This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2009 proceedings.

Optimal Fast Hashing

Yossi Kanizo
Dept. of Computer Science
Technion, Haifa, Israel
ykanizo@cs.technion.ac.il

David Hay
Dept. of Electrical Engineering
Politecnico di Torino, Turin, Italy
hay@tlc.polito.it

Isaac Keslassy
Dept. of Electrical Engineering
Technion, Haifa, Israel
isaac@ee.technion.ac.il

***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 CAM (content addressable memory)-based *overflow list* [3], [4]. An objective of an hashing scheme then becomes to reduce the *overflow fraction*, i.e. 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.

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

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

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. Due to space limits, some proofs are omitted and can be found in [13].

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

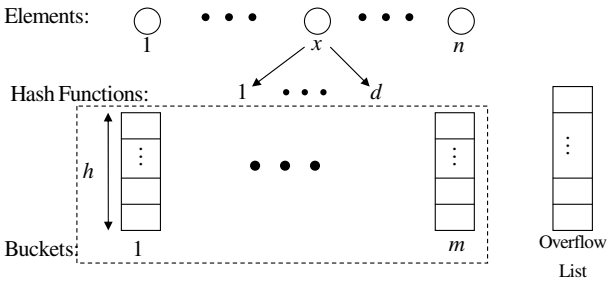


Fig. 1. Illustration of the hashing model.

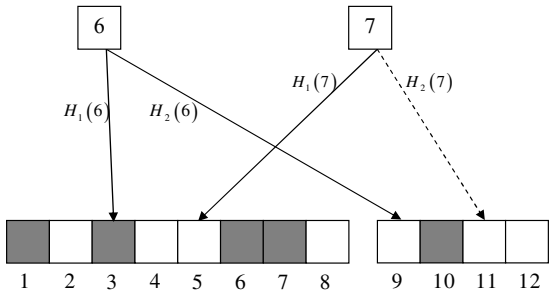


Fig. 2. Illustration of MHT scheme

Note that our sequential insertion framework does not allow schemes that move elements after their initial insertion, such as [4], [14]. 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 [14] 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. The proof is omitted and can be found in [13].

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.

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. The proof can be found in [13].

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 this lower bound is computed for the case where in every bucket $i \in \{1 \cdots m\}$, $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$, i.e. the weighted average of all distributions is uniform.

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

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.

Also, 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.

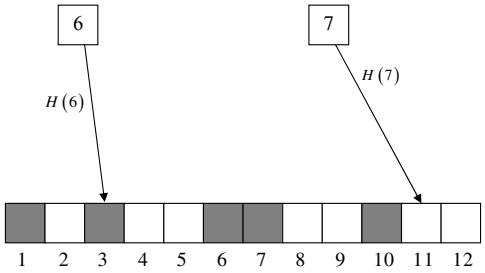


Fig. 3. Illustration of SIMPLE scheme

Fig. 3 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 adopt 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

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

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 $\vec{f}(t) \stackrel{\triangle}{=} (f_1(t), \ldots, f_d(t))$ 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 [15]–[17], 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].

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

$$f_i(t) = \begin{cases} \frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} & i < h \\ 1 - \sum_{k=0}^{h-1} \frac{1}{k!}\left(\frac{n}{m}t\right)^k e^{-\frac{n}{m}t} & i = h \end{cases} \quad (2)$$

Note that the solution is actually Poisson$\left(\lambda = \frac{n}{m}t\right)$. 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 Bin$\left(nt, \frac{1}{m}\right)$, and the corresponding limit distribution is 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

$$\begin{aligned} \gamma_{\text{SIMPLE}}(t) &= 1 - \frac{m}{n}\sum_{i=0}^{h-1} i \cdot \frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} \\ &\quad -\frac{m}{n}\cdot h \cdot \left(1 - \sum_{k=0}^{h-1}\frac{1}{k!}\left(\frac{n}{m}t\right)^k e^{-\frac{n}{m}t}\right) \\ &= 1 - t + \frac{m}{n}\sum_{i=h+1}^{\infty}(i-h)\cdot\frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} \quad (3) \end{aligned}$$

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:

$$\begin{aligned} \gamma^t_{\text{SIMPLE}}(t) &= 1 - \frac{m}{nt}\sum_{i=0}^{h-1} i \cdot \frac{1}{i!}\left(\frac{n}{m}t\right)^i e^{-\frac{n}{m}t} \\ &\quad -\frac{m}{nt}\cdot h \cdot \left(1 - \sum_{k=0}^{h-1}\frac{1}{k!}\left(\frac{n}{m}t\right)^k e^{-\frac{n}{m}t}\right) \quad (4) \end{aligned}$$

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.

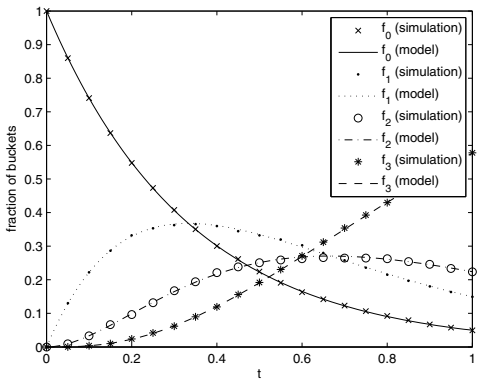
The equations above are only true as long as the average number of memory accesses per element is at most a. Since this average number equals t at time t, the process is stopped at $t = a$. Then, the optimality of the scheme is obtained by substituting $c = \frac{n}{mh}$ and $t = a$ in Equation (3), and comparing it to $\gamma_{\text{LB}}(a)$ of Theorem 1. ■

We finish by the following simple example:

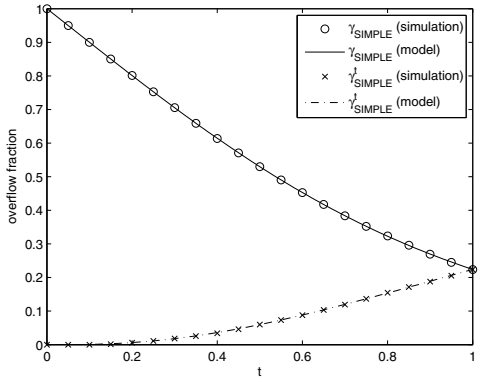
Example 5: For the case where $h = 1$, the overflow fraction $\gamma_{\text{SIMPLE}}(t)$ is given by:

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

Therefore, for $a = 1$, it meets the lower bound at $t = 1$, using



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



(b) Overflow fraction.

Fig. 4. Model and simulation results for the SIMPLE scheme given load $c = 1$, bucket size $h = 3$, and memory size $m = 10,000$.

$c = \frac{n}{m}$:

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

And if $c = 1$, we get:

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

C. 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. 4(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. 4(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

We now introduce the GREEDY scheme, a natural extension to the SIMPLE 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. 5 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

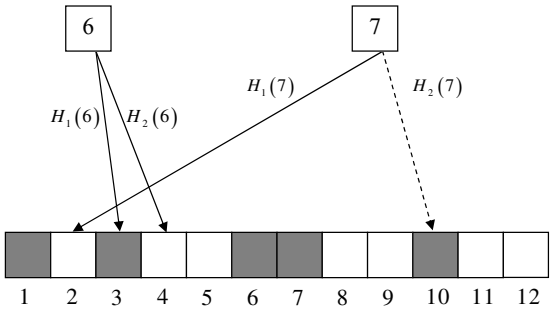


Fig. 5. Illustration of GREEDY scheme

buckets storing i elements at time t, then

$$\frac{df_i(t)}{dt} = \begin{cases} -\frac{n}{m} f_0(t)\, g(t) & i = 0 \\ \frac{n}{m} f_{h-1}(t)\, g(t) & i = h \\ \frac{n}{m}\left(f_{i-1}(t) - f_i(t)\right) g(t) & \text{otherwise} \end{cases} \quad (5)$$

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 (1), 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

$$\frac{df^a_{\text{GREEDY}}(t)}{dt} = \sum_{k=1}^{d-1} k \cdot (f_h(t))^{k-1} (1 - f_h(t)) + d \cdot (f_h(t))^{d-1}, \quad (6)$$

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. The proof of the following theorem is omitted and can be found in [13].

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 $a^{co}_{\text{GREEDY}} = f^a_{\text{GREEDY}}(1)$.

Although in general it is difficult to obtain analytically a closed form solution for the differential equations describing the GREEDY scheme, we can do it for the following simple example. The exact derivation steps appear in [13].

Example 6: When $h = 1$ and $d = 2$:

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

and the overflow fraction is:

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

Finally, the cut-off point is:

$$a^{co}_{\text{GREEDY}} = \frac{m}{n} \cdot \ln\left(\frac{e^{2\frac{n}{m}} + 1}{2}\right).$$

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

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

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

We found that the GREEDY scheme is optimal for $a \leq a^{co}_{\text{GREEDY}}$. We now want to find a scheme that is optimal beyond a^{co}_{GREEDY}. Therefore, we consider another hashing scheme, *the multi-level hash table* (MHT), and evaluate its range of optimality. We later compare the performance of the MHT and GREEDY schemes in Section VII.

We defined MHT in Example 1 (Section II). As in previous schemes, to keep an average number of memory accesses per element of at most a, we stop inserting elements when a total of $a \cdot n$ memory accesses is reached, and the remaining elements are placed in the overflow list.

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:

$$\frac{df_{i,j}(t)}{dt} = \begin{cases} -\frac{n}{\alpha_j m} f_{0,j}(t)\, g_j(t) & i = 0 \\ \frac{n}{\alpha_j m} f_{h-1,j}(t)\, g_j(t) & i = h \\ \frac{n}{\alpha_j m}\left(f_{i-1,j}(t) - f_{i,j}(t)\right) g_j(t) & \text{otherwise} \end{cases} \tag{7}$$

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 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 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 $f^a_{\text{MHT}}(t)$:

$$\frac{df^a_{\text{MHT}}(t)}{dt} = \sum_{k=1}^{d-1} k \cdot g_k(t)\left(1 - f_{h,k}(t)\right) + d \cdot g_d(t), \tag{8}$$

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, do not come back to T_j. Thus, within each subtable T_j, MHT behaves like SIMPLE, 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 (2) and (4), $f_i^{\text{SIMPLE}}(t)$ and $\gamma_{\text{SIMPLE}}^t(t)$ only depend 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 $f_i^{\text{SIMPLE}}(t,m,n,h)$ and $\gamma_{\text{SIMPLE}}^t(t,m,n,h)$. We obtain the following theorem, which is valid for any arbitrary partition of the subtables. The proof is in [13].

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 $f_{\text{MHT}}^a(t) \le a$, the functions $n_j(t)$, $\gamma_j^t(t)$ and $f_{i,j}(t)$ satisfy

$$n_j(t) = n \cdot t \cdot \prod_{k=1}^{j-1} \gamma_k^t(t), \quad (9)$$

$$\gamma_j^t(t) = \gamma_{\text{SIMPLE}}^t(1, \alpha_j m, n_j(t), h), \quad (10)$$

$$f_{i,j}(t) = f_i^{\text{SIMPLE}}(1, \alpha_j m, n_j(t), h). \quad (11)$$

Last, 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

$$\gamma_{\text{MHT}} = \prod_{j=1}^{d} \gamma_j^t(1). \quad (12)$$

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. The proof of the following theorem can be found in [13].

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 $p(a) = \gamma_{\text{SIMPLE}}^t(1,m,a\cdot n,h)$ 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 $p(a)$:

$$\alpha_j = \left(\frac{1-p(a)}{1-p(a)^d}\right) p(a)^{j-1}; \quad (13)$$

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

$$a_{\text{MHT}}^{co} = \frac{1-p(a_{\text{MHT}}^{co})^d}{1-p(a_{\text{MHT}}^{co})}. \quad (14)$$

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}.

Although in general it is difficult to obtain analytically a closed-form solution for the differential equations describing the MHT scheme, we can do it for the following simple example. More details appear in [13].

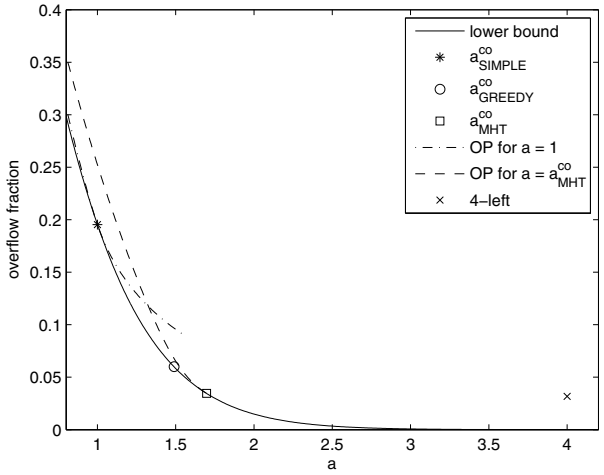


Fig. 6. Overflow fraction as a function of a with $d=4$, $h=4$, $c=1$. OP(a) denotes the optimal partition of MHT for a as obtained by Theorem 6.

Example 7: When $h=1$, $d=2$, and $c=1$, i.e. $n=m$, we can find that the overflow fraction $\gamma_{\text{MHT}}(t)$ is given by:

$$\gamma_{\text{MHT}}(t) = \alpha_1 e^{-\frac{1}{\alpha_1}\cdot t} + \alpha_2 e^{\left(-\frac{1}{\alpha_2}\cdot t - \frac{\alpha_1}{\alpha_2} e^{-\frac{1}{\alpha_1}\cdot t} + \frac{\alpha_1}{\alpha_2}\right)},$$

and the cut-off point a_{MHT}^{co} is given by:

$$a_{\text{MHT}}^{co} = 1 + 2 \cdot W\left(\frac{1}{2} e^{-\frac{1}{2}}\right) \approx 1.4777,$$

where the Lambert W function is the inverse function of the function $\omega(x) = xe^x$ [18]. At the cut-off point, assuming an optimal partition, the overflow fraction is

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

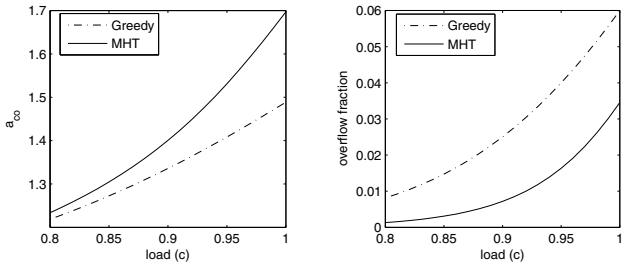
Likewise, if $n = 0.1m$, the cut-off point is $a_{\text{MHT}}^{co} = 1.0507$ and the corresponding overflow fraction is $\gamma_{\text{MHT}}(t=1) = 0.26\%$.

VII. Comparative Evaluation

Fig. 6 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 $a_{\text{SIMPLE}}^{co} \stackrel{\triangle}{=} 1$ with $\gamma_{\text{SIMPLE}} = 19.5\%$ (Theorem 3), and $a_{\text{GREEDY}}^{co} = 1.488$ 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. 7. 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 [13], 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. 7(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 also larger. Additionally, Fig. 7(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.

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

This work was partly supported by the European Research Council Starting Grant n° 210389. We would also like to thank Ran Ginosar for useful discussions.

References

[1] T. H. Cormen, C. E. Leiserson *et al.*, *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

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

[3] D. Fotakis, R. Pagh *et al.*, "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 *et al.*, "Hash-based techniques for high-speed packet processing," *Unpublished manuscript*.

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

[7] Y. Azar, A. Z. Broder *et al.*, "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 *et al.*, "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] Y. Kanizo, D. Hay *et al.*, "Optimal fast hashing," Comnet, Technion, Israel, Technical Report TR08-05, 2008. [Online]. Available: http://comnet.technion.ac.il/ isaac/papers.html

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

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

[16] 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.

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

[18] R. Corless, G. Gonnet *et al.*, "On the Lambert W function," *Advances in Computational Mathematics*, vol. 5, pp. 329–359, 1996.