Access-Efficient Balanced Bloom Filters

Yossi Kanizo[a,*], David Hay[b], Isaac Keslassy[c]

[a] *Dept. of Computer Science, Technion, Haifa, 32000, Israel.*
[b] *School of Computer Science and Engineering, Hebrew Univ., Jerusalem, 91905, Israel.*
[c] *Dept. of Electrical Engineering, Technion, Haifa, 32000, Israel.*

Abstract

Bloom Filters particularly suit network devices, because of their low theoretical memory-access rates. However, in practice, since memory is often divided into blocks and Bloom Filters hash elements into several arbitrary memory blocks, Bloom Filters actually need high memory-access rates. Unfortunately, a simple solution of hashing all Bloom Filter elements into a single memory block yields high false positive rates.

In this paper, we propose to implement load-balancing schemes for the choice of the memory block, along with an optional overflow list, resulting in better false positive rates while keeping a high memory-access efficiency. To study this problem, we define, analyze and solve a fundamental *access-constrained balancing problem*, where incoming elements need to be optimally balanced across resources while satisfying average and instantaneous constraints on the *number of memory accesses* associated with checking the current load of the resources. We then use these results and suggest a new access-efficient Bloom Filter scheme in networking devices, called the Balanced Bloom Filter. Finally, we show that with a worst-case operation cost of up to 3 memory accesses for each element and an overflow list size of at most 0.5% of the elements, our scheme can reduce the false positive rate by up to two orders of magnitude compared to a filter that hashes all elements into a single memory block.

Keywords: High-Speed Networks, Bloom Filters, Load-Balancing

*Corresponding author (phone: +972-4-829-3864, fax: +972-4-829-3900).
Email addresses: ykanizo@cs.technion.ac.il (Yossi Kanizo), dhay@cs.huji.ac.il (David Hay), isaac@ee.technion.ac.il (Isaac Keslassy)

1. Introduction

1.1. Motivation

The *Bloom Filter* is a space-efficient randomized data structure that supports approximate set membership queries [1]. Its accuracy is measured by its false positive rate (FPR), i.e. the probability that a set membership query returns TRUE, while the element is not in the set; Bloom Filters always have zero false negative rate. Bloom Filters are often used in network applications for caching, routing, forwarding, traffic monitoring, and traffic measurements [2, 3]. Examples of well-known products using Bloom Filters and their variants are Mellanox's IB Switch System [4], Google's database system BigTable [5], and the Web Proxy Cache Squid [6].

Unfortunately, while efficient in memory space, *Bloom Filters require a significant number of memory accesses*. For instance, a Bloom Filter with 30 bits per element yields negligible FPR, but also requires to access about $30 \cdot \ln 2 \approx 21$ memory bits per query. Note that each of these bits can reside in an arbitrary location over the memory space. Thus, such a Bloom Filter would need a prohibitive memory access bandwidth in networking devices when either implemented in an off-chip setting (that is, requiring to access 21 memory blocks per query) or distributed over a network (equivalently, it may be required to access 21 nodes per query).

One proposal to improve the access-efficiency of Bloom Filters is to use a Blocked Bloom Filter [7, 8], in which each element is first hashed using a *single* hash function to one of the memory blocks, and then the memory block operates as a local Bloom Filter. Although this technique is clearly access-efficient, since each element requires to access a single memory block, it also suffers from a high FPR, due to a typical *load imbalance* between the memory blocks. Such an imbalance is inherent in this scheme due to the fact that, given n elements and n memory blocks, the maximum block load is $O(\log n / \log \log n)$ with high probability, while the average block load is 1 [9].

To tackle this problem, our basic approach is to *use load-balancing schemes*, making the number of elements in each memory block as balanced as possible. We further propose to use an overflow list that stores elements hashed to overloaded memory-blocks. The overflow list is typically small. For example, it can be implemented using a content-addressable memory (CAM), which supports parallel-lookup operations.

We study this problem using a *general* access-efficient approach to the *balancing problem*, a fundamental problem that lies at the core of many operations and applications in modern distributed and communication systems.

We model the balancing problem using a *balls and bins model* [10], and more specifically its *sequential multiple-choice* variant [9]. In this model, n balls are placed in m bins. Before placing a ball, d bins are chosen according to some distribution (e.g., uniformly at random) and the ball is placed in one of these bins following some rule (for example, in the least occupied bin). Moreover, we consider an extension of this model that allows a small fraction γ of the balls not to be placed in the bins; these balls are either disregarded or stored in a dedicated overflow list, usually implemented in an expensive memory (a similar model was considered, for example, in [11]). The quality of the balancing is measured by the load on the bins: The resulting load at each bin induces a certain cost, which is calculated by an arbitrary non-decreasing convex *cost function* ϕ. Our goal is to minimize the overall expected cost of the system.

We further impose the following restriction: each operation can look at up to $a < d$ bins *on average*, before deciding where to place the ball. Note that in most reasonable scenarios, checking the status of a bin (e.g., its occupancy) corresponds to either a memory access or a probe over the network. Thus, our restriction can be viewed as imposing a *memory access budget* on the insertion algorithm. *Given this access budget, we aim at achieving the highest-quality balancing.*

The above restrictions imply that a lookup operation (e.g., for updating) is expected to take $a < d$ memory accesses in average for an existing element, and in any case no more than d memory accesses.

1.2. Our Contributions

In this paper, we propose a new access-efficient Bloom Filter architecture for networking devices, called Balanced Bloom Filter. We first maintain balancing schemes to distribute the elements between the memory blocks. At high level, before inserting an element, these schemes query the load of a memory block, and in case it exceeds some threshold the choose another memory block. Memory blocks are chosen by applying hash functions on the inserted element. We also propose to use an optional overflow list to store elements that all hash functions used map to already overloaded memory blocks.

To study this problem, we explore the *optimality region* of the balancing problem. Namely, we consider different balancing schemes at different loads, and determine several selections of the access budget a and the overflow fraction γ such that the balancing scheme is *optimal* with respect to the cost function ϕ. In particular, given some access budget a within a predetermined range, we will show that our scheme is optimal for some $\gamma(a)$, and for any γ satisfying $\gamma \geq \gamma(a)$, thus defining an *optimality region* over the (a, γ) plane.

To show optimality, we first provide lower bounds on the minimum cost of each instance of the problem. The lower bound depends on the access budget a, the number of hash functions d, and the overflow list size, but, quite surprisingly, does not depend on the cost function ϕ. Our lower bounds hold when all hash functions have uniform distribution or when their overall distribution is uniform (in the latter case, the hash function distributions can be different). Then, we provide three different schemes that meet the lower bounds on different access budgets; we further find the minimum size of the overflow list that should be provided in order to achieve optimality. All our analytical models are compared with simulations showing their accuracy.

We conclude by showing how, with a proper choice of the cost function ϕ, the balancing problem can be directly used to optimize Bloom Filters. For example, for an average number of access operations of $a = 1.2$, and $\gamma = 0.5\%$ of the elements stored in the overflow list, the FPR is reduced by up to two orders of magnitude.

Further, since the cost function ϕ is general, our approach can have many applications, such as other Bloom Filter implementations [12], Counting Bloom Filter variants [13, 14], and other applications. We further show how the solution of this problem can be used to construct linked-list–based hash tables with optimal variance.

Paper Organization. We first describe our basic architecture of the Balanced Bloom Filters in Section 2. Then, the optimal balancing problem is defined in Section 3, followed by our lower bound results in Section 4. The three optimal schemes and their analysis are presented in Sections 5, 6, and 7, while a comparative study appears in Section 8. In Section 9 we show how the solution of the balancing problem can be used to construct access-efficient Balanced Bloom Filters. Then, we verify our analysis using trace-driven experiments in Section 10. Finally, Section 11 surveys related work. For brevity, the detailed proofs and an additional application of the access-efficient load-balancing problem are presented in the appendices.

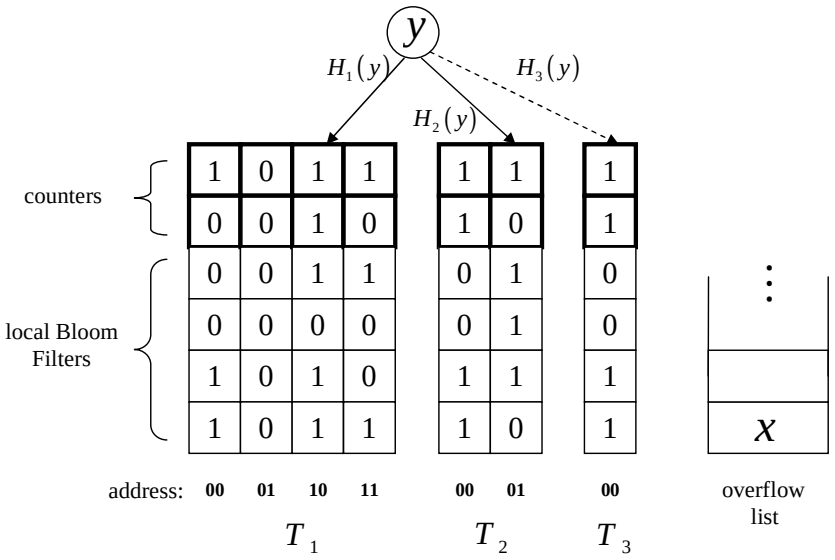


Figure 1: Illustration of a Balanced Bloom Filter implementation based on MHT, with three subtables.

2. Balanced Bloom Filters

In this section, we present the basic architecture of Balanced Bloom Filter. Our architecture follows the two guidelines of balancing the elements between the memory blocks and the usage of an overflow list.

In our basic architecture, illustrated in Fig. 1, each memory block functions as a local Bloom Filter, with the only modification that the memory block also saves some bits for a counter storing the number of elements inserted locally.

Although any balancing scheme can be used, for the purpose of the section we rely on a *special case* of the *multi-level hash table* (MHT) balancing scheme. The memory is divided into d separate subtables $T_1, \ldots, T_d$, with a uniform hash function for each one of these subtables. Upon an element arrival, it is placed in the first subtable in which the corresponding mapped memory block has load lower than a pre-defined threshold h. If no such memory block exists, the element is placed in the overflow list.

A lookup operation follows the same steps as an insertion operation: The local Bloom Filters of the mapped memory blocks are queried one by one, until either the element is found, or a Bloom Filter with load less than h is queried. If all Bloom Filters have full load and the element was not found,

then the overflow list is also queried.

Fig. 1 illustrates an insertion of a new element with $h = 3$. The memory consists of 3 subtables of decreasing size, with 4, 2 and 1 memory blocks, respectively. Each memory block is of size 6 bits, with 2 bits for the counter and 4 for the local Bloom Filter. When element y arrives, it is first hashed into the memory block of subtable T_1 with address 10. The counter at this memory block indicates that 3 elements have already been inserted. Since this load is equal to the threshold $h = 3$, the scheme then tries to insert the element into subtable T_2. In this subtable, the element is hashed into the memory block with address 01, where there are 2 elements. Since $2 < h$, the element is inserted into this memory block. The dashed arrow to subtable T_3 illustrates a hash function that is not actually performed. In addition, the element x is in the overflow list because all of its corresponding buckets were full upon its insertion.

3. Problem Statement

To further study our problem, we first define and solve the *optimal access-constrained balancing problem* in the following sections. In this section, we define the notations and settings of this balancing problem.

Let $\mathcal{B}$ be a set of m *buckets* (or *bins*) of unbounded size, and let $\mathcal{E}$ be a set of n *elements* (or *balls*) that should be distributed among the buckets. In addition, denote by $r = \frac{n}{m}$ *the element-per-bucket ratio.*

Assume also that there exists an *overflow list* [11], i.e. a special bucket of bounded size $\gamma \cdot n$ (namely, at most a fraction γ of the elements can be placed in the list), which can be used by the insertion algorithm at any time. For example, depending on the application, the overflow list may correspond to a dedicated memory—e.g., content-addressable memory (CAM)—in hardware-implemented hash-table, or to the loss ratio when the balancing scheme is allowed to drop elements.

Elements are inserted into either one of the m buckets or the overflow list, according to some balancing scheme with at most d hash-functions per element, which is defined as follows (a similar definition appears in [15]).

Definition 1. *A* balancing scheme *consists of:*
(i) *d hash-function probability distributions over bucket set $\mathcal{B}$, used to generate a* hash-function set $\mathcal{H} = \{H_1, \ldots, H_d\}$ *of d independent random hash functions;*

(ii) *an* insertion algorithm *that places each element $x \in \mathcal{E}$ in one of the d buckets $\{H_1(x), \ldots, H_d(x)\}$ or in the overflow list. The insertion algorithm is an* online *algorithm, which places the elements one after the other with no knowledge of future elements.*

The access-efficiency of a balancing scheme is measured by the number of *bucket accesses* needed to store the incoming elements. We assume that a balancing scheme needs to access a bucket to obtain any information on it. We do not count accesses to the overflow list.

We further consider two constraints, which can be seen as either power- or throughput-constraints depending on the application. First, we require that the *average* number of bucket accesses per element insertion must be bounded by some constant $a \geq 0$. In addition, notice that the *worst-case* number of bucket accesses per element insertion is always bounded by d, because an element does not need to consider any of its d hash functions more than once. These two constraints are captured by the following definition:

Definition 2. *An $\langle a, d, r \rangle$ balancing scheme is a balancing scheme that inserts all elements with an average (respectively, maximum) number of bucket accesses per insertion of at most a (respectively, d), when given an element per bucket ratio r.*

We are now ready to define the *optimal balancing problem*, which is the focus of this paper. Let $\phi : \mathbb{N} \mapsto \mathbb{R}$ be the *cost function* mapping the occupancy of a bucket to its real-valued cost. We assume that ϕ is *non-decreasing* and *convex*. Our goal is to minimize the expected overall cost:

Definition 3. *Let O_j be a random variable that counts the number of elements in the j-th bucket. Given γ, a, d and r, the* OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM *consists of finding an $\langle a, d, r \rangle$ balancing scheme that minimizes*

$$\phi^{\mathrm{BAL}} = \lim_{m \to \infty} \frac{1}{m} \sum_{j=1}^{m} E\left(\phi\left(O_j\right)\right). \tag{1}$$

Whenever defined, let $\phi^{\mathrm{BAL}}_{\mathrm{OPT}}$ denote its optimal cost.

For example, in the trivial case of the identity cost function $\phi(x) = x$ and no overflow list ($\gamma = 0$), $\phi^{\mathrm{BAL}}_{\mathrm{OPT}}$ corresponds to the average load per bucket, which is exactly r, no matter what insertion algorithm or hash functions are used.

4. Theoretical Lower Bounds

We next show a lower bound on the achievable value of the optimal cost $\phi_{\text{OPT}}^{\text{BAL}}$, as a function of the number of buckets m, the number of elements n, the average number of bucket accesses a, and the overflow fraction γ. (The proof appears in Appendix B.1).

The lower bound is derived using a modified *offline* setting. In this setting, each bucket access is considered as a distinct element, as if initially $a \cdot n$ distinct elements were hashed to the buckets, using a single hash function each. After storing all elements, we conceptually choose exactly $(a - 1 + \gamma) \cdot n$ of the elements in a way that minimizes the cost function ϕ^{BAL}, resulting in exactly $(1 - \gamma) \cdot n$ elements in the buckets. Since the cost function ϕ is convex, then the marginal cost is the largest in the most occupied buckets. Therefore, a cost-minimizing removal process would remove element by element, picking the next element to remove in the most occupied bucket at each time.

Since we picked these elements in an offline manner, we necessarily perform better than any online setting. Thus, we bound the achievable value of $\phi_{\text{OPT}}^{\text{BAL}}$.

Theorem 1. *When all hash functions are uniform, the optimal expected limit balancing cost $\phi_{\text{OPT}}^{\text{BAL}}$ in the* OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM *is lower-bounded by*

$$\phi_{\text{LB}}^{\text{BAL}} = \sum_{j=0}^{k_0+1} P_{\text{LB}}(i) \cdot \phi(i), \tag{2}$$

where k_0 is the largest integer such that

$$\frac{a \cdot r \cdot \Gamma(k_0, a \cdot r)}{(k_0 - 1)!} + k_0 \cdot \left(1 - \frac{\Gamma(k_0 + 1, a \cdot r)}{k_0!}\right) < r(1 - \gamma), \tag{3}$$

$\Gamma(s, x) = \int_x^\infty t^{s-1} e^{-t} dt$ is the upper incomplete gamma function, and P_{LB} is a specific distribution that depends only on a, r, and γ, but does not depend on the cost function ϕ.

Specifically, the distribution P_{LB} is defined as follows:

$$P_{\text{LB}}(i) = \begin{cases} e^{-a \cdot r} \frac{(a \cdot r)^i}{i!} & 0 \le i < k_0 \\ e^{-a \cdot r} \frac{(a \cdot r)^{k_0}}{k_0!} + e_0 + k_0 + 1 - \\ \quad k_0 p_0 - p_0 - r \cdot (1 - \gamma) & i = k_0 \\ -e_0 - k_0 + k_0 p_0 + r \cdot (1 - \gamma) & i = k_0 + 1 \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

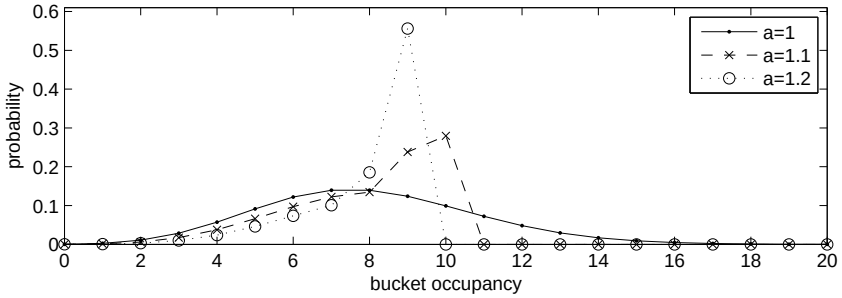


Figure 2: The probability density function of the distribution P_{LB} over the bucket occupancies with load $r = 8$, overflow fraction $\gamma = 0$, and different values of average access rate a.

where $e_0 = \frac{a \cdot r \cdot \Gamma(k_0, a \cdot r)}{(k_0 - 1)!}$, *and* $p_0 = \frac{\Gamma(k_0+1, a \cdot r)}{(k_0)!}$.

Interestingly, the element-elimination algorithm does not depend on the precise convex cost function ϕ. This is why the resulting bucket-load distribution P_{LB} is independent of ϕ as well.

In addition, this distribution P_{LB} is defined on a compact space. As a result, it can also be shown that if a sequence of cost functions $\{\phi_k\}$ converges pointwise to some cost function ϕ, then the sequence of lower bounds converges as well to the corresponding lower bound on ϕ. This can then be used to extend the cost functions to the maximum-load metric commonly used in the literature [9, 16, 17].

Fig. 2 shows the lower-bound distribution $P_{\text{LB}}(i)$ for load $r = \frac{n}{m} = 8$, overflow fraction $\gamma = 0$ and average access rate $a \in \{1, 1.1, 1.2\}$. Note that when $a = 1$, all elements use a single lookup, and therefore there is no element elimination in the offline algorithm. The distribution P_{LB} simply follows a Poisson distribution with parameter $\lambda = r$, as shown using the solid line. Then, for larger values of a, the element elimination algorithm reduces the probability of having a large bin load.

We also consider a setting where $\ell \leq d$ different distributions over the buckets are used by the d hash functions. Denote these distributions by $f^1, \ldots, f^\ell$, and assume that distribution f^i is used by a fraction k_i of the total bucket accesses, with $\sum_{i=1}^{\ell} k_i = 1$. We now show that Theorem 1 holds also in this case when $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$.

Theorem 2. *If* $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$ *then the optimal expected limit balancing cost* $\phi_{\text{OPT}}^{\text{BAL}}$ *in the* OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM *has the same lower bound as in Theorem 1.*

PROOF. The number of elements mapped by the hash functions with distribution f_p to bucket i follows approximately a Poisson distribution with rate $k_p \cdot a \cdot n \cdot f_p(i)$ (see proof of Theorem 1). Since the sum of Poisson random variables is also a Poisson random variable, the total number of elements mapped to the bucket i follows a Poisson distribution with rate $an \sum_{p=1}^{l} k_p f_p(i)$.

Thus, the proof of Theorem 1 implies that if for every bucket i, this rate equals $\frac{an}{m}$, we get the same limit lower bound balancing cost $\phi_{\text{LB}}^{\text{BAL}}$.

□

5. SINGLE - A Single-Choice Balancing Scheme

We have found a lower-bound for the optimal cost. In the sequel, we focus on finding values of a and γ in which we can *match this bound*.

For better intuition, we start by analyzing a simplistic balancing scheme, denoted SINGLE, that is associated with 2 parameters h and p. This scheme only uses a single uniformly-distributed hash function H. Each element is stored in bucket $H(x)$ if it has less than h elements. In case there are exactly h elements, the element is stored in the bucket with probability p and in the overflow list with probability $1-p$. Otherwise, the element is stored in the overflow list.

5.1. Description by Differential Equations

In recent years, several balancing schemes have been modeled using a deterministic system of differential equations [18, 19, 15, 20]. In this section, we adopt this approach and provide a succinct description of SINGLE.

In this approach, we consider the element insertion process as performed between the time $t = 0$ and $t = 1$, that is, at time $t = \frac{j}{n}$ the j-th element is inserted. Furthermore, let $F_i\left(\frac{j}{n}\right)$ denote the fraction of buckets in the hash table that store exactly i elements at time $\frac{j}{n}$, just before element j is inserted, and $\vec{F}\left(\frac{j}{n}\right)$ be the vector of all $F_i\left(\frac{j}{n}\right)$'s. Also, let $\Delta F_i\left(\frac{j+1}{n}\right) \triangleq F_i\left(\frac{j+1}{n}\right) - F_i\left(\frac{j}{n}\right)$ denote the change in the fraction of buckets that store exactly i elements

between times $\frac{j}{n}$ and $\frac{j+1}{n}$. Then

$$\mathbb{E}\left(\Delta F_i\left(\frac{j+1}{n}\right)|\vec{F}\left(\frac{j}{n}\right)\right)=\begin{cases}-\frac{1}{m}F_0\left(\frac{j}{n}\right) & i=0\\ \frac{1}{m}\left(F_{h-1}\left(\frac{j}{n}\right)-p\cdot F_h\left(\frac{j}{n}\right)\right) & i=h\\ \frac{1}{m}\cdot p\cdot F_{h-1}\left(\frac{j}{n}\right) & i=h+1\\ \frac{1}{m}\left(F_{i-1}\left(\frac{j}{n}\right)-F_i\left(\frac{j}{n}\right)\right) & \text{otherwise}\end{cases} \tag{5}$$

At time $t=0$, $F_i(0)=1$ if $i=0$ and 0 otherwise.

The probability that element j hits a bucket storing i elements is $F_i\left(\frac{j}{n}\right)$. Thus, in the first equation, the fraction of empty buckets decreases when element j reaches an empty bucket, which occurs with probability of $F_0\left(\frac{j}{n}\right)$. Likewise, in the second equality, the fraction of buckets that store h elements increases when element j hits a bucket storing $h-1$ elements (with probability of $F_{h-1}\left(\frac{j}{n}\right)$), and decreases with probability of p when the element hits a buckets storing h elements (with total probability of $p\cdot F_h\left(\frac{j}{n}\right)$). In the third equality, the fraction of buckets storing $h+1$ elements increases with probability of q if element j hits a bucket storing h elements. Last, in all other cases, the fraction of buckets storing i elements increases if element j hits a bucket storing $i-1$ elements, and decreases if it hits a bucket storing i elements. Any such increment or decrement is by a value of $\frac{1}{m}$, thus, all equations are multiplied by $\frac{1}{m}$.

By dividing both sides of the equation by $\frac{1}{n}$ and considering the fact that n is large, so that the values of $\Delta F_i\left(\frac{j+1}{n}\right)$ are comparatively very small, we can use the *fluid limit* approximation, which is often very accurate [18]:

$$\frac{df_i(t)}{dt}=\begin{cases}-\frac{n}{m}f_0(t) & i=0\\ \frac{n}{m}\left(f_{i-1}(t)-p\cdot f_i(t)\right) & i=h\\ p\cdot\frac{n}{m}f_{h-1}(t) & i=h+1\\ \frac{n}{m}\left(f_{i-1}(t)-f_i(t)\right) & \text{otherwise}\end{cases} \tag{6}$$

More formally, let $\vec{f}(t)\stackrel{\triangle}{=}(f_1(t),\ldots,f_d(t))$ be the solution of the above set of linear differential equations when assuming $f_0(0)=1$ and $f_i(0)=0$ for each $i\neq 0$. Then, by Kurtz theorems [21, 22, 23], the probability that $\vec{f}$ deviates from $\vec{F}$ by more than some constant ε decays exponentially as a function of n and ε^2 [18]. For further intuition behind this statement, refer to [18] and [24, Chapter 3.4].

Fig. 3 shows the evolution over time of $f_0,\ldots,f_3$ where $r=2.5$, $p=0.5$ and $h=2$, comparing the model with simulated values. In the simulation we used $n=25,000$, and so $m=10,000$.

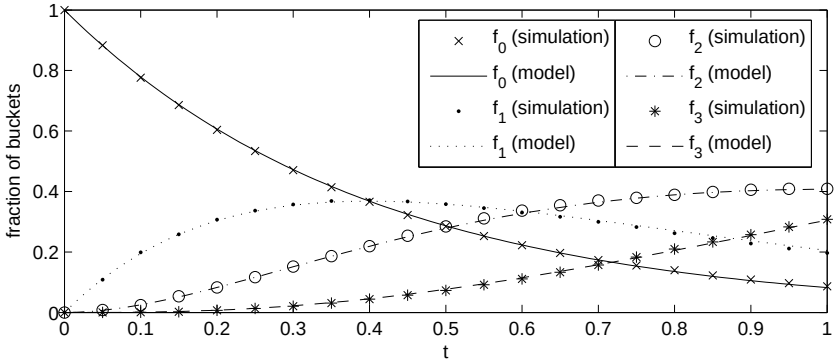


Figure 3: simulation vs. analytical model for SINGLE with $r = 2.5$, $p = 0.5$ and $h = 2$

5.2. Optimality Result

In this section, we solve the system of differential equations yielding the following optimality result.

Theorem 3. *Consider the* SINGLE *balancing scheme with m buckets and n elements, and use the notations of k_0, p_0, e_0 and P from Theorem 1. Then for any value of γ, the* SINGLE *scheme solves the* OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM *for $a = 1$ whenever it satisfies the two following conditions:*
(i) $h = k_0$*;*
(ii) p *is the solution of the following fixed-point equation:*
$\frac{e^{-p\cdot r}}{(1-p)^h} - \frac{e^{-r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(r\cdot(1-p))^i}{i!} = P(k_0)$.

PROOF. We solve the differential equations one by one, substituting the result of the equation for $\frac{df_i(t)}{dt}$ into the equation for $\frac{df_{i+1}(t)}{dt}$. The first equation depends only on $f_0(t)$, and we get immediately that $f_0(t) = e^{-\frac{n}{m}t}$, or $f_0 = e^{-r\cdot t}$. Each equation for $\frac{df_i(t)}{dt}$, where $i \leq h$, depends only on $f_{i-1}(t)$ and $f_i(t)$, and we get that for $i < h$, $f_i(t) = \frac{1}{i!}(r\cdot t)^i e^{-r\cdot t}$.

For $f_h(t)$, we get that for $0 \leq p < 1$

$$f_h(t) = \frac{e^{-p\cdot r\cdot t}}{(1-p)^h} - \frac{e^{-r\cdot t}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(r\cdot t\cdot(1-p))^i}{i!} \tag{7}$$

and for $p = 1$,

$$f_h(t) = \frac{1}{h!}(r\cdot t)^h e^{-r\cdot t}. \tag{8}$$

We also use the fact that $\sum_{i=0}^{h+1} f_i = 1$ to get $f_{h+1}(t)$.

By substituting $t = 1$ in $f_i(t)$, for $i < h$, we find that $f_i(1) = \frac{1}{i!}(r)^i e^{-r}$. We note that it is also the probability that an arbitrary bucket stores i elements, and that it is equal to $P_{\text{LB}}(i)$, thus mimicking the distribution of $P_{\text{LB}}(i)$ for $i < h$.

We are left to show that there exists such a $p \in [0,1]$ so that using its value for $f_h(1)$ will result in the exact expression for $P_{\text{LB}}(h)$. When substituting $p = 0$, we get that $f_h(1) = 1 - e^{-r} \sum_{i=0}^{h-1} \frac{r^i}{i!}$ which is clearly larger than $P_{\text{LB}}(h)$ (it is equal when $P_{\text{LB}}(h+1) = 0$). On the other hand, when substituting $p = 1$, we get that $f_h(1) = \frac{1}{h!}(r)^h e^{-r}$ which is lower than $P_{\text{LB}}(h)$. Thus, using the Intermediate Value Theorem (all functions are clearly continuous), there exists some $p \in [0,1]$ such that $f_h(1) = P_{\text{LB}}(h)$. Since $\sum_{i=0}^{h+1} f_i(1) = \sum_{i=0}^{h+1} P_{\text{LB}}(i) = 1$, we get also that $f_{h+1}(1) = P_{\text{LB}}(h+1)$. □

6. SEQUENTIAL - A Multiple-Choice Balancing Scheme

We now introduce the SEQUENTIAL scheme, which is also associated with two parameters h and p. In the SEQUENTIAL scheme, we use an ordered collection of d hash functions $\mathcal{H} = \{H_1, \ldots, H_d\}$, such that all functions are independent and uniformly distributed. Upon inserting an element x, the scheme successively reads the buckets $H_1(x), H_2(x), \ldots, H_d(x)$, and places x in the first bucket that satisfies one of the following two conditions: *(i)* the bucket stores less than h elements, or, *(ii)* the bucket stores exactly h elements, and x is inserted with probability p. If the insertion algorithm fails to store the element in all the d buckets, x is stored in the overflow list. Last, to keep an average number of bucket accesses per element of at most a, the process stops when a total of $a \cdot n$ bucket accesses has been reached; the remaining elements are placed in the overflow list.

Using the approach presented in Section 5.1, we describe the dynamics of the SEQUENTIAL scheme as a system of differential equations, which appear in Appendix A.1. The resulting system of equations is hard to solve analytically, implying it cannot be used directly to show optimality (naturally, the system can be solved numerically to provide a numerical approximation of the expected balancing cost).

Therefore, we take a different approach. We analyze the SEQUENTIAL scheme by reducing it to the SINGLE scheme: Since both the SINGLE and SEQUENTIAL schemes use the same uniform distribution, a new attempt to

insert an element after an unsuccessful previous attempt in the SEQUENTIAL scheme is equivalent to creating a new element in the SINGLE scheme and then trying to insert it. In other words, the number of elements successfully inserted by the SEQUENTIAL scheme after considering n elements and using a total of $a \cdot n$ bucket accesses is the same as the number of elements successfully inserted by the SINGLE scheme after considering $a \cdot n$ elements. (The proof appears in Appendix B.2.)

Theorem 4. *Consider the* SEQUENTIAL *balancing scheme with m buckets and n elements, and use the notations of k_0, p_0, e_0 and P from Theorem 1. The* SEQUENTIAL *scheme solves the* OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM *whenever it satisfies the three following conditions:*
(i) $h = k_0$*;*
(ii) *all $a \cdot n$ memory accesses are exhausted before or immediately after trying to insert the n-th element;*
(iii) *p is the solution of the following fixed-point equation:*
$\frac{e^{-p \cdot a \cdot r}}{(1-p)^h} - \frac{e^{-a \cdot r}}{(1-p)^h} \sum_{i=0}^{h-1} \frac{(a \cdot r \cdot (1-p))^i}{i!} = P(k_0)$.
Moreover, the optimality region is given by the overflow list of size $\gamma_0 \cdot n$ that results in exhausting all $a \cdot n$ memory immediately after trying to insert the n-th element.

7. The Multi-Level Hash Table (MHT) Balancing Scheme

The *multi-level hash table* (MHT) balancing scheme conceptually consists of d separate subtables $T_1, \ldots, T_d$, where T_i has $\alpha_i \cdot n$ buckets, and d associated hash functions $H_1, \ldots, H_d$, defined such that H_i never returns values of bucket indices outside T_i.

Using the MHT scheme, element x is placed in the smallest i that satisfies one of the following two conditions: *(i)* the bucket $H_i(x)$ stores less than h elements, or, *(ii)* the bucket $H_i(x)$ stores exactly h elements, and element x is then inserted with probability p. If the insertion algorithm fails to store the element in all the d tables, x is placed in the overflow list. Since that smallest i with available space is used, the bucket accesses for each element x are sequential, starting from $H_1(x)$ until a place is found or all d hash functions are used (and the element is stored in the overflow list).

The description of the dynamics of the MHT scheme using a system of differential equations (that appears in Appendix A.2) is generally too difficult

to solve. As in SEQUENTIAL, this can be circumvented by relying on our results from the SINGLE scheme.

We skip to the optimality theorem of the MHT scheme.

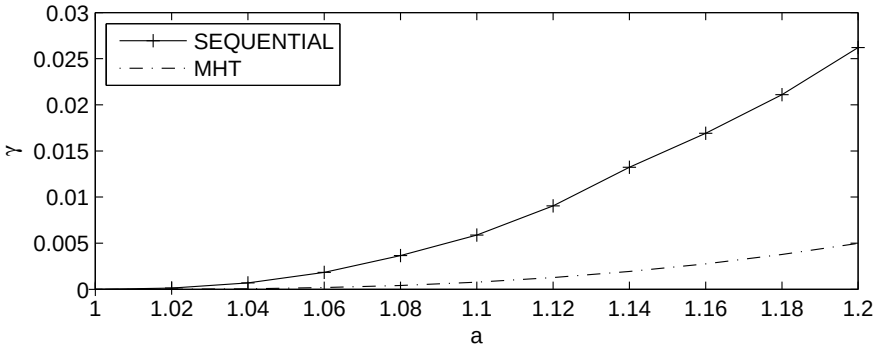
Theorem 5. *Consider an* $\langle a, d, r\rangle$ MHT *balancing scheme in which each subtable* T_j *has* $\alpha_j \cdot m$ *buckets, with* $\sum \alpha_j = 1$*, and use the notations of* k_0*,* p_0*,* e_0 *and* P *from Theorem 1. Further, let* $p(a)$ *denote the overflow fraction of the* SINGLE *scheme with* $a \cdot n$ *elements. Then, the* $\langle a, d, r\rangle$ MHT *scheme solves the* OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM *whenever it satisfies the following four conditions:*
(i) $h = k_0$*;*
(ii) *all* $a \cdot n$ *memory accesses are exhausted before or immediately after trying to insert the* n*-th element;*
(iii) p *is the solution of the following fixed-point equation:*
$\frac{e^{-p\cdot a\cdot r}}{(1-p)^h} - \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(a\cdot r\cdot(1-p))^i}{i!} = P(k_0)$;
(iv) *the subtable sizes* $\alpha_j \cdot m$ *follow a geometric decrease of factor* $p(a)$*:* $\alpha_j = \left(\frac{1-p(a)}{1-p(a)^d}\right) p(a)^{j-1}$.
Moreover, the optimality region is given by the overflow list of size $\gamma \cdot n$ *that results in exhausting all* $a \cdot n$ *memory accesses immediately after trying to insert the* n*-th element. Furthermore, if all four conditions are met then all buckets have an identical occupancy distribution.*

8. Comparative Evaluation and Analysis

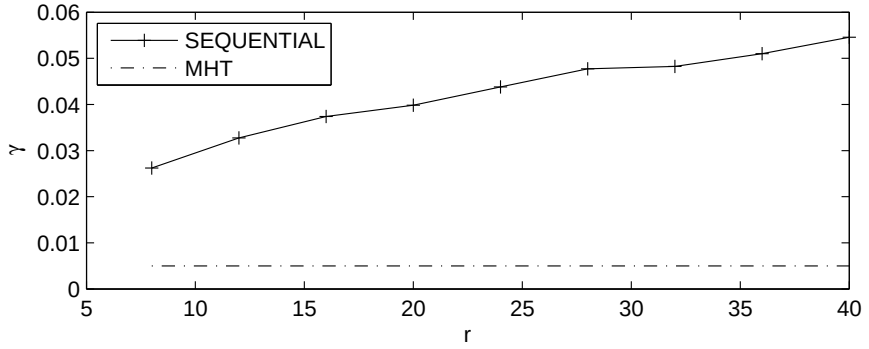
Fig. 4(a) shows the optimality region of SEQUENTIAL and MHT with element-per-bucket ratio $r = 8$, and $d = 3$ hash functions. For each value of the average number of bucket accesses a, it shows the minimum value of the overflow fraction γ that suffices to solve the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM. For instance, we can see that for $a \approx 1.1$, SEQUENTIAL achieves optimality for an overflow fraction equal to or larger than approximately 1%. In addition, Fig. 4(b) shows the optimality region of SEQUENTIAL and MHT with $a = 1.2$ and $d = 3$ for different values of r. We can see that MHT scales better to higher loads.

9. Analysis of the Balanced Bloom Filter

In Section 2 we provided a high-level overview of the Balanced Bloom Filter. In this section, we explain in detail how to use our optimal online



(a) Fixed load $r = 8$; variable average memory access rate a



(b) Fixed average memory access rate $a = 1.2$; variable load r

Figure 4: The overflow fraction γ induced by applying the SEQUENTIAL and MHT schemes with worst-case memory access rate $d = 3$.

schemes to construct the Balanced Bloom Filter, and by this, to achieve a better FPR for networking devices.

9.1. Balanced Bloom Filter with Overflow List

Blocked Bloom Filters [7, 8] form the first attempt to design an access-efficient Bloom Filter. They constrain the k hashed bits to be located in the same memory block, thus causing a single memory access.

The Blocked Bloom Filter mechanism can be modeled using the SINGLE scheme with no overflow list ($\gamma = 0$) and with a cost function ϕ that expresses the FPR incurred to an element in a given memory block given the number

of elements are hashed to this memory block:

$$\phi(i) = \left(1 - \left(1 - \frac{1}{B}\right)^{ki}\right)^{B} \approx \left(1 - e^{-\frac{ki}{B}}\right)^{k}, \tag{9}$$

where B is the size in bits of the memory block and k is the number of hash functions used. Although the SINGLE scheme is optimal, its average number of memory accesses a is 1, thus it achieves poor balancing of the elements resulting in a high FPR. In this section, we explain in detail how to *use our optimal online schemes to achieve a better balancing between the memory blocks, and consequentially, a better FPR.*

Assuming memory blocks of size B bits and bits-per-element ratio β, the number of elements per bucket r is B/β. Using the optimal online balancing schemes described above to implement a Balanced Bloom Filter requires saving $b = \lceil \log_2(k_0 + 2) \rceil$ bits in every memory block to count the elements hashed into each one. Thus, we get $\phi(i) = \left(1 - e^{-\frac{ki}{B-b}}\right)^{k}$.

In the standard Bloom Filter, the optimal FPR is achieved when using $k = r \cdot \ln 2$ hash functions [2]. Although this may not be the best choice in our settings, we will use the same k for simplicity.

Balancing schemes that read more than one memory block on a query operation increase the FPR, because a false positive may result from a query operation on each one of the memory blocks. Thus, since the probability of false positive in every memory block is relatively small, then the overall false positive probability, i.e. the FPR, is modeled by $\sum_{j=1}^{d} P_j \cdot \mathrm{FPR}_j$, where P_j is the probability that a query operation results in reading at least j memory blocks, and FPR_j is the false positive probability of the j-th memory block read. Since in our balancing schemes, there exists some constant FPR, where for all j, $\mathrm{FPR}_j = \mathrm{FPR}$, the expression above reduces to $\sum_{j=1}^{d} P_j \cdot \mathrm{FPR} = \mathbb{E}(P) \cdot \mathrm{FPR}$, where $\mathbb{E}(P)$ is the expected number of memory-block read operations.

Given an average number of memory accesses a in our balancing schemes, $\mathbb{E}(P) \neq a$ in general. Querying the Balanced Bloom Filter with elements that are in the set is expected to have exactly a memory accesses on average. This is because the query memory access pattern follows exactly the insertion memory access pattern. However, for elements that are not in the set, the average number of memory accesses is expected to be $\mathbb{E}(P) \leq d$.

Fig. 5 compares the FPR for different values of bits-per-element ratios with memory block size $B = 256$. The MHT balancing scheme has $a = 1.2$

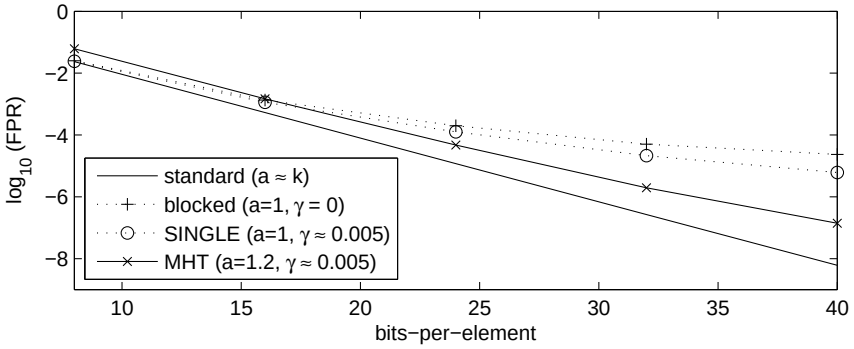


Figure 5: False positive rates of different Bloom Filter schemes with memory block size of $B = 256$ bits and variable load r. SINGLE and MHT use overflow fraction $\gamma \approx 0.5\%$.

and $d = 3$. Thus, by Theorems 1 and 5, we get that the overflow list size needed is $\gamma \approx 0.5\%$. For comparison, Fig. 5 presents the performance of both the SINGLE balancing scheme with the same overflow list size and the Blocked Bloom Filter, which is equivalent to the SINGLE scheme with no overflow list.

Because of the usage of the small overflow list, the SINGLE balancing scheme performs only slightly better than the Blocked Bloom Filter scheme. For low values of bits-per-element ratio, the MHT scheme performs worse. This is due to the need to check multiple memory blocks (up to d) on a query operation. However, demonstrating the power in balancing, for larger values of bits-per-element ratios, the MHT performs better by up to two orders of magnitude, and only one order of magnitude worse than the standard Bloom Filter, which uses an access-inefficient scheme with $a \approx k$ accesses. For example, for a bit-per-element ratio of 24, $k = 17$ hash functions are used, introducing up to 17 memory-read operations in the standard Bloom Filter, which can clearly present memory-throughput and power-consumption issues.

9.2. Balanced Bloom Filter Without Overflow List

In some applications, it may be too expensive to use an overflow list. In such cases, the overflow list can be avoided using a simple modification on our basic scheme. Upon an insertion of a new element, the basic scheme runs as usual, but whenever there is an overflow event, the corresponding element is just put in one of the d choices at random. Because there is only a small fraction of overflow elements, this is not expected to worsen the FPR of the system significantly. However, in the general OPTIMAL ACCESS-CONSTRAINED

BALANCING PROBLEM, we are no longer able to prove theoretically the optimality of the solution, as we could in case one may use an overflow list of a small size.

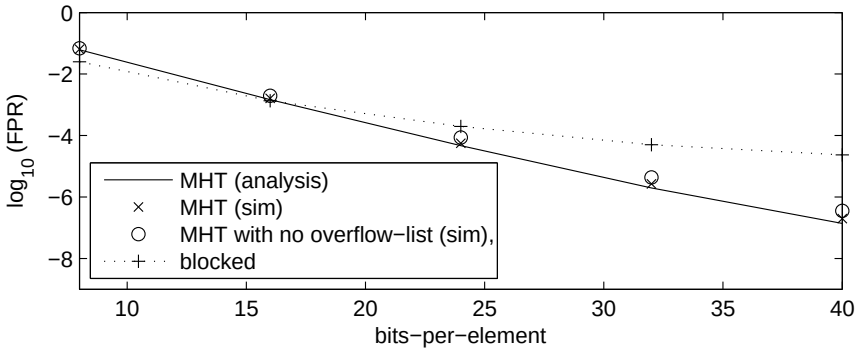
10. Trace-Driven Experiments

We conducted experiments of our proposed MHT-based Balanced Bloom Filter using real-life traces recorded on a single direction of an OC192 backbone link [25], where packets are hashed to the memory blocks using a real 64-bit mix function [26]. Our goal is two-folded. First, we would like to verify that our analysis agrees with results of real-life traces. And second, we would like to evaluate the case where no overflow list is used, as introduced in Section 9.2.

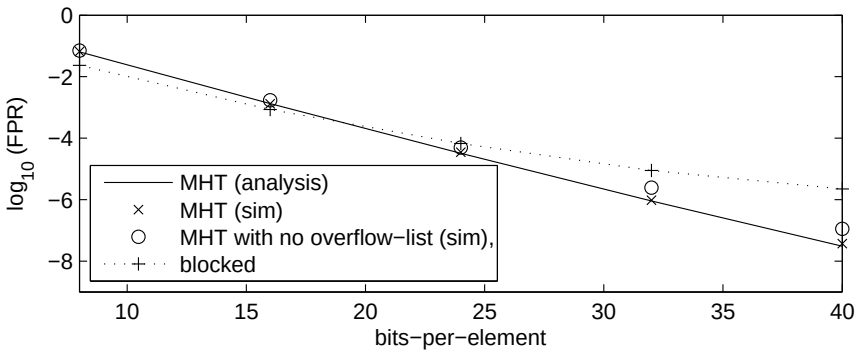
In all experiments, we used the MHT-based Balanced Bloom Filter with $a = 1.2$, $d = 3$, memory block sizes of either $B = 256$ or $B = 512$, a total of 1024 memory blocks, and various values of loads r. By Theorems 1 and 5, we get that the overflow list size required in the basic scheme is $\gamma \approx 0.5\%$. We further find by Theorem 5 the partition of the memory blocks to subtables. Notice that this partition depends on the load r. To hash the elements into the memory blocks, we used a real 64-bit mix function. However, to set the k bits within the corresponding memory block, we used a standard randomization procedure. The total FPR is then computed by the number of set bits and the number of elements recorded by the counter at each memory block.

Fig. 6 shows the FPR found by our experiments of both the basic MHT-based Balanced Bloom Filter scheme, and the variation where no overflow list is used. First, the experiment results of the basic MHT-based scheme verifies the accuracy of out analysis. We note that our experiments also show that the fraction of elements in the overflow list and the average number of access operations per element are approximately $\gamma = 0.5\%$ and $a = 1.2$, respectively. These results further validate our analysis. In addition, if an overflow list is not used, the FPR becomes slightly worse. For a bits-per-element ratio of 40 and a memory block size $B = 256$, our experiments show an FPR of $2.0 \cdot 10^{-7}$ and $3.6 \cdot 10^{-7}$ for the basic scheme and the scheme with no overflow list, respectively. While for the same configuration with a memory block size $B = 512$, experiments show FPRs of $3.7 \cdot 10^{-8}$ and $1.1 \cdot 10^{-7}$.

Finally, Fig. 6 also depicts the FPR of the blocked Bloom Filter [7, 8], and shows that even if no overflow list is used, the FPR can be drastically



(a) memory block size of $B = 256$ bits



(b) memory block size of $B = 512$ bits

Figure 6: Trace-driven experiments of the MHT-based Balanced Bloom Filter scheme different values of r.

reduced: For instance, for a bits-per-element ratio of 40 and a memory block size $B = 256$, our experiments show that the FPR is reduced from $3.4 \cdot 10^{-5}$ to $3.6 \cdot 10^{-7}$. This improvement of approximately two orders of magnitude demonstrates the power of the suggested balancing schemes.

11. Related Work

The Blocked Bloom Filter [8, 7] is one of the first implementations of access efficient Bloom Filters. In the Blocked Bloom Filter, each element is first hashed using a *single* hash function to one of the memory blocks, and then the memory block operates as a local Bloom Filter. Other designs of an access-efficient (or, alternatively, energy-efficient) Bloom Filter exploit the fact that a query returns TRUE if and only if *all* corresponding bits are set to

1 [27, 28, 29]. Thus, the hash functions can be applied sequentially only until some hash function maps to a bit set to 0. In practice, the hash functions may be partitioned into several sets, and the query is performed set by set, where all hash functions in some set are applied in parallel.

We next survey works that are related to the *balancing problem*.

The balancing problem was also extensively investigated in the last decades for various applications involving allocations of resources [30]. Prime examples include task balancing between many machines [31], item distribution over several locations [32], bandwidth allocation in communication channels [30] or within switches and routers [33], and hash-based data structures [34].

Our paper is related to the *sequential static multiple-choice balls-and-bins problem* described above [9, 19]. In this model, n balls are placed sequentially into m bins. Each ball can be placed only in one of d bins that are chosen according to some distribution (e.g., uniformly at random) and the ball is placed in one of these bins following some rule (for example, in the least occupied bin).

The work in [9] shows that when $n = m$ and balls are placed into the least occupied bin, the maximum bin size is $\log\log n/\log 2 + O(1)$ for $d = 2$, compared to approximately $\log n/\log\log n$ for $d = 1$. This result had a large impact on modern algorithms and data structures (see surveys in [19, 35]).

Note that while most papers considered the *maximum load* of the system, while our paper considers the entire load distribution (including the maximum load).

Access-constrained hash-schemes were also considered in [15], which, however, did not consider a cost minimization problem and cannot deal with infinite-size bins. Our paper is different since it deals with a general balancing problem and aims at minimizing a general cost function given a known overflow list size, while [15] considered the size of the overflow list given bounded-size bins.

Finally, we note that concurrently [36], another work has also proposed to balance the elements in the context of Bloom Filters [37]. However, beyond the mere balancing approach, we further provide a theoretical framework for the general problem of balancing the elements subject to a given access budget.

12. Conclusion

In this paper we presented an access-efficient variant of Bloom Filters for networking devices, called the Balanced Bloom Filter. In this variant, each element is first assigned to a memory block, where a local Bloom Filter is maintained. Our basic approach was two-folded. First, we proposed to maintain load-balancing schemes: a simple approach (using only a single hash function), a sequential approach, and a mutli-level hash-table approach. And second, we proposed to use an overflow list that can store elements that are hashed to overloaded buckets.

To study this problem, we presented an access-constrained balancing problem and showed that, for some specific parameters, our proposed load-balancing schemes are optimal.

Since the cost function is very general, the balancing problem can be used in other contexts. We demonstrated how it can be used in order to find optimal balancing schemes that take into account the variance of the query-time and not just its expected or worst-case time.

Acknowledgments

The work was partly supported by the European Research Council Starting Grant n°210389, the European Research Council Starting Grant n°259085, the Alon Fellowship, the ATS-WD Career Development Chair, the Loewengart Research Fund, the Intel ICRI-CI Center, and the Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11).

References

[1] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, COMMUN. ACM 13 (7) (1970) 422–426.

[2] A. Broder, M. Mitzenmacher, Network applications of Bloom filters: A survey, Internet Mathematics 1 (4) (2004) 485–509.

[3] S. Tarkoma, C. Rothenberg, E. Lagerspetz, Theory and practice of Bloom filters for distributed systems, Commun. Surveys Tuts. PP (99) (2011) 1 –25.

[4] Mellanox ib qdr 324p switch system - overview, http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?lang=en&cc=us&taskId=120&prodSeriesId=5033638&prodTypeId=12883&objectID=c01775272.

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, Bigtable: A distributed storage system for structured data, in: OSDI, 2006.

[6] Squid web proxy cache, http://www.squid-cache.org.

[7] F. Putze, P. Sanders, J. Singler, Cache-,hash- and space-efficient Bloom Filters, in: Workshop on Exp. Algorithms, 2007, pp. 108–121.

[8] Y. Qiao, T. Li, S. Chen, One memory access Bloom Filters and their generalization., in: IEEE Infocom, 2011, pp. 1745–1753.

[9] Y. Azar, A. Z. Broder, A. R. Karlin, E. Upfal, Balanced allocations, in: ACM STOC, 1994, pp. 593–602.

[10] N. L. Johnson, S. Kotz, Urn models and their application: an approach to modern discrete probability theory, Wiley NY, 1977.

[11] A. Kirsch, M. Mitzenmacher, U. Wieder, More robust hashing: Cuckoo hashing with a stash, in: ESA, 2008, pp. 611–622.

[12] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, Bloom filters via d-left hashing and dynamic bit reassignment, in: IEEE Allerton, 2006.

[13] O. Rottenstreich, Y. Kanizo, I. Keslassy, The variable-increment counting Bloom filter, in: IEEE Infocom, 2012.

[14] O. Rottenstreich, I. Keslassy, The Bloom paradox: When not to use a Bloom filter?, in: IEEE Infocom, 2012.

[15] Y. Kanizo, D. Hay, I. Keslassy, Optimal fast hashing, in: IEEE Infocom, 2009, pp. 2500–2508.

[16] B. Vöcking, M. Mitzenmacher, The asymptotics of selecting the shortest of two, improved, in: Analytic Methods in Applied Probability, 2002, pp. 165–176.

[17] B. Vöcking, How asymmetry helps load balancing, in: IEEE FOCS, 1999, pp. 131–141.

[18] A. Kirsch, M. Mitzenmacher, The power of one move: Hashing schemes for hardware, in: IEEE Infocom, 2008, pp. 565–573.

[19] M. Mitzenmacher, A. Richa, R. Sitaraman, The power of two random choices: A survey of techniques and results, in: Handbook of Randomized Computing, 2000, pp. 255–312.

[20] Y. Kanizo, D. Hay, I. Keslassy, Hash tables with finite buckets are less resistant to deletions, Computer Networks 56 (4) (2012) 1376–1389.

[21] S. N. Ethier, T. G. Kurtz, Markov processes, John Wiley&Sons, 1986.

[22] T. G. Kurtz, Solutions of ordinary differential equations as limits of pure jump Markov processes, J. of Applied Probability 7 (1) (1970) 49–58.

[23] T. G. Kurtz, Approximation of Population Processes, 1981.

[24] M. Mitzenmacher, The power of two choices in randomized load balancing, Ph.D. thesis, University of California at Berkley (1996).

[25] C. Shannon, E. Aben, K. Claffy, D. E. Andersen, 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`.

[27] T. Kocak, I. Kaya, Low-power Bloom Filter architecture for deep packet inspection, Communications Letters, IEEE 10 (3) (2006) 210 – 212. doi:10.1109/LCOMM.2006.1603387.

[28] M. Paynter, T. Kocak, Fully pipelined Bloom Filter architecture, Communications Letters, IEEE 12 (11) (2008) 855 –857. doi:10.1109/LCOMM.2008.081176.

[29] M. Ahmadi, S. Wong, K-stage pipelined Bloom Filter for packet classification, in: Computational Science and Engineering, 2009. CSE '09. International Conference on, Vol. 2, 2009, pp. 64 –70. doi:10.1109/CSE.2009.408.

[30] Y. Azar, On-line load balancing, Theoretical Computer Science (1992) 218–225.

[31] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, On-line load balancing with applications to machine scheduling and virtual circuit routing, in: ACM STOC, 1993, pp. 623–631.

[32] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, I. Stoica, Load balancing in dynamic structured P2P systems, in: IEEE Infocom, 2004.

[33] I. Keslassy, The load-balanced router, Ph.D. thesis, Stanford Univ. (2004).

[34] G. H. Gonnet, Expected length of the longest probe sequence in hash code searching, J. ACM 28 (2) (1981) 289–304.

[35] A. Kirsch, M. Mitzenmacher, G. Varghese., Hash-based techniques for high-speed packet processing, in: Algorithms for Next Generation Networks, Springer London, 2010, Ch. 9, pp. 181–218.

[36] Y. Kanizo, D. Hay, I. Keslassy, Energy-constrained balancing, Technical report tr09-02, Comnet, Technion, Israel (2009).
URL `http://comnet.technion.ac.il/~isaac/papers.html`

[37] E. Krimer, M. Erez, The power of $1 + \alpha$ for memory-efficient Bloom filters, Internet Mathematics 7 (1) (2011) 28 – 44.

[38] A. D. Barbour, P. G. Hall, On the rate of Poisson convergence, Math. Proc. Cambridge Philos. Soc. 95 (3) (1984) 473–480.

[39] J. M. Steele, Le Cam's inequality and Poisson approximations, American Mathematical Monthly 101 (1994) 48–54.

Appendix A. The Dynamics of the Proposed Schemes

Appendix A.1. The SEQUENTIAL *Scheme*

We start analyzing the SEQUENTIAL scheme by first assuming that there is no constraint on the total number of bucket accesses (that is, $a = \infty$) and characterizing the dynamics of the scheme as a system of differential equations.

As before, let $f_i(t)$ represent the fraction of buckets storing i elements at time t, then

$$\frac{df_i(t)}{dt} = \begin{cases} -\frac{n}{m} \cdot f_0 \cdot (t)\, g(t) & i = 0 \\ \frac{n}{m} \cdot (f_{h-1}(t) - p \cdot f_h(t))\, g(t) & i = h \\ \frac{n}{m} \cdot p \cdot f_h(t)\, g(t) & i = h+1 \\ \frac{n}{m} \cdot (f_{i-1}(t) - f_i(t))\, g(t) & \text{otherwise} \end{cases} \tag{A.1}$$

where

$$\begin{aligned} g(t) &= \sum_{k=0}^{d-1} ((1-p) \cdot f_h(t) + f_{h+1}(t))^k \\ &= \frac{1 - ((1-p) \cdot f_h(t) + f_{h+1}(t))^d}{1 - ((1-p) \cdot f_h(t) + f_{h+1}(t))}, \end{aligned} \tag{A.2}$$

with $f_0(0) = 1$ and $f_i(0) = 0$ for each $i \neq 0$ as an initial condition. Comparing with the differential equations of the SINGLE scheme (Equation (5)), there is an additional factor of $g(t)$. For instance, in the first equation, $f_0(t)$ is replaced by $f_0(t)\, g(t) = \sum_{k=0}^{d-1} \left[((1-p) \cdot f_h(t) + f_{h+1}(t))^k \cdot f_0(t) \right]$, which represents the sum of the probabilities of mapping to an empty bucket after being mapped $k = 0, 1, \ldots, d-1$ times to a bucket in which the element could not be stored; in other blocks, each bucket either already had $h+1$ elements, or had h element but with probability of $1-p$ the element was denied insertion.

We are also interested in the average number of bucket accesses performed during this process. Let $f^a_{\text{SEQUENTIAL}}(t)$ denote the cumulative number of bucket accesses performed by time t, normalized by n. It can be modeled as

$$\frac{df^a_{\text{SEQUENTIAL}}(t)}{dt} = \sum_{k=1}^{d-1} k \cdot (f_n(t))^{k-1}(1 - f_n(t)) + d \cdot (f_n(t))^{d-1}, \tag{A.3}$$

where

$$f_n(t) = ((1-p) \cdot f_h(t) + f_{h+1}(t)); \tag{A.4}$$

and $f^a_{\text{SEQUENTIAL}}(0) = 0$ as an initial condition.

The differential equation reflects the fact that at a given time t, the cumulative number of bucket accesses increases by $1 \leq k < d$ bucket accesses whenever in the first $k-1$ bucket accesses an element is not stored and in the

next one the element is stored. It also increases by d bucket accesses whenever in the first $d-1$ bucket accesses an element is not stored, independently of the bucket state in the d-th bucket access.

Appendix A.2. The MHT *Scheme*

The system of differential equations that characterizes the dynamics of MHT is influenced by the static partitioning of the memory among subtables, which introduces extra variables. Specifically, let $f_{i,j}(t)$ be the fraction of buckets in subtable T_j that store exactly i elements. Then:

$$\frac{df_{i,j}(t)}{dt} = \begin{cases} -\frac{n}{\alpha_j m} f_{0,j}(t)\, g_j(t) & i = 0 \\ \frac{n}{\alpha_j m}\left(f_{h-1,j}(t) - p \cdot f_{h,j}(t)\right) g_j(t) & i = h \\ \frac{n}{\alpha_j m} \cdot p \cdot f_{h,j}(t)\, g_j(t) & i = h+1 \\ \frac{n}{\alpha_j m}\left(f_{i-1,j}(t) - f_{i,j}(t)\right) g_j(t) & \text{otherwise} \end{cases} \tag{A.5}$$

where

$$g_j(t) = \prod_{k=1}^{j-1} \left((1-p) f_{h,k}(t) + f_{h+1,k}(t)\right) \tag{A.6}$$

represents the probability that all the insertion attempts in subtables $T_1, \cdots, T_{j-1}$ do not result in storing the element, and thus that MHT will attempt to insert the element in subtable T_j. By convention $g_1(t) = 1$. The initial conditions are $f_{i,j}(0) = 1$ for $i = 0$ and $f_{i,j}(0) = 0$ otherwise.

As in the SEQUENTIAL scheme, let $f^a_{\text{MHT}}(t)$ denote the cumulative number of bucket accesses done by time t, normalized by n. Then the following differential equation reflects the dynamics of $f^a_{\text{MHT}}(t)$:

$$\frac{df^a_{\text{MHT}}(t)}{dt} = d \cdot g_d(t) + \sum_{k=1}^{d-1} k \cdot g_k(t)\left(1 - (1-p) f_{h,k}(t) - f_{h+1,k}(t)\right), \tag{A.7}$$

with $f^a_{\text{MHT}}(0) = 0$.

This description of the dynamics of the MHT scheme using a system of differential equations is difficult to solve. Our approach relies on the fact that *each subtable follows a local* SINGLE *scheme*. More specifically, all elements attempting to access some subtable T_j only access a single uniformly-distributed bucket in T_j, and if this bucket is full, do not consider any other bucket in T_j. Thus, within each subtable T_j, MHT behaves like SINGLE, with a number of initial elements that depends on previous subtables.

Let $f_i^{\text{SINGLE}}(t)$ be the fraction of buckets that store exactly i elements at time t in the SINGLE scheme. As in the proof of Theorem 3, it is given by:

$$f_i^{\text{SINGLE}}(t) = \begin{cases} \frac{1}{i!}(r\cdot t)^i e^{-r\cdot t} & i < h \\ \frac{e^{-p\cdot r\cdot t}}{(1-p)^h} - \frac{e^{-r\cdot t}}{(1-p)^h}\sum_{j=0}^{h-1}\frac{(r\cdot t\cdot(1-p))^j}{j!} & i = h \\ 1 - \sum_{j=0}^{h-1}\frac{1}{j!}(r\cdot t)^j e^{-r\cdot t} - & \\ \frac{e^{-p\cdot r\cdot t}}{(1-p)^h} + \frac{e^{-r\cdot t}}{(1-p)^h}\sum_{j=0}^{h-1}\frac{(r\cdot t\cdot(1-p))^j}{j!} & i = h+1 \end{cases} \tag{A.8}$$

Also, let $\gamma_{\text{SINGLE}}^t(t)$ be the fraction of the elements that are not stored in the buckets out of all the elements that arrived up to time t. Given all $f_i^{\text{SINGLE}}(t)$'s, it is simply the complementary of the expectation of the number of elements in the buckets up to time t normalized by the total number of elements arrived up to this time:

$$\gamma_{\text{SINGLE}}^t(t) = 1 - \frac{m}{nt}\cdot\sum_{i=0}^{h+1} i f_i^{\text{SINGLE}}(t). \tag{A.9}$$

Let $n_j(t)$ denote the number of elements that are considered in subtable T_j up to time t, and $\gamma_j^t(t)$ denote the fraction of these elements that are not placed in subtable T_j. We will express these using f_i^{SINGLE} and γ_{SINGLE}^t, the corresponding functions in the SINGLE scheme.

Note that as shown in Equations (A.8) and (A.9), $f_i^{\text{SINGLE}}(t)$ and $\gamma_{\text{SINGLE}}^t(t)$ only depend on the time t, the number of elements n, the number of buckets m, the bucket size h and the probability p; thus, we refer to them as $f_i^{\text{SINGLE}}(t,m,n,h,p)$ and $\gamma_{\text{SINGLE}}^t(t,m,n,h,p)$. We obtain the following theorem, which is valid for any arbitrary partition of the subtables.

Theorem 6. *Consider an* $\langle a,d,r\rangle$ MHT *balancing scheme in which for each* $1\le j\le d$, *subtable* T_j *has* $\alpha_j\cdot m$ *buckets, with* $\sum\alpha_j = 1$. *Then, as long as* $f_{\text{MHT}}^a(t)\le a$, *the functions* $n_j(t)$, $\gamma_j^t(t)$ *and* $f_{i,j}(t)$ *satisfy*

$$n_j(t) = n\cdot t\cdot\prod_{k=1}^{j-1}\gamma_k^t(t), \tag{A.10}$$

$$\gamma_j^t(t) = \gamma_{\text{SINGLE}}^t(1,\alpha_j m, n_j(t), h, p), \tag{A.11}$$

$$f_{i,j}(t) = f_i^{\text{SINGLE}}(1,\alpha_j m, n_j(t), h, p). \tag{A.12}$$

PROOF. By the definition of the MHT scheme, it follows immediately that $n_j(t) = \gamma_{j-1}^t(t)\, n_{j-1}(t)$; since $n_1(t) = n \cdot t$ (all elements go through the first subtable), we get that $n_j(t) = n \cdot t \cdot \prod_{k=1}^{j-1} \gamma_k^t(t)$.

Equations (A.11) and (A.12) are immediately derived by setting the right parameters for each SINGLE scheme within each subtable T_j; namely, its total number of buckets is $\alpha_j \cdot m$ and the number of elements by time t is $n_j(t)$. □

Appendix B. Omitted Proofs

Appendix B.1. Proof of Theorem 1

We derive the lower bound on the balancing by computing the best-case distribution of each bucket in an offline setting. We assume that whenever an hash function points to some bucket, an element is inserted into this bucket, having a total of $a \cdot n$ elements at the end of the process. Then, we remove exactly $(a - 1 + \gamma) \cdot n$ elements, which results in $(1 - \gamma) \cdot n$ total elements in the buckets. We remove the elements in a way that minimizes the cost function ϕ^{BAL}.

In fact, by the convexity of the cost function ϕ, minimizing the total cost ϕ^{BAL} can be done by removing the $(a - 1 + \gamma) \cdot n$ elements greedily, each time from one of the most occupied buckets. This is because the marginal cost is the largest (due to convexity) in those buckets. In the sequel, we relate to this process as the *removal process*.

We consider every hash value as a distinct element. Therefore, the number of elements (out of total $a \cdot n$ elements) that are mapped to bucket $j \in \mathcal{B}$ follows a Binomial distribution with $a \cdot n$ independent experiments and a success probability of $\frac{1}{m}$. Let $Q_j(i) = \binom{an}{i} \left(\frac{1}{m}\right)^i \left(1 - \frac{1}{m}\right)^{an-i}$ denote the probability that bucket j stores i elements before the removal process.

Let $M_j(i)$ be the probability that bucket j stores i elements *after* the removal process. As we show now, $M_j(i)$ has to satisfy two constraints. First, since in the removal process elements are only removed (and not inserted), then the probability that some bucket stores less than i elements after the removal process cannot be larger than before the removal process. Thus, for every i:

$$\sum_{k=0}^{i} M_j(k) \geq \sum_{k=0}^{i} Q_j(k). \tag{B.1}$$

Second, since we end up with exactly $(1-\gamma)\cdot n$ elements, then:

$$\sum_{k=1}^{m}\left(\sum_{i=0}^{\infty} i\cdot M_k\left(i\right)\right) = (1-\gamma)\cdot n, \tag{B.2}$$

that is, the expected number of elements in all the buckets after the removal process must be $(1-\gamma)\cdot n$.

As we are looking for a lower bound on the balancing cost, our goal is to pick the bucket distribution that minimizes that cost. Consider bucket j and assume that the expected occupancy after the removal process is E_0. Since all hash functions are uniform, by symmetry E_0 must be at most $\frac{an}{m}$. We construct the following distribution that minimizes the balancing cost of bucket j. The idea is to keep the original probabilities for low values of buffer occupancies, until the point where the expected occupancy E_0 is reached. On this point, we share the remaining probabilities such that we get the exact expected occupancy. Specifically, let k_0 be the largest integer such that

$$\sum_{i=0}^{k_0} i\cdot Q_j\left(i\right) + k_0\cdot\left(1-\sum_{i=0}^{k_0} Q_j\left(i\right)\right) < E_0. \tag{B.3}$$

That is, k_0 is the buffer occupancy until which we keep the original probability. Let $e_0 = \sum_{i=0}^{k_0} i\cdot Q_j\left(i\right)$ and $p_0 = \sum_{i=0}^{k_0} Q_j\left(i\right)$. In the sequel, we use e_0 and p_0 to construct the remainder of the distribution, that is, the probability for buffer occupancies k_0 and k_0+1.

We define the following distribution $P_j\left(i\right)$:

$$P_j\left(i\right) \;=\; \begin{cases} Q_j\left(i\right) & 0\le i < k_0 \\ Q_j\left(i\right) + e_0 + k_0 + 1 \\ \quad -k_0p_0 - p_0 - E_0 & i = k_0 \\ -e_0 - k_0 + k_0p_0 + E_0 & i = k_0+1 \\ 0 & \text{otherwise} \end{cases} \tag{B.4}$$

$P_j\left(i\right)$ satisfies both constraints from Equations (B.1) and (B.2). First, since we kept the original probabilities until buffer occupancy k_0, and then shared the remaining probabilities between k_0 and k_0+1, then for every i, $\sum_{k=0}^{i} P_j\left(k\right) \ge \sum_{k=0}^{i} Q_j\left(k\right)$. Second, let $\widetilde{P}_j\left(i\right)$ be the random variable that corresponds to the distribution $P_j\left(i\right)$. Then, the expected number of

elements in bucket j is:

$$\begin{aligned}\mathbb{E}\left(\widetilde{P}_j(i)\right) &= \sum_{i=0}^{k_0+1} i \cdot P_j(i) \\ &= e_0 - k_0 Q_j(k_0) + k_0 P_j(k_0) + \\ &\quad (k_0+1) P_j(k_0+1) \\ &= E_0\end{aligned} \tag{B.5}$$

Thus, $P_j(i)$ satisfies the two constraints.

We now show that it minimizes the cost function, over all distributions that satisfy both constraints. Let $G_j(i)$ be a distribution over the buffer occupancies after the removal process that satisfies both constraints. Let i_0 be the smallest integer such that $G_j(i_0) \neq P_j(i_0)$; if such i_0 does not exist, we are done since $G_j(i)$ coincides with $P_j(i)$. We will show that $G_j(i_0) > P_j(i_0)$. Also, let i_1 be the largest integer such that $G_j(i_1) > P_j(i_1)$.

We now show that if i_0 and i_1 are defined, then $G_j(i_0) > P_j(i_0)$, and $i_1 - i_0 \geq 2$. We distinguish between 3 cases: $i_0 > k_0$, $i_0 = k_0$ and $i_0 < k_0$.

First, in case of $i_0 > k_0$, for every bucket occupancy $i \leq k_0$, $G_j(i) = P_j(i)$. Thus, $G_j(i) = P_j(i)$ for every i, as $G_j(i)$ satisfies the second constraint (Equation (B.2)), implying that i_0 and i_1 are not defined.

In case $i_0 < k_0$, by the first constraint (Equation (B.1)) and the fact that $P_j(i) = Q_j(i)$ for every $i < i_0$, we get that $G_j(i_0) > P_j(i_0)$. We now show that $i_1 > k_0$, implying that $i_1 - i_0 \geq 2$. Assume on the contrary that $i_1 \leq k_0$, then $G_j(k_0+1) \leq P_j(k_0+1)$ and for every $i > k_0+1$, $G_j(i) = 0$. Let $\widetilde{G}_j(i)$ be the random variable that corresponds to $G_j(i)$. Since $\mathbb{E}\{\widetilde{G}_j(i)\} = \mathbb{E}\{\widetilde{P}_j(i)\}$ and for any random variable X that takes values in $\mathbb{N}$, $\mathbb{E}(X) = \sum_{\ell=1}^{\infty} \Pr\{X \geq \ell\}$, we get that

$$\sum_{i=1}^{k_0+1} \sum_{\ell=i}^{\infty} G_j(\ell) = \sum_{i=1}^{k_0+1} \sum_{k=i}^{\infty} P_j(\ell). \tag{B.6}$$

Thus,

$$\sum_{i=1}^{k_0+1} \left[\sum_{\ell=i}^{\infty} G_j(\ell) - \sum_{\ell=i}^{\infty} P_j(\ell) \right] = 0. \tag{B.7}$$

By the definition of $P_j(i)$, for every $i \leq k_0$, $\sum_{\ell=i}^{\infty} P_j(\ell) = \sum_{\ell=i}^{\infty} Q_j(\ell)$. So,

$$\sum_{i=1}^{k_0} \left[\sum_{\ell=i}^{\infty} G_j(\ell) - \sum_{\ell=i}^{\infty} Q_j(\ell) \right] + G_j(k_0+1) - P_j(k_0+1) = 0. \tag{B.8}$$

The first constraint (Equation (B.1)) states that $\sum_{\ell=0}^{i} G_j(\ell) \geq \sum_{\ell=0}^{i} Q_j(\ell)$, thus, $\sum_{\ell=i}^{\infty} G_j(\ell) \leq \sum_{\ell=i}^{\infty} Q_j(\ell)$. Also, we know that $G_j(k_0+1) \leq P_j(k_0+1)$. Since $G_j(i_0) \neq Q_j(i_0)$, we get that the total sum cannot be zero, that is, at least one element in the sum is negative (but none is positive). Therefore, $i_1 > k_0$.

The last case to consider is when $i_0 = k_0$. If $G_j(i_0) < P_j(i_0)$, then $G_j(i)$ clearly does not satisfy the second constraint (Equation (B.2)) as for every $i < k_0$, $G_j(i_0) = P_j(i_0)$. Therefore, $G_j(i_0) > P_j(i_0)$. Furthermore, the second constraint implies that there must be some integer $i_1 > k_0 + 1$ such that $G_j(i_1) \neq 0$. Therefore, $i_1 - i_0 \geq 2$.

We are now ready to define another distribution, $G'_j(i)$, which also has minimal cost function:

$$G'_j(i) = \begin{cases} G_j(i) - w & i \in \{i_0, i_1\} \\ G_j(i) + w & i \in \{i_0+1, i_1-1\} \\ G_j(i) & \text{otherwise} \end{cases} \tag{B.9}$$

where $w = \min\{G_j(i_0) - P_j(i_0), G_j(i_1) - P_j(i_1)\}$. Notice that $G'_j(i)$ is well-defined since $i_1 - i_0 \geq 2$. In addition, $w > 0$ since $G_j(i_0) > P_j(i_0)$ and $G_j(i_1) > P_j(i_1)$. Hence, $G'_j(i)$, which clearly preserves both constraints, has a cost no larger than $G_j(i)$. By continuing this process, we end up with $P_j(i)$ no matter what $G_j(i)$ is, as $i_1 - i_0$ decreases at each step by at least 1. This implies that $P_j(i)$ minimizes the cost function.

Finally, since we are interested in the *limit* balancing cost lower bound $\phi_{\text{LB}}^{\text{BAL}}$, we consider the limit distribution $P_{\text{LB}}(i)$ of the distribution $P_j(i)$ that was found to be optimal for any finite parameters. This is done by using the Poisson approximation for the binomial distribution $Q_j(i)$ of the buckets occupancy before the removal process [15, 38, 39], where we use the same approximation to find the values of k_0, p_0 and e_0. Also, by symmetry we get that $E_0 = \frac{(1-\gamma)\cdot n}{m} = r \cdot (1-\gamma)$. □

Appendix B.2. Proof of Theorem 4

We compare the SEQUENTIAL scheme with the SINGLE scheme. In the SEQUENTIAL scheme we continually try to insert each element, until either it is placed or all d functions are used.

Note that all hash functions have the same (uniform) distribution over all buckets. Thus, for every i, $f_i(t)$ are independent of the exact elements that are hashed. Therefore, applying $d_1 \leq d$ hash functions on the same element is equivalent to applying a single hash function on d_1 elements. This implies

we can use the results of the SINGLE scheme, in which a total of $a \cdot n$ elements are considered and therefore a total of $a \cdot n$ bucket accesses are performed.

Given an average number of bucket accesses a, we set $h = k_0$, and let SEQUENTIAL operate until a is reached, that is, until time $t_0 \leq 1$ such that $f^a_{\text{SEQUENTIAL}}(t_0) = a$. We apply the SINGLE dynamics to get $f_i(t_0)$, and get that

$$f_i(t_0) = \begin{cases} \frac{1}{i!}(a \cdot r)^i e^{-a \cdot r} & i < h \\ \frac{e^{-p \cdot a \cdot r}}{(1-p)^h} - \frac{e^{-a \cdot r}}{(1-p)^h} \sum_{j=0}^{h-1} \frac{(a \cdot r \cdot (1-p))^j}{j!} & i = h \\ 1 - \sum_{j=0}^{h-1} \frac{1}{j!}(a \cdot r)^j e^{-a \cdot r} - \frac{e^{-p \cdot a \cdot r}}{(1-p)^h} + & \\ \frac{e^{-a \cdot r}}{(1-p)^h} \sum_{j=0}^{h-1} \frac{(a \cdot r \cdot (1-p))^j}{j!} & i = h+1 \end{cases} \tag{B.10}$$

Using the same consideration as in the proof of Theorem 3, we find that there is some $p \in [0,1]$ such that $f_i(t_0)$ is exactly $P_{\text{LB}}(i)$, for every i.

However, note that the SEQUENTIAL scheme cannot bring to any desired number of bucket accesses a, and is limited to $f^a_{\text{SEQUENTIAL}}(1)$. Since $f^a_{\text{SEQUENTIAL}}(t)$ increases as k_0 decreases, and k_0 decreases as γ increases, then the minimum γ such that the SEQUENTIAL scheme achieves optimality, for a given a and r, is given by γ_0 such that $f^a_{\text{SEQUENTIAL}}(1) = a$. Thus, we get the optimality region of this scheme. □

Appendix B.3. Proof of Theorem 5

Given the average number of bucket accesses a, we set $h = k_0$. We would like to let MHT operate until a is reached, that is, until time $t_0 \leq 1$ such that $f^a_{\text{MHT}}(t_0) = a$. However, $f^a_{\text{MHT}}(t)$ depends on the subtables sizes α_j's.

Up to time t_0, in which we aim to exhaust all $a \cdot n$ bucket accesses, we used exactly $n_j(t_0)$ times the hash function $H_j(x)$ in subtable T_j. Since we aim at an optimal balancing cost, the necessary condition on the distributions of the hash functions, given in Theorem 2, immediately implies that

$$\alpha_j = \frac{n_j(t_0)}{n \cdot a}. \tag{B.11}$$

By substituting the expression for α_j in (A.10), we get:

$$\begin{aligned}
\gamma_j^t(t_0) &= \gamma_{\text{SINGLE}}^t\left(1, \frac{n_j(t_0)}{n\cdot a}m, n_j(t), h, p\right) \\
&= 1 - \frac{\frac{n_j(t_0)}{n\cdot a}m}{n_j(t)} \cdot \sum_{i=0}^{h+1} i f_i^{\text{SINGLE}}\left(1, \frac{n_j(t_0)}{n\cdot a}m, n_j(t), h, p\right) \\
&= 1 - \frac{m}{n\cdot a} \cdot \sum_{i=0}^{h+1} i f_i^{\text{SINGLE}}(1, m, a\cdot n, h, p) \\
&= \gamma_{\text{SINGLE}}^t(1, m, a\cdot n, h, p) = p(a) \qquad \text{(B.12)}
\end{aligned}$$

It is important to notice that, quite surprisingly, $\gamma_j^t(t_0)$ does not depend on j.

We now obtain the time t_0 by observing that $n_j(t_0) = n\cdot t_0 \cdot p(a)^{j-1}$, thus $\alpha_j = \frac{t_0\cdot p(a)^{j-1}}{a}$. Since $\sum_{k=1}^d \alpha_k = 1$, we get $\sum_{k=1}^d \frac{t_0 p^{j-1}}{a} = 1$, and therefore t_0 is given by the sum of a geometric series:

$$t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right). \qquad \text{(B.13)}$$

This, in turn, immediately gives us the claimed memory partitioning α_j.

We now turn to show that all bucket distributions are identical. In subtable T_j, the total number of elements considered is $n_j(t_0) = n\cdot t_0\cdot p(a)^{j-1}$, while there are $m\cdot\alpha_j = m\cdot\left(\frac{1-p(a)}{1-p(a)^d}\right)p(a)^{j-1}$ buckets. Hence, by Theorem 6, we get that the fraction of buckets in subtable T_j that store exactly i elements $f_{i,j}(t)$ is given by

$$f_i^{\text{SINGLE}}\left(1, m\cdot\left(\frac{1-p(a)}{1-p(a)^d}\right)p(a)^{j-1}, n\cdot t_0\cdot p(a)^{j-1}, h, p\right) \qquad \text{(B.14)}$$

and by substituting $t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right)$, we get that $f_{i,j}(t) = f_i^{\text{SINGLE}}(1, m, a\cdot n, h, p)$.

Finally, as in the proof of Theorem 4, the MHT scheme cannot bring to any desired average number of bucket accesses a, but is limited to $f_{\text{MHT}}^a(1)$. Since $f_{\text{MHT}}^a(t)$ increases as k_0 decreases, and k_0 decreases as γ increases, then the minimum γ such that the MHT scheme achieves optimality, for a given a and r, is given by γ_0 such that $f_{\text{MHT}}^a(1) = a$. Since $t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right)$, we seek γ such that $a = \frac{1-p(a)^d}{1-p(a)}$. Thus, we get the optimality region of this scheme. □

Appendix C. Variance of the Query-Time in Chain-Based Hash-Tables

The balancing problem can be directly used in order to construct an access-constrained balancing scheme with optimal variance over its query time. In such a scheme, the time it takes to complete a *lookup* operation directly depends on the occupancy of the buckets. For example, suppose that there is only one hash-function H, and some element y is mapped to a bucket $H(y)$; in order to query y we need to go over all elements of $H(y)$ in case y is not in the table, or on average over half of them in case y is in the hash-table.

Note that the *average load* is simple to obtain and equals $\frac{(1-\gamma)n}{m}$. We next show how to find balancing schemes with minimal occupancy variance. This is simply done by defining the appropriate cost function: $\phi(i) = i^2 - \left(\frac{(1-\gamma)n}{m}\right)^2$.

Thus,

$$\begin{aligned}\phi^{\mathrm{BAL}} &= \lim_{m\to\infty}\frac{1}{m}\sum_{j=1}^{m}\mathbb{E}\left(O_j^2-\left(\frac{(1-\gamma)\,n}{m}\right)^2\right)=\lim_{m\to\infty}\frac{1}{m}\sum_{j=1}^{m}\mathbb{E}(O_j^2)-\mathbb{E}(O_j)^2\\ &= \lim_{m\to\infty}\frac{1}{m}\sum_{j=1}^{m}\mathrm{Var}(O_j)=\mathrm{Var}(O), \qquad \text{(C.1)}\end{aligned}$$

where O_j is the random variable representing the occupancy of bucket j. By symmetry, all variables O_j have the same distribution and thus the same variance, which is denoted by Var(O). This immediately implies that the schemes we presented can be used in order to build a hash-table with optimal variance.

Vitae

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.

David Hay received the B.A. (summa cum laude) and Ph.D. degrees in computer science from the Technion Israel Institute of Technology, Haifa, Israel, in 2001 and 2007, respectively. He is currently a Senior Lecturer in the School of Computer Science and Engineering, Hebrew University, Jerusalem Israel. Previously, he was a Post-Doctoral Research Scientist with the Department of Electrical Engineering, Columbia University, New York, NY and with the Department of Electrical Engineering, Politecnico di Torino, Italy. His main research interests are algorithmic aspects of high-performance switches and routers in particular, QoS provisioning and packet classification.

Isaac Keslassy received the 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, Haifa, Israel. His recent research interests include the design and analysis of high-performance routers and on-chip networks. He is the recipient of the Yigal Alon Fellowship, the ATS-WD Career Development Chair and the ERC Starting Grant.