Optimal Fast Hashing

Yossi Kanizo, David Hay, and Isaac Keslassy

***Abstract*—This paper is about designing *optimal high-throughput hashing schemes* that minimize the *total* number of memory accesses needed to build and access an hash table. Recent schemes often promote the use of multiple-choice hashing. However, such a choice also implies a significant increase in the number of memory accesses to the hash table, which translates into higher power consumption and lower throughput. In this paper, we propose to only use choice when needed. Given some target hash table overflow rate, we provide a lower bound on the total number of needed memory accesses. Then, we design and analyze schemes that provably achieve this lower bound over a large range of target overflow values. Further, for the multi-level hash table scheme, we prove that the optimum occurs when its subtable sizes decrease in a geometric way, thus formally confirming a heuristic rule-of-thumb.**

I. Introduction

A. Background

High-speed networks with fast hash-based per-packet decisions call for fast hashing schemes. For example, hash tables are being used for heavy-hitter flow identification, flow state keeping, virus signature scanning, flow counter management, and IP address lookup algorithms.

Traditional hash-table construction schemes rely on either chaining (linked lists) or open addressing (probing) [1]. However, in the case of *hash collisions*, the worst-case insertion time in these schemes cannot be bounded by a constant, making them poorly suited to high-speed networks [2]. Further, when insertion times become unacceptable, the traditional solution of performing a full hash-table rehash, where all elements are rehashed using new hash functions, is also impractical at high speeds.

A typical solution is to restrict the data structure such that the *worst-case* number of memory accesses per element insertion is a constant d. If an element cannot be inserted after d accesses, it is placed in an expensive Content Addressable Memory (CAM) based *overflow list* [3], [4]. The objective of an hashing scheme then becomes to reduce the *overflow fraction*, namely, the fraction of elements that are placed in the overflow list.

Multiple-choice hashing schemes are particularly suited to this worst-case insertion time of d [5], [6]. In these schemes, the hash table is subdivided into many buckets, and each element can only reside in one of d possible buckets. For instance, in the *d-random* and *d-left* schemes [7]–[9], each arriving element uses d hash functions to check the states of d buckets, and then joins the least-occupied one. If all buckets are full, it is placed in the overflow list. These schemes achieve low overflow fractions even when d does not grow with the number of elements to hash, for instance with $d = 4$.

However, these multiple-choice hashing schemes always require d memory accesses per element insertion, even when the hash table is nearly empty. Therefore, implementing them in an off-chip DRAM-based hash table means that for every incoming element to hash, a chip needs to access up to d memory lines in the off-chip DRAM instead of possibly a single one. Assuming for simplicity a uniform memory access time, this means that the links to the DRAM need to work with a speedup of d; or, equivalently, given a total chip in/out pin capacity, this implies that a larger fraction of this capacity is devoted to hashing, yielding a significant throughput decrease as d increases. Therefore, the large *average* number of memory accesses into off-chip DRAM memories translates into a *lower throughput*.

Furthermore, when implementing both on-chip SRAM-based and off-chip DRAM-based hash tables, each memory access uses some power. Neglecting static power consumption and assuming a uniform dynamic power consumption per memory access, a larger *average* number of memory accesses also directly translates into a *higher power consumption*.

Therefore, the average number of memory accesses can directly influence both the throughput and the power consumption. Hence, while we still want to *keep a worst-case bound* d on the number of memory accesses per element, we also want to *significantly reduce the average number* a below the worst-case d used by the above schemes.

Further, while we reduce throughput and power consumption, we do not want to affect *performance*, as measured by the overflow fraction. Therefore, given some allowed worst-case d and average a, the objective of this paper is to find *hashing schemes that minimize the overflow fraction*.

B. Contributions

This paper investigates hashing schemes with low expected overflow fraction given the worst-case and average insertion times d and a.

We consider stateless hashing schemes, in which the only way to know a bucket occupancy is to access it, and therefore allow for a distributed hashing scheme implementation. We do not consider multi-level memory [10], [11] and element deletions [2]. Finally, our results apply asymptotically to hashing schemes with a large number of elements and a large memory size.

We first establish a lower bound on the expected overflow fraction as a function of a. The lower bound also depends on

This work was partly supported by the European Research Council Starting Grant n°210389.

Y. Kanizo is with the Dept. of Computer Science, Technion, Haifa, Israel. Email: ykanizo@cs.technion.ac.il.

D. Hay is with the Dept. of Electronics, Politecnico di Torino, Turin, Italy. Email: hay@tlc.polito.it.

I. Keslassy is with the Dept. of Electrical Engineering, Technion, Haifa, Israel. Email: isaac@ee.technion.ac.il.

other system parameters such as the bucket size and the *load* on the hash table (that is, the ratio between the number of elements and the total memory size).

The lower bound enables us to prove the optimality of the schemes we propose. We provide three hashing schemes and show that each of them is optimal for a specific range of values of a.

Specifically, we first demonstrate that a SIMPLE hashing scheme that relies on a single uniformly-distributed hash function achieves an optimal overflow fraction for $a \leq 1$.

We further show that a multiple-choice GREEDY scheme with d uniformly-distributed hash functions, in which each element successively checks up to d buckets until it can be inserted, achieves optimality for $a \leq a^{co}_{\text{GREEDY}}$, where $a^{co}_{\text{GREEDY}} > 1$ depends on the system parameters.

The optimality range can be further extended using a multi-level hashing (MHT) scheme [3], [11], [12]. In particular, among all MHT schemes, we demonstrate the optimality of those in which the subtable sizes decrease geometrically according to a factor that depends on system parameters, thus confirming a previously-known rule-of-thumb.

Further, while we obtain the optimal expected overflow fraction for a specific value a, we can equivalently find the optimal a for a given expected overflow fraction, and potentially a corresponding optimal scheme. Thus, this paper provides an *optimal fast hashing scheme* given a targeted overflow fraction.

We conclude by providing simulations and quantitative results showing that our models closely reflect simulation results.

C. Related Work

A crucial problem in hash tables is how to avoid collisions between elements. Traditional strategies include *open addressing* and *chaining* [1]. These strategies store all colliding elements in another bucket, resulting in relatively difficult algorithms for insertion, deletion, and lookups that do not ensure a constant run-time in the worst case.

As we already mentioned, a popular approach to alleviate this problem is to use multiple hash functions, with various ways in which they can be used. In [7] the authors considered the *d-random scheme*: Each element is hashed by d hash functions, where the element is placed in the least occupied bucket. If n elements are inserted sequentially to n buckets, the maximum load in a bucket is $\frac{\log\log n}{\log d} + O(1)$ with high probability, as opposed to the case where just one hash table is used, in which the maximum load in a bucket is $\log n\,(1 + O(1))$. In [8], [9] the authors considered and analyzed the *d-left scheme*, where initially the n buckets are divided into d groups of size $\frac{n}{d}$ which are ordered from left to right. Again, each element is hashed by all d hash functions. But this time, every hash function has the range of $[1 \cdots \frac{n}{d}]$, where the first hash function determines a bucket on the first (left-most) group, the second hash function determines a bucket on the second group, and so on. The element is placed in the least occupied bucket, where ties are broken toward the left. This scheme was shown to be better than the d-random scheme. However, both d-random and d-left schemes access the memory exactly d times for every element. We argue that such a usage in memory may be unjustified.

*Multi-level hash tables (*MHT*)* were introduced in [12] and analyzed in [3], [11], [12]. The basic idea is to divide the hash table into d ordered subtables, such that one hash function is used for every subtable, and the element is placed in the first subtable whose hash function returns a non-full bucket. The number of the subtables and the exact partitioning of the memory among the subtables depend on the exact architecture. Specifically, [12] considered the case where n elements should be stored in $O(n)$ buckets of size 1; they show that in order to avoid overflows, the number of hash functions (and subtables) must be at least $\log\log n + O(1)$ and the expected number of rehashing needed is constant. In [11], the authors considered the case where no rehashing is allowed and used the multilevel hash table as the underlying architecture for an on-chip *summary* mechanism. Finally, in [4], the authors focused on the case where $d = 4$, while the scheme proposed allows at most one move per element after it was stored.

Another scheme, called *cuckoo hashing* [13] (originally presented for $d = 2$ and buckets of size 1), takes advantage of the fact that every element has d possible buckets. Whenever an insertion operation fails because all buckets are occupied, the new element switches with an older element, which is rehashed and may switch in turn with another element, and so on. This process continues until all elements are positioned and may require $\Omega(\log n)$ moves with non-negligible probability. Our proposed schemes differ from this scheme as they don't move elements after being hashed, so that they would comply with hardware considerations.

Recently, Kirch et al. [5] survey these schemes and additional hashing-based techniques (such as Bloom Filters) and how they are used for high-speed packet classification.

D. Paper Organization

We start with preliminary definitions in Section II. Section III provides a lower bound on the overflow fraction. Then, in Sections IV, V, and VI, we present and analyze the SIMPLE, GREEDY, and MHT schemes respectively, which we finally evaluate in Section VII.

II. Problem Statement

We adopt the conventional hashing terminology and notations [4], [12]. As illustrated in Fig. 1, we are given a set $\mathcal{E}$ of n elements to insert in an initially-empty *hash table*. The hash table consists of a set $\mathcal{B}$ of m buckets of size h each and of an *overflow list*. Our goal is to find a hashing scheme to insert the elements.

Definition 1: A *hashing scheme*, or hash-table construction scheme, consists in defining:
(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) and an *insertion algorithm* that sequentially inserts the n elements in the hash table. The insertion algorithm places each element $x \in \mathcal{E}$ either in one of the d buckets

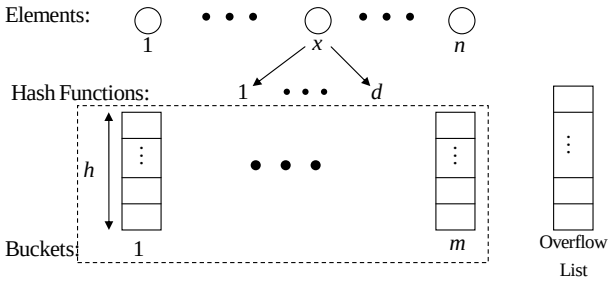


Fig. 1. Illustration of the hashing model.

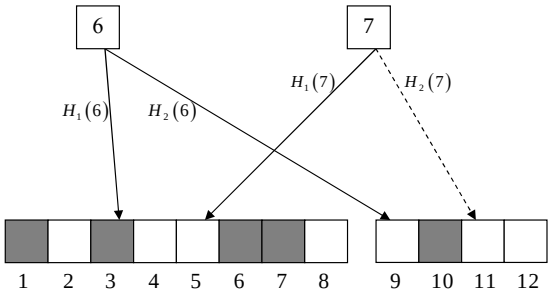


Fig. 2. Illustration of MHT scheme.

$\{H_1(x), \ldots, H_d(x)\}$ or in the overflow list. At most h elements can be placed in each bucket.

Note that our sequential insertion framework does not allow schemes that move elements after their initial insertion, such as [4], [13]. However, our overflow lower bound in Section III does apply to these as well.

*Example 1 (*MHT*):* A *multi-level hash table (*MHT*)* [3], [11], [12] construction scheme conceptually divides the m buckets into d separate subtables, $T_1, \ldots, T_d$, where T_i contains $\alpha_i \cdot m$ buckets, with $\sum \alpha_i = 1$.
(i) MHT generates each hash function H_i according to the uniform distribution over the buckets in T_i;
(ii) and the insertion algorithm successively places each element x in the smallest i such that $H_i(x)$ is not full, and in the overflow list if all such buckets are full.
Fig. 2 illustrates MHT with $m = 12$, $h = 1$ and $d = 2$ (the overflow list is not represented). The gray buckets are the occupied ones. Dashed arrows represent potential memory accesses that are not performed (and exist only to illustrate the mapping of the elements). We can see that element 6 is initially mapped by H_1 to a full bucket in the first subtable, and therefore is inserted in the second subtable, where it is mapped into an empty bucket by H_2. On the contrary, element 7 is directly inserted in the first subtable, and does not use H_2. Therefore, it only uses one memory access.

The throughput of a hashing scheme is measured by the number of *memory accesses* needed to store the incoming elements in the hash table. We define a *memory access time* as the time needed to access a single bucket, read all of its elements, and update them. This definition corresponds for instance to an update (operation) of one word in SRAM or DRAM memory. We assume that a hashing scheme needs to access a bucket in order to obtain any information on it; thus, if the hashing scheme tries to insert an element in a full bucket, it wastes a memory access. We also do not count accesses to the overflow list. We finally allow the hashing scheme to access up to d buckets in parallel at each element insertion before deciding which one to update. Thus, we get the following number of memory accesses for previously known schemes:

Example 2 (d-random and d-left): Inserting an element in the least loaded of d buckets requires d memory accesses [7]–[9].

*Example 3 (*MHT*):* Inserting an element in subtable T_i requires i memory accesses, since in that case we first sequentially access $i-1$ full buckets in subtables $T_1, \ldots, T_{i-1}$, and then access the last non-full bucket in subtable T_i.

We further consider two throughput constraints. First, we impose that the *average* number of memory accesses per element insertion be bounded by some constant $a \geq 0$. In addition, the *worst-case* number of memory 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. Let the load $c = \frac{n}{mh}$ denote the ratio of the number of elements to insert by the total memory size. Then we can formalize these two constraints:

Definition 2: An $\langle a, d, c, h \rangle$ hashing scheme is a hashing scheme that inserts all elements with an average (resp. maximum) number of memory accesses per insertion of at most a (resp. d), when given a load c and a bucket size h.

Let γ denote the *expected overflow fraction* of the elements, i.e. the expected ratio of the number of elements that cannot be stored in the buckets by the total number of elements n. These unstored elements are placed in the overflow list, which usually is more expensive than the memory buckets (e.g., when implemented in a CAM [4]) or requires more memory accesses (e.g., as a linked list). Our goal is to minimize γ:

Definition 3: The OPTIMAL HASH TABLE CONSTRUCTION PROBLEM is to find an $\langle a, d, c, h \rangle$ hashing scheme that minimizes γ as the number of elements n goes to infinity. Whenever defined, let γ_{OPT} denote this optimal expected limit overflow fraction.

The definitions above do not only bound the total number of memory accesses per element insertion but also per element *lookup*. First, since each element can only be placed in one of d buckets, the number of memory accesses needed for a lookup is bounded by d. Second, in most hashing schemes, the lookup operation accesses buckets in the same order as the insertion operation. Therefore, the average number of memory accesses to query a random element in the hash table is also a. So, given a probability p that a queried element is in the hash table, the average number of memory accesses needed for a lookup is bounded by $p \cdot a + (1-p) \cdot d$. Therefore, in most hashing schemes, the bounds on insertion memory accesses directly translate into bounds on lookup memory accesses.

III. Overflow Lower Bound

A. The Cloning Method

In this section, we provide a lower bound on γ_{OPT}, and therefore on the expected limit overflow fraction of *any* $\langle a, d, c, h \rangle$ hashing scheme. We do so by relaxing three conditions.

First, we consider an *offline* case, in which the hashing scheme looks at all elements at once, instead of considering them in the predetermined online sequential order.

Second, we *omit the bound* d on the worst-case number of memory accesses per element, and enable all elements to use *any* number of memory accesses, as long as the average number of memory accesses per element is still at most a.

Last, we *dissociate the memory accesses from the elements*. In other words, we hypothetically consider each memory access to a bucket as if it is made by a *clone* of the initial element and allow the clone to be inserted if the bucket is not full, independently of the other clones. Thus, if one element accesses two buckets, it conceptually corresponds to *two clones* each accessing one of these buckets, potentially corresponding to *two* clone insertions. The number of inserted clones after this dissociation is clearly an upper bound on the actual number of inserted elements. In our case, since n elements make at most a memory accesses per element on average, we will consider a set of at most an clones making one memory access each and evaluate the number of such clones that are inserted.

Conceptually, the cloning relaxation is the most significant one. While it seems to provide a crude bound, we will later see that this bound is actually tight over a range of values of a.

Note that our lower bound also holds for schemes that allow moves, such as cuckoo hashing [13] and one-move schemes [4], since we are assuming an offline non-sequential setting.

B. Identical Hash Function Distributions

Different elements might end up using different hash functions, and therefore generate memory accesses by their clones that are distributed in a different way. We first consider the easier case in which all hash functions have the same distribution, implying that all memory accesses by the clones are distributed in the same way. Then, we later consider the heterogeneous case, in which the hash functions do not necessarily have the same distribution. In both cases, we eventually derive the same lower bound on the expected limit overflow fraction.

We start with the setting in which all hash functions are distributed identically, but the common distribution is not necessarily uniform. In this setting, the following theorem provides a lower bound on γ_{OPT}.

To prove it, we bound the expected fraction of unused memory after the insertion of all elements. We approximate the binomial distribution of the load on each bucket by a Poisson distribution, and supply a bound on the approximation error. Then, we prove the theorem on the Poisson distribution, and apply the bound to conclude.

Theorem 1: Under the constraint that all hash functions are distributed identically, the optimal expected limit overflow fraction γ_{OPT} in the OPTIMAL HASH TABLE CONSTRUCTION PROBLEM is lower bounded by

$$\gamma_{\text{LB}}(a) = 1 - \frac{1}{c} + \frac{1}{ch} e^{-ach} \sum_{k=0}^{h} (h-k) \frac{(ach)^k}{k!},$$

and this lower bound is computed with the uniform distribution.

Proof: We derive the lower bound on the overflow fraction by bounding the expected fraction of unused memory after the insertion of all elements. Let $f : \mathcal{E} \to [0,1]$ be the distribution used by the hash functions, where $f(i)$ is the probability that a hash function maps any element $x \in \mathcal{E}$ to a bucket $i \in \mathcal{B}$, and $\sum f(i) = 1$. The number of elements mapped to the specific bucket number i follows a binomial distribution with $a \cdot n$ trials of success probability $f(i)$ each. Furthermore, as assumed above, we consider the case where m and n are large and we can approximate it by a Poisson distribution of parameter $\lambda_i = a \cdot n \cdot f(i)$. Let Z be the random variable that represents the ratio of the unused memory size by the total memory size. We note that Z clearly depends f. Then, By the linearity of expectation,

$$E(Z) = \frac{1}{mh} \sum_{i=1}^{m} \sum_{k=0}^{h} (h-k) \binom{an}{k} f(i)^k (1-f(i))^{a \cdot n - k}.$$

Using the Poisson approximation,

$$E(Z) \approx \frac{1}{mh} \sum_{i=1}^{m} \sum_{k=0}^{h} (h-k) \frac{(anf(i))^k e^{-a \cdot nf(i)}}{k!}, \quad (1)$$

and we denote it by $E_n(Z)$.

Barbour and Hall Theorem [14] yields that the total variation distance between the Binomial distribution and the Poisson approximation representing the number of elements mapped to bucket i, is bounded by $\lambda_i^{-1} \cdot \left(1 - e^{-\lambda_i}\right) \sum_{j=0}^{an} \left(f(i)\right)^2 \leq f(i)$.

By [15], the total variation distance is equivalent to the following expression:

$$\frac{1}{2} \sum_{j=0}^{\infty} |\Pr(X = j) - \Pr(Y = j)| = \frac{1}{2} \sum_{j=0}^{\infty} \delta_{X,Y}(j),$$

where Y is the approximated random variable distributed Poisson, and X is the actual random variable distributed Binomial.

Thus, we find a bound on the error:

$$\begin{aligned} |E(Z) - E_n(Z)| &\leq \frac{1}{mh} \sum_{i=1}^{m} \sum_{k=0}^{h} (h-k) \cdot \delta_{X,Y}(j) \\ &\leq \frac{1}{mh} \sum_{i=1}^{m} h \sum_{k=0}^{h} \delta_{X,Y}(j) \\ &\leq \frac{1}{mh} \sum_{i=1}^{m} h \cdot 2f(i) \\ &= \frac{2}{m} \end{aligned} \quad (2)$$

this implies that the error is bounded by a constant, and therefore we are able to seek for the extremal points of $E_n(Z)$ and bound $E(Z)$.

By substituting $f(m) = 1 - \sum_{i=1}^{m-1} f(i)$ in the expression for $E_n(Z)$, we get:

$$E_n(Z) = \frac{1}{mh}\sum_{i=1}^{m-1}\sum_{k=0}^{h}(h-k)\frac{(anf(i))^k e^{-a\cdot nf(i)}}{k!} + \frac{1}{mh}\sum_{k=0}^{h}(h-k)\cdot\frac{\left(an(1-\sum_{i=1}^{m-1}f(i))\right)^k e^{-a\cdot n(1-\sum_{i=1}^{m-1}f(i))}}{k!}$$

We further denote the inner sum of the first term for each i as $g_i(f)$ and the sum in the second term as $g_m(f)$, so that $E_n(Z) = \frac{1}{mh}\sum_{i=1}^{m-1} g_i(f) + \frac{1}{mh} g_m(f)$. Note that formally $g_i(f)$ is a function with $m-1$ variables: $f(1), f(2), ..., f(m-1)$, although in practice it only depends on $f(i)$ (unless $i = m$). Thus, in order to find a global minimum we will apply a second derivative test using the corresponding *Hessian Matrix*.

We first deal with the Hessian matrices of $g_i(f)$ with $i \neq m$. Since $g_i(f)$ only depends on $f(i)$, the Hessian matrix $H_{g_i(f)} \equiv [h^i_{jk}] = \left[\frac{\partial^2 g_m(f)}{\partial f(j)\partial f(k)}\right]$ of $g_i(f)$ has only a single non-zero element:

$$h^i_{ii} = \frac{\partial^2 g_i(f)}{\partial f(i)^2} = (a\cdot n)^2\frac{(a\cdot n\cdot f(i))^{h-1}e^{-a\cdot n\cdot f(i)}}{(h-1)!}.$$

On the other hand, the Hessian matrix $H_{g_m(f)}$ of $g_m(f)$ (which depends on $f(1),\ldots,f(m-1)$) has the same elements in all its entries and their value is

$$h^m_{jk} = \frac{\partial^2 g_m(f)}{\partial f(j)\partial f(k)} = (a\cdot n)^2\frac{(a\cdot n\cdot f(m))^{h-1}e^{-a\cdot n\cdot f(m)}}{(h-1)!},$$

where $f(m) = 1 - \sum_{i=1}^{m-1} f(i)$.

It is easy to verify that all eigenvalues of these Hessian matrices $H_{g_i(f)}$ are non-negative, implying that all the functions $g_i(f)$ are convex. Thus, $E(Z)$ is also a convex function, as a sum of convex functions. Finally, in order to find an external point we differentiate the expression:

$$\frac{\partial E(Z)}{\partial f(i)} = -a\cdot\frac{n}{mh}\cdot e^{-a\cdot nf(i)}\cdot\sum_{k=0}^{h-1}\frac{(anf(i))^k}{k!} + a\cdot\frac{n}{mh}\cdot e^{-a\cdot n(1-\sum_{i=1}^{m-1}f(i))}\cdot\sum_{k=0}^{h-1}\frac{\left(an(1-\sum_{i=1}^{m-1}f(i))\right)^k}{k!}$$

Comparing this expression to 0 yields a system of $m-1$ equations with $m-1$ variables; one of its solutions f_u corresponds to the uniform distribution, in which $f(i) = \frac{1}{m}$ for all $i \in \{1,\cdots,m\}$. By applying the *second derivative test* on the Hessian matrix $H_{E(Z)} = \sum_{i=1}^{m} H_{g_i(f)}$, we verify that it is positive definite at f_u and thus f_u is a local minimum of $E(Z)$. By convexity, we deduce that it is in fact a global minimum over all the region.

Thus, the minimum expectation of unused memory is $E_{\min}(Z) = e^{-a\cdot\frac{n}{m}}\frac{1}{h}\sum_{k=0}^{h}(h-k)\frac{\left(a\frac{n}{m}\right)^k}{k!}$, implying that at least $n - (mh - mhE_{\min}(Z))$ elements are overflowed. Since we are interested in the limit expectation overflow fraction, i.e. $n, m \to \infty$, we get that the error bound in Equation 2 tends to zero, and therefore the result expression is tight. Finally, the claimed expression of the minimum overflow fraction of elements by substituting $c = \frac{n}{mh}$.

■

Under the assumptions above, we derive the following example:

Example 4: If $h = 1$,

$$\gamma_{\text{LB}}(a) = 1 - \frac{1}{c} + \frac{1}{c}e^{-ac}.$$

Thus, for $c = 1$, i.e. $n = m$, we get

$$\gamma_{\text{LB}}(a) = e^{-a},$$

and the lower-bound decreases exponentially as a function of the average number of memory accesses per insertion a. Therefore, for any constant a, an $\langle a, d, c, h\rangle = \langle a, d, 1, 1\rangle$ hashing scheme can never reach a zero-overflow result.

C. Multiple Hash Function Distributions

We now 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 memory accesses, with $\sum_{i=1}^{\ell} k_i = 1$. We now show that Theorem 1 holds also in this case.

Theorem 2: The optimal expected limit overflow fraction γ_{OPT} is lower bounded by

$$\gamma_{\text{LB}}(a) = 1 - \frac{1}{c} + \frac{1}{ch}e^{-ach}\sum_{k=0}^{h}(h-k)\frac{(ach)^k}{k!},$$

and is reached when for each bucket $i \in \{1\cdots m\}$, $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$, namely, the weighted average of all distributions is uniform.

Proof: As in the proof of Theorem 1, the number of elements mapped to bucket i by all hash functions follows approximately a Poisson distribution with rate $\lambda_i = an\sum_{p=1}^{\ell} k_p f_p(i)$. Let Y be the random variable that follows this Poisson distribution, and X be the actual random variable. Then, from [14], [15], we get:

$$\begin{aligned}
\sum_{j=0}^{\infty}\delta_{X,Y}(j) &\leq \lambda_i^{-1}\cdot\left(1-e^{-\lambda_i}\right)\sum_{p=1}^{\ell}\sum_{j=1}^{an\cdot k_p}(f_p(i))^2 \\
&\leq \lambda_i^{-1}\sum_{p=1}^{\ell} an\cdot k_p(f_p(i))^2 \\
&= \frac{\sum_{p=1}^{\ell} k_p(f_p(i))^2}{\sum_{p=1}^{\ell} k_p f_p(i)} \\
&\leq \frac{\max_{p\in\{1,\cdots,\ell\}}\{f_p(i)\}\sum_{p=1}^{\ell} k_p f_p(i)}{\sum_{p=1}^{\ell} k_p f_p(i)} \\
&= \max_{p\in\{1,\cdots,\ell\}}\{f_p(i)\}
\end{aligned}$$

Therefore,

$$\begin{aligned}
|E(Z) - E_n(Z)| &\leq \frac{1}{mh}\sum_{i=1}^{m}\sum_{k=0}^{h}(h-k)\cdot\delta_{X,Y}(j) \\
&\leq \frac{1}{mh}\sum_{i=1}^{m} h\sum_{k=0}^{h}\delta_{X,Y}(j) \\
&\leq \frac{2}{m}\sum_{i=1}^{m}\max_{p\in\{1,\cdots,\ell\}}\{f_p(i)\} \\
&= \frac{2\cdot\ell}{m}
\end{aligned}$$

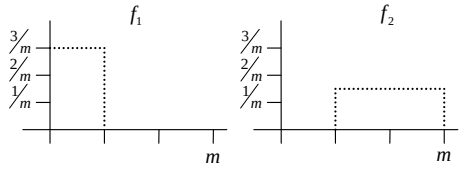


Fig. 3. An example of two different distributions (probability mass functions) f_1 and f_2 that reach the lower bound with $k_1 = \frac{1}{3}$ and $k_2 = \frac{2}{3}$.

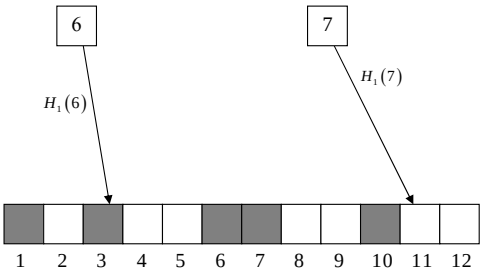


Fig. 4. Illustration of SIMPLE scheme.

Thus, the proof of Theorem 1 implies that to get global minimum over the overflow fraction, for every bucket i, $an\sum_{p=1}^{\ell} k_p f_p(i)$ must equal $\frac{an}{m}$. This implies that

$$\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$$

is a necessary condition for optimal overflow fraction.

Further, since any legal combination of k_p and f_p can be considered as a single distribution, Theorem 1 implies that a lower value of the overflow fraction cannot be found if the above condition is violated, and thus is also a sufficient condition. ■

Note that while any offline algorithm may pick its own values explicitly, we would typically like to have an online hashing scheme in which the values of k_p are picked *implicitly* so that $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$. In Fig. 3, there is an example of two distributions that under $k_1 = \frac{1}{3}$ and $k_2 = \frac{2}{3}$ reach the lower bound:

$$f_1(i) = \begin{cases} \frac{3}{m} & i \leq \frac{m}{3} \\ 0 & \text{otherwise} \end{cases}$$

and

$$f_2(i) = \begin{cases} \frac{3}{2m} & i > \frac{m}{3} \\ 0 & \text{otherwise} \end{cases}$$

IV. SIMPLE - A SINGLE-CHOICE HASHING SCHEME

We now want to find simple hashing schemes that can potentially achieve the overflow fraction lower bound γ_{LB}, and therefore the optimal overflow fraction γ_{OPT}.

We start by analyzing a simplistic hashing scheme, denoted SIMPLE. This scheme only uses a single uniformly-distributed hash function H. Each element is stored in bucket $H(x)$ if it is not full, and in the overflow list otherwise.

Furthermore, to keep an average number of memory accesses per element of at most a, not all elements can be inserted when $a < 1$. Therefore, in that case, the process stops when a total of $a \cdot n$ memory accesses is reached, and the remaining elements are placed in the overflow list as well.

Fig. 4 illustrates SIMPLE with $m = 12$ and $h = 1$ (the overflow list is not represented). We can see that element 6 is mapped by H to a full bucket, and therefore cannot be inserted. Thus, it joins the overflow list. On the contrary, element 7 is directly inserted in an empty bucket.

Following our notations in Definition 2, SIMPLE is an $\langle a, 1, c, h\rangle$ hashing scheme; we will show that it is optimal for $a \leq 1$.

A. Description by Differential Equations

In recent years, several hashing schemes have been modeled using a deterministic system of differential equations [4], [6]. We adapt this approach in order to describe the SIMPLE scheme.

We start by considering that the j-th element is inserted in the hash table at time $\frac{j}{n}$; namely, all elements are handled by time $t = 1$. 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

$$\mathbf{E}\left(\Delta F_i\left(\frac{j+1}{n}\right) | \vec{F}\left(\frac{j}{n}\right)\right) = \begin{cases} -\frac{1}{m} F_0\left(\frac{j}{n}\right) & i = 0 \\ \frac{1}{m} F_{h-1}\left(\frac{j}{n}\right) & i = h \\ \frac{1}{m}\left(F_{i-1}\left(\frac{j}{n}\right) - F_i\left(\frac{j}{n}\right)\right) & \text{otherwise} \end{cases} \quad (3)$$

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

The first equality shows that the fraction of empty buckets only decreases when element j reaches an empty bucket, which happens with probability $F_0\left(\frac{j}{n}\right)$. Likewise, in the second equality, the fraction of full buckets only increases when element j hits a bucket of size $h-1$. Last, in the third equality, the fraction of elements of size i either increases with probability $F_{i-1}\left(\frac{j}{n}\right)$, or decreases with probability $F_i\left(\frac{j}{n}\right)$. Any such increment or decrement is by a value of $\frac{1}{m}$.

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 [4]:

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

More formally, let $\vec{f}(t) \triangleq (f_1(t), \ldots, f_d(t))$ 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 [16]–[18], 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 [4]. For further intuition behind this statement, refer to [4] and [19, Chapter 3.4].

B. Optimality of the SIMPLE Scheme

We solve analytically the system of differential equations to obtain the overflow fraction of the scheme and show that it is *identical* to the lower bound given in Theorem 1. Since SIMPLE does not perform more than one memory access per operation, this yields the following theorem.

Theorem 3: The SIMPLE scheme solves the OPTIMAL HASH TABLE CONSTRUCTION PROBLEM for $a \leq 1$, $d = 1$, and any values of c and h.

Proof: We solve the differential equations one by one, substituting the result of equation i into equation $i+1$. The first equation depends only on $f_0(t)$, thus $f_0 = e^{-\frac{n}{m}t}$. Each other equation i depends on $f_{i-1}(t)$ and $f_i(t)$. Finally, for $f_h(t)$, we use the fact that $\sum_{i=0}^{h} f_i = 1$ and substitute all the previous solutions. The resulting values are

$$f_i(t) = \begin{cases} \frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} & i < h \\ 1 - \sum_{k=0}^{h-1} \frac{1}{k!}\left(\frac{n}{m}t\right)^k e^{-\frac{n}{m}t} & i = h \end{cases} \quad (4)$$

Note that the solution is the Poisson distribution with $\lambda = \frac{n}{m}t$. This is no surprise, due to fact that at a given time t, the total number of mapped elements into a specific bucket is distributed according to $\text{Bin}\left(nt, \frac{1}{m}\right)$, thus the corresponding limit distribution is $\text{Poisson}\left(\lambda = \frac{n}{m}t\right)$.

We define the *overflow fraction at time* t as the fraction of all n elements that has not been inserted into the buckets by time t and denote it $\gamma_{\text{SIMPLE}}(t)$. Thus, $\gamma_{\text{SIMPLE}}(t=0) = 1$, since at the start no elements have been inserted yet. Then, the $\gamma_{\text{SIMPLE}}(t)$ function is decreasing as more elements are inserted, until it reaches the final overflow fraction $\gamma_{\text{SIMPLE}} = \gamma_{\text{SIMPLE}}(t=1)$. Using the solutions above, right before the j-th elements is hashed, the overflow fraction at time $t = \frac{j}{n}$ is

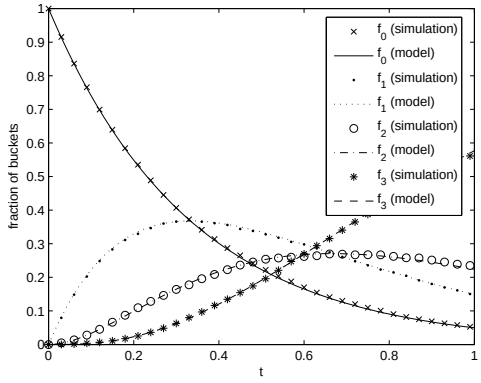
$$\begin{aligned} \gamma_{\text{SIMPLE}}(t) &= 1 - \frac{m}{n}\sum_{i=0}^{h-1} i \cdot \frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} \\ &\quad -\frac{m}{n}\cdot h \cdot \left(1 - \sum_{k=0}^{h-1}\frac{1}{k!}\left(\frac{n}{m}t\right)^k e^{-\frac{n}{m}t}\right) \\ &= 1 - t + \frac{m}{n}\sum_{i=h+1}^{\infty}(i-h)\cdot\frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} \quad (5) \end{aligned}$$

Alternatively, one can also consider the *cumulative overflow fraction* at time t; namely, considering only the $n \cdot t$ elements that are handled by this time (and normalizing according to $n \cdot t$ and not n). This cumulative overflow fraction is:

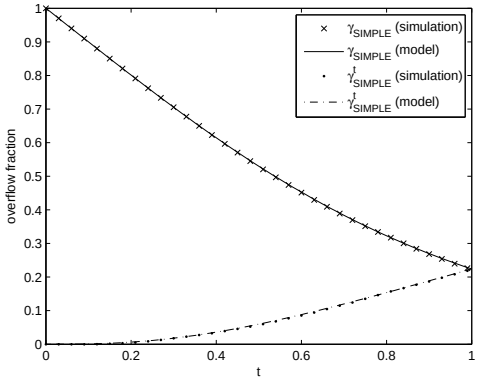
$$\begin{aligned} \gamma^t_{\text{SIMPLE}}(t) &= 1 - \frac{m}{nt}\sum_{i=0}^{h-1} i \cdot \frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} \\ &\quad -\frac{m}{nt}\cdot h \cdot \left(1 - \sum_{k=0}^{h-1}\frac{1}{k!}\left(\frac{n}{m}t\right)^k e^{-\frac{n}{m}t}\right) \quad (6) \end{aligned}$$

At time $t = 1$, we get $\gamma_{\text{SIMPLE}} = \gamma_{\text{SIMPLE}}(1) = \gamma^t_{\text{SIMPLE}}(1)$, which is the overflow fraction of the scheme.

The equations above are only true as long as the average number of memory accesses per element is at most a. Since this average number equals t at time t, the process is stopped at $t = a$. Then, the optimality of the scheme is obtained by substituting $c = \frac{n}{mh}$ and $t = a$ in Equation (5) and comparing it to $\gamma_{\text{LB}}(a)$ of Theorem 1. ■



(a) Fraction of buckets that store $i \in \{0, \ldots, 3\}$ elements.



(b) Overflow fraction.

Fig. 5. Model and simulation results for the SIMPLE scheme given load $c = 1$, bucket size $h = 3$, and memory size $m = 10,000$.

C. Case Study - A SIMPLE $\langle 1, 1, c, 1\rangle$ Hashing Scheme

In the SIMPLE scheme, each element is hashed exactly once, therefore, both d, the maximum number of memory accesses, and a, the average number of memory accesses, equal to 1. We want to consider the case where the bucket size is also 1, and derive the expressions for $f_0(t)$:

$$f_0(t) = e^{-\frac{n}{m}t} = e^{-c\cdot t},$$

and $f_1(t) = 1 - f_0(t)$.

Next, the overflow fraction $\gamma_{\text{SIMPLE}}(t)$ is given by:

$$\gamma_{\text{SIMPLE}}(t) = 1 - \frac{m}{n}\cdot\left(1 - e^{-\frac{n}{m}t}\right).$$

Therefore, for $a = 1$, it meets the lower bound at $t = 1$, using $c = \frac{n}{m}$:

$$\gamma_{\text{SIMPLE}}(t=1) = \gamma_{\text{LB}}(a=1) = 1 - \frac{1}{c}\cdot\left(1 - e^{-c}\right).$$

And if $c = 1$, we get:

$$\gamma_{\text{SIMPLE}}(t=1) = e^{-1} = 36.8\%.$$

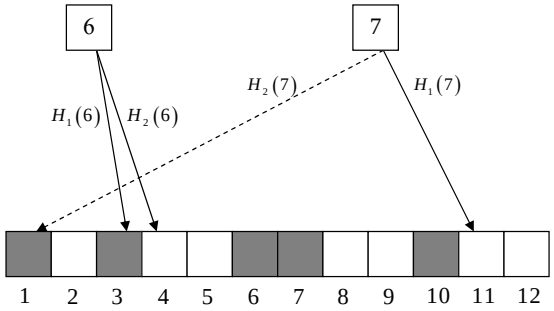


Fig. 6. Illustration of GREEDY scheme.

D. Simulation Results

We now compare the analytical results of the SIMPLE scheme with simulation results. Since the SIMPLE scheme uses a hash function with uniform distribution, we simulated it by successively choosing a bucket for each element uniformly at random. We used a load $c = 1$, a bucket size $h = 3$, and $m = 10,000$ buckets.

Fig. 5(a) shows how the bucket occupancies evolve over time. All buckets are empty at the beginning, while at the end, 57.68% of the buckets are full, i.e. hold three elements, 22.40% hold two elements, 14.94% hold a single element and 4.98% of the buckets are empty. For all functions, our fluid model appears to closely match simulations.

Fig. 5(b) shows how the overflow fraction and the cumulative overflow fraction evolve over time. As previously explained, the overflow fraction is a monotonically-decreasing function that starts at 1 while the cumulative overflow fraction is a monotonically-increasing function that starts at 0. Both functions get the same value at the end. Here again, for both functions, our fluid model appears to closely match simulations.

V. GREEDY - A MULTIPLE-CHOICE HASHING SCHEME

In the GREEDY scheme, we use an ordered set of d hash functions $\mathcal{H} = \{H_1, \ldots, H_d\}$, such that all the hash 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 non-full bucket. If all these buckets are full, x is placed in the overflow list. Last, as in SIMPLE, to keep an average number of memory accesses per element of at most a, the process stops when a total of $a \cdot n$ memory accesses is reached, and the remaining elements are placed in the overflow list as well.

Fig. 6 illustrates GREEDY with $m = 12$, $h = 1$ and $d = 2$. We can see that element 6 is initially mapped by H_1 to a full bucket. It is therefore mapped again by H_2, and inserted in an empty bucket. On the contrary, element 7 is directly inserted in an empty bucket, and therefore does not need a second memory access.

A. Description by Differential Equations

We model the dynamics of the GREEDY scheme as a system of differential equations, in which time is scaled according to element arrivals. As before, let $f_i(t)$ represent the fraction of buckets storing i elements at time t, then

$$\frac{df_i(t)}{dt} = \begin{cases} -\frac{n}{m} f_0(t)\, g(t) & i = 0 \\ \frac{n}{m} f_{h-1}(t)\, g(t) & i = h \\ \frac{n}{m}\left(f_{i-1}(t) - f_i(t)\right) g(t) & \text{otherwise} \end{cases} \quad (7)$$

where

$$g(t) = \sum_{k=0}^{d-1} f_h(t)^k = \frac{1 - f_h(t)^d}{1 - f_h(t)},$$

with $f_0(0) = 1$ and $f_i(0) = 0$ for each $i \neq 0$ as an initial condition. Compared to the differential equations of the SIMPLE scheme from Equation (3), there is an additional factor $g(t)$. For instance, in the first equation, $f_0(t)$ is replaced by $f_0(t)\, g(t) = \sum_{k=0}^{d-1} \left[f_h(t)^k \cdot f_0(t) \right]$, which represents the sum of the probabilities of entering an empty bucket after $k = 0, 1, \ldots, d-1$ hits at full buckets.

The process stops when reaching a total of $a \cdot n$ memory accesses, thus we keep count of the total number of memory accesses. Let $f^a_{\text{GREEDY}}(t)$ denote the cumulative number of memory accesses done by time t, normalized by n. It can be modeled as

$$\frac{df^a_{\text{GREEDY}}(t)}{dt} = \sum_{k=1}^{d-1} k \cdot (f_h(t))^{k-1} (1 - f_h(t)) + d \cdot (f_h(t))^{d-1}, \quad (8)$$

with $f^a_{\text{GREEDY}}(0) = 0$ as an initial condition. We stop the process when either $t = 1$ or $f^a_{\text{GREEDY}}(t)$ reaches a. The differential equation reflects the fact that at a given time t, the cumulative number of memory accesses increases by $1 \leq k < d$ memory accesses whenever the first $k-1$ memory accesses hit full buckets and the next one hits a non-full bucket. It also increases by d memory accesses whenever the first $d-1$ memory accesses hit full buckets, independently of the bucket state in the d-th memory access.

B. Optimality of the GREEDY Scheme

We now want to show the optimality of the GREEDY scheme over a range of values of a. In general, the above differential equations are hard to solve analytically, and thus cannot help in showing optimality — even though they can of course be solved numerically and yield a numerical approximation of the expected overflow fraction.

Instead, to show the optimality of the GREEDY scheme, we reduce it to the optimality of the SIMPLE scheme using the *cloning* method. Since both the SIMPLE and GREEDY schemes use the same uniform distribution, a new attempt to insert an element after hitting a full bucket in the GREEDY scheme is equivalent to creating a new element (or clone) in the SIMPLE scheme and then trying to insert it. In other words, the number of clones successfully inserted by the GREEDY scheme after considering n elements and using a total of $a \cdot n$ memory accesses is the same as the number of elements successfully inserted by the SIMPLE scheme after considering $a \cdot n$ clones and using a single memory access per clone.

We next show that GREEDY is an *optimal* $\langle a, d, c, h \rangle$ hashing-scheme for $a \leq f^a_{\text{GREEDY}}(1)$ and any values of d, c

and h. We call the value $f^a_{\text{GREEDY}}(1)$ the *cut-off point* of the GREEDY scheme and denote it by a^{co}_{GREEDY}; beyond this average number of memory accesses per element, the GREEDY scheme is not necessarily optimal anymore. It is important to notice that the differential equations, described in Section V-A, are used only to obtain the optimality range (that is, to calculate the value of a^{co}_{GREEDY}).

Theorem 4: The GREEDY scheme solves the OPTIMAL HASH TABLE CONSTRUCTION PROBLEM for $a \leq a^{co}_{\text{GREEDY}}$ and any values of d, c and h, where $a^{co}_{\text{GREEDY}} = f^a_{\text{GREEDY}}(1)$.

Proof: We compare the GREEDY scheme with the SIMPLE scheme. In the GREEDY scheme, we continually try to insert each element, until either it is placed or all d function are used. Note that all hash functions have the same (uniform) distribution over all buckets. Thus, for every i, $f_i(t)$–and therefore also $\gamma_{\text{GREEDY}}(t)$–are independent of the exact elements that are hashed. Moreover, applying $d_1 \leq d$ hash function on the same element is equivalent to applying a single hash function on d_1 different elements. This implies that the results of the SIMPLE scheme hold also in this case with a different time scale: The insertion process continues beyond time 1, until time a, in which a total of $a \cdot n$ memory accesses are performed. Thus, the overflow fraction is obtained by substituting $t = a$ in Equation (6).

Note, however, that the total number of memory accesses after all elements are considered is given by $f^a_{\text{GREEDY}}(1)$. thus if a is larger than $f^a_{\text{GREEDY}}(1)$ it does not impose an effective restriction on the scheme; this, in turn, implies that by increasing a beyond $f^a_{\text{GREEDY}}(1)$ one cannot improve the scheme performance.

Hence, the GREEDY scheme solves the OPTIMAL HASH TABLE CONSTRUCTION PROBLEM for $a \in [1, a^{co}_{\text{GREEDY}}]$, because both expressions are identical and GREEDY does not perform more than a^{co}_{GREEDY} memory accesses per insertion. Therefore, the limit overflow fraction as a function of the average memory accesses is given by:

$$\gamma_{\text{GREEDY}}(a) = 1 - \frac{1}{c} + \frac{1}{ch}\sum_{i=0}^{h}(h-i)\cdot\frac{(a_e ch)^i}{i!}e^{-a_e ch},$$

where $a_e = \min\{a, f^a_{\text{GREEDY}}(1)\}$. ∎

C. Case Study - A GREEDY $\langle a, 2, c, 1\rangle$ Hashing-Scheme

Although in general it is difficult to obtain analytically the value of a^{co}_{GREEDY} for general values of h and d, in the following section we describe how it can be obtained when $h = 1$ and $d = 2$

In this specific case, $f_0(t) + f_1(t) = 1$ and therefore the first differential equation in (7) is $\frac{df_0(t)}{dt} = -\frac{n}{m}\left(2f_0(t) - (f_0(t))^2\right)$, where $f_0(0) = 1$ is the given initial condition. Since this is a Bernoulli differential equation, it can be solved analytically implying that

$$f_0(t) = \frac{2}{1+e^{2\frac{n}{m}t}}$$
$$f_1(t) = \frac{-1+e^{2\frac{n}{m}t}}{1+e^{2\frac{n}{m}t}}$$

and the overflow fraction is given by

$$\gamma_{\text{GREEDY}}(t) = 1 - \frac{m}{n}\frac{-1+e^{2\frac{n}{m}t}}{1+e^{2\frac{n}{m}t}} \tag{9}$$

Finally, in order to compute $f^a_{\text{GREEDY}}(t)$, we substitute $f_0(t)$ and $f_1(t)$ in Equation (8) and solve the resulting differential equation: $\frac{df^a_{\text{GREEDY}}(t)}{dt} = \frac{2e^{2\frac{n}{m}t}}{1+e^{2\frac{n}{m}t}}$. Integrating the right side and applying the initial condition implies:

$$f^a_{\text{GREEDY}}(t) = \frac{m}{n}\cdot\ln\left(\frac{1+e^{2\frac{n}{m}t}}{2}\right), \tag{10}$$

and by Theorem 4, the range of optimality in this specific case is $a \in [1, \frac{n}{m}\cdot\ln\left(\frac{1+e^{2\frac{n}{m}}}{2}\right)]$.

In particular, if $n = m$, the cut-off point is $a^{co}_{\text{GREEDY}} = \ln\left(\frac{e^2+1}{2}\right) \approx 1.4338$ and the corresponding overflow fraction is

$$\gamma_{\text{GREEDY}}(t=1) = \frac{2}{e^2+1} \approx 23.8\%.$$

Likewise, if $n = 0.1m$, the cut-off point is $a^{co}_{\text{GREEDY}} = 10\ln\left(\frac{e^{0.2}+1}{2}\right) \approx 1.0499$ and the corresponding overflow fraction is

$$\gamma_{\text{GREEDY}}(t=1) = 1 - 10\cdot\frac{e^{0.2}-1}{e^{0.2}+1} \approx 0.33\%.$$

Note that if a is within the range of optimality, by solving the equation $f^a_{\text{GREEDY}}(t) = a$, we can obtain the time in which the bound a is reached and no more memory accesses are allowed. In our case this time turns to be

$$t = \frac{1}{2}\cdot\frac{m}{n}\cdot\ln\left(2e^{\frac{m}{n}a} - 1\right). \tag{11}$$

Note that by substituting t in Equation (9), we get:

$$\begin{aligned}\gamma_{\text{GREEDY}}(a) &= 1 - \frac{m}{n}\frac{-1+e^{2\frac{n}{m}\left(\frac{1}{2}\cdot\frac{m}{n}\cdot\ln\left(2e^{\frac{m}{n}a}-1\right)\right)}}{1+e^{2\frac{n}{m}\left(\frac{1}{2}\cdot\frac{m}{n}\cdot\ln\left(2e^{\frac{m}{n}a}-1\right)\right)}} \\ &= 1 - \frac{m}{n}\frac{-1+\left(2e^{\frac{m}{n}a}-1\right)}{1+\left(2e^{\frac{m}{n}a}-1\right)} \\ &= 1 - \frac{m}{n} + \frac{m}{n}e^{-\frac{m}{n}a}\end{aligned} \tag{12}$$

which matches exactly the result in Theorem 4.

VI. THE MULTI-LEVEL HASH TABLE (MHT) SCHEME

In this section, we consider another hashing scheme, *the multi-level hash table* (MHT). We show that it is optimal beyond $a \leq a^{co}_{\text{GREEDY}}$, thus improving upon the GREEDY scheme, described in the previous section. We later compare in details the performance of these schemes in Section VII.

In MHT, each of the hash functions maps to a different subtable and therefore has a different distribution. Theorem 2 states that in this case, the overflow fraction lower bound γ_{LB} is computed using a weighted average distribution that is uniform across all buckets. As we later show, the MHT scheme implicitly complies with this condition when the subtable sizes follow a specific geometric decrease.

A. Description by Differential Equations

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

$$\frac{df_{i,j}(t)}{dt} = \begin{cases} -\frac{n}{\alpha_j m} f_{0,j}(t)\, g_j(t) & i = 0 \\ \frac{n}{\alpha_j m} f_{h-1,j}(t)\, g_j(t) & i = h \\ \frac{n}{\alpha_j m}\left(f_{i-1,j}(t) - f_{i,j}(t)\right) g_j(t) & \text{otherwise} \end{cases} \tag{13}$$

where $g_j(t) \triangleq \prod_{k=1}^{j-1} f_{h,k}(t)$ represents the probability that all the insertion attempts in subtables $T_1, \cdots, T_{j-1}$ meet full buckets, and thus MHT attempts 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 GREEDY scheme, let $f^a_{\text{MHT}}(t)$ denote the cumulative number of memory accesses done by time t, normalized by n. Then the following differential equation reflects the dynamics of $f^a_{\text{MHT}}(t)$:

$$\frac{df^a_{\text{MHT}}(t)}{dt} = \sum_{k=1}^{d-1} k \cdot g_k(t)\left(1 - f_{h,k}(t)\right) + d \cdot g_d(t), \tag{14}$$

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

B. Reduction to the SIMPLE Scheme

As in the GREEDY scheme, we prove the optimality of the MHT scheme by reducing it to the SIMPLE scheme, and do not rely on the differential equations, which are hard to solve analytically.

Our approach relies on the fact that *each subtable follows a local* SIMPLE *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, never return to T_j. Thus, within each subtable T_j, MHT behaves like the SIMPLE scheme, with a number of initial elements that depends on previous subtables.

More formally, let $n_j(t)$ denote the number of elements that are considered in subtable T_j up to time t, and $\gamma^t_j(t)$ denote the fraction of these elements that are not placed in subtable T_j. We will express these using f^{SIMPLE}_i and γ^t_{SIMPLE}, the corresponding functions in the SIMPLE scheme. Note that, as shown in Equations (4) and (6), $f^{\text{SIMPLE}}_i(t)$ and $\gamma^t_{\text{SIMPLE}}(t)$ depend only on the time t, the number of elements n, the number of buckets m, and the bucket size h; thus, we refer to them as $f^{\text{SIMPLE}}_i(t, m, n, h)$ and $\gamma^t_{\text{SIMPLE}}(t, m, n, h)$. We obtain the following theorem, which is valid for any partition of the subtables.

Theorem 5: Consider an $\langle a, d, c, h\rangle$ MHT hashing 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 $f^a_{\text{MHT}}(t) \le a$, $n_j(t)$, $\gamma^t_j(t)$, and $f_{i,j}(t)$ satisfy:

$$n_j(t) = n \cdot t \cdot \prod_{k=1}^{j-1} \gamma^t_k(t), \tag{15}$$

$$\gamma^t_j(t) = \gamma^t_{\text{SIMPLE}}(1, \alpha_j m, n_j(t), h), \tag{16}$$

$$f_{i,j}(t) = f^{\text{SIMPLE}}_i(1, \alpha_j m, n_j(t), h). \tag{17}$$

In addition, if the average number of memory accesses does not reach a by the end of the process, the overflow fraction of MHT is given by

$$\gamma_{\text{MHT}} = \prod_{j=1}^{d} \gamma^t_j(1). \tag{18}$$

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

The claimed result is immediately derived by setting the right parameters for each SIMPLE 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 $n_j(t)$. ■

C. Optimality of the MHT Scheme

We now prove that MHT is optimal on a given range of a, and in particular we show that the overflow fraction γ_{MHT} of the MHT scheme reaches the overflow fraction lower bound γ_{LB} for such a. Further, we demonstrate that MHT is optimal when its subtable sizes follow a specific geometric decrease.

Theorem 6: Consider an $\langle a, d, c, h\rangle$ MHT hashing scheme in which each subtable T_j has $\alpha_j \cdot m$ buckets, with $\sum \alpha_j = 1$. Further, let $p(a) = \gamma^t_{\text{SIMPLE}}(1, m, a \cdot n, h)$ denote the overflow fraction of the SIMPLE scheme with $a \cdot n$ elements. Then, for any values of d, c, and h, the $\langle a, d, c, h\rangle$ MHT scheme solves the OPTIMAL HASH TABLE CONSTRUCTION PROBLEM whenever it satisfies the two following conditions:
(i) The subtable sizes $\alpha_j \cdot m$ follow a geometric decrease of factor $p(a)$:

$$\alpha_j = \left(\frac{1 - p(a)}{1 - p(a)^d}\right) p(a)^{j-1}; \tag{19}$$

(ii) $a \le a^{co}_{\text{MHT}}$, where a^{co}_{MHT} is given by the solution of the following fixed-point equation:

$$a^{co}_{\text{MHT}} = \frac{1 - p\left(a^{co}_{\text{MHT}}\right)^d}{1 - p\left(a^{co}_{\text{MHT}}\right)}. \tag{20}$$

Proof: Consider a specific time t_0, in which we exhausted all $a \cdot n$ memory accesses. Up until this time, we used exactly $n_j(t_0)$ times the hash function $H_j(x)$ in subtable T_j. Since we aim at an optimal overflow fraction, the necessary condition on the distributions of hash function, given in Theorem 2, immediately implies that

$$\alpha_j = \frac{n_j(t_0)}{n \cdot a}. \tag{21}$$

By substituting the expression for α_j in (15), we get:

$$\begin{aligned}\gamma_j^t(t_0) &= \gamma_{\text{SIMPLE}}\left(1, \frac{n_j(t_0)}{n\cdot a}m, n_j(t_0), h\right)\\ &= 1 - \frac{\frac{n_j(t_0)}{n\cdot a}mh}{n_j(t_0)} + \frac{\frac{n_j(t_0)}{n\cdot a}m}{n_j(t_0)}\\ &\quad\cdot\sum_{i=0}^{h-1}(h-i)\cdot\frac{1}{i!}\left(\frac{n_j(t_0)}{\frac{n_j(t_0)}{n\cdot a}m}\right)^i e^{-\frac{n_j(t_0)}{\frac{n_j(t_0)}{n\cdot a}m}}\\ &= 1 - \frac{mh}{na} + \frac{m}{na}\sum_{i=0}^{h-1}(h-i)\cdot\frac{1}{i!}\left(\frac{na}{m}\right)^i e^{-\frac{na}{m}}\\ &= \gamma_{\text{SIMPLE}}^t(1, m, a\cdot n, h)\\ &= p(a)\end{aligned}$$

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:

$$t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right). \tag{22}$$

This, in turn, immediately gives us the claimed memory partition α_j of Equation (19).

We now turn to show that the overflow fraction is indeed optimal. Notice that overflowed elements arise in two situations:

1) Elements which are rejected by subtable T_d. By the time t_0, the fraction of overflowed elements out of these which were considered by T_d is $\gamma_d^t(t_0) = p(a)$. Furthermore, since the total number of elements considered is $n_d(t_0)$, we get that the total number of elements which were moved by T_d to the overflow list is $p(a)n_d(t_0) = n\cdot t_0\cdot p(a)^{d-1}p(a) = n\cdot t_0\cdot p(a)^d$.
2) Elements arriving after time t_0 and are rejected without any consideration, because the number of total memory accesses is exhausted. By the definition of t_0, the number of such elements is $n - nt_0$.

Hence, the overflow fraction of the MHT scheme with the above memory partitioning is:

$$\begin{aligned}\gamma_{\text{MHT}}(a) &= \frac{n - n\cdot t_0 + n\cdot t_0\cdot p(a)^d}{n}\\ &= 1 - t_0\left(1 - p(a)^d\right)\\ &= 1 - a(1-p(a))\\ &= 1 - \frac{mh}{n} + \frac{m}{n}\sum_{i=0}^{h-1}(h-i)\cdot\frac{1}{i!}\left(\frac{na}{m}\right)^i e^{-\frac{na}{m}}\end{aligned}$$

By comparing $\gamma_{\text{MHT}}(a)$ with Theorem 1, we immediately conclude that the MHT scheme with the above partitioning solves the OPTIMAL HASH TABLE CONSTRUCTION PROBLEM for the given a.

Finally, we observe by Equation (22) that as a increases, the time t_0, in which all memory accesses are exhausted, also increases. This implies that preserving the tightness to the lower bound can continue only until a_{MHT}^{co}, when $t_0 = 1$. ■

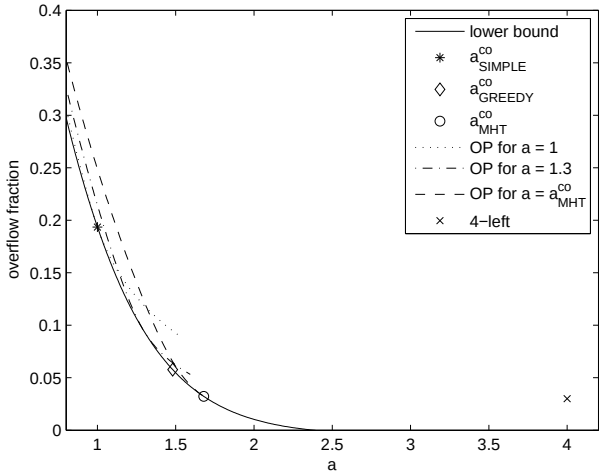


Fig. 7. Overflow fraction as a function of a with $d = 4$, $h = 4$, $c = 1$. OP(a) denotes the optimal partition of MHT for a as obtained by Theorem 6.

In the above theorem, for any $a \le a_{\text{MHT}}^{co}$, we found a specific partition of MHT that achieves optimality. Note that the overflow fraction can still be improved beyond a_{MHT}^{co}, albeit without preserving tightness to the overflow fraction lower bound γ_{LB}.

D. Case Study - A MHT $\langle a, 2, 1, 1\rangle$ Hashing-Scheme

In the following section, we consider a specific MHT scheme with $h = 1$, $d = 2$, and $m = n$ (i.e. with load $c = 1$). For this specific case, we can provide closed-form expressions for the cut-off point and the overflow ratio, by directly solving the set of differential equations in Equation (13). In case $h = 1$ and $d = 2$, this set is reduced to:

$$\begin{aligned}\frac{df_{0,1}(t)}{dt} &= -\frac{1}{\alpha_1}f_{0,1}(t)\\ \frac{df_{1,1}(t)}{dt} &= \frac{1}{\alpha_1}f_{0,1}(t)\\ \frac{df_{0,2}(t)}{dt} &= -\frac{1}{\alpha_2}f_{0,2}(t)\cdot f_{1,1}\\ \frac{df_{1,2}(t)}{dt} &= \frac{1}{\alpha_2}f_{0,2}(t)\cdot f_{1,1}\end{aligned} \tag{23}$$

where $f_{0,1}(0) = f_{0,2}(0) = 1$ and $f_{1,1}(0) = f_{1,2}(0) = 0$.

As in the SIMPLE scheme, we immediately get that $f_{0,1}(t) = e^{-\frac{1}{\alpha_1}t}$ and $f_{1,1}(t) = 1 - e^{-\frac{1}{\alpha_1}\cdot t}$. By substituting $f_{1,1}$ in Equation (23), we get that:

$$f_{0,2}(t) = e^{\left(-\frac{1}{\alpha_2}\cdot t - \frac{\alpha_1}{\alpha_2}e^{-\frac{1}{\alpha_1}\cdot t} + C_1\right)}$$

Where C_1 is a constant, that is determined by the initial condition, and equals to $\frac{\alpha_1}{\alpha_2}$. It is also immediately follows that $f_{1,2}(t) = 1 - f_{0,2}(t)$. Thus, the overflow fraction $\gamma_{\text{MHT}}(t)$ is

given by:

$$\begin{aligned}\gamma_{\text{MHT}}(t) &= \frac{n-(\alpha_1 m f_{1,1}(t)+\alpha_2 m f_{1,2}(t))}{n}\\ &= 1-\alpha_1 f_{1,1}(t)-\alpha_2 f_{1,2}(t)\\ &= 1-\alpha_1\left(1-e^{-\frac{1}{\alpha_1}\cdot t}\right)\\ &\quad -\alpha_2\left(1-e^{\left(-\frac{1}{\alpha_2}\cdot t-\frac{\alpha_1}{\alpha_2}e^{-\frac{1}{\alpha_1}\cdot t}+\frac{\alpha_1}{\alpha_2}\right)}\right)\\ &= \alpha_1 e^{-\frac{1}{\alpha_1}\cdot t}\\ &\quad +\alpha_2 e^{\left(-\frac{1}{\alpha_2}\cdot t-\frac{\alpha_1}{\alpha_2}e^{-\frac{1}{\alpha_1}\cdot t}+\frac{\alpha_1}{\alpha_2}\right)}\end{aligned}$$

In order to compute $f^a_{\text{MHT}}(t)$, we substitute $f_{0,1}(t), f_{1,1}(t)$ and $f_{0,2}(t)$ in Equation (14) and solve the resulting differential equation:

$$\begin{aligned}\frac{df^a_{\text{MHT}}(t)}{dt} &= 2\cdot\left(1-e^{-\frac{1}{\alpha_1}\cdot t}\right)+1\cdot\left(e^{-\frac{1}{\alpha_1}\cdot t}\right)\\ &= 2-e^{-\frac{1}{\alpha_1}\cdot t},\end{aligned}$$

whose solution is:

$$f^a_{\text{MHT}}(t)=2t+\alpha_1 e^{-\frac{1}{\alpha_1}\cdot t}-\alpha_1$$

For the case we consider, $p(a)=1-\frac{1}{a}+\frac{1}{a}e^{-a}$. Thus, by Theorem 6, we get a fixed point equation for a^{co}_{MHT}:

$$a^{co}_{\text{MHT}}=2-\frac{1}{a^{co}_{\text{MHT}}}+\frac{1}{a^{co}_{\text{MHT}}}e^{-a^{co}_{\text{MHT}}}$$

with one positive solution:

$$a^{co}_{\text{MHT}}=1+2\cdot W\left(\frac{1}{2}e^{-\frac{1}{2}}\right)\approx 1.4777 \qquad (24)$$

where the Lambert W function is the inverse function of the function $\omega(x)=xe^x$ [20].

At the cut-off point, assuming an optimal partition, the overflow fraction is

$$\gamma_{\text{MHT}}(t=1)=e^{-a^{co}_{\text{MHT}}}\approx 22.8\%.$$

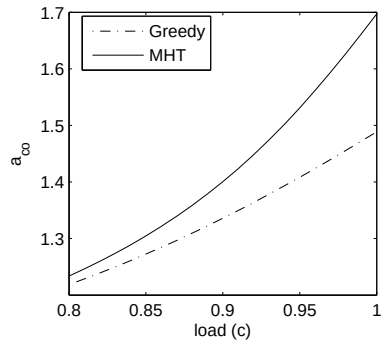
Likewise, if $n=0.1m$, the cut-off point is $a^{co}_{\text{MHT}}=1.0507$ and the corresponding overflow fraction is $\gamma_{\text{MHT}}(t=1)=0.26\%$.

VII. Comparative Evaluation

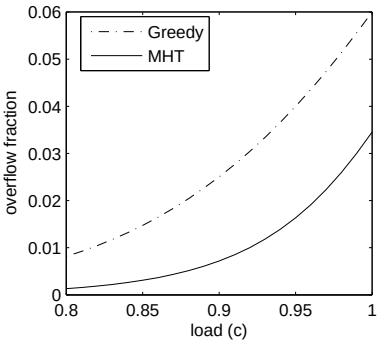
Fig. 7 illustrates the influence of the memory partition on the overflow fraction and the optimality of MHT. It was obtained with $d=4$, $h=4$ and $c=1$. All values were derived from the analytical formulas above, except for the d-left hashing scheme, for which we ran simulations with $m=4,000$, $n=16,000$ and d equally-sized subtables.

First, the solid line plots the overflow fraction lower-bound $\gamma_{\text{LB}}(a)$ from Theorem 1. Thus, no scheme can achieve an asymptotic overflow fraction below this line.

As elements are successively inserted and the total number of memory accesses $a\cdot n$ increases, the overflow fractions $\gamma_{\text{SIMPLE}}(a)$ and $\gamma_{\text{GREEDY}}(a)$ of the SIMPLE and the GREEDY schemes follow this lower-bound line, respectively until $a^{co}_{\text{SIMPLE}}\triangleq 1$ with $\gamma_{\text{SIMPLE}}=19.5\%$ (Theorem 3), and $a^{co}_{\text{GREEDY}}=1.488$ with $\gamma_{\text{GREEDY}}=6.00\%$ (Theorem 4).



(a) Cut-off point a^{co} as a function of the load c.



(b) Overflow fraction γ as a function of the load c.

Fig. 8. Cut-off points of GREEDY and MHT schemes, and the corresponding overflow fraction, as a function of the load c, with bucket size $h=4$ and $d=4$ hash functions.

On the contrary, in the case of MHT, for a given partition, $\gamma_{\text{MHT}}(a)$ does not go down along the lower-bound line. As shown in the proof of Theorem 6 [21], for any given a, an MHT scheme using the optimal geometrically-descending partition for a will be strictly above the lower-bound line, then reach it at a, then rebound and be above it again. This is indeed illustrated using the optimal partitions for $a=1$ and $a=a^{co}_{\text{MHT}}=1.697$. The corresponding optimal overflow fractions are $\gamma_{\text{MHT}}(a=1)=\gamma_{\text{SIMPLE}}=19.5\%$ and $\gamma_{\text{MHT}}(a=a^{co}_{\text{MHT}})=3.45\%$.

Last, we compare the performance of MHT with that of the d-left algorithm, in which $a=d=4$ [8], [9]. It can be seen that d-left achieves an overflow fraction of $\gamma_{d-\text{left}}=3.17\%$, which is not far from the overflow fraction of MHT with $a=a^{co}_{\text{MHT}}$, while on average MHT saves more than half of the memory accesses.

We conclude by comparing the MHT and GREEDY schemes. Fig. 8(a) compares the respective cut-off points $a=a^{co}_{\text{GREEDY}}$ and $a=a^{co}_{\text{MHT}}$ of GREEDY and MHT under different loads, with $h=4$ and $d=4$. Clearly, the cut-off point of MHT is larger, implying that its range of optimality is larger. Additionally, Fig. 8(b) shows the corresponding overflow fractions $\gamma_{\text{GREEDY}}(a=a^{co}_{\text{GREEDY}})$ and $\gamma_{\text{MHT}}(a=a^{co}_{\text{MHT}})$, illustrating how MHT can achieve a lower overflow fraction.

VIII. Conclusion

In this paper we considered hash-based data structures that have become crucial algorithmic building blocks for contemporary network elements that handle and analyze large amounts of data at very high speeds. For instance, for each arriving packet, routers need to perform an address lookup and several flow classification and identification operations, each often relying on a hash table scheme.

Unlike traditional hash tables which guarantee only amortized constant-time operations, in a networking setting hash tables should provide a constant worst-case bound of d per operation. Moreover, as the *average* cost per operation may dictate the overall performance of the network element (e.g., its throughput or its power consumption), we considered hash tables that also provide a constant bound a on this quantity.

Given a and d, we first presented a lower bound on the overflow fraction—the fraction of elements that cannot be stored in the hash table without violating these restrictions. Then, we studied three hashing schemes: a simple single-choice scheme (SIMPLE), a greedy multiple-choice scheme (GREEDY), and a multi-level scheme (MHT). For all these schemes, we first obtained an expression of their overflow fraction as a function of a. By comparing with our lower bound, we concluded that these schemes provide *optimal fast hashing* for a specific range of a's. In comparison, recently-proposed schemes, such as d-left, are shown by simulations to be far from the lower bound.

On the practical side, we were able to find the expected overflow fraction of the evaluated schemes, which determines the size of the required overflow list (for example, when implemented in CAM). Also, for the well-studied *multi-level hash table* scheme, we were able to prove that one can achieve optimal performance when the subtable sizes follow a specific geometric decrease. This confirms a widely-known rule-of-thumb.

Acknowledgment

We would like to thank Ran Ginosar for useful discussions.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[2] S. Kumar, J. Turner, and P. Crowley, "Peacock hashing: Deterministic and updatable hashing for high performance networking," in *IEEE Infocom*, 2008, pp. 556–564.

[3] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis, "Space efficient hash tables with worst case constant access time," in *Symposium on Theoretical Aspects of Computer Science*, 2003, pp. 271–282.

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

[5] A. Kirsch, M. Mitzenmacher, and G. Varghese., *Hash-Based Techniques for High-Speed Packet Processing*. DIMACS, 2009, to appear. [Online]. Available: http://www.eecs.harvard.edu/ michaelm/postscripts/dimacs-chapter-08.pdf

[6] M. Mitzenmacher, A. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds., 2000, pp. 255–312.

[7] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," in *ACM STOC*, 1994, pp. 593–602.

[8] B. Vöcking and M. Mitzenmacher, "The asymptotics of selecting the shortest of two, improved," in *Analytic Methods in Applied Probability: In Memory of Fridrih Karpelevich*, 2002, pp. 165–176.

[9] B. Vöcking, "How asymmetry helps load balancing," in *IEEE FOCS*, 1999, pp. 131–141.

[10] H. Song, S. Dharmapurikar, J. S. Turner, and J. W. Lockwood, "Fast hash table lookup using extended Bloom filter: an aid to network processing," in *ACM SIGCOMM*, 2005, pp. 181–192.

[11] A. Kirsch and M. Mitzenmacher, "Simple summaries for hashing with choices," *IEEE/ACM Trans. Networking*, vol. 16, no. 1, pp. 218–231, 2008.

[12] A. Z. Broder and A. R. Karlin, "Multilevel adaptive hashing," in *ACM-SIAM SODA*, 1990, pp. 43–53.

[13] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *European Symposium on Algorithms*, 2001, pp. 121–133.

[14] A. D. Barbour and P. G. Hall, "On the rate of poisson convergence," *Math. Proc. Cambridge Philos. Soc.*, vol. 95, no. 3, pp. 473–480, 1984.

[15] J. M. Steele, "Le cam's inequality and Poisson approximations," *American Mathematical Monthly*, vol. 101, pp. 48–54, 1994.

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

[17] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *J. of Applied Probability*, vol. 7, no. 1, pp. 49–58, 1970.

[18] ——, *Approximation of Population Processes*, 1981.

[19] M. Mitzenmacher, "The power of two choices in randomized load balancing," Ph.D. dissertation, University of California at Berkley, 1996.

[20] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth, "On the Lambert W function," *Advances in Computational Mathematics*, vol. 5, pp. 329–359, 1996.

[21] Y. Kanizo, D. Hay, and I. Keslassy, "Optimal fast hashing," Comnet, Technion, Israel, Technical Report TR08-05, 2008. [Online]. Available: http://comnet.technion.ac.il/ isaac/papers.html