Hash Tables With Finite Buckets Are Less Resistant to Deletions

Yossi Kanizo, David Hay, and Isaac Keslassy

***Abstract*—We show that when memory is bounded, i.e. buckets are finite, dynamic hash tables that allow insertions and deletions behave significantly worse than their static counterparts that only allow insertions. This behavior differs from previous results in which, when memory is unbounded, the two models behave similarly.**

We show the decrease in performance in dynamic hash tables using several hash-table schemes. We also provide tight upper and lower bounds on the achievable overflow fractions in these schemes. Finally, we propose an architecture with content-addressable memory (CAM), which mitigates this decrease in performance.

I. Introduction

A. Background

Networking devices often use *dynamic* hash tables, in which elements keep arriving and departing, and not *static* ones that are built only once. However, for simplicity, device designers typically model the performance of the dynamic hash tables using models of the static hash tables. This paper shows that these static models can lead to a significant *under-estimation of the drop rate* in the dynamic case.

This under-estimation of the drop rate can potentially affect the performance of networking devices. Hash tables form the core building block of many networking device operations, such as flow counter management, flow state keeping, elephant traps, virus signature scanning, and IP address lookup algorithms. If memory is allocated to the dynamic hash tables according to the static model, many more elements might need to be dropped from the hash tables than initially estimated.

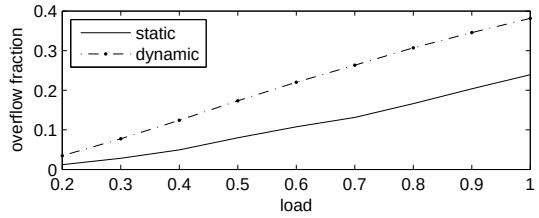
Using the static model seems natural. In fact, dynamic hash tables are known for being *typically harder to model* than static ones, sometimes even lacking any mathematical analysis [1]. Therefore, the static model appears to be a simpler and more accessible option to the network designer.

More significantly, past studies have also found the *same asymptotic behavior* in dynamic and in static hash tables, in at least three cases:
(a) In the static case in which n elements are uniformly hashed into n infinite buckets, the maximum bucket size is known to be approximately $\log n / \log \log n$ with high probability [2], [3]. The dynamic case yields the same result, assuming alternate departures and arrivals of random elements while keeping n elements in the hash table after each arrival.
(b) Likewise, when inserting each element in the least-loaded of two random buckets (d-random algorithm with $d = 2$), the maximum bucket size is $\log \log n / \log 2 + O(1)$ in the static case; and again, the dynamic case yields the same result [3], [4].
(c) Similarly, using the asymmetric d-left algorithm, the static case and the dynamic case yield again the same bound on the maximum bucket size [5].
Therefore, as illustrated in these three cases, given a large number of elements, it appears that the network designer could use the simpler static model for the dynamic case.

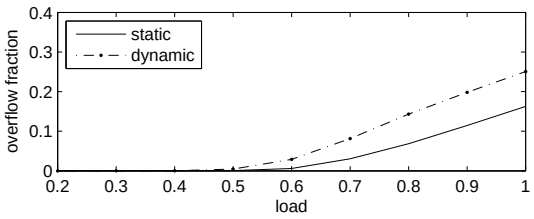
Y. Kanizo is with the Dept. of Computer Science, Technion, Haifa, Israel. Email: ykanizo@cs.technion.ac.il.

D. Hay is with the Dept. of Electrical Engineering, Columbia Univ., NY, USA. Email: hdavid@ee.columbia.edu.

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



(a) d-random with a stash



(b) Cuckoo hashing with a stash

Fig. 1. Overflow fraction with 2 hash functions and bucket size 1, using both the static and the dynamic model.

In this paper, we focus on the realistic scenario in which buckets are finite, as used in networking devices, contrarily to the infinite-bucket case assumed above. We show that the dynamic hash table can exhibit a *significantly worse* drop rate than its static counterpart.

B. Intuitive Example

Fig. 1 plots the system overflow fraction as a function of the load, i.e. the fraction of elements not placed in the buckets as a function of the average number of elements per bucket. It shows the overflow fraction for both a static system, where there are only insertions, and a dynamic system, where we alternate between deletions and insertions while a fixed load is maintained [4], [6]. To measure the overflow fraction, it relies on an overflow list, called *stash*, to which new elements are

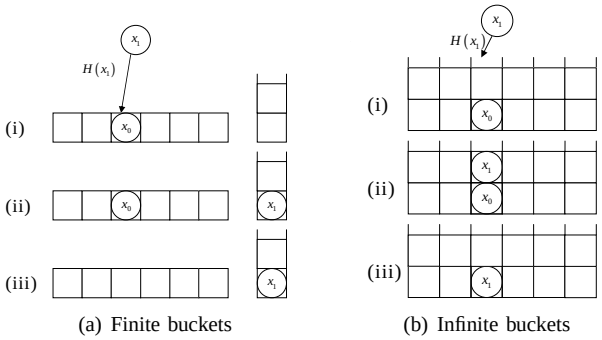


Fig. 2. An example demonstrating the degradation of performance in dynamic hash tables.

moved when they cannot be inserted in the hash table. Fig. 1(a) and 1(b) show the overflow fraction of the d-random algorithm with a stash [4] and the cuckoo hashing with a stash [7], [8]. The overflow fractions are obtained in simulations using 2048 buckets, 10^6 rounds with one random element deletion and one element insertion in each round, and a standard pseudorandom number generator to obtain hash values[1].

Both figures clearly show a non-negligible degradation in the overflow fraction of the dynamic system. For instance, the cuckoo hashing scheme with load of 0.6 yields an overflow fraction of 0.52% and 2.97% in the static and dynamic models, respectively. Moreover, while for cuckoo hashing scheme with load of 0.5 the overflow fraction in the static model goes to 0 [9], it does so more slowly in the dynamic case. For instance, for $m = 1024$ we got an overflow fraction in the static and dynamic models of 0.05% and 0.44%, where for $m = 16384$ we got 0.0012% and 0.0606%, respectively.

The intuition behind this difference in behavior is that if the bucket size is bounded, once an element is placed in the overflow list it stays there regardless of whether the corresponding bucket become available later upon deletion. Therefore, the order of the insertion and deletion operations directly affects the performance. This is typically not the case in the unbounded bucket case, and the difference can cause a drastic degradation in the scheme performance.

Fig. 2 illustrates this degradation in performance, using the same scenario both for the finite and the infinite bucket sizes. For the case of finite buckets, we assume bucket sizes of 1, an overflow list, and an insertion algorithm that uses only one hash function. We consider the following scenario: Let t be the time when a new element x_1 is hashed to a full bucket j that already stores element x_0 (step (i) in both Fig. 2(a) and Fig. 2(b)). If a finite bucket is used, then x_1 is moved to the overflow list (step (ii) in Fig. 2(a)), while in the infinite-bucket case, x_1 is simply stored in bucket j (step (ii) in Fig. 2(b)). Let $t' > t$ be the time when element x_0 is deleted. Assuming that element x_1 is not deleted before t', it stays in the overflow list in the finite-bucket case, while in the infinite-bucket case it is stored in bucket j (step (iii)).

Therefore, in the dynamic case with finite bucket sizes, element x_1 is in the overflow list, even though its corresponding bucket j is empty. This could never happen in the static case (elements are stored in the overflow list only after their corresponding buckets are full, and full buckets cannot become empty). It also could never happen in the dynamic case with infinite buckets (there is no overflow list).

[1]Simulations with ten times more buckets or rounds yielded near-identical results.

C. Our Contributions

In this paper, we show that dynamic hash tables with finite buckets behave worse than static ones.

We start by considering a simplistic dynamic scheme with a single hash function. We model this hashing scheme analytically using three different models: a discrete-time Markov chain, a continuous model with a birth-death chain, and a fluid model with a continuous-time Markov process. We find that this simplistic dynamic scheme performs notably worse than its corresponding static scheme.

Then, we derive a lower bound on the overflow fraction in the dynamic model of *any* hash-table scheme that uses uniform hash functions and does not move back elements once they were placed in the overflow list. We prove that when the *average* number of memory accesses per insertion a increases, the overflow fraction can decrease as slowly as $\Omega(1/a)$. This indicates that the bad performance of dynamic schemes is fundamental, and is hard to solve by simply using additional memory accesses.

Next, we introduce an online multiple-choice scheme. We demonstrate that this scheme reaches that lower bound and therefore is optimal up to a certain rate of memory access, which depends on the system parameters.

However, due to the slow decrease of the lower bound, optimality may be insufficient for certain applications. Therefore, we suggest changing the assumptions and moving back elements from the overflow list when a bucket becomes available upon deletion. We propose the M-B (Moving-Back) scheme that uses a CAM (content-addressable memory) device that stores the elements along with their hash values. A parallel lookup operation is used once an element is deleted and its bucket becomes non-full. This operation, supported by the CAM, finds an element in the overflow list that can be moved back to the bucket. This scheme is shown to beat the initial lower bound without a CAM.

Finally, we evaluate in this paper all proposed schemes using simulations as well as experiments with real hash functions applied on real-life traces.

Paper Organization: We start with preliminary definitions in Section II. Section III presents and analyzes the single-choice SINGLE scheme, while Sections IV and V provide a lower bound on the overflow fraction. Then, in Section VI we present and analyze the multiple-choice MULTIPLE scheme, and in Section VII we present the CAM-based M-B scheme which, upon deletion, moves back elements from the overflow list. Finally, we evaluate all the analytical results in Section VIII.

Note that, for the sake of readability, some proofs are presented in the appendices of this paper.

II. Problem Statement

A. Terminology and Notations

This paper considers *single- and multiple-choice hash schemes with a stash* [10], [11]. Such schemes consist of two data structures: *(i)* A *hash table* of total memory size $m \cdot h$, partitioned into m buckets of size h; *(ii)* An *overflow list*, usually stored in an expensive CAM. Note that the overflow list can also be absent, in which case overflow elements are simply dropped.

As in traditional hash tables, the schemes should support three basic operations: element insertions, element deletions, and lookups. We call the (infinitely long) sequence of these operations the *arrival sequence* of the scheme. In the paper, we focus mostly on a specific arrival sequence, alternating between departures of a random element (picked uniformly at random) and insertions of a new element [4], [6].

Multiple-choice hashing schemes employ up to d probability distributions over the set of buckets; these distributions are then used to generate a *hash-function set* $\mathcal{H} = \{H_1, \ldots, H_d\}$ of d independent hash functions. For each element x and each operation, the scheme can consider only the buckets $\{H_1(x), \ldots, H_d(x)\}$ (and the overflow list). In addition, we assume that the scheme must access a bucket to obtain any information on it (thus, if the hashing scheme tries to insert an element in a full bucket, it must access the bucket first).

Our goal is to minimize the *expected overflow fraction* of the scheme, i.e. the fraction of elements that are placed in the overflow list, subject to the (total and average) number of *memory accesses*. We count as one memory access reading and updating all the elements of a single bucket (this corresponds to the common practice of sizing the bucket size by the width of the memory word) and we do not count accesses to the overflow list. We further assume that up to d buckets can be read in parallel before deciding which one to update, requiring a total of d memory accesses.

Formally, the hashing scheme and the optimization problem are captured by the following two definitions, where the load c is the ratio of the total number of elements n by the total memory size mh: $c = \frac{n}{mh}$.

Definition 1: When the load is c and the bucket size is h, an $\langle a, d, c, h\rangle$ *hashing scheme* is a scheme with an expected (respectively, maximum) number of memory accesses per element of at most a (respectively, d).

Definition 2: The OPTIMAL DYNAMIC HASH TABLE PROBLEM is to find an $\langle a, d, c, h\rangle$ hashing scheme that minimizes the expected overflow fraction γ as the number of elements n goes to infinity. Whenever defined, let γ_{OPT} denote this optimal expected limit overflow fraction.

B. Arrival Models

Throughout the paper, we will use three different models for the arrivals and departures of elements: a *discrete model* with a finite number of elements; a *continuous model* with a finite number of elements; and a *fluid model* based on differential equations with an infinite number of elements. Our objective is to model a constant load, i.e. a constant number of elements in the system, so that departing elements are replaced by arriving elements.

Discrete Model — In the *discrete model*, we assume that time is divided into time-slots of unit duration, and start at time $t = 0$ with n elements in the overflow list. At the start of each time-slot $t > 0$, an element is chosen uniformly at random among all n elements in the system to depart. Next, at the end of time-slot t, a new element arrives and is inserted according to the hashing scheme into either a non-full bucket or the overflow list. Therefore, by the end of each time-slot t, there are always n elements in the system, either in the hash table or in the overflow list.

Continuous Model — The second model is a *continuous-time model*, starting again at time $t = 0$ with n elements in the overflow list. In this model, each element stays in the system for an exponentially-distributed duration of average 1. Therefore, at each infinitesimal time-interval $[t, t + \delta t]$, the probability that a given element departs is $n \cdot \delta t + o(\delta t)$. For each element departure, another element is automatically generated and inserted in the system according to the hashing algorithm into either a non-full bucket or the overflow list. Again, there are always n elements in the system at each time t, ensuring a constant load.

Since there are n departures per time-unit on average instead of a single one, the continuous system can be seen as a speeded-up version of the discrete system. In fact, when only looking at the system during the discrete element departure times, which follow exponentially-distributed inter-departure times, *we obtain the discrete model again*.

Incidentally, although each element departure triggers the arrival of another element with different hashed buckets, we will sometimes refer by simplicity to the departed element as if it was reinserted.

Fluid Model — The last model is the *fluid model*, which attempts to model the behavior of the continuous system as the number of elements n and the number of buckets m go to infinity with a constant limit ratio $ch = \lim_{n\to\infty} \frac{n}{m}$. In the fluid model, we will often analyze the system using differential equations, and will be mainly interested in their fixed-point solutions. Again, we will assume that at $t = 0$, all elements are in the overflow list.

In the fluid model, as in the finite continuous-time model, elements stay in the system for an exponentially-distributed duration of average 1, and therefore the departure rate from each bucket is proportional to the bucket size.

In addition, as in the other models, element departures trigger element arrivals. Note that in the continuous model, the average arrival rate per bucket is $\frac{n}{m}$, since the arrival rate is n and there are m buckets. Therefore, in the fluid model, we model a constant average arrival rate per bucket of $ch = \lim_{n\to\infty} \frac{n}{m}$. Likewise, in the continuous model, when arriving elements use a uniformly-distributed hash function, they hash into each bucket at a rate equal to the average rate of $\frac{n}{m}$. In the fluid model, since we consider an infinite number of buckets, a *uniformly-distributed hash function* is not well defined. By extension, and for simplicity, we will define such a function as one that enables the same arrival rate of ch to all buckets.

Furthermore, we will define the average hashing rate per element a such that it is valid at any time t. We will also assume that elements may not use hash functions that pick with higher probability buckets with lower occupancy, i.e. that the average hashing rate limit of a is valid given any bucket size. Thus, if one tenth of the buckets are empty, a uniform hash will find one tenth of its buckets empty as well. Of course, an element might still decide to enter a bucket with lower occupancy with higher probability.

Model Alternatives — In general, to model system scaling, we would be interested in using either the discrete or continuous finite models, and then in studying how their solution scales with n. However, given the complex interactions between the n elements, these models often prove intractable. Therefore, we will use the fluid model in these cases, and will most often *not be able* to prove convergence of the discrete or continuous models to the fluid model. Likewise, we will not always prove convergence of the differential equations to the fixed-point solutions. This is, of course, a limit of our analysis.

On the other hand, for the single-choice hashing scheme (Section III), we provide a full analysis with the three models, and prove that the limit of the discrete and continuous finite models behaves indeed like in the fluid model. In simulations, we will also show that the scaled systems converge fast to their fluid model. We refer to [12] for a more complete discussion of the sufficient conditions for the convergence to the fluid-limit fixed-point solution.

III. A Single-Choice Hashing Scheme

We start by analyzing a simplistic hashing scheme, which uses only a single uniformly-distributed hash function H to insert elements in the hash table. Each element x is stored in bucket $H(x)$, if it is not full, and in the overflow list otherwise. Since an element uses exactly one hash function, its average number of memory accesses per element is $a = 1$. Of course, this simplistic scheme would probably not be implemented in advanced networking devices. However, it provides a better intuition on the reasons behind the performance degradation in dynamic hash-table schemes.

Discrete Model — We first develop an analytical model for the scheme within the discrete framework presented in Section II. Let $p_k(t)$ denote the fraction of buckets that have k elements at the end of time-slot t, and $p(t) = (p_1(t), \ldots, p_h(t))$. Using this discrete model, we obtain the following result on the limits of the distribution of p and of the overflow fraction. The full proof appears in Appendix A-A and is based on a birth-death Markov chain that models the occupancy of an arbitrary bucket over time.

Theorem 1: Let $C = \sum_{\ell=0}^{h} \binom{n}{\ell} (\frac{1}{m-1})^{\ell}$. In the discrete model,
(i) the distribution of $p(t)$ converges to the Engset distribution π^n [13], [14]; namely,

$$\pi_k^n = \frac{1}{C} \cdot \binom{n}{k} \cdot \left(\frac{1}{m-1}\right)^k. \tag{1}$$

(ii) the overflow fraction converges to

$$\frac{1}{C} \cdot \binom{n}{h} \cdot \left(\frac{1}{m-1}\right)^h \cdot \left(1 - \frac{h}{n}\right). \tag{2}$$

It is interesting to note that Equation (1) can be rewritten as a truncated binomial expression

$$\pi_k^n = \frac{\binom{n}{k}\left(\frac{1}{m}\right)^k\left(1-\frac{1}{m}\right)^{n-k}}{\sum_{l=0}^{h}\binom{n}{l}\left(\frac{1}{m}\right)^l\left(1-\frac{1}{m}\right)^{n-l}}, \tag{3}$$

which hints at the following interesting equivalent system: the bucket occupancy is distributed as if the n elements were assigned uniformly at random among the m buckets, and then the buckets with more than h elements were completely cleared out and had *all* their elements put in the overflow list. This is in contrast with the static system in which only elements exceeding the bucket capacity of h are placed in the overflow list. Therefore, it nicely illustrates the difference between the static and dynamic cases.

A detailed example of the behavior of the scheme in the dynamic and static setting appears in Appendix B-A, which shows a simplistic setting where, as the number of buckets increases (with fixed load), the dynamic case yields an overflow fraction of 50%, while the static case has an overflow fraction of only $e^{-1} \approx 36.79\%$.

Continuous Model — We now turn to the continuous model in which elements stay in the system for an exponentially-distributed duration of average 1. It turns out that the continuous model yields similar results to those of the discrete model (Theorem 1).

Theorem 2: In the continuous model, the single-choice hashing scheme has the same stationary distribution and overflow fraction as in the discrete model.

Fluid Model — We now analyze the *infinite* system using a fluid model. In the fluid model, as in the finite continuous-time model, elements stay in the system for an exponentially-distributed duration of average 1, and therefore the departure rate from each bucket is proportional to the bucket size. In addition, when an element departs, a new element in inserted into the hash table (or in the overflow list if the corresponding bucket is full). As explained in Section II, the arrival rate to each bucket is therefore $ch = \lim_{n\to\infty} \frac{n}{m}$.

The following theorem, which is based on the M/M/h/h continuous-time Markov process [14], shows the performance of the scheme under the fluid model. (The full proof is in Appendix A-C).

Theorem 3: In the fluid model,
(i) the distribution of $p(t)$ converges to the stationary distribution π^{∞}, where

$$\pi_k^{\infty} = \frac{(ch)^k}{k!} \Big/ \sum_{l=0}^{h} \frac{(ch)^l}{l!}, \quad k = 0, \ldots, h. \tag{4}$$

(ii) the overflow fraction converges to π_h^{∞} and follows the Erlang-B formula.

We have seen that the discrete and continuous models with n elements yield a stationary distribution π^n, while the fluid model yields a fixed-point distribution π^{∞}. We will now show that as expected, when scaling n to infinity, π^n converges to π^{∞}, and so does the associated overflow fraction. (Proof in Appendix A-D.)

Corollary 4: When $n \to \infty$ with $\frac{n}{m} \to ch$,
(i) the stationary distribution converges to the fixed-point distribution of the fluid model: $\pi^n \to \pi^\infty$; and
(ii) the overflow fraction of the discrete (continous) model converges to the overflow fraction of the fluid model.

Finally, we generalize the scheme to deal with *probabilistic insertions*. Namely, there exists some $\alpha \in [0,1]$ such that each arriving element is either hashed into a bucket as before with probability α, or placed directly in the overflow list with probability $1-\alpha$, yielding an average number of memory accesses α (or equivalently, a total number of memory accesses $\alpha n \leq n$, less than the number of elements). Using the fluid model for simplicity, we obtain the following result. While this probabilistic scheme is probably not useful in practice (since the average memory access rate is seldom less than 1), we will later demonstrate that it is *optimal* under specific conditions.

Theorem 5: In the fluid model, given the single-choice hashing scheme with an insertion probability α, we obtain $a = \alpha \leq 1$, and
(i) the distribution of $p(t)$ converges to the stationary distribution π^∞, where

$$\pi_k^\infty = \frac{(\alpha ch)^k}{k!} \Big/ \sum_{\ell=0}^{h} \frac{(\alpha ch)^\ell}{\ell!}, \quad k = 0, \ldots, h. \tag{5}$$

(ii) the overflow fraction converges to $(1-\alpha) + \alpha \cdot \pi_h^\infty$.

Proof: The differential equations are the same as in the proof of Theorem 3 when replacing ch by αch, since α simply changes the arrival rate. The distribution results are then immediate. In addition, in the fixed-point equations, an arriving element either overflows immediately with probability $1-\alpha$, or checks with probability α a bucket that can be full with probability π_h^∞, hence the overflow equation follows as well. ∎

IV. Overflow Lower Bound

Our objective is to find a lower bound on the optimal expected limit overflow fraction γ_{OPT} in the OPTIMAL DYNAMIC HASH TABLE PROBLEM, and therefore on the expected overflow fraction γ of any $\langle a, d, c, h\rangle$ hashing scheme, when assuming a fluid model. We will study the simpler case with a single uniformly-distributed hash function, as defined in Section II. The more general case with several hash functions using different subtable-based distributions appears in Section V.

The proof relies on the following result from [15]. Consider an Erlang blocking model with N servers, and suppose that the arrival rate depends on the system. Let λ_k be the arrival rate when there are k transmissions in progress, $k = 0, 1, \ldots, N-1$. Then we have:

Lemma 1 (Theorem 4.2 in [15]): For all increasing mappings $f : \mathbb{R} \to \mathbb{R}$ and for all $t > 0$, $\mathbb{E}f(X)$ is concave increasing as a function of λ_k, for $k = 0, 1, \ldots, N-1$.

We use this lemma to prove the lower-bound result.

Theorem 6: In the fluid model, under the assumptions above where all buckets have the same probability of being hashed into, the optimal expected fixed-point overflow fraction γ_{OPT} in the OPTIMAL DYNAMIC HASH TABLE PROBLEM is lower-bounded by

$$\gamma_{\text{LB}}^\infty(a) = 1 - a + a \cdot \frac{r^h}{h!} \Big/ \sum_{l=0}^{h} \frac{r^l}{l!}, \tag{6}$$

where $r = ach$.
Note again that the Erlang-B formula appears in the lower-bound on the overflow. This yields the following optimality result:

Theorem 7: In the fluid model, the single-choice hashing scheme is optimal for every average number of memory accesses a in $[0,1]$ (and in particular for $a = 1$).

Proof: For the SINGLE scheme, there is a single hashed bucket per element, and it is accessed with probability α, therefore $a = \alpha$. For $a \leq 1$, we get

$$\gamma_{\text{LB}}^\infty(a) \overset{(a)}{=} (1-\alpha) + \alpha \cdot \frac{(\alpha ch)^h}{h!} \Big/ \sum_{l=0}^{h} \frac{(\alpha ch)^l}{l!} \overset{(b)}{=} \gamma_{\text{SINGLE}}^\infty$$

where (a) comes from Equation (6), $r = ach$ and $a = \alpha$, and (b) from Theorem 5. ∎

Example 1: We illustrate the significance of the lower bound by considering a simple system with buckets of size $h = 1$, implying $\gamma_{\text{LB}}^\infty(a) = 1 - a + a \cdot \frac{c \cdot a}{1 + c \cdot a} = 1 - \frac{a}{1 + c \cdot a}$.
In particular, for a load $c = 1$, corresponding to the scaling case where the number of buckets is kept equal to the number of elements and therefore $\lim_{n\to\infty} \frac{n}{m} = 1$, we get $\gamma_{\text{LB}}^\infty(a) = 1 - \frac{a}{1+a} = \frac{1}{1+a}$, which shows that the lower-bound decreases slowly as $\Theta(1/a)$ when the average number of memory accesses per insertion a increases.
For instance, to get a 1% drop rate we need each element to access an average of at least $a = 99$ buckets. Of course, this is impossible to implement in high-speed networking devices. Thus, this lower bound is essentially an *impossibility result*, which shows that it is not easy to obtain efficient hash tables with deletions.

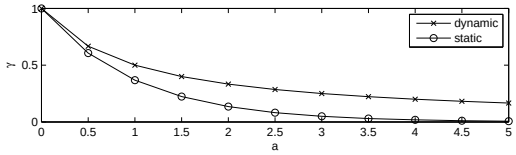


Fig. 3. Overflow fraction as a function of the average memory access rate a.

Fig. 3 compares this drop rate lower-bound with the drop rate lower-bound in the static case, which is equal to e^{-a} [11]. As a is increased, the figure shows how dynamic hash tables are significantly less efficient than their static counterparts.

V. Lower Bound with Multiple Hash-Function Distributions

We now consider a setting with a set $\mathcal{I}$ of $I = |\mathcal{I}|$ subtables, where subtable $i \in \mathcal{I}$ uses a fraction α^i of all buckets. We will allow for the d hash functions to use up to d different distributions $\{f_j\}_{1 \leq j \leq d}$ over the I subtables, where each distribution f_j assigns a probability f_j^i to subtable $i \in \mathcal{I}$, with $\sum_{i \in \mathcal{I}} f_j^i = 1$, and then uniformly picks buckets within

each subtable (as defined in Section II). We also assume that each distribution f_j is used by a fraction κ_j of the total memory accesses. Therefore, subtable i is accessed with a total probability of $\beta^i = \sum_{j=1}^{d} \kappa_j \cdot f_j^i$, with $\sum_{i \in \mathcal{I}} \beta^i = 1$. The following result establishes that the lower-bound is reached when the hash table is used in a uniform way, i.e. the probability β^i of accessing a subtable is equal to its fraction α^i in the table, and therefore the lower-bound is the same as established previously in Theorem 6.

Theorem 8: In the fluid model with multiple distributions as defined above, the lower-bound $\gamma_{\text{LB}}^{\infty}(a)$ on the fixed-point overflow fraction is the same as with a unique uniform hash function, and is reached iff for all $i \in [1, I]$, $\beta^i = \alpha^i$, i.e. the weighted average of all distributions is uniform.

VI. A Multiple-Choice Hashing Scheme

We now introduce a natural extension to the single-choice hashing scheme that uses 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. To keep an average number of memory accesses per element of at most a, the algorithm attempts to insert x into the hash table with a probability α, otherwise it is directly placed in the overflow list. A detailed example appears in Appendix B-B.

We evaluate the performance of this scheme analytically using the fluid model. (Proof in Appendix A-G).

Theorem 9: Assume the multiple-choice hashing scheme with a hashing probability α. Using the fluid-model fixed-point distribution π^{∞},

(i) π^{∞} satisfies $\pi_k^{\infty}(a) = \dfrac{\frac{(ach)^k}{k!}}{\sum_{l=0}^{h} \frac{(ach)^l}{l!}}$, for each $k = 0, \ldots, h$;

(ii) the average bucket access rate a satisfies the fixed-point equation $a = \alpha \cdot \frac{1-\pi_h^{\infty}(a)^d}{1-\pi_h^{\infty}(a)}$;

(iii) the overflow fraction is equal to the lower-bound, and is therefore *optimal*, for $a \in [0, a^{co}]$, where a^{co} satisfies the fixed-point equation $a^{co} = \frac{1-\pi_h^{\infty}(a^{co})^d}{1-\pi_h^{\infty}(a^{co})}$.

The following example illustrates our results.

Example 2: For the case where $h = 1$, solving the fixed-point equation yields $a^{co} = \frac{2c-1+\sqrt{1+4c^2}}{2c}$. Therefore, for a load of one element per bucket, i.e. $c = \lim_{n\to\infty} \frac{n}{m} = 1$, we get $a^{co} = \frac{1+\sqrt{5}}{2} \approx 1.62$, and the corresponding overflow fraction is $\gamma_{\text{LB}}^{\infty}(a^{co}) = 1.5 - \frac{\sqrt{5}}{2} \approx 38.2\%$. Likewise, for a load of $c = 0.1$, we get $a^{co} = \frac{-0.8+\sqrt{1+0.04}}{0.2} \approx 1.099$, with the corresponding overflow fraction $\gamma_{\text{LB}}^{\infty}(a^{co}) \approx 0.98\%$.

VII. Moving Back Elements

So far, we have found optimal schemes for a range of values of a, the average number of memory accesses per element. However, although optimal, the expected overflow fraction may still be too large.

In the literature, several solutions exist to reduce the drop rate (or collision probability) in a dynamic system. One such solution uses limited hash functions in order to be able to rebalance the hash table in case of deletion [16]. However, this approach gives up randomness, and the efficiency of a similar approach appears limited [6]. Another solution, based on the *second-chance* scheme [10], moves elements from one bucket to another by storing hints at each bucket [6]. However, we found in simulations that this solution was less effective than our suggested scheme presented below for higher loads, while it was more effective for lower loads. Detailed simulation results are found in Section VIII.

To reduce the overflow fraction, we suggest a scheme that allows moving elements back from the overflow list to the buckets upon a *deletion* operation[2]. This scheme can be combined with any insertion scheme.

A. Description

Our scheme, called the moving-back scheme (M-B), relies on a (binary) CAM. In general, a CAM stores keys in entries. Given some key k, a parallel lookup is performed over all entries and the index of the first (that is, highest priority) entry that contains k is returned from the CAM. In many cases, this index is later used in order to access in regular memory a direct-access array that contains the value associated with k. CAMs enable constant-time operations, however they are more expensive and consume more power than regular memory. It is a common practice to implement the overflow list in a CAM [1], [10], [11], relying on the fact that the number of elements in the overflow list is small.

Our scheme uses an auxiliary CAM, besides the primary CAM used to store the element of the overflow list: For each element x that is stored in the i-th entry of the primary CAM, we store the values $\{H_1(x), H_2(x), \ldots, H_d(x)\}$ in entries $d \cdot i, d \cdot i + 1, \ldots, d \cdot i + (d-1)$ of the auxiliary CAM.

When an element is deleted from a bucket j that was previously full, we need to move an element x from the overflow list to bucket j such that j is the result of applying at least one of the hash-functions on x. We can locate such an element in constant time by querying the auxiliary CAM with key j. Suppose the entry returned by the auxiliary CAM is ℓ, then x is located in entry $\lfloor \ell/d \rfloor$ of the primary CAM.

We note that upon moving an element back to the hash table, one should update the corresponding entries of the primary and auxiliary CAMs. An efficient way to update is to write the value $m+1$ in these entries, such that when a new element is inserted into the overflow list, one can query the auxiliary CAM with the value $m+1$ to decide in which entry (of the primary CAM) to put the new element.

B. Analysis

We first derive the exact overflow fraction in the case of the SINGLE scheme, and later provide an approximate model for the MULTIPLE scheme, which is confirmed by simulations.

[2]We also consider a scheme that works upon *insertion*, however the details are omitted due to lack of space; moving back elements upon deletion performs better in general.

Theorem 10: Consider the SINGLE scheme with M-B for moving back elements from the TCAM and a symmetric insertion algorithm. The overflow fraction is given by:

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

Proof: Whenever a deletion occurs, the CAM device performs a lookup operation for any element that can be moved back to the bucket. Since every element has only one hash value, all elements that are corresponding to some bucket can be viewed as its own pending list. Since the element we pick to delete follows a random process that is independent of any other random process in our system, and also the load is fixed, we get that the overflow fraction follows the static case exactly, which is given in [11]. ∎

Theorem 11: Consider the MULTIPLE scheme with M-B for moving back elements from the CAM and a symmetric insertion algorithm. Let X_t^i is the occupancy of bucket i at step t and $P_0, \ldots, P_h$ be the equilibrium probabilities of the occupancy of each buffer. The probabilities can be modeled by the following Markov chain:

$$\begin{aligned} P_{kj}^i &= \Pr(X_t = j | X_{t-1} = k) \\ &= \begin{cases} g \cdot \frac{1}{m} & j = k+1, k < h \\ \frac{k}{n} & j = k-1, h > k > 0 \\ \frac{k}{n} \cdot e^{-\frac{\gamma chd}{P_h}} & j = k-1, k = h \end{cases} \end{aligned}$$

where $g = \sum_{l=1}^{d} {P_h}^{l-1} = \frac{1-{P_h}^d}{1-P_h}$, and the overflow fraction γ is given by $\gamma = 1 - \frac{1}{ch} \cdot \sum_{i=0}^{h} i \cdot P_i$.

Proof: The Markov chain is the same as in the regular MULTIPLE scheme, except when an element is deleted from a full bucket. In this case, it is possible that one of the overflow elements in the CAM is moved back to the bucket. It is possible only in this case because all elements in the CAM have hashes to full buckets.

We now approximate the probability that none of the elements has an hash value to that bucket: The total number of hashes is $\gamma \cdot n \cdot d$, where all the hashes are to full buckets. The number of full buckets is $P_h \cdot m$. The probability that a single hash does not point to the specific bucket is $\frac{P_h \cdot m - 1}{P_h \cdot m} = 1 - \frac{1}{P_h \cdot m}$. And the probability that none of them points to the specific bucket is given by

$$\left(1 - \frac{1}{P_h \cdot m}\right)^{\gamma nd} \approx e^{-\frac{\gamma nd}{P_h \cdot m}} = e^{-\frac{\gamma chd}{P_h}}.$$

Multiplying the above expression by the probability that one of the elements is picked for deletions in case the bucket is full yields the claimed Markov chain. ∎

VIII. Experimental Results

A. Simulations

Fig. 4 compares all the schemes. It was obtained with $d = 4$ choices, bucket size $h = 4$, $n = 4{,}096$ elements and $m = 1{,}024$ buckets, yielding a load $c = 1$.

The solid line plots the overflow fraction lower-bound $\gamma_{\text{LB}}(a)$ from Theorem 6. Simulations show that the proposed M-B scheme beats the lower bound with an overflow fraction of 4.6%, emphasizing the strength of this architecture. Of course, the lower bound does not apply to this case, since it moves back elements from the CAM.

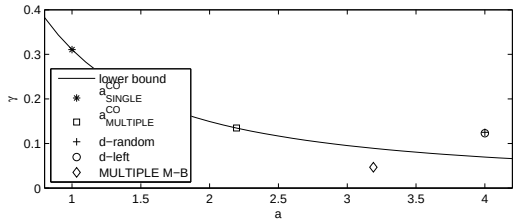


Fig. 4. Overflow fraction as a function of a with $d = 4$, $h = 4$, $c = 1$.

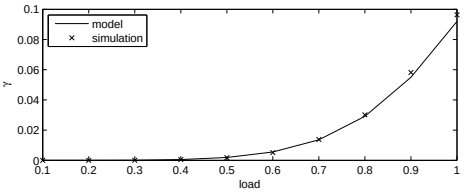


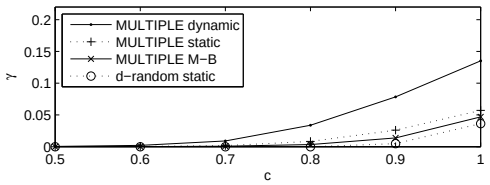
Fig. 5. M-B with MULTIPLE scheme, for $h = 4$, $d = 2$ and different loads

As follows from Theorems 7 and 9, the overflow fractions $\gamma_{\text{SINGLE}}(a)$ and $\gamma_{\text{MULTIPLE}}(a)$ of the single-choice (SINGLE) and the multiple-choice (MULTIPLE) hashing schemes follow the lower-bound line, respectively until $a^{co}_{\text{SINGLE}} = 1$ with $\gamma_{\text{SINGLE}} = 31.1\%$, and $a^{co}_{\text{MULTIPLE}} = 2.195$ with $\gamma_{\text{MULTIPLE}} = 13.5\%$. Therefore, they are clearly optimal up to a certain point.

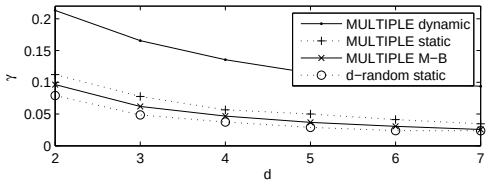
We also test our models from Section VII-B. Fig. 5 shows the accuracy of our M-B model. We ran simulations with $m = 1024$, $h = 4$, $d = 2$ and different loads. The maximum gap is for load $c = 1$ where our model predicts an overflow fraction of 9.20%, whereas simulations show an overflow fraction of 9.68%. For lower values of c, the model is much more accurate. For instance, for load $c = 0.5$, our model predicts an overflow fraction of 0.19% compared to an overflow fraction of 0.18% found via simulations.

We further evaluate the performance of our proposed M-B scheme. Quite surprisingly, when using the MULTIPLE scheme (of Section VI), the M-B scheme outperforms the *static* case of the MULTIPLE scheme (see Fig. 6), and performs similarly to the static d-random scheme (in the static case, d-random performs better than our multiple-choice scheme, albeit consuming significantly more energy [11]). This can be explained intuitively as follows: our moving-back strategy moves back an element to the only corresponding bucket which is not full; this is equivalent to inserting the element to the least occupied bucket as in the d-random hashing scheme.

Finally, we compare the performance of our proposed M-B scheme with the performance of the hint-based scheme proposed in [6]. Note that our M-B scheme can be used with any insertion scheme. Thus, for fair comparison, since the hint-based scheme uses the second-chance scheme [10] for insertions, we also used the second-chance scheme for our proposed M-B scheme. We ran simulations with $m = 4096$,

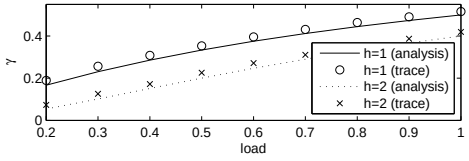


(a) $h = 4$ and $d = 4$

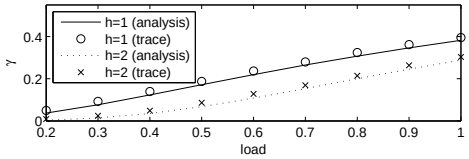


(b) $h = 4$ and $c = 1$

Fig. 6. Overflow fraction of the proposed moving-back (M-B) scheme (via simulations).



(a) SINGLE



(b) MULTIPLE

Fig. 7. Experiment using real-life traces and hash functions with SINGLE and MULTIPLE (d=2).

$h = 1$, $d = 4$ and different loads. As proposed in [6], the memory level sizes are exponentially decreasing with factor 2.

Fig. 8 shows that our M-B scheme is more effective than the hints-based scheme for higher loads, while it is less effective for lower loads. For instance, for a load of 0.6, the M-B and the hints-based schemes yield overflow fractions of 1.08% and 0.78%, respectively. For a load of 0.7, they yield 3.88% and 4.59%.

B. Experiments Using Real-Life Traces

We have also conducted experiments using real-life traces recorded on a single direction of an OC192 backbone link [17]. Our goal is to compare the average overflow fraction retrieved using our models for SINGLE and MULTIPLE with the corresponding overflow fraction when using a real hash function on a real-life trace. We used a 64-bit mix function [18] to implement two 16-bit hash functions. We used $m = 10{,}000$ buckets, and set a number of elements n as corresponding to various values of h and c. To keep a constant desired load, we alternated $100{,}000$ times between an arrival (insertion) of a new TCP packet according to the trace, and the departure (deletion) of a random TCP packet. The hash functions were given the source and destination IP tuple as well as the sequence and acknowledgment numbers of the TCP packets. Therefore, the hash table stores the latest TCP packets, and can retrieve any needed packet based on its header. It can be used to monitor ongoing TCP flows, given a target number n of packets that are stored at any time. Its objective in our experiments was mainly to test the correctness of our model.

Fig. 7 shows that the results of our experiments are relatively close to our model. The maximum gap is for the SINGLE scheme with $h = 1$ and $c = 0.3$. Our model predicts an overflow fraction of 23.08%, while the experiment yields 25.67%.

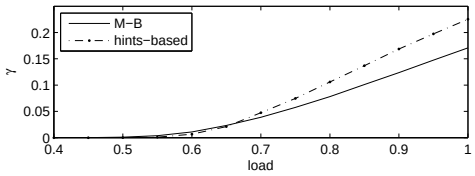


Fig. 8. The M-B and the hints-based schemes, for $h = 4$, $d = 1$ and different loads.

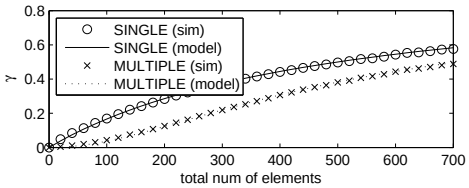


Fig. 9. Marginal overflow fraction of 100 on-off flows with $m = 500$, $h = 1$ and $d = 2$

C. Experiments Using an On-off Arrival Model

We also consider a queueing model where at each step i, b_i elements arrive according to k independent on-off bursty flows of elements [19]; then, after the arrival phase, one element is randomly deleted. Therefore, the number of elements in the system keeps changing, contrarily to the previous models with a constant load.

Fig. 9 shows the marginal overflow fraction under the above queueing model with $k = 100$ on-off flows of elements. Each flow has rate $\rho = 0.0095$ and average burst size of 10 elements. The figure shows that, given the number of elements currently in the system, the marginal overflow fraction is approximately the one we found for the constant-load case, both for SINGLE and MULTIPLE.

Moreover, by the distribution of the number of elements in the system given by the queueing model, we are able to heuristically approximate the overall expected number of elements in the overflow list. More precisely, we take the sum-product of the queue size distribution by the distribution of the overflow fraction as a function of the load. In the case of SINGLE this model gives an expected number of overflow elements of 61.63, while simulations yield 61.41. Likewise, for MULTIPLE, we obtain 40.17 and 40.26, respectively. Therefore, this heuristic model proves quite accurate.

IX. Conclusion

In this paper we demonstrated that when the memory is bounded, dynamic schemes behave significantly worse than their static counterparts. This decrease in performance is inherent to the problem, as shown by our lower bounds.

Moreover, we considered two hashing schemes that we proved to be optimal: a single-choice hashing scheme that was used to demonstrate our approach and techniques, and a multiple-choice scheme that inserts the elements greedily.

However, due to the slow decrease of the lower bound, optimality may be insufficient for certain applications. Therefore, we suggested moving back elements from the overflow list as soon as a deletion occurs. We have shown through simulations that this strategy beats the lower bound of the dynamic case (where moving back elements is not allowed).

We also conducted an extensive experimental study to verify the accuracy of our model, the behavior of the models under realistic (rather than fully-random) hash functions, and under variable-load arrival models.

Acknowledgment

This work was partly supported by the European Research Council Starting Grant n° 210389 and by the Legacy Heritage Fund program of the Israel Science Foundation (grant 11816/10).

References

[1] A. Kirsch, M. Mitzenmacher, and G. Varghese., *Hash-Based Techniques for High-Speed Packet Processing*. DIMACS, 2010, ch. 9.

[2] G. H. Gonnet, "Expected length of the longest probe sequence in hash code searching," *J. ACM*, vol. 28, no. 2, pp. 289–304, 1981.

[3] M. Mitzenmacher, A. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*, vol. 1, 2000, pp. 255–312.

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

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

[6] A. Kirsch and M. Mitzenmacher, "On the performance of multiple choice hash tables with moves on deletes and inserts," in *Allerton*, 2008, pp. 1284–1290.

[7] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More robust hashing: Cuckoo hashing with a stash," *SIAM J. on Computing*, vol. 39, no. 4, p. 1543, 2009.

[8] R. Kutzelnigg, "A further analysis of cuckoo hashing with a stash and random graphs of excess r," *Submitted for publication*, 2009.

[9] M. Drmota and R. Kutzelnigg, "A precise analysis of cuckoo hashing," *Submitted for publication*, 2009.

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

[11] Y. Kanizo, D. Hay, and I. Keslassy, "Optimal fast hashing," in *IEEE Infocom*, 2009, pp. 2500–2508.

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

[13] R. B. Cooper, *Introduction to Queueing Theory*, 2nd ed. North Holland, 1981.

[14] L. Kleinrock, *Queueing Systems*. Wiley, 1975.

[15] P. Nain, "Qualitative properties of the Erlang blocking model with heterogeneous user requirements," INRIA, Tech. Rep. 1018, April 1989.

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

[17] C. Shannon, E. Aben, K. claffy, and D. E. Andersen, "CAIDA Anonymized 2008 Internet Trace equinix-chicago 2008-03-19 19:00-20:00 UTC (DITL) (collection)," http://imdc.datcat.org/collection/.

[18] T. Wang, "Integer hash function," http://www.concentric.net/~Ttwang/tech/inthash.htm.

[19] A. Adas, "Traffic models in broadband networks," *IEEE Communications Magazine*, vol. 35, pp. 82–89, 1997.

[20] S. Asmussen, *Applied Probabilities and Queues*, 2nd ed. Springer, 2003.

[21] W. Grassmann, "The convexity of the mean queue size of the $M/M/c$ queue with respect to the traffic intensity," *Journal of Applied Probability*, vol. 20, pp. 916–919, 1983.

[22] K. R. Krishnan, "The convexity of loss rate in an Erlang loss system and sojourn in an Erlang delay system with respect to arrival and service rates," *IEEE Trans. Commun.*, vol. 38, no. 9, pp. 1314–1316, 1990.

[23] A. Harel, "Convexity properties of the Erlang loss formula," *Operations Research*, vol. 38, no. 3, pp. 499–505, 1990.

Appendix A
Proofs

A. Proof of Theorem 1

We model the hash table using a *discrete-time Markov chain* that represents the occupancy X_t^i of an arbitrary bucket i at the end of time-slot t. We will see that this is possible because the process is memoryless from time-slot to time-slot, and because when conditioned on the occupancy of bucket i, its arrival and departure probabilities are independent of the states of the other buckets or of the overflow list.

At the end of each time-slot $t-1$, there are X_{t-1}^i elements in bucket i. Then, at the start of time-slot t, the element that departs is chosen uniformly at random out of the n elements in the system. Therefore, the probability that it belongs to one of the X_{t-1}^i elements in bucket i is $\frac{X_{t-1}^i}{n}$.

The element is then reinserted into the system. The probability that it is hashed by the uniformly-distributed hash function H into bucket i out of m buckets is $\frac{1}{m}$.

We can now build the state transition matrix. The bucket occupancy obviously increases iff there is no departure while there is an arrival, while it decreases iff there is a departure but no arrival. For $1 \leq j,k \leq h$, the transition probability from occupancy j to occupancy k is

$$\begin{aligned} P_{jk}^i &= \Pr\left(X_t^i = k | X_{t-1}^i = j\right) \\ &= \begin{cases} \frac{1}{m}\cdot\left(1-\frac{j}{n}\right) & k = j+1, k \geq 1, \\ \frac{j}{n}\cdot\left(1-\frac{1}{m}\right) & k = j-1, k \leq h-1, \\ \left(1-\frac{j}{n}\right)\left(1-\frac{1}{m}\right)+\frac{j}{n}\frac{1}{m} & k = j. \end{cases} \end{aligned}$$

The birth-death Markov chain is clearly irreducible, positive, recurrent and aperiodic. Therefore, it converges to its stationary distribution π^n. In addition, since the state transition matrix does not depend on i, by ergodicity, the distribution of $p(t)$ converges to π^n as well. π^n satisfies:

$$\pi_k^n = \frac{\frac{1}{m}\cdot\left(1-\frac{j}{n}\right)}{\frac{j+1}{n}\cdot\left(1-\frac{1}{m}\right)}\cdot\pi_{k-1}^n, \quad k = 1,\ldots,h,$$

with

$$\sum_{k=0}^{h}\pi_k^n = 1.$$

For $0 \leq k \leq h$,

$$\begin{aligned}\pi_k^n &= \left(\prod_{j=0}^{k-1} \frac{\frac{1}{m}\cdot\left(1-\frac{j}{n}\right)}{\frac{j+1}{n}\cdot\left(1-\frac{1}{m}\right)}\right)\cdot\pi_0^n \\ &= \left(\prod_{j=0}^{k-1}\frac{n-j}{j+1}\right)\cdot\left(\frac{1}{m-1}\right)^k\cdot\pi_0^n \\ &= \binom{n}{k}\cdot\left(\frac{1}{m-1}\right)^k\cdot\pi_0^n = \frac{\binom{n}{k}\cdot\left(\frac{1}{m-1}\right)^k}{\sum_{l=0}^{h}\binom{n}{l}\cdot\left(\frac{1}{m-1}\right)^l},\end{aligned}$$

with $\binom{n}{l}=0$ when $l>n$ by convention.

Lastly, bucket i overflows in a given time-slot t when it contains h elements, no element leaves bucket i, and an element arrives to bucket i. In addition, the probability that the element arriving at time t is sent to the overflow list is the sum of all individual bucket overflow probabilities. Therefore, by ergodicity, the total overflow probability at time-slot t converges to

$$\gamma_{\text{SINGLE}}^n = m\cdot\left(\pi_h^n\cdot\left(1-\frac{h}{n}\right)\cdot\frac{1}{m}\right) = \pi_h^n\cdot\left(1-\frac{h}{n}\right).$$

Note that when assuming that elements are chosen uniformly at random at each time slot among all n elements to depart and be reinserted in the system, independently of the system state, then the probability for a given element to be in the overflow list converges to γ_{SINGLE}^n as well; and therefore, the expected size of the overflow list is $(\gamma_{\text{SINGLE}}^n\cdot n)$. ■

B. Proof of Theorem 2

As in the discrete model, we build a birth-death chain to model the occupancy X_t^i of an arbitrary bucket i at time t.

First, during any time interval $[t,t+\delta t]$, the probability that an element in bucket i of size X_t^i is chosen to depart is $X_t^i\cdot\delta t+o(\delta t)$, since each element stays for an exponentially-distributed amount of time of average 1 and there are X_t^i such elements in the bucket. However, since SINGLE uses a single uniformly-distributed hash function to hash elements into the buckets, a chosen element might be reinserted into the same bucket i with probability $\frac{1}{m}$. The element thus departs the bucket with probability $1-\frac{1}{m}$. Therefore, the departure probability from bucket i during time interval $[t,t+\delta t]$ is $X_t^i\cdot\left(1-\frac{1}{m}\right)\cdot\delta t+o(\delta t)$.

Likewise, during any time interval $[t,t+\delta t]$, an element leaves the other buckets with probability $\left(n-X_t^i\right)\cdot\delta t+o(\delta t)$, since they contain $n-X_t^i$ elements. Therefore, since the single-choice hashing scheme (SINGLE) uses a single uniformly-distributed hash function to hash elements into the buckets, the probability that an element is hashed into bucket i during time interval $[t,t+\delta t]$ is $\frac{n-X_t^i}{m}\cdot\delta t+o(\delta t)$.

Therefore, we obtain the following transition rate matrix for the continuous-time birth-death Markov process: for $1\leq j,k\leq h$, the transition rate Q_{jk}^i from occupancy j to occupancy k is

$$Q_{jk}^i=\begin{cases}\frac{n-j}{m} & k=j+1,k\geq 1,\\ j\cdot\left(1-\frac{1}{m}\right) & k=j-1,k\leq h-1,\\ -\left(\frac{n-j}{m}+j\cdot\left(1-\frac{1}{m}\right)\right) & k=j.\end{cases}$$

$\left\{X_t^i\right\}_{t\geq 0}$ is ergodic (e.g., from Corollary 2.5 in p. 74 of [20]), and therefore its distribution converges to the stationary distribution π^n. By ergodicity, the distribution of $p(t)$ converges to π^n as well. Furthermore, since the transition rates from any j to any $k\neq j$ in the continuous-time Markov process are a scaled version of the transition probabilities in the discrete-time Markov chain by a scaling factor n, the stationary distribution π^n is the same as in the discrete model.

Finally, since we assume instantaneous recirculation of departing elements, a recirculated element is lost when it is hashed into a bucket different from the one it just left, and this bucket is full. In particular, during time interval $[t,t+\delta t]$, the probability that an element is recirculated to a full bucket i of size h from another bucket is $\left(\frac{n-h}{m}\Pr\left(X_t^i=h\right)\right)\cdot\delta t+o(\delta t)$, since there are $n-h$ elements in the other buckets, each recirculating at rate 1 and then hashed into bucket i with probability $\frac{1}{m}$. Since the n elements are recirculated at a total rate of n, and in particular are recirculated into bucket i at rate $\frac{n}{m}$, the fraction of all such elements that overflow converges to

$$\frac{\frac{n-h}{m}\cdot\pi_h^n}{\frac{n}{m}}=\gamma_{\text{SINGLE}}^n,$$

which concludes the proof. ■

C. Proof of Theorem 3

In the fluid model, the departures from buckets of size $k\geq 1$ cause the fraction $p_k(t)$ to decrease at rate $k\cdot p_k(t)$, since each of the k elements departs at rate 1, and therefore the k elements depart at a total rate of k. Since the buckets of size k with a departing element have a new size $k-1$, the departures from such buckets increase in turn $p_{k-1}(t)$ at the same rate $k\cdot p_k(t)$.

Likewise, the arrivals to buckets of size $k<h$, which occur at rate $ch=\lim_{n\to\infty}\frac{n}{m}$, cause the fraction $p_k(t)$ to decrease at rate $ch\cdot p_k(t)$, and the fraction $p_{k+1}(t)$ to increase at the same rate.

Therefore, we obtain the following differential equation which characterizes the birth-death process:

$$\frac{dp_k(t)}{dt}=\begin{cases}ch\cdot p_{k-1}(t)+(k+1)p_{k+1}(t) & \\ \qquad -(ch+k)p_k(t) & \text{for } k\in[1,h-1],\\ p_1(t)-ch\cdot p_0(t) & \text{for } k=0,\\ ch\cdot p_{h-1}(t)-hp_h(t) & \text{for } k=h,\end{cases}$$

with $\sum_{k=0}^h p_k=1$. Assume the system is initially empty, i.e. $p_k(0)=\mathbf{1}_{k=0}$.

Consider a fixed point π^∞ of the birth-death process, i.e. assume that for any $k\in[0,h]$, $\frac{d\pi_k^\infty(t)}{dt}=0$. Solving the differential equation above yields the following balance equations: for $k\in[1,h]$,

$$ch\cdot\pi_{k-1}^\infty=k\cdot\pi_k^\infty,$$

with $\sum_{k=0}^{h} \pi_k^\infty = 1$. Therefore,

$$\pi_k^\infty = \frac{(ch)^k}{k!} \cdot \pi_0^\infty = \frac{\frac{(ch)^k}{k!}}{\sum_{l=0}^{h} \frac{(ch)^l}{l!}},$$

corresponding to the stationary distribution in the $M/M/h/h$ loss system [14]. In addition, the drop rate in the fluid model is

$$\gamma_{\text{SINGLE}}^\infty = \pi_h^\infty,$$

following the well-known *Erlang-B formula*. Finally, since the differential equations are exactly those of the ergodic $M/M/h/h$ continuous-time Markov process, $p(t)$ converges to π^∞ [14], [20]. ■

D. Proof of Corollary 4

For each $n \in \mathbb{N}^* \cup \{\infty\}$, $\sum_{k=0}^{h} \pi_k^n = 1$ and $\pi_0^n > 0$, so for $k \in [0, h]$, π_k^n/π_0^n is defined and

$$\pi_k^n = \frac{\pi_k^n/\pi_0^n}{\sum_{l=0}^{h} \pi_l^n/\pi_0^n}.$$

Therefore, to prove the convergence of $\{\pi^n\}_{n\geq 1}$, which is a sequence of finite vectors, we only need to prove the point convergence of π_k^n/π_0^n to $\pi_k^\infty/\pi_0^\infty$. We get

$$\begin{aligned}
\pi_k^n/\pi_0^n &= \binom{n}{k}\left(\frac{1}{m-1}\right)^k \\
&= \frac{1}{k!} \cdot \left(\frac{n}{m}\right)^k \cdot \frac{(1) \cdot \left(1-\frac{1}{n}\right) \cdot \cdots \cdot \left(1-\frac{k-1}{n}\right)}{\left(1-\frac{1}{m}\right)^k} \\
&= \frac{1}{k!} \cdot (ch)^k \cdot (1+o(1)) = \pi_k^\infty/\pi_0^\infty \cdot (1+o(1)),
\end{aligned}$$

which concludes the proof of the convergence of π^n to π^∞.

Lastly, $\gamma_{\text{SINGLE}}^\infty = \pi_h^\infty$ and $\gamma_{\text{SINGLE}}^n = \pi_h^n \cdot \left(1-\frac{h}{n}\right)$. Since $\left(1-\frac{h}{n}\right) = 1 + o(1)$, the convergence of γ_{SINGLE}^n to $\gamma_{\text{SINGLE}}^\infty$ follows. ■

E. Proof of Theorem 6

For any $k \in [1, h]$, let $p_k(t)$ denote the fraction of buckets of size k. As shown with the single-choice hashing scheme (SINGLE), in the fluid model, the departures from buckets of size $k \geq 1$ decrease the fraction $p_k(t)$ at rate $k \cdot p_k(t)$, and increase the fraction $p_{k-1}(t)$ at the same rate $k \cdot p_k(t)$.

Likewise, the element arrival rate before hashing is $ch = \lim_{n\to\infty} \frac{n}{m}$, and the hashing rate per element is a, therefore the hashing rate is ach. Let $r = ach$. Since elements can only decide to enter a bucket after hashing into it, we know that their post-hashing arrival rate to any bucket is bounded from above by r. Of course, the decision of whether to enter a bucket after hashing into it might depend on the bucket occupancy. Therefore, let $r_k(t)$ denote the average arrival rate to the fraction of buckets of size k at time t, with $0 \leq r_k(t) \leq r$. These arrivals cause the fraction $p_k(t)$ to decrease at rate $r_k(t) \cdot p_k(t)$, and the fraction $p_{k+1}(t)$ to increase at the same rate.

Combining departures and arrivals, we obtain the following differential equation characterizing the birth-death process:

$$\frac{dp_k(t)}{dt} = \begin{cases} r_{k-1}(t)p_{k-1}(t) + (k+1)p_{k+1}(t) & \\ \qquad -(r_k(t)+k)p_k(t) & \text{for } k \in [1, h-1], \\ p_1(t) - r_0(t)p_0(t) & \text{for } k = 0, \\ r_{h-1}(t)p_{h-1}(t) - hp_h(t) & \text{for } k = h, \end{cases}$$

with $\sum_{k=0}^{h} p_k = 1$ and $p_k(0) = \mathbf{1}_{k=0}$.

Consider a fixed point π of the birth-death process, i.e. assume that for any $i \in [0, h]$, $\frac{d\pi_k(t)}{dt} = 0$. Then the finite vector $(\pi_0(t), \ldots, \pi_h(t))$ is independent of t, and therefore the arrival rate $r_k(t)$ to a bucket of size k is independent of t as well. Denote this constant arrival rate as r_k, where $0 \leq r_k \leq r$. Solving the differential equation above yields the following balance equations: for each $k \in [1, h]$,

$$r_{k-1}\pi_{k-1} = k\pi_k, \tag{7}$$

with $\sum_{k=0}^{h} \pi_k = 1$. Therefore, π satisfies

$$\pi_k = \frac{\prod_{j=0}^{k-1} r_j}{k!} \cdot \pi_0 = \frac{\frac{\prod_{j=0}^{k-1} r_j}{k!}}{\sum_{l=0}^{h} \frac{\prod_{j=0}^{l-1} r_j}{l!}}.$$

Using Lemma 1, we find that the average bucket occupancy $\mathbb{E}X^\pi$ under π is upper-bounded by the average bucket occupancy $\mathbb{E}X^{\bar{\pi}}$ under $\bar{\pi}$, where $\bar{\pi}$ is the fixed-point distribution when $r_k = r$. This is because $f : X \to X$ is increasing [20], and $r_k \leq r$ for $k \in [0, h-1]$. Therefore,

$$\mathbb{E}X^\pi \leq \mathbb{E}X^{\bar{\pi}}. \tag{8}$$

Finally, note that without element losses, we would have the average occupancy equal to the average number of elements per bucket, i.e., $ch = \lim_{n\to\infty} \frac{n}{m}$. Therefore, the average fraction of lost elements is equal to

$$\begin{aligned}
\gamma^\infty &= \frac{ch - \mathbb{E}X^\pi}{ch} \overset{(a)}{=} 1 - a \cdot \frac{\mathbb{E}X^\pi}{r} \overset{(b)}{\geq} 1 - a \cdot \frac{\mathbb{E}X^{\bar{\pi}}}{r} \\
&\overset{(c)}{=} 1 - a \cdot \frac{r \cdot (1-\bar{\pi}_h)}{r} \overset{(d)}{=} 1 - a + a \cdot \frac{\frac{r^h}{h!}}{\sum_{l=0}^{h} \frac{r^l}{l!}},
\end{aligned}$$

where (a) uses $r = ach$, (b) relies on Equation (8), (c) uses a standard Erlang-B result [13], [14], and (d) comes from Equation (8) with $r_k = r$. ■

F. Proof of Theorem 8

As in the proof of Theorem 6, in each subtable i, we focus on the fixed-point distribution π^i, which satisfies

$$\pi_k^i = \frac{\frac{\prod_{j=0}^{k-1} r_j^i}{k!}}{\sum_{l=0}^{h} \frac{\prod_{j=0}^{l-1} r_j^i}{l!}},$$

with $r_j^i \leq \frac{\beta^i}{\alpha^i} \cdot r$. This is because the rate at which the elements check a bucket in subtable $i \in \mathcal{I}$ is proportional to the ratio of their probability β^i of picking subtable i by the proportional size α_i of subtable i. In addition, even if the elements check

a bucket of size j, they can decide not to enter it. The rate r_j^i at which they enter it depends both on the size j and on the subtable i, although it needs to be upper-bounded by the rate $\frac{\beta^i}{\alpha^i} \cdot r$ at which they checked it.

From the proof of Theorem 6, in each subtable $i \in \mathcal{I}$, we know that the average occupancy is upper-bounded by the case in which we have equality

$$r_j^i = \frac{\beta^i}{\alpha^i} \cdot r.$$

We now want to find the vector β that maximizes the average occupancy of the whole system. Let $\bar{\pi}\left(\frac{\beta^i}{\alpha^i} r\right)$ denote the distribution that maximizes the average occupancy in subtable $i \in \mathcal{I}$, and define $f : \mathbb{R}^+ \to \mathbb{R}^+$ with $f(x) = \mathbb{E}X^{\bar{\pi}(x)}$. Then we want to find

$$\max \sum_{i \in \mathcal{I}} \alpha^i \cdot f\left(\frac{\beta^i}{\alpha^i} r\right) \qquad \text{s.t.} \quad \sum_{i \in \mathcal{I}} \alpha^i = 1, \quad \sum_{i \in \mathcal{I}} \beta^i = 1.$$

f is known to be strictly concave [21]–[23] (the concavity also follows from Lemma 1). Therefore

$$\sum_{i \in \mathcal{I}} \alpha^i \cdot f\left(\frac{\beta^i}{\alpha^i} r\right) \overset{(a)}{\leq} f\left(\sum_{i \in \mathcal{I}} \cdot \alpha^i \cdot \frac{\beta^i}{\alpha^i} r\right) = f\left(r \cdot \sum_{i \in \mathcal{I}} \beta^i\right) \overset{(b)}{=} f(r),$$

where (a) uses concavity and $\sum_{i \in \mathcal{I}} \alpha^i = 1$, and (b) uses $\sum_{i \in \mathcal{I}} \beta^i = 1$, with equality iff $\frac{\beta^i}{\alpha^i} r$ independent of i, i.e. $\beta^i = \alpha^i$ for all $i \in \mathcal{I}$, because $\sum_{i \in \mathcal{I}} \beta^i = \sum_{i \in \mathcal{I}} \alpha^i$. Finally, as in the proof of Theorem 6, the same upper bound on the average bucket occupancy corresponds to the same lower bound on the overflow fraction. ■

G. Proof of Theorem 9

For a given rate a, the differential equations are the same as for the single-choice hashing scheme (SINGLE) and satisfy the same fixed-point distribution (Theorem 7).

Let us now compute a. Whenever an element arrives, the probability that it uses its l^{th} hash function H_l, for $1 \leq l \leq d$, is $\alpha \cdot (\pi_h^\infty)^{l-1}$; namely, the product of the probability α that it is not directly placed in the overflow list by the probability that the first $l-1$ hash functions mapped into full buckets. Then, the l^{th} trial is successful with probability $1 - \pi_h^\infty$. Finally, there were d unsuccessful trials with probability $\alpha \cdot (\pi_h^\infty)^{d-1}$. Therefore, the average number of trials per element is:

$$a = \left(\sum_{l=1}^{d-1} l \cdot \alpha \cdot (\pi_h^\infty)^{l-1} (1 - \pi_h^\infty)\right) + d \cdot \alpha \cdot (\pi_h^\infty)^{d-1}$$

Using the general formula

$$\sum_{k=1}^{K} k x^{k-1} = \frac{1 - x^{K+1} - (1-x) \cdot (K+1) x^K}{(1-x)^2},$$

we get

$$\begin{aligned} a &= \alpha \left[\frac{1 - (\pi_h^\infty)^d - (1 - \pi_h^\infty) \cdot d \cdot (\pi_h^\infty)^{d-1}}{1 - \pi_h^\infty} + d \cdot (\pi_h^\infty)^{d-1}\right] \\ &= \alpha \cdot \frac{1 - (\pi_h^\infty)^d}{1 - \pi_h^\infty}. \end{aligned}$$

Finally, this can only hold for $\alpha \leq 1$; once we reach $\alpha = 1$, we obtain a_{MULTIPLE}^{co}. ■

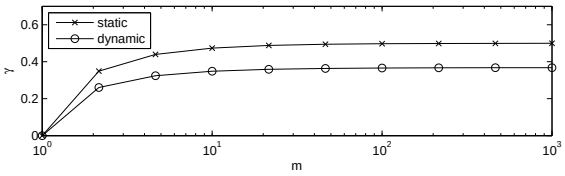


Fig. 10. Overflow fraction as a function of the number of buckets m.

APPENDIX B
EXAMPLES

A. Single-Choice: Dynamic vs. Static

For the case where $h = 1$ the overflow fraction γ_{SINGLE}^n reduces to $\gamma_{\text{SINGLE}}^n = \frac{n-1}{m+n-1}$. Denoting the load $c = \frac{n}{m}$, $\gamma_{\text{SINGLE}}^n = \frac{c - \frac{1}{m}}{1 + c - \frac{1}{m}} \xrightarrow{m \to \infty} \gamma_{\text{SINGLE}} = \frac{c}{1+c}$, where γ_{SINGLE} is the limit overflow fraction as we scale the system while keeping the load constant to c. For instance, for $c = 1$, we get

$$\gamma_{\text{SINGLE}} = 50\%. \tag{9}$$

In other words, when scaling the system with the same number of elements and buckets, we find that we asymptotically lose 50% of the elements.

Note that in such a scaling, we lose a fraction $\gamma_{\text{SINGLE}}^n = \frac{1 - \frac{1}{m}}{2 - \frac{1}{m}} = \frac{m-1}{2m-1}$ of the elements. This fraction corresponds for instance to no losses with $m = 1$; to $1/3$ of the elements lost with $m = 2$; and to 40% of the elements lost with $m = 3$; the overflow fraction then continuing to increase monotonically and converge to γ_{SINGLE}.

Now we compare the *dynamic* overflow fraction γ_{SINGLE}^n with the *static* overflow fraction, denoted σ_{SINGLE}^n, given a bucket size of 1. First, assuming a load $c = 1$, the overflow fraction is equal to the fraction of unused buckets, because the number of elements is equal to the number of buckets, and buckets can contain at most one element. Therefore, since each element chooses a bucket uniformly at random, we get

$$\sigma_{\text{SINGLE}}^n = \left(1 - \frac{1}{m}\right)^m \xrightarrow{m \to \infty} \sigma_{\text{SINGLE}} = e^{-1}.$$

Thus, the static system has a clearly lower overflow fraction.

Fig. 10 illustrates the overflow fraction as a fraction of m in both the static and dynamic cases. The dynamic case clearly always causes a higher overflow fraction. In addition, both converge fast from below to their limit values.

More generally, for any arbitrary load $c \leq 1$, the limit static overflow fraction is known [11] to be $\sigma_{\text{SINGLE}} = 1 - \frac{1 - e^{-c}}{c}$. Therefore, as $c \to 0$, we asymptotically get $\gamma_{\text{SINGLE}} = c + O(c^2)$, and $\sigma_{\text{SINGLE}} = \frac{c}{2} + O(c^2)$, so for low loads, the large dynamic system has about twice the overflow of the large static one.

B. Illustration of the MULTIPLE Scheme

Fig. 11 illustrates the multiple-choice hashing scheme (MULTIPLE) with $m = 12$, $h = 1$, $d = 2$ and $q = 1$. We can see that element x_1 is initially mapped by H_1 to a full bucket. It is therefore mapped again by H_2, and inserted in an empty bucket. Then, the element in bucket number 6 is

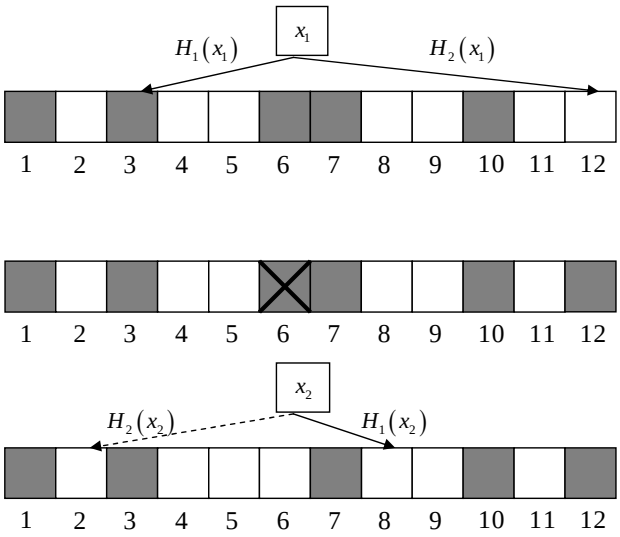


Fig. 11. Illustration of the MULTIPLE scheme

deleted. On the next step, element x_2 is directly inserted in an empty bucket, and therefore does not need a second memory access.