given as the difference between the estimated flow size $\hat{f}$ and its correct value *f*. However, the bound on the error $\hat{f}-f$ increases with the amount of measured traffic, and even for a small amount of traffic is still positive. Moreover, the guarantee is probabilistic, and error larger than ϵN can occur in some measurement instances. In this paper, we study the question of whether there exist designs that realize conditions for accurate flow size estimation?

Our approach is inspired by a recently suggested construction for another popular sketch named the Bloom filter that supports a simpler task of answering set membership queries [3]. The Bloom filter is a popular data structure widely used in many networking algorithms [4], [5], in fields as diverse as packet classification, monitoring, routing, filtering, caching, set synchronization and blockchain [6], [7], [8], [9], [10]. The Bloom filter is used for set representation, supporting element insertion and answering membership queries. There are two kinds of errors for membership queries in the representation of a set *S*: a false positive (when an element $x \notin S$ is reported as a member of *S*) and a false negative (when an element $x \in S$ is reported as a nonmember of *S*).

The Bloom filter encounters false positives and has no false negatives. Error probability (ratio of non-member elements reported as members of the set) decreases when more memory is allocated and increases when a larger set *S* is represented. The Bloom filter stores an array of bits, where a set of hash functions is used to map elements to the bits. With initial values of zero bits, the elements of *S* are first inserted to the filter, setting all bits pointed to by the hash functions. Upon a query, the bits mapped by the queried element are examined and a positive answer is given in case all these bits are set. A recent approach [11] presented the EGH construction for the Bloom filter which ensures it has no false positives when the represented set taken from a finite universe satisfies a given upper bound *d* on its size. This restriction is referred as the Bloom filter False Positive Free Zone (FPFZ). The construction makes use of simple carefully-designed functions mapping elements to the filter. Insertion and lookups are similar to those in the original Bloom filter.

Our (simple) observation is that properties of a similar mapping can also guarantee *accurate flow size estimation* when applied to flows mapped to the counter array of the CM. More generally, the FPFZ Bloom filter construction can be used also for accurate estimation in other counting schemes and data structures. However, we also identify that the memory complexity of the Bloom filter EGH construction [11] might not be practical in the context of CM where the number of active flows (namely flows with observed traffic) might be large. The existing EGH construction allocates an amount of memory affected by the universe size and the maximal set size for which false positives should be avoided. It requires a memory of $O(d^2 \log n)$ such that false positives are avoided whenever the set represented *S*, taken from a universe of size *n* satisfies $|S| \leq d$.

We explain that in the context of the CM, the elimination of overestimation is possible when the number of active (non-zero) flows is bounded similarly to the restriction on the set size in the Bloom filter without false positives. The universe from which elements can appear corresponds to the set of flows with potential traffic. Accordingly, this existing EGH construction requires memory that increases quadratically with the number of active flows, restricting in practice the potential scope of measurement tasks with accurate flow size estimation.

Accordingly, to make this adaptation towards the CM practical we also need to come up with new constructions that scale better for larger sets or equivalently larger number of active flows. We design new constructions of the Bloom filter false positive free zone that better fit larger sets, namely constructions with memory complexity which grows linearly with *d*, the maximal set size for which false positives are avoided. We explain how these constructions allow accurate flow size estimation in the CM even for a relatively large number of active flows.

Our main notations are summarized in Table I and formally defined in Section III.

B. Contributions

As the first contribution of this paper, we suggest two constructions of Bloom filters with a false positive free zone that allow filter memory complexity which is *linear* in the size of the represented set. In particular, the new constructions require memory of $(d+1)\sqrt{n}$ or $((t-1)\cdot d+1)\cdot \sqrt[t]{n}$ bits where *t* is a controllable parameter. They achieve an advantage over the existing EGH filter for larger sets. Note that to achieve the linear dependency of the memory in the maximal set size *d*, the dependency in the universe size *n* is more expensive.

The second main contribution of this work is showing how constructions for Bloom filter with a false positive free zone can be used to guarantee accuracy in flow size estimation in Count-Min sketch [2]. We believe this work can serve as the basis towards the design of new type of measurement data structures for a wider range of measurement tasks having guarantees (which are not statistical) on the accuracy.

Paper outline: Section II discusses previous related works. In Section III, Bloom filters with a False Positive Free Zone (FPFZ) are formally defined. In Section IV we describe the OLS filter as a first construction. It relies on Orthogonal Latin Squares. Next, in Section V we propose as a second construction the POL filter that relies on the representation of universe elements through polynomials and their values. In Section VI we demonstrate the design of the CM supporting flow size estimation without overestimations when the number

of active flows is bounded. In Section VII we discuss potential generalizations and improvements. In Section VIII, practical configurations of the different filters are presented and compared to illustrate the trade-offs; the false positive probability of the filters beyond their FPFZ is evaluated by simulation; the accuracy of the proposed counting sketches is also evaluated by simulation and finally a practical application of the FPFZ to detect new sources in a network is discussed and evaluated. Conclusions and future work ideas appear in Section IX.

II. Related Work

The work of [11] presented the notion of the false positive free zone and suggested the EGH filter (discussed in Section III) as its construction. On the contrary, previous works tried to reduce the probability for false positives in Bloom filters rather than to eliminate them completely. We overview such works. Likewise, the careful mapping of elements into the filter in our proposed constructions of the false positive free zone is related to broader approaches to reduce and avoid collisions in hashing.

Reducing the Bloom filter false positive probability: For a given set size, the false positive probability is a function of the allocated memory. The tradeoff of the Bloom filter itself was shown to be close to an ideal one implied by properties of information theory. To support deletions, often counters are used instead of bits, requiring more memory for a given false positive probability. A line of work focused on data structure constructions for improving this tradeoff while supporting deletions. A work by Bonomi *et al.* [12] describes a scheme where a fingerprint of an element is stored in a memory location allowing more accurate answer to queries. The use of fingerprints stored in several locations was considered in [13] to further reduce the false positive rate. The VI-CBF [14] uses variable counter increments for encoding some element information to reduce false positives. Other works refer to tradeoff between the false positive probability to that of the false negative probability. In the (traditional) Bloom filter, false negatives are completely avoided. An approach suggested by Donnet *et al.* [15] is to randomly select bits of ones and reset them to zero. Reducing the ratio of 1s clearly reduces the false positive probability but also implies false negatives. A similar idea was described by Laufer *et al.* as the Generalized Bloom Filter [16]. Finally, [17], proposes the use of asymmetric error correction codes to reduce the false positive rate of filters used to perform multi-set membership checking. The cuckoo filter [18] allows set representation with low false positive probability by keeping element fingerprints. The adaptive cuckoo filter exploits correlation in the queries to avoid having the same false positive repeating many times [19].

Collision resolution and avoidance techniques: Hash tables and other data structures with hashing-based mapping often suffer from hash collisions. Similar to the Cuckoo filter, the *cuckoo hash table* [20] relies on flexibility to insert an element to one of multiple entries (as indicated by multiple hash functions) in order to reduce element collisions and achieve high occupancy of tables earlier to the detection of an insertion failure. *Open addressing* [21] solves collisions in hashing by computing prioritized alternative locations for an element. A sequential location can be computed in various ways, e.g., by adding a constant to the address or rehashing. Chaining simply allows an entry to keep a list of located elements. *Hopscotch hashing* [22] allows storing an element within a small neighborhood around a counter mapped by a hash function.

Improving the accuracy of the Count-Min Sketch (CM): The approach we present allows accurate flow size estimation for the Count-Min Sketch (CM) [2] following an assumption on the number of measured non-zero flows. The CM simply includes a two dimensional array and upon its arrival a flow is hashed multiple times and a single counter in each array is increased by the flow size. The flow is estimated as the minimum among the counter values it is hashed to. CM guarantees the following accuracy: With $u = \lceil \ln \frac{1}{\delta} \rceil$ arrays, each of $w = \lceil \frac{e}{\epsilon} \rceil$ counters, the estimation $\hat{f}_x$ for a flow x of size f_x satisfies w.p. of at least $1 - \delta$ that $\hat{f}_x - f_x \leq \epsilon \cdot \sigma$, where σ is the total size (sum of flow size) of the complete measured stream of elements.

Several works tried to improve the accuracy of the CM, although to the best of our knowledge our work is the first to provide guarantees for an exact estimation. Estan and Varghese [23] proposed an alternative counter update policy, known as *conservative update* that can be used when only flow insertions (rather than deletions) are supported. Consider a flow x of size f_x with an arrival of additional c packets. Prior to the counter update the flow size estimation $\hat{f}_x$ is computed. Then, rather than incrementing all counters by c, a counter value is updated to the maximum between its current value to $\hat{f}_x + c$. Such update policy also ensures a flow size is never underestimated and can reduce the overestimation by up to an order of magnitude. Lee *et al.* [24] described a method based on least squares optimization for flow size estimation computed over all counters of the CM rather than taking the minimum among particular counter values. A linear system of equations is constructed based on the mapping of flows to counters and the counters values. The number of variables equals the number of all potential flows. With the cost of solving the system (using the facts that flow size is non-negative and upper bounded by particular counter values), the approach can achieve estimates of higher accuracy. Having an estimation which can just be an overestimation for the flow size (which is not always accurate) yields a biased estimation. Krishnamurthy *et al.* [25] suggested an *unbiased estimation* based on taking the median of values computed for the counters. Let σ be again the sum of all flow values, let u be the number of subarrays and w be the number of counters in each subarray. Let C_x^i be the counter value for flow x in subarray i. For each subarray a value $\hat{f}_x^i = \frac{C_x^i - \sigma/w}{1 - 1/w}$ is computed. The estimation $\hat{f}_x$ for f_x is given as the median for $\{\hat{f}_x^1, \ldots \hat{f}_x^u\}$.

III. Preliminaries

This section provides an overview of the Bloom filter data structure, a formal definition of its false positive free zone and describes the existing construction of filters with a false positive free zone. It also gives a brief overview of linear block

TABLE I
SUMMARY OF MAIN NOTATIONS

Symbol	Meaning
S	represented set
U	universe from which elements are selected
n	universe size $\|U\|$
d	maximal set size in the false positive free zone
m	required filter length
k	number of hash functions
M	binary matrix representing the hash functions

TABLE II
COMPLEXITY OF THE SUGGESTED OLS AND POL FILTERS ALLOWING MEMORY COMPLEXITY WHICH IS LINEAR IN THE MAXIMAL SET SIZE IN COMPARISON WITH THE EXISTING EGH FILTER. t IS A CONTROLLABLE PARAMETER

filter	memory complexity	# probes
EGH [26]	$O(d^2 \cdot \log n)$	$O(d \cdot \log n)$
OLS	$(d+1)\sqrt{n}$	$d+1$
POL	$((t-1)\cdot d+1)\cdot \sqrt[t]{n}$	$(t-1)\cdot d+1$

codes. This provides the reader the background needed to follow the proposed constructions presented in Section IV and Section V. The notations used in the paper are summarized in Table I. Table II provides the parameters of the new suggested constructions named OLS and POL including in addition to the memory of each filter, the required number of probes, the number of bits that are updated upon an insertion or read in a query. Reducing the number of probes can allow higer efficiency, especially for large filters for which the accessed bits are not necessarily in a very small range.

A. Bloom Filter and Its False Positive Free Zone

A Bloom Filter is a well-known simple data structure used to represent a set of elements $S = \{x_1, \ldots, x_l\}$ from a universe *U* using an array of *m* bits [3].

As illustrated in Fig. 2, the Bloom filter uses *k* uniformly-distributed hash functions over the range $1, \ldots, m$ of its *m*-bit filter. For each element in the set *S*, *k* hash entries are calculated using the hash functions and the corresponding bits are set to one. For instance, in the figure, two hash functions are used and the bits pointed to by *u*, *x* and *y* are set to one. To test whether an element $z \in U$ is in *S*, we check whether all of its *k* corresponding bit locations mapped by *z* are set to one. If this is not the case, we know that $z \notin S$. If all of them are set, the Bloom filter states that $z \in S$, although this might be a false positive error. For each $z \notin S$, the probability of a false positive error, can be approximated by $(1-p_0)^k$, where $p_0 = (1-1/m)^{nk}$ is the probability that a specific bit is still zero after the insertion of *n* elements.

While the above probability of a Bloom filter for a false positive depends on its number of elements, it still remains positive while holding a small number of elements and even a single element. For some applications, it is required to guarantee the avoidance of false positives implying that any positive membership indication is correct. For a finite universe, the false positive free zone guarantees the avoidance of false positives as long as the number of set elements remains small. We follow the Bloom filter false positive free zone definition from [11].

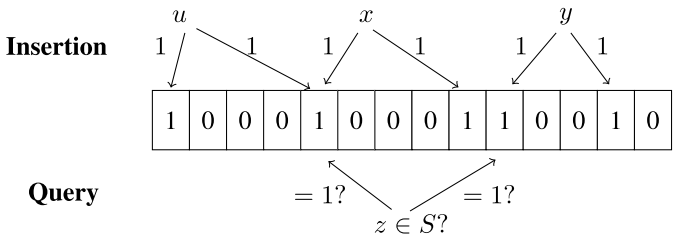


Fig. 2. The Bloom filter (representing a set $S = \{u, x, y\}$): An element is mapped by hash functions to bits in the array. Bits are set upon element insertion and are examined upon a membership query.

Definition 1: The **false positive free zone** of a filter allows a universe $U = \{1, \ldots, n\}$ of size *n* and a maximal set size *d*, if for any filter $S \subseteq U$ satisfying $|S| \leq d$, the query operator of an element $y \in U$ always returns the true answer and in particular avoids false positives.

Note that we do not restrict the set size from temporarily being larger than *d* but allow then the occurrence of false positives.

The construction of a filter of size *m* is described through a binary matrix *M* of size $m \times n$. The columns refer to the *n* universe elements, each described through a binary vector of length *m* showing the bits it maps to the *k* hash functions. The requirements on *M* for enabling the FPFZ are as follows.

Property 1: Let *M* be a $m \times n$ binary matrix such that for $j \in [1, n]$ the j^{th} column describes the bits mapped by an element $j \in U$ where $|U| = n$ through the set of hash functions. Let *S* be a set of elements of size $|S| \leq d$. The matrix *M* implies a FPFZ of size *d* if for each such *S* and element $j \notin S$, there is an index $i \in [1, m]$ such that $M_{i,j} = 1$ and $\forall j' \in S, M_{i,j'} = 0$. Namely, the membership of element *j* in the filter describing *S* can be eliminated based on the index *i*.

B. The EGH Filter

The EGH filter was described as the first construction of a Bloom filter with a false positive free zone (Definition 1) in [11]. For a universe $U = \{1, \ldots, n\}$ it defines the mapping of elements into a filter of *m* bits through a matrix of size *m* $\times$ *n*. The filter is organized in blocks such that each block length is a prime integer, starting from the smallest primes $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, etc. To provide the false positive free zone property for all represented sets of size $|S| \leq d$, it is required to use *k* primes such that $\prod_{i=1}^{k} p_i \geq n^d$. The required filter length is then given by the sum $m = \sum_{j=1}^{k} p_j$. For an element $x \in U$ the mapping of *x* into the filter is simply given by computing *x* mod p_i for all *k* primes and storing the values.

The filter for a set *S* is given by the cumulative OR of the columns that correspond to the $|S|$ elements. A membership element query returns positive if all *k* corresponding bits are set in the filter. The false positive free zone property guarantees that a negative answer is returned to the query of all elements in $U \backslash S$, namely each such element is mapped to at least one bit not set by any of *S* elements. Moreover, as in

the traditional Bloom filter, false negatives do not occur. The asymptotic memory required filter is $O(d^2 \log n)$.

C. Linear Block Codes

In communications or storage, errors can occur and alter the data transmitted or stored. If those errors are not corrected they could lead to performance degradation or even to system failure. Therefore, many communications and storage systems use error detection and correction codes [27]. One of the most common types of codes used are systematic binary linear block codes that protect a data block of v bits using a codeword of w bits formed by the original v data bits and w–v additional bits used to store parity checks on a subset of the v bits. The code can be defined by its generator matrix G and parity check matrix H. Both matrices are binary. The first one is used to obtain the encoded data by multiplying the generator matrix with the data to encode such that additions are done modulo 2. Conversely, the parity check matrix H detects errors. This is done again by multiplication, with the w bits transmitted or stored. The resulting vector is commonly known as the syndrome and when it is not the zero vector, an error is detected.

As an example, let us consider a simple Hamming code that can correct single bit errors [28] and a block of size $v = 4$ for which w–$v = 3$ additional parity check bits are needed. The generator matrix for the code (of size $w \times v$) is shown in Fig. 3(a). The corresponding parity check matrix H (of size $(w$–$v) \times w$) can be found in Fig. 3(b). Each row corresponds to a parity check equation and each column to a data or parity bit. If data bits are b_1, b_2, b_3, b_4 from left to right and parity bits are p_1, p_2, p_3 also from left to right, we would have: $p_1 = b_1$ `xor` b_3 `xor` b_4 where `xor` is used to implement modulo-2 addition. The leftmost data bit b_1 for instance participates on parity checks p_1 and p_2.

We could think of the parity check matrix H (excluding the columns of the parity check bits) as our M matrix of size $m \times n$. In that case, the size of the filter m would correspond to the number of parity check bits w–v and the number of data bits v to the false positive free universe size n. To provide a false positive free zone, we need that the union or Boolean sum (or bitwise OR) of any up to d columns does not contain any other column. Unfortunately, this is not generally the case for an arbitrary matrix. For example, going back to the Hamming code, the column that corresponds to b_4 (fourth column in Fig. 3(b)) contains the other three data columns. The same applies to various error correction codes, for example to Bose-Chaudhuri-Hocquenghem (BCH) codes that have some columns that contain other columns. There is however an exception for which the desired properties hold: Orthogonal Latin Square (OLS) codes. In the next section, those codes are described and we explain how they can be used to build filters with a free positive zone.

$$\left|\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right|$$

(a) Generator matrix G.

$$\left|\begin{array}{ccccccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right|$$

(b) Parity check matrix H.

Fig. 3. Linear codes: Generator matrix and parity check matrix for a linear code, encoding $v = 4$ data bits through $w = 7$ bits.

TABLE III
OLS FILTER ADDITIONAL NOTATIONS

Symbol	Meaning
s	Latin square order (dimension size)
H	Latin square parity matrix
Q	Column indices for elements in S
$c = \|S\|$	Set size beyond the FPFZ limit

IV. ORTHOGONAL LATIN SQUARE BLOOM FILTERS WITH A FALSE POSITIVE FREE ZONE

In this section we present our first construction: the OLS filter. A first subsection discusses the features of the Orthogonal Latin Square (OLS) codes and how they can be used to construct the filter. Then a practical filter example is given followed by a formal proof regarding the filter false positive free zone. The rest of the section presents an analytical comparison with the existing EGH filter and discusses the false positive rate of the OLS filter when it operates beyond its false positive free zone. Additional notations for the OLS filter are summarized in Table III.

A. Description of the Proposed Filters

As we mentioned, the design of the matrices required for the false positive free Bloom filters is to some extent similar to the design of binary linear block codes for error detection and correction. Those codes are defined by their generator or by their parity check matrices. In the case of the error correction codes, the operations are XOR (modulo-2 addition) instead of binary OR. In particular, a class of codes known as Orthogonal Latin Square (OLS) codes [29], has been widely studied to protect memories from errors as they have a modular construction and can be decoded in parallel with simple circuitry [30].

An Orthogonal Latin Square (OLS) code is constructed using Latin Squares. A Latin square of order s is a matrix of size s by s where elements $0, 1, \ldots, s-1$ are placed in the matrix so that each of them appears only once in each row and each column [31]. We refer to s as the order of the Latin square. An example of two Latin squares is given in Fig. 4(a).

Two Latin squares are called orthogonal if when superimposing the squares, every ordered pair of elements appears exactly once. Taking the two Latin squares of Fig. 4(a), we obtain the matrix shown in Fig. 4(b) when they are superimposed. For the superimposing, in each pair of values the left value is taken from the first Latin square and the right value from the right Latin square. It can be seen that each ordered pair of values appears only once. A set of Latin squares is mutually orthogonal if all the pairs of Latin squares in the set are orthogonal. This means that when all the Latin squares in the set are superimposed, a pair of ordered sequences of elements has at most one position with the same value as if

0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

0	1	2	3	4
2	3	4	0	1
4	0	1	2	3
1	2	3	4	0
3	4	0	1	2

(a) Two Latin squares of order 5.

0,0	1,1	2,2	3,3	4,4
1,2	2,3	3,4	4,0	0,1
2,4	3,0	4,1	0,2	1,3
3,1	4,2	0,3	1,4	2,0
4,3	0,4	1,0	2,1	3,2

(b) Superposition of the two Latin squares.

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

0	1	2	3	4
0	1	2	3	4
0	1	2	3	4
0	1	2	3	4
0	1	2	3	4

(c) Additional matrices for OLS code construction.

Fig. 4. Latin squares and superposition: In (a), two Latin squares of order 5. Their superposition is shown in (b). (c) shows two additional matrices used for the construction of the OLS code.

there are two, then the Latin squares that correspond to those positions would not be orthogonal.

The construction of orthogonal Latin square codes [29] makes use of two additional special matrices (which are not Latin squares). These two matrices, shown in Fig. 4(c), have fixed values along all rows or columns. By the definition of Latin squares, if one such matrix is superimposed with a Latin square (and in particular those from Fig. 4(a)), every ordered pair of elements appears exactly once.

Orthogonal Latin Square (OLS) codes are constructed from a set of mutually orthogonal Latin squares. The code covers s^2 data bits and each Latin square adds s parity bits that check the bits that have the corresponding value. To do so, each value in each Latin square represents a parity bit that covers the bits on which it appears. For example, in Fig. 4(a), the first parity check bit of the first Latin square (that corresponds to value 0) would check bits 0, 9, 13, 17 and 21. As each data bit corresponds to a position in the Latin square, it participates exactly in one of the s parity checks associated with a Latin square. Additionally, since the Latin squares are mutually orthogonal, each pair of parity checks has at most a bit in common.

Therefore, by construction, an OLS code built using $d+1$ Latin squares has a parity check matrix H satisfying:

- Each pair of columns only has at most a single position with a bit of one in common.
- Each column that corresponds to a data bit has the same number of ones: $d+1$.
- Each column that corresponds to a data bit has size $(d+1)\cdot s$ and has a single bit of one in the first s bits, another on the second group of s bits and so on.

Based on the properties of the matrices, we can see that an OLS code with $d+1$ ones in the columns can be used to implement a Bloom filter with a zero false positive universe given by the size of the data block of the code for up to d elements (at least $d+1$ elements are needed to create a false positive).

In general, the Bloom filters based on OLS codes built from a Latin Square of size s has the following parameters:

- Universe size: $n = s^2$.
- Bloom filter size: $(d+1)\cdot s$.
- Free positive zone: d.

Elements insertion and membership queries can be performed in the OLS filter simply in a way similar to these operations in the original Bloom filter with specific mappings of the hash functions described by the matrix where each element in the universe is associated with a column. An insertion of an element is performed by setting in the filter the bits set in the column. A query is performed by checking that all bits set in the column are set also in the filter. Similarly, membership of multiple elements can be done by checking that all bits that should be set for one of them are set in the filter. The insertion and query operations are illustrated in Fig. 5.

B. Illustrative Example

As an example, the $20 \times 25 = (d+1)\cdot s \times s^2$ matrix M of an OLS filter with $d=3$, $n=25$ and $s=\sqrt{n}=5$ is shown in Fig. 6. The filter length is $m=(d+1)\cdot s=20$ defined for a universe of size $n=25$ and guarantees no false positives for sets up to size $d=3$. It includes $d+1=4$ parts of s bits. In this case, the first two sub-matrices used are these from Fig. 4(c) and the third and fourth correspond to the Latin squares on Fig. 4(a). In case the maximal set size is smaller $d=2$ a shorter filter of length $(2+1)\cdot 5=15$ can be satisfying. Such a filter is simply obtained by considering only the first $d+1=2+1=3$ first sub-matrices, each of $s=5$ rows.

C. False Positive Free Zone Property

In the following main theorem, we prove the OLS filter completely avoids false positives in a well-defined zone.

Theorem 1: Consider an OLS filter defined for a universe $U=\{1,\ldots,n=s^2\}$. For $d\geq 1$, an OLS filter of size $(d+1)\cdot s$ bits completely avoids false positives for any represented set of size $|S|\leq d$.

Proof: The proof relies on the structure of the partial parity check matrix of the OLS code for which an example is shown in Fig. 6. We refer to some combination of at most d columns from the matrix and explain that their OR does not (fully) cover any other column. A column is divided into $d+1$ parts of s bits with exactly a single bit of one in each part. Let j be a column index and Q be a set of column indexes satisfying $|Q|\leq d, j\notin Q$. We explain that column j cannot be covered by the cumulative OR for the columns of Q. By contradiction: To be fully covered, all the $d+1$ bits of one of the columns j have to be covered by the (at most) d columns that refer to Q. By the pigeonhole principle, one column in Q covers at least two bits of one of column j. This contradicts the fact that each pair of columns in the matrix has at most a single bit of one in common. Thus at least one of bits of one in column j is not set in the cumulative OR for the columns in Q. Accordingly, a query of a general element in $U\backslash S$ would not lead to a false positive. ■

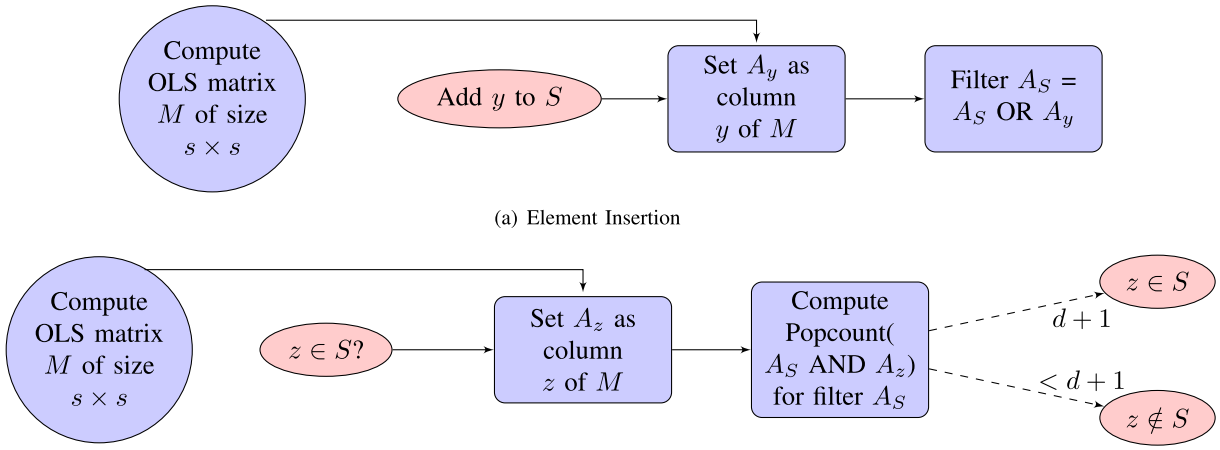


Fig. 5. OLS filter: Illustration of basic operations for a set *S*, universe size *n*, maximal set size *d* and filter parameter $s = \sqrt{n}$. Computation of the OLS matrix *M* is done once and not for each operation. A_S is the filter representing the set *S*.

D. Analytical Comparison With Previously Proposed Filters

We compare the performance of the proposed OLS filter with the only previously known construction with a false positive free zone [11] known as the EGH filter. To do so, we compare the minimal filter size required for a given universe size and a maximal allowed set size such that false positives are guaranteed to be avoided. We denote by *m*(*n*, *d*) the minimal required filter size for a universe $U = \{1, \ldots, n\}$ of size *n* and a maximal set size *d*. Clearly, the function *m*(*n, d*) is (weakly) monotonically increasing in both parameters.

We first refer to the memory complexity of the EGH filter of [11]. Recall that the filter is composed of *k* segments that refer to the *k* smallest primes that their product equals at least n^d. The EGH filter length equals the sum of the primes and has complexity of $m(n, d) = O(d^2 \cdot \log_2 n)$. The property was shown in [11]. The minimal filter length *m*(*n*, *d*) is given as the sum of the first *k* primes $m(n, d) = \sum_{j=1}^{k} p_j$ where *k* is determined as the minimal value for which the inequality $\prod_{j=1}^{k} p_j \geq n^d$. holds. By relying on [26], it was concluded that the bound can be satisfied with $p_k \leq \lceil 2d \log(n) \rceil$.

We now refer to the suggested OLS based filters with its described construction.

Theorem 2: The memory complexity of the OLS filter satisfies: $m(n, d) = (d + 1) \cdot \sqrt{n}$.

Proof: The value *m*(*n, d*) simply expresses the Bloom filter length, given as the length of a column in a given construction of the OLS filter. To allow a maximal set size *d*, a column of the code generator matrix has $d + 1$ parts, each part of *s* bits where the universe size is $n = s^2$. The OLS filter value for a set is simply given as the OR of multiple columns and its length equals the length of a column $(d + 1) \cdot \sqrt{n}$. ■

The above behaviors demonstrate that the OLS should be used when the represented set *d* is large while the universe size is still restricted at some degree. Intuitively, this is described as the following relationship between the two complexities.

Corollary 1: For an asymptotically maximal set size *d*, the OLS filter requires less memory than the EGH filter.

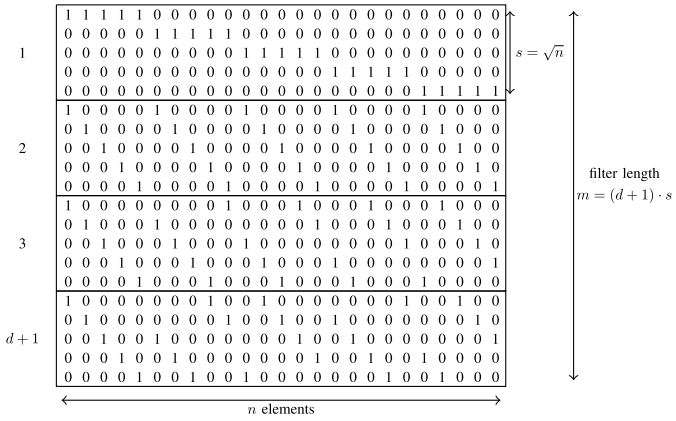


Fig. 6. Parity check matrix *H* of size $(d + 1) \cdot s \times s^2 = 20 \times 25$ for an Orthogonal Latin Square code with universe size $|U| = n = s^2 = 25$ for $s = 5$ and maximal set size $d = 3$. The required filter length is $m = (d + 1) \cdot s = (d + 1) \cdot \sqrt{n} = 4 \cdot 5 = 20$.

This analysis shows that depending on the application the proposed filters can provide an advantage in some settings over the filters proposed in [11].

E. Beyond the False Positive Free Zone

As mentioned, the proposed OLS filters are false positive free when the set size is equal or smaller than *d*. However, they can also operate with larger sets and for those will provide a non-zero false positive rate. Determining the expected false positive rate of the OLS filters is not straightforward as the bit positions set by each element have non trivial dependencies. For example, if two elements share a position with a one, they by design do not share any other position. Accurately modeling those dependencies is complex while the focus of this work is the false positive free zone. In the following, a simple approximation of the false positive rate of OLS filters is presented. We refer to *c* satisfying $c \geq d + 1$ as the number of elements in the filter.

To estimate the false positive rate of the OLS filter, let us consider that we perform a lookup of an element not stored in the filter when *c* elements have been stored. We can look

TABLE IV
POL FILTER ADDITIONAL NOTATIONS

Symbol	Meaning
P	Polynomial
$t-1$	Polynomial degree
$a_0, \ldots, a_{t-1}$	Polynomial coefficients
$\sqrt[t]{n}$	Modulo range size
$[0, (t-1)\cdot d]$	Polynomial input values

at each of the $d+1$ groups of s bits independently and the element has exactly a single bit of 1 in each of them. The probability that for the first group this bit is set equals the probability that at least one of the c elements has set it to one which is given by:

$$1-\left(\frac{s-1}{s}\right)^{c} \tag{1}$$

For the second group of s bits, we know that an element that matched on the first group cannot match the searched element (due to the properties of the matrix) and therefore the probability in the second group can be approximated by:

$$1-\left(\frac{s-1}{s}\right)^{c-1} \tag{2}$$

The same reasoning applies to the rest of the groups and the filter false positive rate is approximated as:

$$f_{OLS} \approx \prod_{i=0}^{d}\left(1-\left(\frac{s-1}{s}\right)^{c-i}\right) \tag{3}$$

This formula for f_{OLS} has similarity to the traditional approximation for the false positive probability of the Bloom filter:

$$f_{BF} \approx \left(1-\left(\frac{s-1}{s}\right)^{c}\right)^{d+1}. \tag{4}$$

It gives a lower false positive rate as elements can only match in one group. This property is verified in the simulation results presented in the evaluation section.

V. POLYNOMIAL BASED FILTERS WITH A FALSE POSITIVE FREE ZONE

A. Motivation

We have been discussing so far two approaches for the design of a Bloom filter with a false positive free zone. Let n again describe the size of the universe $U=\{0,\ldots,n-1\}$ and d the maximal set size for which false positives should be avoided. The existing EGH filter requires memory of $O(d^2\cdot\log n)$ bits [11] and our suggested OLS filter from Section IV achieves complexity of (exactly) $(d+1)\sqrt{n}$ bits. We saw that the EGH filter becomes expensive for larger d while the OLS becomes less attractive for large n. In this section, we would like to allow even larger flexibility in the design, trying to avoid the expensive components of d^2 in the complexity of the EGH filter as well as the $\sqrt{n}$ in that of the OLS filter. Accordingly, we describe a construction with memory size of $((t-1)\cdot d+1)\cdot\sqrt[t]{n}$ where t is an integer parameter we control. This construction is based on Polynomials and we refer to it as the *POL filter*. Additional notations for the POL filter are summarized in Table IV.

B. Construction Based on Polynomials

Let t satisfy $((t-1)\cdot d+1) < \sqrt[t]{n}$ and assume for simplicity that $\sqrt[t]{n}$ is an integer. We associate an element $y\in U$ with a polynomial P_y of degree t–1 such that each of its t coefficients $a_0,\ldots,a_{t-1}$ belong to $[0,\sqrt[t]{n}-1]$. The coefficients are selected such that $P_y(x)=\sum_{i=0}^{t-1}a_i\cdot x^i$ satisfies

$$P_y\left(\sqrt[t]{n}\right)=\sum_{i=0}^{t-1}a_i\cdot\left(\sqrt[t]{n}\right)^i=y.$$

Note that $a_0,\ldots,a_{t-1}$ are well defined given $y,\sqrt[t]{n}$ as the coefficients in a representation of y in base $\sqrt[t]{n}$. For instance

$$a_0=y \bmod \sqrt[t]{n},\quad a_1=\left((y-a_0)/\sqrt[t]{n}\right) \bmod \sqrt[t]{n}.$$

We characterize each element $y\in U$ by $((t-1)\cdot d+1)$ values for $x\in[0,(t-1)\cdot d]$ of the polynomial $P_y(x)$ such that calculations are performed modulo $\sqrt[t]{n}$. We describe each such value in $|[0,\sqrt[t]{n}-1]|=\sqrt[t]{n}$ bits such that a single bit that corresponds to the value equals one while all others are zero. Similarly, we describe the ordered list of values in a binary vector of length

$$((t-1)\cdot d+1)\cdot\sqrt[t]{n}.$$

We refer to a generator matrix with $n=|U|$ columns. For $y\in[1,n]$ the y^{th} column keeps the $((t-1)\cdot d+1)\cdot\sqrt[t]{n}$ bits describing the element $y\in U$. The Bloom filter in that construction is of length $((t-1)\cdot d+1)\cdot\sqrt[t]{n}$. Equivalently, a column in the matrix is composed of $((t-1)\cdot d+1)$ groups, each of $\sqrt[t]{n}$ bits. For $y\in U$, the j^{th} group (for $j\in[0,(t-1)\cdot d]$), refers to the value $P_y(j)$.

A pseudocode for the POL filter construction for a given set appears in Algorithm 1. Earlier to the insertion of elements, the filter is initialized as an array of zero bits. The filter for a set S is (as in other constructions) simply given by the cumulative OR for the corresponding $|S|$ columns of the matrix. Its length is $((t-1)\cdot d+1)\cdot\sqrt[t]{n}$ and we can see it as composed of $((t-1)\cdot d+1)$ groups.

The insertion and query operations work as follows.

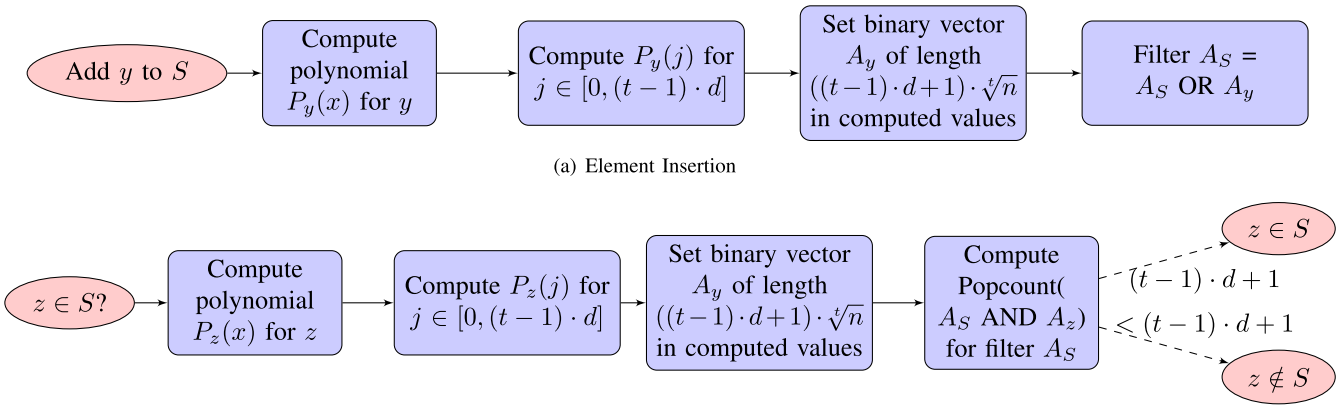
Element Insertion: Upon an insertion of an element $y\in S$, the vector is set as the OR of the previous vector with the column for y. Equivalently, upon such an insertion of an element y, in each of group $j\in[0,(t-1)\cdot d]$ of bits, we set the bit that refers to $P_y(j)$ (if it has not been set before).

Element Query: Upon a query of an element $y\in U$, $((t-1)\cdot d+1)$ bits are accessed. The bits that correspond to $P_y(j)$ for $j\in[0,(t-1)\cdot d]$, one in each group, are examined. A positive answer to the query is returned if all bits are set. If at least one of them is zero, a negative answer is returned for the query.

The insertion and query operations are illustrated in Fig. 7.

C. Illustrative Example

For $n=7^3=343$ for instance, we have $\sqrt{n}=7^{1.5}\approx 18.52$ and setting $t=3$ implies $\sqrt[t]{n}=7$. Here, the length of



(a) Element Insertion

(b) Element Query

Fig. 7. POL filter: Illustration of basic operations for a set *S*, universe size *n*, maximal set size *d* and filter parameter *t*. A_S is the filter representing the set *S*.

Algorithm 1: POL FILTER CONSTRUCTION

Input: Set S with $x \in U = \{0,\ldots,n-1\}$, parameters d, t
Result: POL filter A_S for the set S
begin
- $A_S = (0) \times ((t-1)\cdot d+1)\cdot \sqrt[t]{n}$
- **for** *y in S* **do**
  - $A_y = (0) \times ((t-1)\cdot d+1)\cdot \sqrt[t]{n}$
  - $y' = y$
  - **for** $i \in [0,(t-1)]$ **do**
    - $a_i = y' \bmod \sqrt[t]{n}$
    - $y' = (y' - a_i)/\sqrt[t]{n}$
  - $P_y(x) = \sum_{i=0}^{t-1} a_i \cdot x^i$
  - **for** $j \in [0,(t-1)\cdot d]$ **do**
    - $A_y[j\cdot \sqrt[t]{n} + (P_x(j) \bmod \sqrt[t]{n})] = 1$
  - $A_S = A_S$ OR A_y
- Return A_S

the OLS filter is $(d+1)\lceil\sqrt{n}\rceil = 19(d+1)$. The POL filter length is $((t-1)\cdot d+1)\cdot\sqrt[t]{n} = (2d+1)\cdot 7 = 14d+7$, shorter than that of the OLS filter for any $d \geq 1$. The generator matrix of the POL filter has $n = 343$ columns and when $d = 2$ each column is of $14d + 7 = 35$ bits. We demonstrate some of the columns of the matrix, each composed of $(t-1)\cdot d+1 = 5$ groups of $\sqrt[t]{n} = 7$ bits. For instance the element $y = \sqrt[t]{n} = 7$ corresponds to polynomial $P_y(x) = x$, and we have a column with the following 35 bits (1000000 0100000 0010000 0001000 0000100), where spaces are shown just to simplify the presentation and do not exist. For instance in the second group the bit that corresponds to $P_y(1) = 1$ is set. Likewise for $y = (\sqrt[t]{n})^2 + 1 = 50$ the polynomial is $P_y(x) = x^2 + 1$ and we have a column of the following 35 bits (0100000 0010000 0000010 0001000 0001000), where for instance in the fifth group we have $P_y(4) = 4^2 + 1 = 17 = 3$ (modulo $\sqrt[t]{n} = 7$).

D. Guarantees and Analytical Cost

We show a guarantee on the false positive zone holds.

Theorem 3: Consider a filter with the described construction defined for a universe $U = \{1,\ldots,n\}$. For $d \geq 1$ let t be an integer satisfying $((t-1)\cdot d+1) < \sqrt[t]{n}$ such that $\sqrt[t]{n}$ is prime. The filter of size $((t-1)\cdot d+1)\cdot\sqrt[t]{n}$ bits completely avoids false positives for any represented set of size $|S| \leq d$.

Proof: Consider an element $y \in U$ for which the filter provides a positive indication. This happens when for $j \in [0,(t-1)\cdot d]$, the corresponding bit of *y* was set by an element in *S*. Namely, for all the $(t-1)\cdot d+1$ values of *j*, the polynomial value of $P_y(j)$ equals to one of the $|S|$ polynomial values computed for the same *j*. Since $|S| \leq d$, by the pigeonhole principal there is necessarily a member $w \in S$ such that

$$|\{j|P_w(j) = P_y(j)\}| \geq \lceil((t-1)\cdot d+1)/d\rceil = t.$$

The two polynomials $P_w(j), P_y(j)$, each of degree t collide on at least t distinct values. By $\sqrt[t]{n}$ being a prime, it implies that P_w exactly equals P_y. Accordingly, $y = w$ and $y \in S$. ■

The following theorem follows from the construction details and describes the memory complexity of the POL filter.

Theorem 4: The memory complexity of the POL filter can be selected as $m(n,d) = ((t-1)\cdot d+1)\cdot\sqrt[t]{n}$ for an integer t satisfying $((t-1)\cdot d+1) < \sqrt[t]{n}$ and in addition $\sqrt[t]{n}$ is prime.

E. Beyond the False Positive Free Zone

In the case of POL filters, the dependency between bits that have a one in common is less stringent than for OLS filters as they can have two ones in common. This is much weaker dependency that should not have a major impact on the false positive rate of the POL filters when they operate beyond their false positive free zone. Therefore, for POL filters, it seems that the traditional approximation of the false positive rate for a Bloom filter given by Eq. (4) could be used. This will also be corroborated by the simulation results presented in Section VIII.

VI. ACCURATE COUNT-MIN SKETCH (CM) ESTIMATIONS

We demonstrate how the concept of the false positive free zone can be applicable beyond set representation with Bloom filters towards accuracy in the more advanced task of flow size estimation with counter based sketches. There can be two possible errors in the estimation of a flow size: Overestimation and

underestimation. This connection is not limited to constructions from Sections IV and V and applies also to that of [11] (that has not been discussed beyond Bloom filters). Here we focus on the state-of-the-art data structure for flow size estimation, the Count-Min Sketch (CM) suggested by Cormode and Muthukrishnan in 2005 [2] that can observe overestimations. Interestingly, we are the first to introduce the notion of estimation error free zone in such sketches.

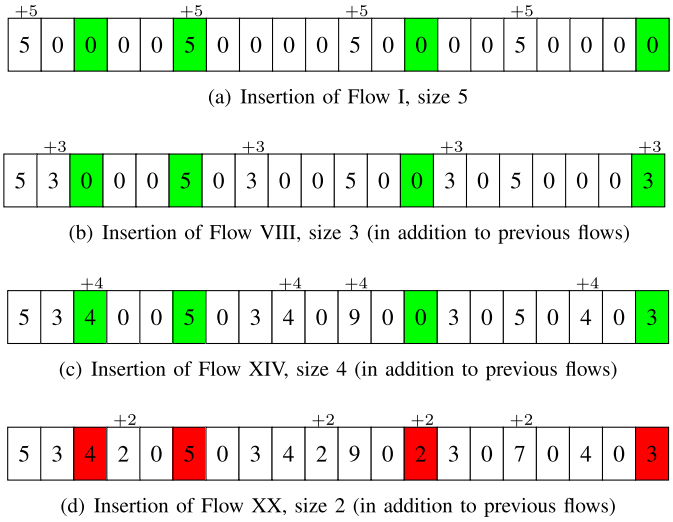
The CM relies on a two-dimensional array of counters initialized to values of 0s. A set of k hash functions are used to map a flow to k counters, one in each of its rows. The structure naturally fits that of the OLS filter with a single bit set in each subset of $\sqrt{n}$ bits of the filter as well as that of the POL filters with a single bit set in each subset of $\sqrt[t]{n}$ bits. Upon the arrival of a flow, each of these counters is incremented by the length of the flow sequence. To estimate the size of a flow, its k counters are considered and the size is estimated as the minimal value among the k counters. Since multiple flows can contribute to the same counter, the computed value can be larger than the exact one in case other flows contributed to all k counters implying overestimations. On the other hand, the CM completely avoids underestimations. A tradeoff exists between the level of accuracy and the amount of allocated memory such that more allocated memory reduces collisions among flows. Similarly, reducing the number of active flows improves accuracy. However, to the best of our knowledge there are no known results on how to completely avoid flow size overestimations even when the number of flows is small.

We describe how we can avoid the typical overestimation of flow size when constraints on the number of estimated flows and those among them with non-zero amount of traffic are satisfied. This can be achieved using a mapping of flows to CM counters given as a mapping implying a false positive free zone for the Bloom filter. We formalize that in the following theorem.

Theorem 5: Consider a CM sketch defined for up to n flows using an array of m counters. Let M be a $m \times n$ binary matrix implying a Bloom filter false positive free zone for sets $|S| \leq d$ selected from a universe of size n. Assume a mapping of flows to counters in the CM sketch that follows the matrix M. In case at most d of the flows have a non-zero flow size, the flow size reported by the CM is exactly its right value for all n flows. Moreover, with $d+1$ non-zero flows, the size of each non-zero flow is reported correctly.

Proof: We show that flow size is computed correctly for a flow regardless whether it is of size zero or has a positive size. Consider an arbitrary flow. By the definition of M, it follows that in the CM the flow maps to one counter such that none of the flows in S is mapped to this counter. Besides this estimated flow there are at most d other non-zero flows. Accordingly, the counter has a value which exactly equals the right flow size. Since the CM reports a flow size based on the minimal among several counters the existence of one counter with the correct value implies an exact size estimation. ■

Fig. 8 illustrates an example for a CM computed with $m = 20$ counters. It measures $n = 25$ potential flows with a mapping of flows to counters as of the OLS Bloom filter construction from Fig. 6 with $d = 3$. Four flows I, VIII, XIV and XX (namely 1, 7, 14 and 20) are observed with mappings described by the corresponding columns in Fig. 6. For instance flow I has mapping described by the most left column and flow X by the tenth from left column. Flow sizes are 5, 3, 4 and 2, respectively. The intermediate counter values following the insertions of the first i flows for $i \in [1, 4]$ are shown in Fig. 8 (a)–(d).



(a) Insertion of Flow I, size 5

(b) Insertion of Flow VIII, size 3 (in addition to previous flows)

(c) Insertion of Flow XIV, size 4 (in addition to previous flows)

(d) Insertion of Flow XX, size 2 (in addition to previous flows)

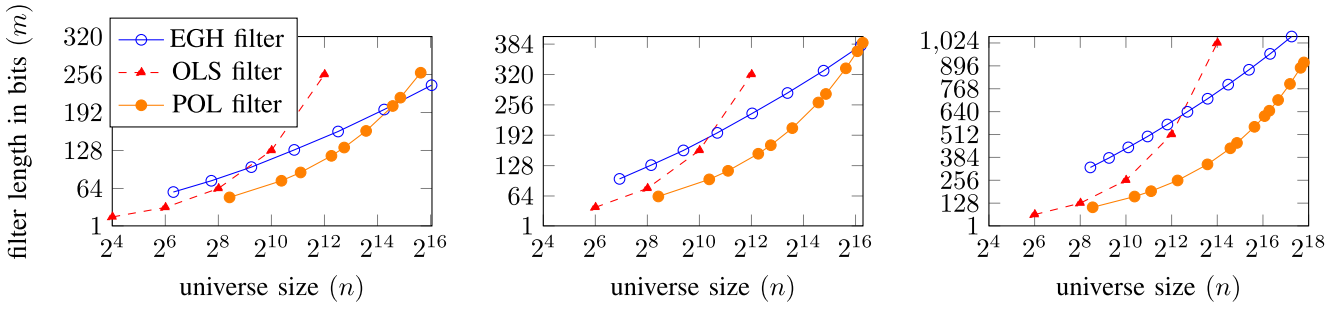
Fig. 8. Accurate flow size estimation with the CM sketch. Mapping is based on the OLS filter construction from Fig. 6 with $n = 25$, $d = 3$ and $m = 20$ counters. No overestimation for any flow is observed following the first $d = 3$ insertions or in earlier steps. Following the fourth insertion, while flow XI did not have traffic, its query returns an overestimation value of min(4, 5, 2, 3) = 2. Counters accessed in query of flow XI are shown in colors.

By Theorem 5, when up to $d = 3$ flows are observed, all non-observed flows yields a correct estimation of 0. As an example, counters mapped by flow XI for which traffic has not been observed are shown in color and their minimum is taken as the estimation for its size. Indeed, an estimation of 0 is observed with up to three flows. However, following the insertion of the fourth flow, all pointed counters are positive and the flow size is observed as min(4, 5, 2, 3) = 2 although the flow did not have traffic. On the other hand, even with $d + 1$ non-zero flows, the size of each of them is estimated correctly, e.g., for flow I as min(5, 5, 9, 7) = 5 and for flow XX as min(2, 2, 2, 7) = 2. Upon the appearance of additional flows, overestimation can occur also in the estimation of flows with traffic.

VII. Discussion

In this section we discuss important points such as implementation, extensions and alternatives of the proposed approach.

Implementation: In general, the proposed filters require to access several memory words and check different bits as a traditional Bloom filter (see Figures 5 and 7). However, when the filter fits in a memory word (or a few words), queries can be efficiently implemented by applying an AND operation of the relevant column of the auxiliary matrix with the filter and counting the number of ones. When that number is the same as the ones in the column a positive is obtained.



(a) maximal set size $d = 3$ (b) maximal set size $d = 4$ (c) maximal set size $d = 7$

Fig. 9. Illustrative comparison of the length in bits m vs. universe size n for the suggested and existing filters.

This implementation leverages the popcount instruction for counting the number of ones that is included in many modern processors [32], so that a query is completed in two instructions: AND and popcount and a comparison.

Perfect Hashing: The proposed schemes bear some similarity with perfect hashing [33] as they try to avoid collisions. However, differently from perfect hashing, collisions do not need to be totally avoided as elements can collide on some of the hash functions as long as the positions for d elements do not cover all the positions of any other element. This enables additional flexibility that is used to reduce the space needed as in general, the filter size m is smaller than the universe size n. Exploring the relationship between perfect hashing and Bloom filters with a false positive zone can yield new Bloom filter constructions that further improve the efficiency of the proposed schemes and it is thus an interesting area for future work.

Combining Hash Tables With Bloom Filters: One can consider the following alternative approach based on combining a hash table and a traditional Bloom filter as of [3]. Upon representing a set S, map its first d elements (or less if $|S| < d$) to the hash table. If $|S| > d$ insert the elements beyond the first $|S|$ to the Bloom filter. An element query returns a positive answer if the element can be found in the hash table or the Bloom filter returns a positive answer. If elements are stored only in the hash table (necessarily in such as case $|S| \leq d$) there is no need to query the Bloom filter. This can be indicated by counters for the elements in each part. Note that in such a case an element that is not in S would not be found in the hash table so accordingly no false positives can occur when the Bloom filter is not in use. An important advantage of the scheme can be its simplicity making use of traditional components. On the other hand, note that the time to find an element in the hash table might not be constant and is affected by the selected method to solve hash collisions of elements. Also, with this scheme if elements removals are also supported, potentially there are a total of at most d elements but some of them appear in the Bloom filter. This can result in false positives. On the contrary, the Bloom filter and Count-Min Sketch constructions proposed in the paper guarantee no false positives and no flow size overestimation in all cases of at most d represented elements.

VIII. Practical Configurations and Evaluation

This section first discusses practical configurations of the proposed OLS and POL filters and the EGH filter of [11] giving numerical examples that complement the asymptotic analysis. Then, the false positive rate of the proposed filters when they operate beyond their FPFZ is evaluated by simulation and compared to that of a traditional Bloom filter. Last, accurate flow size estimation with the Count-Min Sketch is also demonstrated. Our open-source code is available online [34].

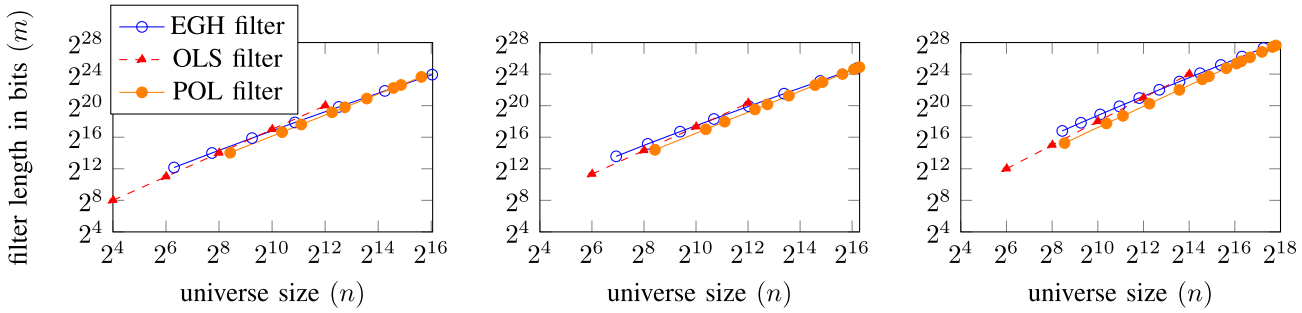
A. Practical Configurations - FPFZ Size

Configurations for the existing EGH filter and the proposed OLS and POL filters for some practical values of the universe size and filter length are summarized in Fig. 9. Note that the values of the maximal set size d shown are the worst case to compare the proposed filters against the EGH filters as they become more competitive as d increases. From the plots it can be seen, that for example, when $d = 2, 3$, the OLS filters provide better parameters when n is smaller than 256. For $d = 4$ the range increases to n being smaller than 1024. In the case of the POL filter with $t = 3$, the ranges are extended to over 2000, 16000 and 64000 for $d = 2, 3, 4$, respectively.

The last plot shows, for $d = 7$ and filter size smaller than 512 bits, the benefits of the proposed filters when the maximal set size increases. The OLS and POL filters can reduce the filter size or extend the universe size.

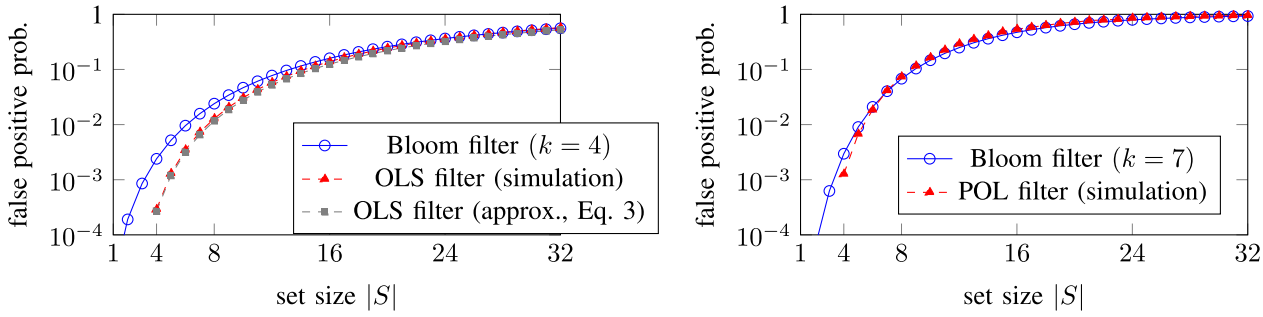
These examples show how the proposed filters can be used to either extend the universe size for the same filter size or to reduce the filter size for a given universe size when filter size is small. Note that this is the case, for example when the filter is embedded as a field of the packets in applications like SDN attribute encoding [35] or multicast addressing [36]. Recall (Table II) that the memory complexity of the OLS and POL filters grows with the universe size n and $\sqrt{n}$ or $\sqrt[t]{n}$, respectively, unlike the logarithmic dependency of the EGH filter. Accordingly, for large enough universe size both OLS and POL filters require more memory than the EGH filter.

Fig. 10 also shows the size of the auxiliary matrices representing the mapping of universe elements to the filter. A matrix size is simply given by the filter length times the universe size. Accordingly, differences between the various lengths of



(a) maximal set size $d = 3$ (b) maximal set size $d = 4$ (c) maximal set size $d = 7$

Fig. 10. Auxiliary matrix size comparison vs. universe size n for the suggested and existing filters.



(a) $n = 256, d = 3, m = 64$ (b) $n = 343, d = 3, m = 49$

Fig. 11. False positive probability beyond the false positive free zone for the OLS and POL filters.

the three filters are not dominant for large universe size n. Similarly, to the results in Fig. 9 the OLS and POL filters have a meaningful advantage for $d = 7$ over the EGH filter and their advantage was analyzed to be even greater for larger values of d.

B. Bloom Filter False Positive Probability

To check that the proposed filters have the expected false positive free zone, they have been implemented and then the entire universe has been tested for one thousand combinations of d elements in the set. The false positive rate when the filter stores more than d elements was also measured.

We considered the following OLS filters and POL filters:

- Filter OLS *O:* $n = 256$, $d = 3$, $m = 64$.
- Filter POL *P:* $n = 343$, $d = 3$, $m = 49$.

Those values correspond to universes of a few hundred elements which are sufficient for applications using a Bloom filter in packet headers to convey information to network nodes [36].

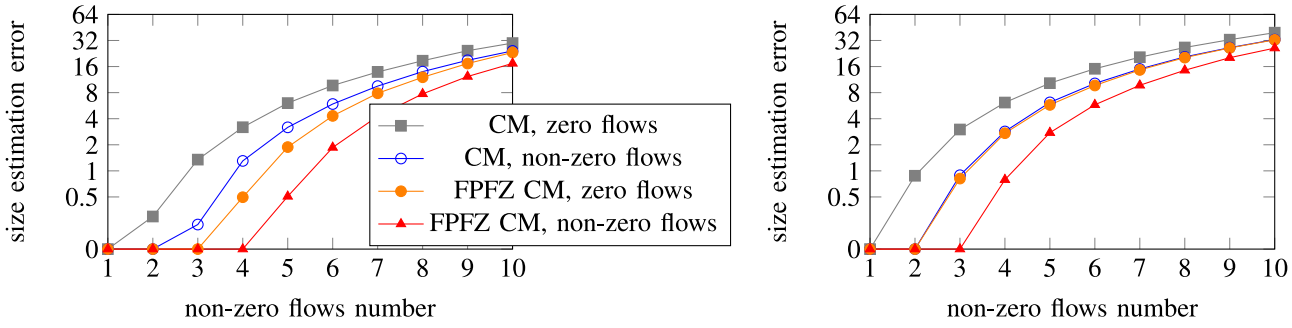
Simulation results are shown in Fig. 11 and compared with the approximated false positive probability of a traditional Bloom filter [3], [4] and the OLS filter approximated accuracy from Eq. (3). No false positives have been observed for sets of at most size d, for both OLS and POL filters. Beyond d, the false positive rate of the OLS filters matches reasonably well that predicted by Eq. (3) and is lower than that of a traditional Bloom filter. Instead, the POL filter false positive rate matches well that of the traditional Bloom filter as expected. Similar results obtained for other filter configurations are not shown due to space limitations.

C. Count-Min Sketch Accuracy

We compare the accuracy of the Count-Min Sketch (CM) in estimating flow size in two possible variants. The first is based on random mapping of elements into counters (with distinct mappings to any pair of elements). The second is based on Section VI and relies on element mapping implying a false positive free zone, replacing bits by counters of the CM. In particular we refer to a mapping of the OLS filter from Fig. 6.

We assume a universe of $n = 25$ flows. We vary in the range [1, 10] the number of flows with a positive size. The size of a non-zero flow is uniformly-distributed in the range [1, 100]. We track flow size in the CM and estimate a flow size as the typical CM's estimation by taking the minimum among the counters a flow maps to. Recall the CM either computes a flow size accurately or overestimates its size. We are interested in learning the cases where an exact estimation is guaranteed as well as in the average flow size overestimation. As indicated by Theorem 5, we can observe a difference in the accuracy of the estimation of the flows of size zero and those of the non-zero flows. Intuitively, with a fixed number of non-zero flows, flows of size zero are potentially affected more by other flows. Fig. 12 presents the overestimation in logarithmic scale.

First, Fig. 12(a) refers to a filter of length $m = 20$ allowing a false positive free zone with $d = 3$ for a universe with a number of elements that equals the total flow number $n = 25$. Consider the CM with the FPFZ mapping illustrated in the first two right curves. With up to four non-zero flows, each of these flows is evaluated correctly. With at most three non-zero flows, all flows of size zero also observe exact size estimation.



(a) filter length $m = 20$ for maximal set size $d = 3$

(b) filter length $m = 15$ for maximal set size $d = 2$

Fig. 12. Comparison of the accuracy of the Count-Min sketch (CM) with a mapping based on the false positive free zone vs. random mapping. Construction is based on the OLS filter from Fig. 6.

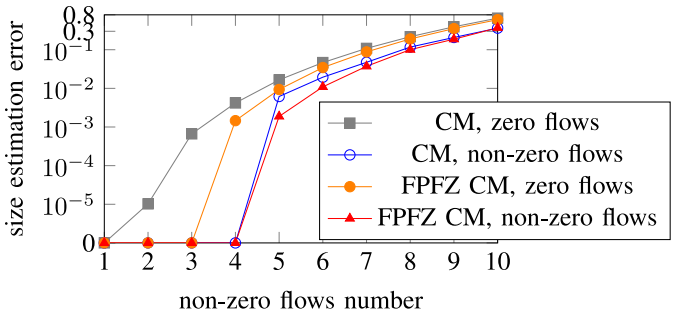


Fig. 13. Large universe (with $n = 1331$ flows): Count-Min sketch (CM) accuracy with a mapping based on the false positive free zone vs. random mapping. Construction is based on a POL filter of length $m = 77$ for $d = 3$.

This exactly follows Theorem 5. On the other hand, with the traditional random mapping the original CM, we observe an overestimation of a flow of size zero even when there are only two non-zero flows. Flows of size zero are sometimes overestimated when there are two or more non-zero flows while due to the uniqueness of the hashing of the elements cannot be overestimated by a single non-zero flow. In addition, the graph shows an improvement in the average overestimation. For instance, with five non-zero flows, their overestimation falls from 3.18 in the traditional CM to only 0.51 in the FPFZ CM. Overestimation of zero flows decreases then from 6.06 to 1.88.

Fig. 12(b) shows the results for the corresponding filter of length $m = 15$, implying a false positive free zone for $d = 2$. This refers to the first three bit groups in the OLS filter from Fig. 6. As implied by the analysis, setting $d = 2$ further limits the number of non-zero flows for which accurate size estimations are guaranteed. E.g., for the new CM variant three non-zero flows are sufficient to imply an overestimation in their size and two are sufficient to imply an overestimation in the size of zero flows. Similarly, a larger overestimation is observed for both the new CM variant based on the FPFZ and the traditional CM. However, the higher accuracy of the new variant is still meaningful. For instance, again with five non-zero flows the overestimation of zero flows is 10.29 in the traditional CM and equals 5.77 in the FPFZ CM, a reduction of 43.9% in the overestimation.

We also examined a scenario of a larger universe of $n = 1331$ flows. We made use of a mapping described by a POL filter of length length $m = 77$ that applies a FPFZ with $d = 3$. In the filter each element is mapped to seven bits. As above we examined the accuracy for a given number (denoted by x) of non-zero flows when the size of each is uniformly-distributed in the range [1, 100]. The results are shown in Fig. 13 indicating an improvement in the accuracy for the size estimation of zero flows as well as for the non-zero flows. In general, the larger filter and the larger number of counters an element is mapped to allowing smaller values of the overestimation in comparison with the previous experiment.

Also here, as implied by Theorem 5 with $d = 3$ in the FPFZ CM there is no overestimation for zero flows when $x \leq d = 3$ and for non-zero flows when $x \leq d + 1 = 4$. On the other hand, for the CM overestimation was observed for zero flows even when $x = 2$. Considering larger x values, for zero flows we see an improvement of 43.5% in the error for $x = 5$, 25.7% for $x = 6$ such that it becomes smaller for larger x values and equals 8.6% for $x = 10$. For non-zero flows the partial improvement equals 69.5% for $x = 5$ 43.8% for $x = 6$ and is again more meaningful for smaller x values.

D. Application to New Source Detection

To show the benefits of the proposed FPFZ in a practical application, we discuss its applicability for the detection of new sources in a network. This can be useful for example to closely monitor suspicious traffic. In a network, we are interested in tracking the appearance of a new active source IP address in a /24 IPv4 subnetwork.[1] We use a Bloom filter to represent the set of already known sources IP addresses. A new source is detected upon a negative query answer. As Bloom filters suffer from false positives, with a small probability a new source is not detected. This can be avoided by using a filter with a FPFZ.

The number of possible IP addresses in a /24 subnetwork is 256. Therefore, for example the OLS filter with $n = 256$ and $m = 64$ can provide a FPFZ of three elements. Instead, for an EGH the smallest filter that provides that FPFZ requires $m = 100$ bits. Using a filter with a FPFZ of three elements means that the first four sources are always detected correctly. To show the potential benefits of the FPFZ, three publicly

[1]The /24 is the most commonly used prefix length on Internet scale routers.

TABLE V
PERCENTAGE OF /24 SUBNETWORKS WITH MORE THAN PARTICULAR NUMBER OF DISTINCT SOURCE IPS

Trace	Packets	Flows	IPs > 4	IPs > 8	IPs > 16
CHI15	35.1M	1.7M	6.69%	1.53%	0.39%
CHI16	33.4M	1.6M	5.66%	1.30%	0.40%
MAWI	9.4M	0.3M	2.48%	1.25%	0.56%

TABLE VI
FILTER SIZE (IN BITS) FOR GUARANTEED CORRECT DETECTION IN /24 SUBNETWORKS (UNIVERSE SIZE $n = 256$)

Filter	IPs ≤ 4 $d = 3$	IPs ≤ 8 $d = 7$	IPs ≤ 16 $d = 15$
EGH filter	100	328	1060
OLS filter	64	128	256
POL filter	49	105	217

TABLE VII
AUXILIARY MATRIX SIZE (IN BITS, UNIVERSE SIZE $n = 256$)

Filter	IPs ≤ 4 $d = 3$	IPs ≤ 8 $d = 7$	IPs ≤ 16 $d = 15$
EGH filter	25600	83968	271360
OLS filter	16384	32768	65536
POL filter	12544	26880	55552

available packet traces have been used. The first two traces are from the CAIDA network monitor in Chicago (CHI15, CH16) [37], the third from MAWI [38], [39]. For each trace, the source IP addresses are extracted and the number of different addresses for each /24 subnetwork with observed traffic are counted. Then the percentage of /24 subnetworks that have more than 4, 8, or 16 different IP addresses is measured. The results are shown in Table V. Most /24 subnetworks have four or less source IPs and thus would not suffer false positives when using a 64 bit OLS filter with a FPFZ of three elements. Table VI shows the corresponding required filter length for the three schemes. For instance, with $d = 7$ filter lengths of 328, 128 and 105 bits for the EGH, OLS and POL filters (with parameter $t = 3$), respectively in the examined three traces allowed correct detection in 98.47%, 98.70% and 98.75% of the /24 subnetworks. This illustrates the benefits of the new FPFZ constructions in memory perspectives. While running time of insertion and query operations can highly depend on the environment, we expect it to be linear with the number of probes. Table VII shows the size in bits of the auxiliary matrix describing the mapping between the universe elements for each filter. The matrix size is simply the filter length times the universe size. For the above application parameters, both the OLS and POL filters require matrices smaller than than of the EGH filter, reaching size of roughly 65K and 55K bits rather than that of 271K bits for the EGH filter.

Similar to the complexity summarized in Table II, we present in Table VIII the particular probes number for the filters for the mentioned d values and universe size $n = 256$. While the POL filter shows numbers close to that of the EGH filter, the numbers of probes are minimized for the OLS filter. This follows the inherent properties of the construction of the various filters expressed in Table II. This highlights the superiority of the OLS with regards to the number of probes also for large values of d. Note that for the POL filter, a tradeoff exists between the filter size and the hash functions number through the parameter t.

TABLE VIII
PROBES (UNIVERSE SIZE $n = 256$)

Filter	IPs ≤ 4 $d = 3$	IPs ≤ 8 $d = 7$	IPs ≤ 16 $d = 15$
EGH filter	9	15	25
OLS filter	4	8	16
POL filter	7	15	31

This simple example shows how the proposed filters can be useful to avoid or reduce the number of false positives upon detecting new sources in subnetworks. The same reasoning can be extended to obtain exact packet count using the proposed FPFZ mapping in the Count-Min Sketch.

IX. CONCLUSION AND FUTURE WORK

Allowing accurate flow size estimation is a required capability in many networking applications. Existing solutions only provide probabilistic guarantee for a bounded size error. We explain the tight connection between constructions that avoid false positive in the Bloom filter and accurate flow size estimation in the Count-Min sketch (CM). Our first contribution is the description of two novel constructions for the Bloom filter false positive free zone while allowing its existence in memory complexity which is linear in the size of the represented set. The constructions are designed for scenarios of larger sets and suggest new tradeoffs between the universe size and the set size, improving an existing tradeoff for cases of increasing set size. Our main contribution is the applicability of the false positive free zone to provide accuracy in supporting functionality beyond set membership such as flow size estimation in the Count-Min sketch. Future work will focus on trying to find additional sketches for which the Bloom filter free zone can guarantee exact counting. A particular interest will be in those recently suggested generic data structures that support multiple measurement tasks.

REFERENCES

[1] O. Rottenstreich, P. Reviriego, E. Porat, and S. Muthukrishnan, "Constructions and applications for accurate counting of the Bloom filter false positive free zone," in *Proc. ACM Symp. SDN Res. (SOSR)*, 2020, pp. 135–145.
[2] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The Count-Min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[3] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[4] A. Z. Broder and M. Mitzenmacher, "Survey: Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2003.
[5] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing Bloom filter: Challenges, solutions, and comparisons," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1912–1949, 2nd Quart., 2019.
[6] N. Dayan, M. Athanassoulis, and S. Idreos, "Monkey: Optimal navigable key-value store," in *Proc. ACM Int. Conf. Manag. Data*, 2017, pp. 79–94.
[7] B. Donnet, B. Gueye, and M. A. Kaafar, "Path similarity evaluation using Bloom filters," *Comput. Netw.*, vol. 56, no. 2, pp. 858–869, 2012.
[8] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 131–155, 1st Quart., 2012.

[9] L. Luo *et al.*, "Efficient multiset synchronization," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1190–1205, Apr. 2017.
[10] O. Rottenstreich, "Sketches for blockchains," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, 2021, pp. 254–262.
[11] S. Z. Kiss, É. Hosszu, J. Tapolcai, L. Rónyai, and O. Rottenstreich, "Bloom filter with a false positive free zone," *IEEE Trans. Netw. Service Manag. (TNSM)*, early access, Feb. 12, 2021, doi: 10.1109/TNSM.2021.3059075.
[12] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting Bloom filters," in *Proc. Eur. Symp. Algorithms (ESA)*, 2006, pp. 684–695.
[13] S. Pontarelli, P. Reviriego, and J. A. Maestro, "Improving counting Bloom filter performance with fingerprints," *Inf. Process. Lett.*, vol. 116, no. 4, pp. 304–309, 2016.
[14] O. Rottenstreich, Y. Kanizo, and I. Keslassy, "The variable-increment counting Bloom filter," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1092–1105, Aug. 2014.
[15] B. Donnet, B. Baynat, and T. Friedman, "Retouched Bloom filters: Allowing networked applications to trade off selected false positives against false negatives," in *Proc. ACM CoNEXT Conf.*, 2006, pp. 1–12.
[16] R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte, "A generalized Bloom filter to secure distributed network applications," *Comput. Netw.*, vol. 55, no. 8, pp. 1804–1819, 2011.
[17] H. Dai, Y. Zhong, A. X. Liu, W. Wang, and M. Li, "Noisy Bloom filters for multi-set membership testing," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Sci.*, 2016, pp. 139–151.
[18] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than Bloom," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, 2014, pp. 75–88.
[19] M. D. Mitzenmacher, S. Pontarelli, and P. Reviriego, "Adaptive cuckoo filters," in *Proc. Workshop Algorithm Eng. Exp. (ALENEX)*, 2018, pp. 36–47.
[20] R. Pagh and F. F. Rodler, "Cuckoo hashing," *J. Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.
[21] D. Knuth, *The Art of Computer Programming. Part 3, Sorting and Searching*. Reading, MA, USA: Addison-Wesley, 1973.
[22] M. Herlihy, N. Shavit, and M. Tzafrir, "Hopscotch hashing," in *Proc. Int. Symp. Distrib. Comput. (DISC)*, 2008, pp. 350–364.
[23] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2002, pp. 323–336.
[24] G. M. Lee, H. Liu, Y. Yoon, and Y. Zhang, "Improving sketch reconstruction accuracy using linear least squares method," in *Proc. ACM Internet Meas. Conf. (IMC)*, 2005, p. 24.
[25] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. ACM Internet Meas. Conf. (IMC)*, 2003, pp. 234–247.
[26] D. Eppstein, M. T. Goodrich, and D. S. Hirschberg, "Improved combinatorial group testing algorithms for real-world problem sizes," *SIAM J. Comput.*, vol. 36, no. 5, pp. 1360–1375, 2007.
[27] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.
[28] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Techn. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.
[29] M. Hsiao, D. Bossen, and R. Chien, "Orthogonal Latin square codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 390–394, 1970.
[30] P. Reviriego, S. Pontarelli, A. Sánchez-Macián, and J. A. Maestro, "A method to extend orthogonal Latin square codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 7, pp. 1635–1639, Jul. 2014.
[31] A. D. Keedwell and J. Dénes, *Latin Squares and Their Applications*. Amsterdam, The Netherlands: Elsevier, 2015.
[32] R. Ramanarayanan, S. Mathew, V. Erraguntla, R. Krishnamurthy, and S. Gueron, "A 2.1ghz 6.5mw 64-bit unified popcount/bitscan datapath unit for 65nm high-performance microprocessor execution cores," in *Proc. 21st Int. Conf. VLSI Design*, Hyderabad, India, 2008, pp. 273–278.
[33] S. R. Blackburn, "Perfect hash families: Probabilistic methods and explicit constructions," *J. Comb. Theory A*, vol. 92, no. 1, pp. 54–60, 2000.
[34] (2021). *Bloom Filter Free Zone Constructions*. [Online]. Available: https://drive.google.com/drive/folders/1x7tOAzRn6ricjoBHcAmy_DCp9jzcNOSa
[35] R. MacDavid, R. Birkner, O. Rottenstreich, A. Gupta, N. Feamster, and J. Rexford, "Concise encoding of flow attributes in SDN switches," in *Proc. Symp. SDN Res.*, 2017, pp. 48–60.
[36] M. Särelä, C. E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott, "Forwarding anomalies in Bloom filter-based multicast," in *Proc. INFOCOM 30th IEEE Int. Conf. Comput. Commun.*, 2011, pp. 2399–2407.
[37] *The CAIDA UCSD Anonymized Internet Traces*. Caida, La Jolla, CA, USA, 2015. Accessed: Feb. 19, 2015. [Online]. Available: http://www.caida.org/data/passive/passive_2015_dataset.xml
[38] (2015). *MAWILab Traffic Trace*. Jul. 20, 2015. [Online]. Available: http://www.fukuda-lab.org/mawilab/v1.1/ 2015/07/20/20150720.html
[39] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. ACM CoNEXT 6th Int. COnf.*, 2010, pp. 1–12.

Ori Rottenstreich (Member, IEEE) received the B.Sc. degree in computer engineering and the Ph.D. degree in electrical engineering from Technion. He is an Assistant Professor with the Department of Computer Science and the Department of Electrical Engineering with the Technion, Haifa, Israel. Previously, he was a Postdoctoral Research Fellow with Princeton University.

Pedro Reviriego (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an Engineer with Teldat, Madrid, working on router implementation. In 2000, he joined Massana to work on the development of 1000BASE-T transceivers. From 2004 to 2007, he was a Distinguished Member of Technical Staff with the LSI Corporation, working on the development of Ethernet transceivers. From 2007 to 2018, he was with Nebrija University. He is currently with Universidad Carlos III de Madrid working on high-speed packet processing and fault-tolerant electronics.

Ely Porat received the Ph.D. degree in computer science from Bar Ilan University, Ramat Gan, Israel, in 2001. Thereafter, in parallel to a military service, he was an Assistant Professor with Bar Ilan University and became a Professor. He also holds a Visiting Professor position with the University of Michigan, Ann Arbor, MI, USA, and Tel Aviv University, Tel Aviv, Israel. His primary research field is pattern matching, but he has also worked on streaming algorithms, data structures, coding theory, group testing, compressed sensing, distributed algorithms, and game theory.

S. Muthukrishnan was a Professor with Rutgers University when this work was performed. He is an Algorithms Researcher with interest in theoretical and applied algorithms, databases, networking, and online advertising.