Efficient Randomized Web-Cache Replacement Schemes Using Samples from Past Eviction-Times *

Konstantinos Psounis[†], Balaji Prabhakar[‡]
[†]Department of Electrical Engineering
[‡]Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305
kpsounis@leland.stanford.edu, balaji@isl.stanford.edu

September 6th 2001

Abstract

The problem of document replacement in web caches has received much attention in recent research, and it has been shown that the eviction rule "replace the least recently used document" performs poorly in web caches. Instead, it has been shown that using a combination of several criteria, such as the recentness and frequency of use, the size, and the cost of fetching a document, leads to a sizeable improvement in hit rate and latency reduction. However, in order to implement these novel schemes, one needs to maintain complicated data structures. We propose randomized algorithms for approximating any existing web-cache replacement scheme and thereby avoid the need for any data structures.

At document-replacement times, the randomized algorithm samples N documents from the cache and replaces the least useful document from the sample, where usefulness is determined according to the criteria mentioned above. The next $M < N$ least useful documents are retained for the succeeding iteration. When the next replacement is to be performed, the algorithm obtains $N - M$ new samples from the cache, and replaces the least useful document from the $N - M$ new samples and the M previously retained. Using theory and simulations, we analyze the algorithm and find that it matches the performance of existing document replacement schemes for values of N and M as low as 8 and 2 respectively. Interestingly, we find that retaining a small number of samples from one iteration to the next leads to an exponential improvement in performance as compared to retaining no samples at all.

*This research is supported in part by a Stanford Graduate Fellowship, a Terman Fellowship, and an Alfred P. Sloan Foundation Fellowship. A preliminary version of this paper appears in the proceedings of INFOCOM '01 [12].

1 Introduction

The enormous popularity of the World Wide Web in recent years has caused a tremendous increase in network traffic due to HTTP requests. Since the majority of web documents are static, caching them at various network points provides a natural way of reducing traffic. At the same time, caching reduces download latency and the load on web servers.

A key component of a cache is its replacement policy, which is a decision rule for evicting a page currently in the cache to make room for a new page. A particularly popular rule for page replacement replaces the least recently used (LRU) page. This is due to a number of reasons: As an online algorithm it is known to have the best competitive ratio[1], it only requires a linked list to be efficiently implemented as opposed to more complicated data structures required for other schemes, and takes advantage of temporal correlations in the request sequence.

Suppose that we associate with any replacement scheme a *utility function*, which sorts pages according to their suitability for eviction. For example, the utility function for LRU assigns to each page a value which is the time since the page's last use. The replacement scheme would then replace that page which is most suitable for eviction.

Whereas for processor caches LRU and its variants have worked very well [16], it has recently been found [4] that LRU is not suitable for web caches. This is because some important differences distinguish a web cache from a processor cache: (i) The size of web documents are not the same, (ii) the cost of fetching different documents varies significantly, and (iii) sort term temporal correlations in web request sequences are not as strong. Thus, a utility function that takes into account not only the recency of use of a web document, but also its size, cost of fetching, and frequency of use can be expected to perform significantly better. Recent work proposes many new cache replacement schemes that exploit this point (e.g. LRU-Threshold[1], GD-Size[4], GD*[7], LRV[13], SIZE[18], Hybrid[19]).

However, the data structures that are needed for implementing these new utility functions turn out to be complicated. Most of them require a priority queue in order to reduce the time to find a replacement from $O(K)$ to $O(\log K)$, where K is the number of documents in the cache. Further, these data structures need to be *constantly updated* (i.e., even when there is no eviction), although they are solely used for eviction.

This prompts us to consider randomized algorithms which do not need any data structures to support the eviction decisions. For example, the particularly simple Random Replacement (RR) algorithm evicts a document drawn at random from the cache [9]. However, as might be expected, the RR algorithm does not perform very well.

[1] That is, there is no other deterministic online algorithm with a smaller competitive ratio [9].

We propose to combine the benefits of both the utility function based schemes and the RR scheme. Thus, consider a scheme which draws N documents from the cache and evicts the least useful document in the sample, where the "usefulness" of a document is determined by the utility function. Although this basic scheme performs better than RR for small values of N, we find a big improvement in performance by refining it as follows: After replacing the least useful of N samples, the identity of the next $M < N$ least useful documents is retained in memory. At the next eviction time, $N - M$ new samples are drawn from the cache and the least useful of these $N - M$ and M previously retained is evicted, the identity of the M least useful of the remaining is stored in memory, and so on.

Intuitively, the performance of an algorithm that works on a few randomly chosen samples depends on the quality of the samples. Therefore, by deliberately tilting the distribution of the samples towards the good side, which is precisely what the refinement achieves, one expects an improvement in performance. Rather surprisingly, we find that the performance improvement can be *exponential* for small values of M (e.g. 1, 2 or 3). As the value of M increases one expects a degradation in performance because bad samples are being retained and not enough new samples are being chosen. This suggests there is an optimal value of M. We analytically demonstrate the above observations and obtain an approximate formula for the optimal value of M as a function of N.

The rest of the paper is organized to reflect the three main parts of the paper: (i) a description of the proposed randomized algorithm for document replacement (Section 2), (ii) an analysis of its general features (Sections 3 and 4), and (iii) a simulation comparison, using web traces, of its performance relative to the deterministic algorithms it approximates (Section 5). Finally, Section 6 discusses implementation issues and further motivates the use of the randomized algorithm in practice, and Section 7 concludes the paper.

We conclude the introduction with a few remarks about the theoretical contributions of the paper. A preliminary version of the paper, presented at INFOCOM '01 [12], contains many of the algorithmic ideas and theoretical statements presented here. The present paper contains the complete details of the analytical steps (e.g. complete proofs of theoretical statements). The main algorithmic contribution of the paper is the demonstration that carrying information between iterations will greatly improve the performance of iterative randomized algorithms. While this has been applied to web-cache replacement policies in this paper, it is equally applicable to other iterative randomized algorithms of interest in networking (e.g. load balancing [15], switch scheduling [17], [14]). Thus, the theoretical methods used here may have wider applicability. In particular, the coupling method used to establish that there is a right amount of information to carry between iterations (see Section 3.1 and the Appendix), and then approximately determining this right amount of information using an exponential martingale argument (Section 4) seem to be of interest in their own right. Finally, other additions to [12] are more extensive

simulations using weekly NLANR traces [21], and a section devoted to implementation issues and to the practical motivation for this work.

2 The Algorithm

The first time a document is to be evicted, N samples are drawn at random from the cache and the least useful of these is evicted. Then, the next $M < N$ least useful documents are retained for the next iteration. And when the next replacement is to be performed, the algorithm obtains $N - M$ new samples from the cache, and replaces the least useful document from the $N - M$ new samples and the M previously retained. This procedure is repeated whenever a document needs to be evicted. In pseudo-code the algorithm is shown in Figure 1.

```
if (eviction) {
   if (first_iteration) {
      sample(N);
      evict_least_useful;
      keep_least_useful(M);
   } else {
      sample(N-M);
      evict_least_useful;
      keep_least_useful(M);
   }
}
```

Figure 1: The randomized algorithm.

An error is said to have occurred if the evicted document *does not* belong to the least useful n^{th} percentile of all the documents in the cache, for some desirable values of n. Thus, the goal of the algorithm we consider is to minimize the probability of error. We shall say that a document is *useless* if it belongs to the least useful n^{th} percentile[2].

It is interesting to conduct a quick analysis of the algorithm described above in the case where $M = 0$ so as to have a benchmark for comparison. Accordingly, suppose that all the documents are divided into $100/n$ bins according to usefulness and N documents are sampled uniformly and independently from the cache. Then the probability of error equals $(1 - n/100)^N$,[3] which approximately equals $e^{-nN/100}$. By increasing N this

[2] Note that samples that are good eviction candidates will be called "useless" samples since they are useless for the cache.

[3] Although the algorithm samples without replacement, the values of N are so small compared to the

probability can be made to approach 0 exponentially fast. (For example, when $n = 8$ and $N = 30$, the probability of error is approximately 0.08. By increasing N to 60, the probability of error can be made as low as 0.0067.)

But it is possible to do much better without doubling N! That is, even with $N = 30$, by choosing $M = 9$, the probability of error can be brought down to 2.4×10^{-6}. In the next few sections we obtain models to further understand the effect of M on performance.

We end this section with the following remark. Whereas it is possible for a document whose id is retained in memory to be accessed between iterations, making it a "recently used document", we find that in practice the odds of this happening are negligibly small[4]. Hence, in all our analysis, we shall assume that documents which are retained in memory are not accessed between iterations.

3 The Model and Preliminary Analysis

In this section we derive and solve a model that describes the behavior of the algorithm precisely. We are interested in computing the probability of error, which is the probability that none of the N documents in the sample is useless for the cache, for any given N and n and for all $M (0 \leq M < N)$.

We proceed by introducing some helpful notation. Of the M samples retained at the end of the $(m-1)^{th}$ iteration, let Y_{m-1} $(0 \leq Y_{m-1} \leq M)$ be the number of useless documents. At the beginning of the m^{th} iteration, the algorithm chooses $N - M$ fresh samples. Let A_m, $0 \leq A_m \leq N - M$ be the number of useless documents coming from the $N - M$ fresh samples. In the m^{th} iteration, the algorithm replaces one document out of the total $Y_{m-1} + A_m$ available (so long as $Y_{m-1} + A_m > 0$) and retains M documents for the next iteration. Note that it is possible for the algorithm to discard some useless documents because of the memory limit of M that we have imposed.

Define $X_m = \min(M+1, Y_{m-1} + A_m)$ to be precisely the number of useless documents in the sample just prior to the m^{th} document replacement, that the algorithm would ever replace at eviction times. If $X_m = 0$, then the algorithm commits an error at the m^{th} eviction. It is easy to see that X_m is a Markov chain and satisfies the recursion

$$X_m = \min(M + 1, X_{m-1} - 1_{(X_{m-1}>0)} + A_m),$$

and that A_m is binomially distributed with parameters $N - M$ and $n/100$. For a fixed N and n, let $p_k(M) = P(A_m = k)$, $k = 0, \ldots, N - M$, denote the probability that k useless documents for the cache, and thus good eviction candidates, are acquired during

overall size of the cache that $(1 - n/100)^N$ almost exactly equals the probability of error.

[4] Trace driven simulations in Section 5 support our observation.

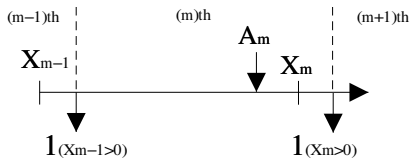

Figure 2: Sequence of events per iteration. Note that eviction takes place prior to resampling.

a sampling. When it is clear from the context we will abbreviate $p_k(M)$ to p_k. Figure 2 is a schematic of the above Markov chain.

Let T_M denote the transition matrix of the chain X_m for a given value of M. The form of the matrix depends on whether M is smaller or larger than $N/2$. Since we are interested in small values of M, we shall suppose that $M \leq N/2$ [5]. It is immediate that T_M is irreducible and has the general form

$$T_M = \begin{pmatrix} p_0 & p_1 & p_2 & \dots & p_M & 1-\sum_{i=0}^{M} p_i \\ p_0 & p_1 & p_2 & \dots & p_M & 1-\sum_{i=0}^{M} p_i \\ 0 & p_0 & p_1 & \dots & p_{M-1} & 1-\sum_{i=0}^{M-1} p_i \\ 0 & 0 & p_0 & \dots & p_{M-2} & 1-\sum_{i=0}^{M-2} p_i \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & p_0 & 1-p_0 \end{pmatrix}$$

As may be inferred from the transition matrix, the Markov chain models a system with one deterministic server, binomial arrivals, and a finite queue size equal to M (the system's overall size is $M+1$). An interesting feature of the system is that as M increases, the average arrival rate, $E(A_m) = (N-M)n/100$, decreases linearly and the maximum queue size increases linearly.

Let $\pi = (\pi_0, \dots, \pi_{M+1})$ denote the stationary distribution of the chain X_m. Clearly π_0 is the probability of error as defined above. Let $A = (a_{ij})$ be an $(M+2) \times (M+2)$ matrix, with $a_{ij} = 1$ for all i, j. Let $a = (a_i)$ be a $1 \times (M+2)$ matrix with $a_i = 1$ for all i. Since T_M is irreducible, $I - T_M + A$ is invertible [10] and

$$\pi = a \cdot (I - T_M + A)^{-1}. \tag{1}$$

Figure 3 shows a collection of plots of π_0 versus M for different values of N and n. The minimum value of π_0 is written on top of each figure. We note that given N and n there

[5] Figure 3 suggests that the M at which the probability of error is minimized is less than $N/2$.

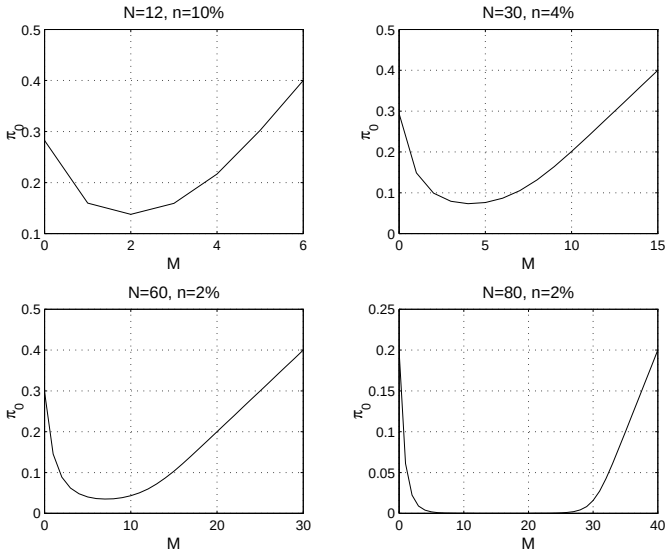

Figure 3: Probability of error (π_0=probability not a useless document for the cache is replaced) versus number of documents retained (M).

are values of $M > 0$ for which the error probability is very small compared to its value at $M = 0$. We also observe that there is no need for N to be a lot bigger than the number of bins $100/n$ for the probability of error to be as close to zero as desired, since even for $N = 2 \cdot 100/n$ the minimum probability of error is extremely small. Finally, we notice that for small values of M there is a huge reduction in the error probability and that the minimum is achieved for a small M. As M increases further the performance deteriorates linearly.

The exponential improvement for small M can be intuitively explained as follows. For concreteness, suppose that $M = 1$ and that the Markov chain X_m has been running from time $-\infty$ onwards (hence it is in equilibrium at any time $m \geq 0$). The relationship $\{X_m = 0\} \subset \{A_m = 0; A_{m-1} \leq 1\}$ immediately gives that $P(X_m = 0) \leq P(A_m = 0)[P(A_{m-1} = 0) + P(A_{m-1} = 1)]$. Supposing that $N \geq 2 \cdot 100/n$, $P(A_m = 0) \approx e^{-3}$ and $P(A_m = 1) \approx 3e^{-3}$. Therefore $P(X_m = 0) \leq 4e^{-6}$. Compare this number with the case $M = 0$, where $P(X_m = 0) = P(A_m = 0) \approx e^{-3}$, and the claimed exponential improvement is apparent.

The linear increase of π_0 for large M, evident from Figure 3, can be seen from the following argument. As M increases, the average arrival rate decreases and the queue size increases. Thus, for large M, the queue virtually never overflows and good eviction candidates are not discarded due to a lack of storage. The problem is that the smaller arrival rate brings fewer good eviction candidates, making the linear decrease of the

arrival rate the dominant phenomenon in performance degradation. Recall that for a queue with arrival rate λ and service rate μ when the overflow probability is negligible, the probability that it is empty, π_0, equals $1 - \lambda/\mu$. In our case $\pi_0 = 1 - (N - M)\frac{n}{100}$, showing the linear increase in π_0 with M.

3.1 A Closer Look at the Performance Curves

From the above discussion and Figure 3 we may deduce that the error probability as a function of M and for given N and n has the following features: As M increases from 0, the error probability decreases exponentially, flattens out, and then increases linearly. In particular, it appears that error probability is a convex function of M. The rest of this section shows that this convexity is a general feature of the algorithm and holds for arbitrary values of N and n.

To establish convexity directly it would have helped greatly if $\pi_0(M)$ could be expressed as a function of the elements of T_M in closed form. Unfortunately, this is not the case and we must use an indirect method, which seems interesting in its own right. Our method consists of relating $\pi_0(M)$ to the quantity $D(t, M, \lambda(M))$, which is the number of overflows in the time interval $[0, t]$ from a buffer of size M with average arrival rate $\lambda(M)$.

Let $\overline{D}(M) \stackrel{\triangle}{=} \lim_{t\to\infty} D(t, M, \lambda(M))/t$.

Theorem 1 *The probability of error is convex in M.*

Proof: Let $A^M[0, t]$ be the number of arrivals in $[0, t]$. Then the probability the system is full as observed by arrivals, or equivalent the probability of drops, equals

$$P_{drop} = \lim_{t\to\infty} \frac{D(t, M, \lambda(M))}{A^M[0, t]} = \lim_{t\to\infty} \frac{D(t, M, \lambda(M))}{t} \frac{t}{A^M[0, t]}.$$

Lemma 3 below implies that $\overline{D}(M)$ is convex in M.

Proceeding, equating effective arrival and departure rates we obtain

$$\begin{aligned} \lambda(M) \cdot (1 - P_{drop}(M)) &= (1 - P_{empty}(M)), \\ \text{or } P_{empty}(M) &= 1 - \lambda(M) \cdot (1 - P_{drop}(M)), \\ \text{or } P_{empty}(M) &= 1 - \lambda(M) + \overline{D}(M). \end{aligned} \tag{2}$$

Since $\lambda(M) = (N - M)\frac{n}{100}$ is linear in M, and $\overline{D}(M)$ is convex in M, Equation (2) implies $P_{error}(M) = P_{empty}(M)$ is convex in M. ∎

To complete the proof it remains to show that $\overline{D}(M)$ is convex in M. The proof of the convexity of $\overline{D}(M)$ is carried out in Lemmas 1, 2, and 3 below. In the following we abbreviate $D(t, M, \lambda(M))$ to $D(t, M)$ when the arrival process does not depend on M, and to $D(t, \lambda(M))$ when the buffer size is constant, regardless of the value of M. Lemma 1 shows that $D(t, M)$ is convex in M for all $t > 0$. Lemma 2 shows the convexity of $\lim_{t\to\infty} D(t, \lambda(M))/t$. Finally, Lemma 3 shows the convexity of $\overline{D}(M)$.

For now, we only give a sketch of the somewhat combinatorially involved proofs of these results. The full proofs can be found in the appendix.

Lemma 1 *$D(t, M)$ is a convex function of M, for each $t > 0$.*

Sketch of proof: To prove convexity it suffices to show that the second order derivative of the number of drops is non-negative; i.e., that $(D(t, M-1) - D(t, M)) - (D(t, M) - D(t, M+1)) \geq 0$. This can be done by comparing the number of drops $D(t, M-1)$, $D(t, M)$, and $D(t, M+1)$ from systems with buffer sizes $M-1$, M, and $M+1$ respectively, under identical arrival processes.

Essentially, the comparison entails considering four situations for buffer occupancies in the three systems, as illustrated in Figure 17 in the appendix. ■

Let $D(\lambda(M)) \stackrel{\triangle}{=} \lim_{t\to\infty} D(t, \lambda(M))/t$.

Lemma 2 *$D(\lambda(M))$ is a convex function of M when $\lambda(M) = (N - M)\frac{n}{100}$.*

Sketch of proof: We need to show $D(\lambda(M-1)) - 2D(\lambda(M)) + D(\lambda(M+1)) \geq 0$ by considering three systems with same buffer sizes and binomially distributed arrival processes with average rates $\lambda(M-1)$, $\lambda(M)$ and $\lambda(M+1)$. Thus, there will be common arrivals and exclusive arrivals as categorized below:

(a) An arrival occurs at all three systems.

(b) An arrival occurs only at the system with buffer size $M-1$.

(c) An arrival occurs at the two systems with buffer sizes $M-1$ and M and there is no arrival at the system with buffer size $M+1$.

Due to the arrival rates being as in the hypothesis of the lemma, category (b) and (c) arrivals are identically distributed. Using this and combinatorial arguments one can then show that $D(\lambda(M))$ is convex. ■

N	$min(\pi_0)$			M^*		
	n=10	n=20		n=10	n=20	
8	0.3643	0.0593		1	2	
10	0.2450	0.0110		1	3	
12	0.1378	0.0011		2	4	
	n=5	n=10		n=5	n=10	
20	0.1946	0.0013		2	5	
	n=4	n=8		n=4	n=8	
30	0.0732	2.4454^{-6}		4	9	
	n=3	n=6	n=9	n=3	n=6	n=9
40	0.0558	8.0595^{-8}	4.6629^{-15}	5	12	16
	n=2	n=4	n=6	n=2	n=4	n=6
50	0.1354	1.8678^{-6}	9.5368^{-14}	4	13	18
60	0.0350	8.3933^{-11}	-	7	19	-
70	0.0025	-	-	11	-	-
80	3.1553^{-5}	-	-	16	-	-

Table 1: Optimum values of π_0 and M for various N and n

Lemma 3 *$\overline{D}(M)$ is a convex function of M when $\lambda(M) = (N - M)\frac{n}{100}$.*

Sketch of proof: We consider three systems of buffer sizes $M-1$, M and $M+1$, whose arrival processes are Binomially distributed with rates $\lambda(M-1)$, $\lambda(M)$ and $\lambda(M+1)$. This is a combination of Lemma 1 and 2. ■

4 On the Optimal Value of M

The objective of this section is to derive an approximate closed form expression for the optimal value of M for a given N and n. This expression (Equation (6)) is simple and, as Table 2 shows, it is quite accurate over a large range of values of N and n.

Let $M^* = \arg\min\{\pi_0(M)\}$ be the optimal value of M. As remarked earlier, even though the form of the transition matrix, T_M, allows one to write down an expression for $\pi_0(M)$, this expression doesn't allow one to calculate M^*. Thus, we have to numerically solve Equation (1), compute $\pi_0(M)$ for all $M \leq N/2$, and read off M^* for various values of N and n, as in Table 1. This table is to be read as follows. Suppose N=30 and $n = 4\%$. Then minimum value of π_0 is 0.0732 and it is achieved at $M^* = 4$.

Even though there is no exact closed form solution from which one might calculate M^*, we can derive an approximate close form solution using elementary martingale

theory. Recall that X_m is the number of useless documents in the sample and that $X_{m+1} = ((X_m - 1 + A_m) \wedge (M+1)) \vee 0$[6]. The boundaries at 0 and $M+1$ complicate the analysis of this Markov Chain (MC). The idea is to work with a MC that has no boundaries and then use the *Optional Stopping Time Theorem* to take into account the boundaries at 0 and $(M+1)$.

Since we want to operate away from the event $\{X_m = 0\}$, we assume that the MC will have a positive drift towards the boundary at $(M+1)$. Let $Z_m^{'} = (M+1) - X_m$. Then $Z_m^{'}$ will have a negative drift towards the boundary at 0.

Let Z_m be the corresponding unreflected MC which follows the equation $Z_{m+1} = Z_m + 1 - A_m$. Consider the exponential martingale (MG)

$$W_m = \exp(\theta Z_m),$$

where θ will be chosen so that $E(W_{m+1}|W_m) = W_m$. Since

$$\begin{aligned} E(W_{m+1}|W_m) &= E(\exp(\theta Z_{m+1})|Z_m) \\ &= E(\exp(\theta(Z_m + 1 - A_m))|Z_m) \\ &= W_m \cdot \exp(\theta) \cdot E(\exp(-\theta A_m)), \end{aligned}$$

we obtain that θ must satisfy the equation

$$E(\exp(-\theta A_m)) = \exp(-\theta).$$

Since A_m is binomially distributed with parameters $N-M$ and $p = n/100$, $E(\exp(-\theta A_m)) = (p \cdot \exp(-\theta) + 1 - p)^{N-M}$. Therefore, we require that θ solve the equation

$$(N-M) \cdot \ln(1 - p(1 - \exp(-\theta))) = -\theta. \tag{3}$$

Since there is no general closed form solution to the above equation, using a Taylor's approximation for $\ln(1-x)$ and $\exp(x)$ and keeping terms up to θ^2 we get:

$$\theta = \frac{2}{p(1-p)} \left(p - \frac{1}{(N-M)} \right). \tag{4}$$

Let $T = \min\{m > 0 : Z_m \leq 0 \text{ or } Z_m = M+1\}$. Then T is a bounded stopping time and we can use the Optional Stopping Time Theorem [6] to get

$$E(W_0) = E(W_T).$$

[6] The symbols $\wedge$ and $\vee$ denote the minimum and maximum operations, respectively.

Let $Z_0 = 0$. Then $W_0 = 1$ and thus

$$\begin{aligned} & E(\exp(\theta Z_T)) = 1 \\ \Rightarrow \quad & E(\exp(\theta Z_T) 1_{(Z_T = M+1)}) \leq 1 \\ \Rightarrow \quad & \exp(\theta(M+1)) \cdot P(Z_T = M+1) \leq 1 \\ \Rightarrow \quad & P(X_T = 0) \leq \exp(-\theta(M+1)). \end{aligned} \tag{5}$$

From (5), to minimize the probability of error it suffices to maximize $\theta(M) \cdot (M+1)$ over M. Using Equation (4) and elementary algebra we conclude that the optimal value is given by

$$M^o = N - \sqrt{(N+1)100/n}. \tag{6}$$

Some comments about the approximations we have made in the above derivation are in order. We have dropped the term $E(\exp(\theta Z_T) 1_{(Z_T \leq 0)})$ to obtain the bound in inequality (5). This term is not negligible, in general. However, it is negligible for positive θ, and Z_T sufficiently negative[7], which holds for reasonably large values of $N \times n$. When the term is not negligible, the approximation for M^o in (6) can differ from the true optimum by 1 or 2.

In Table 2 we compare the results for the optimal M obtained by: (i) Equation (6), denoted by M^o, (ii) numerically solving Equation (3) for θ and then using the bound in (5), denoted by M^+, and (iii) by the MC model, denoted by M^*. This table is to be read as follows: For example, suppose $N = 40$ and $n = 6$, the optimal M equals (i) $M^o = 13.8$, (ii) $M^+ = 12$, and (iii) $M^* = 12$. Note that M^o and M^+ are very close to M^* unless the number of samples is very close to the number of bins.

So far we derived an approximate closed form expression for M^*. Here we comment on the performance improvement when a memory of M^* is used. We have mentioned that introducing memory leads to an exponential improvement in performance. A direct way of verifying this is to compare the number of samples needed to achieve a certain performance with and without memory. In particular, fix an n and for each $N > 0$, compute the minimum probability of error obtainable using the optimal value, M^*, of memory. Then compute the number of samples needed to achieve the same probability of error without memory. Denote by N^* and N^0 the number of samples needed to obtain the same error probability with and without memory, respectively. Figure 4 plots N^* versus N^0 for four different values of n as the error probability varies.

There is no closed form expression for the curves in Figure 4. However, one can obtain an upper bound on N^0 as follows. One of the events that will lead to an error is when

[7] Note that θ is positive when the number of samples $N - M$ are more than the number of bins $100/n$, and Z_T becomes more negative the more and better samples we have.

N	M^o, M^+, M^*								
	$n=10$			$n=20$					
8	0	1	1	1.3	1	2			
10	0	0	1	2.6	2	3			
12	0.5	1	2	3.9	3	4			
	$n=5$			$n=10$					
20	0	0	2	5.5	5	5			
	$n=4$			$n=8$					
30	2.2	2	4	10.3	9	9			
	$n=3$			$n=6$			$n=9$		
40	3.0	3	5	13.8	12	12	18.7	15	16
	$n=2$			$n=4$			$n=6$		
50	0	0	4	14.3	13	13	20.8	18	18
60	4.8	5	7	20.9	18	19	-	-	-
70	10.4	10	11	-	-	-	-	-	-
80	16.4	15	16	-	-	-	-	-	-

Table 2: Comparison of optimum values of M for various N and n, calculated from MG approximation (M^o, M^+) and from MC (M^*).

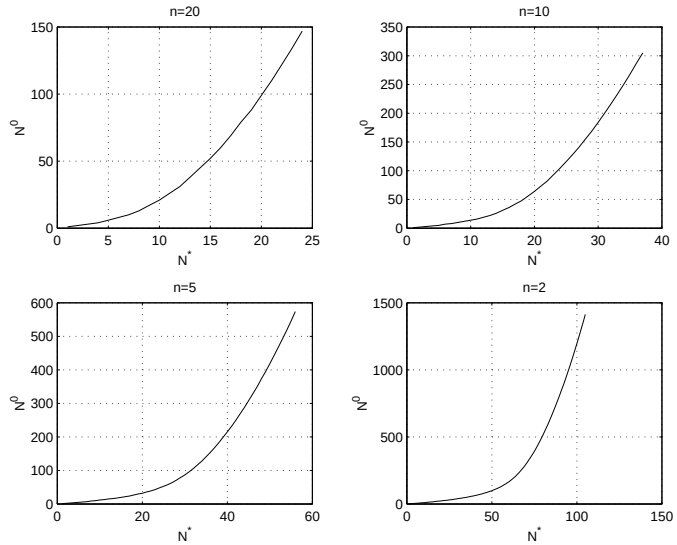

Figure 4: Comparison of number of samples required to achieve same probability of error with (N^*) and without (N^0)memory.

all the $N - M$ fresh samples of all the consecutive $M + 1$ iterations will be bad samples. Thus $P_{error} > (1 - n/100)^{(N-M)(M+1)}$ or $N^0 < (N - M) \cdot (M + 1)$. In a different context, Shah et al. [15] proved that $N^0 > (N - M - \frac{M}{2}) \cdot (M + 1)$.

5 Trace Driven Simulations

We present simulations using web traces to study the performance of our algorithm under real traffic. In particular, we approximate deterministic cache replacement schemes using our randomized algorithm, and compare their performances. Recall that any cache replacement algorithm is characterized by a utility function that determines the suitability of a page for eviction. The main issues we wish to understand are:

- How good is the performance of the randomized algorithm according to realistic metrics like hit rate and latency? It is important to understand this because we have analyzed performance using the frequency of eviction from designated percentile bins as a metric. This metric has a strong positive correlation with realistic metrics but doesn't directly determine them.
- Our analysis in the previous sections assumes that documents retained in memory are not accessed between iterations. Clearly, in practice, this assumption can only hold with a high probability at best. We show that this is indeed the case and determine the probability that a sample retained in memory is accessed between iterations.
- How long do the best eviction candidates stay in the cache? If this time is very long (on average), then the randomized scheme would waste space on "dead" items that can only be removed by a cache flush.

Of the three items listed above, the first is clearly the most important and the other two are of lesser interest. Accordingly, the bulk of the section is devoted to the first question and the other two are addressed towards the end.

5.1 Deterministic Replacement Algorithms

We shall approximate the following two deterministic algorithms: (i) LRU, and (ii) GD-Hyb, which is a combination of the GD-Size [4] and the Hybrid [19] algorithms. LRU is chosen because it is the standard cache replacement algorithm. GD-Hyb represents the class of new algorithms which base their document replacement policy on multiple criteria (recentness of use, size of document, cost of fetching documents, and frequency of use). We briefly describe the details of the deterministic algorithms mentioned above.

1. *LRU.* The utility function assigns to each document the most recent time that the document was accessed.

2. *Hybrid [19].* The utility function f assigns eviction values to documents according to the formula

$$f = (L + \frac{W_1}{B})F^{W_2}/S,$$

where L is an estimate of the latency for connecting with the corresponding server, B is an estimate of the bandwidth between the proxy cache and the corresponding server, F is the number of times the document has been requested since it entered the cache (frequency of use), S is the size of the document, and W_1, W_2 are weights[8]. Hybrid evicts the document with the smallest value of f.

3. *GD-Size [4].* Whenever there is a request for a document, the utility function f adds the reciprocal of the document's size to the currently minimum eviction value among all the documents in the cache, and assigns the result to the document. Thus, the eviction value for document i is given by

$$f_i = \min(f_j : j \text{ in cache}) + \frac{1}{S_i},$$

Note that the quantity $\min(f_j : j$ in cache$)$ is increasing in time and it is used to take into account the recentness of a document. Indeed, since whenever a document is accessed its eviction value is increased by the currently minimum eviction value, the most recently used documents tent to have larger eviction values. GD-Size evicts the document with the smallest value of f.

4. *GD-Hyb* uses the utility function of Hybrid in place of the quantity $1/S$ in the utility function of GD-Size. Thus, its utility function is as follows:

$$\begin{aligned} f &= \min(f) + f', \quad \text{where} \\ f' &= (L + \frac{W_1}{B})F^{W_2}/S. \end{aligned}$$

We shall refer to the randomized versions of LRU and GD-Hyb as RLRU and RGD-Hyb respectively. Note that the RGD-Hyb algorithm uses the $\min(f)$ among the samples, and not the global $\min(f)$ among all documents in the cache.

So far we have described the utility functions of some deterministic replacement algorithms. Next, we comment on the implementation requirements of those schemes.

LRU can be implemented with a linked list that maintains the order in which the cached documents were accessed so far. This is due to the "monotonicity" property of its utility

[8] In the simulations we use the same weights as in [19].

function; whenever a document is accessed, it is the most recently used. Thus, it should be inserted at the bottom of the list and the least recently used document always resides at the top of the list. However, most algorithms, including those that have the best performance, lack the monotonicity property and they require to search all documents to find which to evict. To reduce computation overhead, they must use a priority queue to drop the search cost to $O(logK)$, where K is the number of documents in the cache. In particular, Hybrid, GD-Size, and GD-Hyb must use a priority queue.

The authors in [13] propose an algorithm called LRV (Lowest Relative Value). This algorithm uses a utility function that is based on statistical parameters collected by the server. By separating the cached documents into different queues according to the number of times they are accessed, or their relative size, and by taking into account within a queue only time locality, the algorithm maintains the monotonicity property of LRU *within* a queue. LRV evicts the best among the documents residing at the head of these queues. Thus, the scheme can be implemented with a constant number of linked lists, and finds an eviction candidate in constant time. However, its performance is inferior to algorithms like GD-Size [4]. Also, the cost of maintaining all these linked lists is still high.

The best cache replacement algorithm is in essence the one with the best utility function. In this paper we don't seek the best utility function. Instead, we propose a low cost, high performance, robust algorithm that treats all the different utility functions in a unified way.

5.2 Web Traces

The traces we use are taken from Virginia University, Boston University, and National Laboratory for Applied Network Research (NLANR). In particular:

- The *Virginia* [18] trace consists of every URL request appearing on the Computer Science Department backbone of Virginia University with a client inside the department, naming any server in the world. The trace was recorded for a 37 day period in September and October 1995. There are no latency data on that trace thus it can not be used to evaluate RGD-Hyb.
- The *Boston* [5] trace consists of all HTTP requests originating from 32 workstations. It was collected in February 1995 and contains latency data.
- The *NLANR* [21] traces consist of two sets, one daily trace, and a weekly trace. The daily trace was recorded the 23rd of September 2000, while the weekly trace

Traces	total requests	unique requests	one-timers	θ
Virginia	54434	27395	20603	0.59
Boston	104496	34178	22157	0.69
NLANR daily	295295	167862	122966	0.49
NLANR weekly	2876105	1012831	658319	0.72

Table 3: Trace workload characteristics.

was collected from the 22nd to the 28th of September 2000[9]. Both of them contain latency data.

We only simulate requests with a known reply size. Table 3 presents the workload characteristics of the traces, namely the total number of requests, the number of unique requests, the number of one-timers, and a popularity-parameter ,θ , resulting from fitting the popularity distribution of the documents of each trace to a Zipf-like distribution[10].

5.3 Results

The performance criteria used are three:

(i) the hit rate (HR), which is the fraction of client-requested URLs returned by the proxy cache,

(ii) the byte hit rate (BHR), which is the fraction of client requested bytes returned by the proxy cache, and

(iii) the latency reduction (LR), which is the reduction of the waiting time of the user from the time the request is made till the time the document is fetched to the terminal (download latency), over the sum of all download latencies.

For each trace, HR, BHR, and LR are calculated for a cache of infinite size, that is a cache large enough to avoid any evictions. Then, they are calculated for a cache of size $0.5\%, 5\%, 10\%$, and 20% of the minimum size required to avoid any evictions. This size is 500MB, 900MB, 2GB, and 20GB, for Virginia, Boston, daily NLANR, and weekly NLANR trace respectively. Table 4 shows the absolute values of HR, BHR, and LR for the four traces, for an infinite size cache. Notice that the HR of the NLNAR traces is lower than the HR of the other traces, since NLANR caches are second-level caches[11].

[9] NLANR traces consist of daily and weekly traces of many sites; the one we used is the PA site.

[10] Let p_i be the probability of requesting the i^{th} most popular document of a trace. In practice, $\{p_i\}$ is known to be Zipf-like [2]; i.e., $p_i \propto 1/i^{\theta}$, where θ is the popularity-parameter of the trace.

[11] First-level caches exploit most of the temporal locality in web-request sequences.

Traces	HR	BHR	RL
Virginia	42.40%	32.57%	-
Boston	54.38%	36.05%	33.72%
NLANR daily	24.74%	16.00%	13.91%
NLANR weekly	31.13%	17.50%	23.28%

Table 4: Absolute values of HR, BHR, and LR for the four traces, for an infinite size cache.

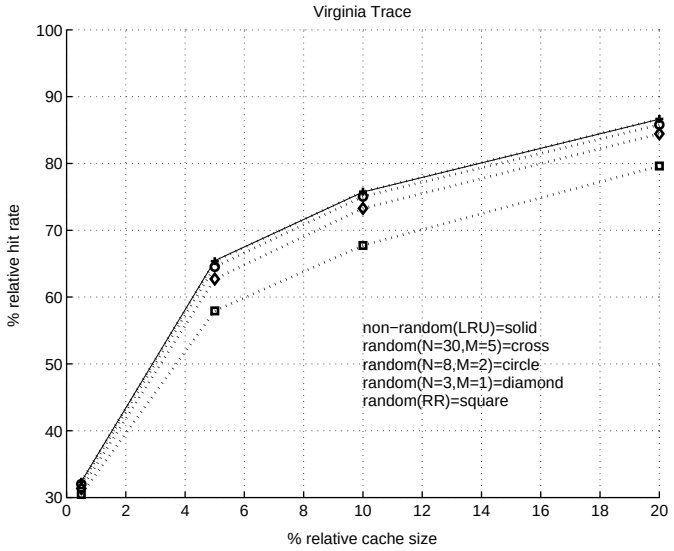

Figure 5: Hit rate comparison between LRU and RLRU.

All the traces give similar results. Below, we show the performance of RLRU using the Virginia trace, then present simulation results using the Boston trace and examine the performance RGD-Hyb, and finally evaluate both RLRU and RGD-Hyb using the longer and newer NLANR traces.

We examine how well the randomized algorithm can approximate LRU, using the Virginia trace. Figure 5 presents the ratio of the HR of LRU, RLRU, and RR over the HR achieved by an infinite cache, using the Virginia trace. As expected, the more the samples the better the approximation of LRU by RLRU. Note that $N = 8$ and $M = 2$ are enough to make RLRU perform almost as good as LRU, and even $N = 3$, $M = 1$ give good results. Figure 6 presents the ratio of the BHR of LRU, RLRU, and RR, over the BHR achieved by an infinite cache. Note again that $N = 8$ and $M = 2$ are enough to make RLRU perform almost as good as LRU.

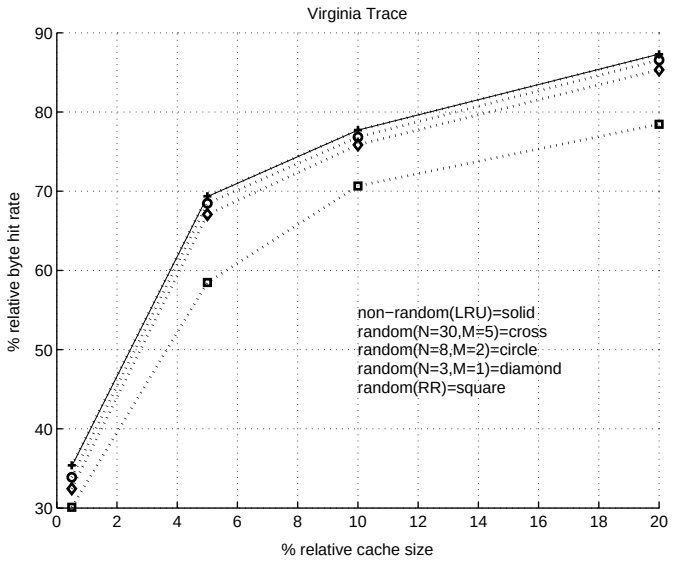

Figure 6: Byte Hit rate comparison between LRU and RLRU.

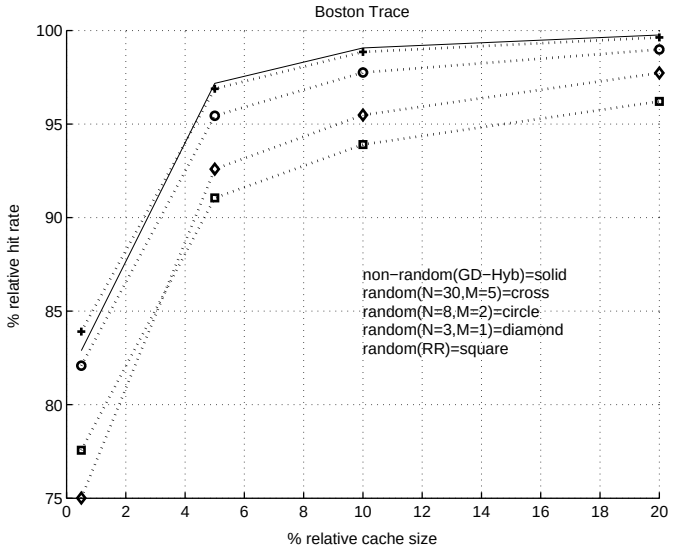

Figure 7: Hit rate comparison between GD-Hyb and RGD-Hyb.

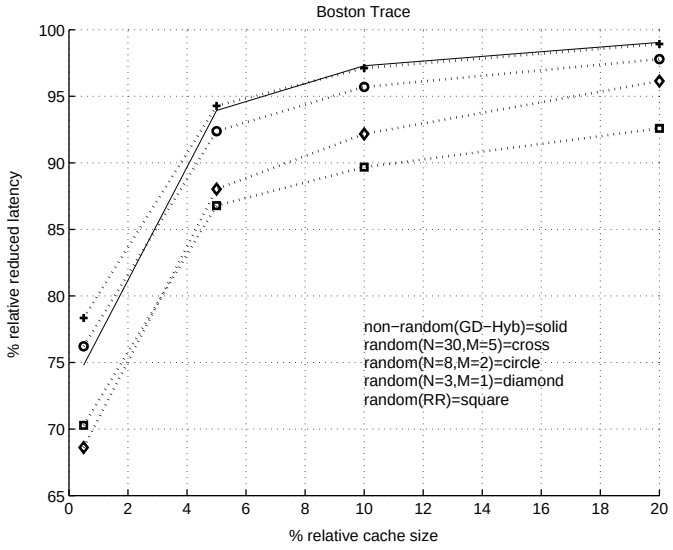

Figure 8: Latency reduction comparison between GD-Hyb and RGD-Hyb.

Next, we examine how well the randomized algorithm can approximate GD-Hyb. We present results from the Boston trace. Figure 7 presents the ratio of the HR of GD-Hyb, RGD-Hyb, and RR, over the HR achieved by an infinite cache. As expected, the more the samples the better the approximation of GD-Hyb by RGD-Hyb. The performance curve of RGD-Hyb for N=8 and M=2 is very close to the performance curve of GD-Hyb. For N=30 and M=5 the curves almost coincide. Note that the Boston trace has low correlation among the requests and as a result RR performs relatively well.

Figure 8 presents the ratio of LR achieved by GD-Hyb, RGD-Hyb, and RR, over the LR achieved by an infinite cache. It is again the case that values of N and M as low as 8 and 2 respectively are enough for RGD-Hyb to perform very close to GD-Hyb.

The traces we used so far are taken from universities, and they are relatively small and old. Next, we present the results from the daily NLANR trace. Our goal is twofold: to evaluate the randomized algorithm under more recent and larger traces, and examine the performance of RLRU and RGD-Hyb in the same trace.

Figure 9 and 10 present the ratio of HR of various schemes over the HR achieved by an infinite cache. The former figure compares LRU to RLRU, and the later GD-Hyb to RGD-Hyb. In Figure 9 RLRU nearly matches LRU for N and M as small as 8 and 2 respectively. In Figure 10 RGD-Hyb requires 30 samples and a memory of 5 to approximate very close GD-Hyb, but its performance is superior to LRU. Indeed GD-Hyb achieves around 100% of the infinite cache performance while LRU achieves below 90%. Note that this trace has a lot more correlation on its requests than the Boston

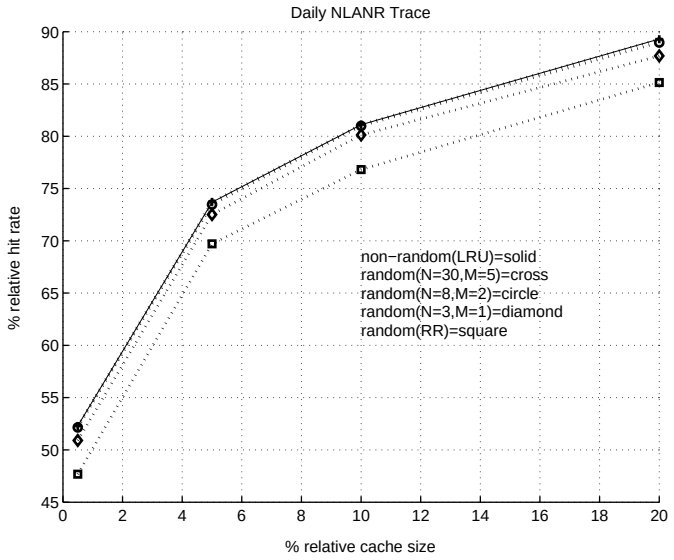

Figure 9: Hit rate comparison between LRU and RLRU.

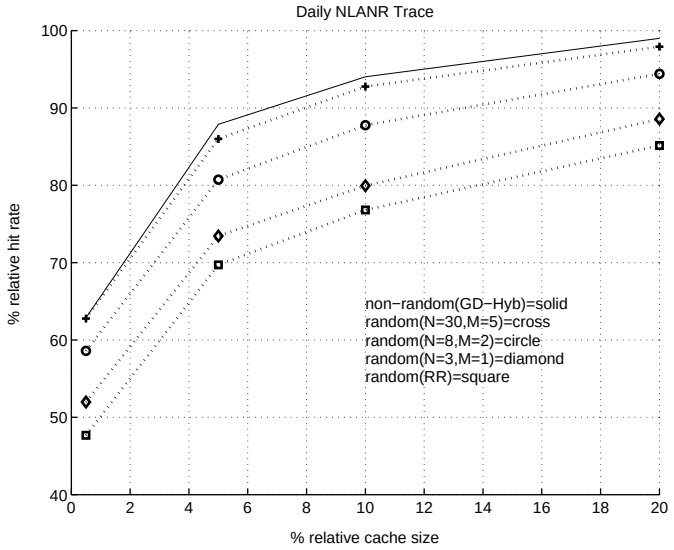

Figure 10: Hit rate comparison between GD-Hyb and RGD-Hyb.

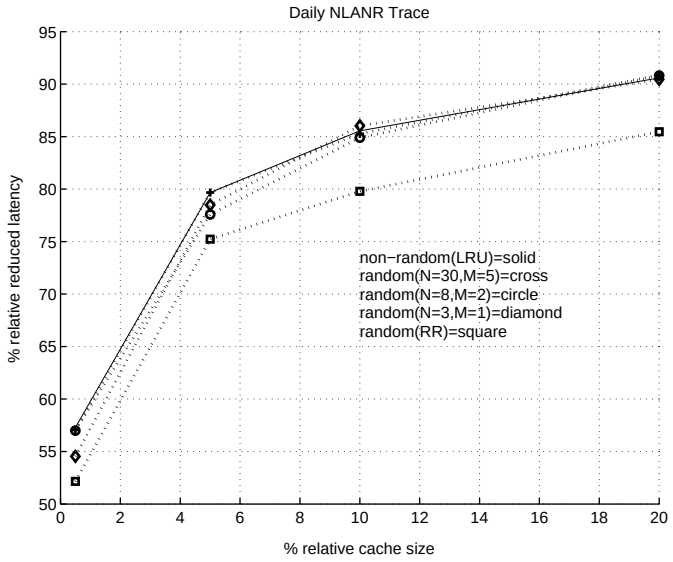

Figure 11: Latency reduction comparison between LRU and RLRU.

trace since RR's performance is 15% worse than GD-Hyb's.

Figure 11 and 12 present the ratio of RL of various schemes over the RL achieved by an infinite cache. The former figure compares LRU to RLRU, and the later GD-Hyb to RGD-Hyb. In Figure 11 RLRU nearly matches LRU for N and M as small as 3 and 1 respectively. In Figure 12 RGD-Hyb performs slightly better than GD-Hyb. This somewhat unexpected result is caused because GD-Hyb makes some suboptimal eviction decisions on that particular trace that could be removed by fine-tuning the two terms in its utility function[12]. Despite that fact, GD-Hyb performs better than LRU in respect to RL, since LRU does not take into account any latency information.

The last trace from which we present results is the weekly NLANR trace. This trace consists of roughly 3 million requests and thus approximates reality better. Cache size is now an issue. In particular, 20% of the maximum size required to avoid any evictions corresponds to 4GB.

Figure 13 and 14 present the ratio of HR of various schemes over the HR achieved by an infinite cache. The former figure compares LRU to RLRU, and the later GD-Hyb to RGD-Hyb. Since these figures are similar to Figure 9 and 10, the performance of the randomized algorithm appears to be unaffected by the trace size. Note that the

[12]Since all the deterministic cache replacement schemes rely on heuristics to predict future requests, their eviction decision may be suboptimal. Thus, randomized approximations of these algorithms may occasionally perform better.

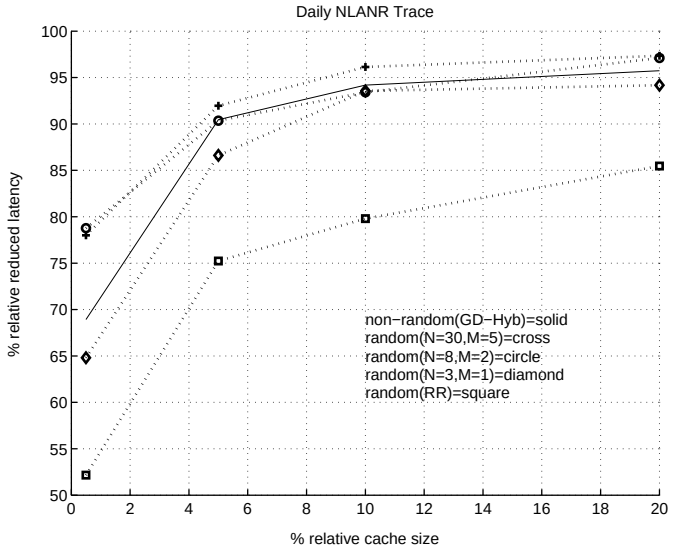

Figure 12: Latency reduction comparison between GD-Hyb and RGD-Hyb.

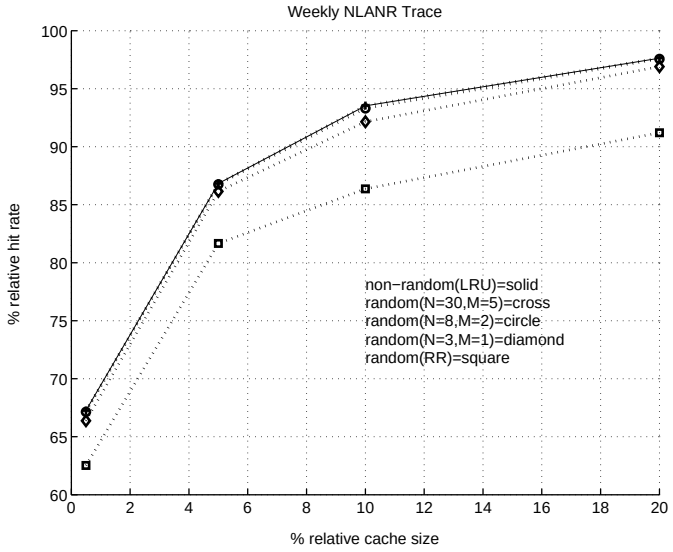

Figure 13: Hit rate comparison between LRU and RLRU.

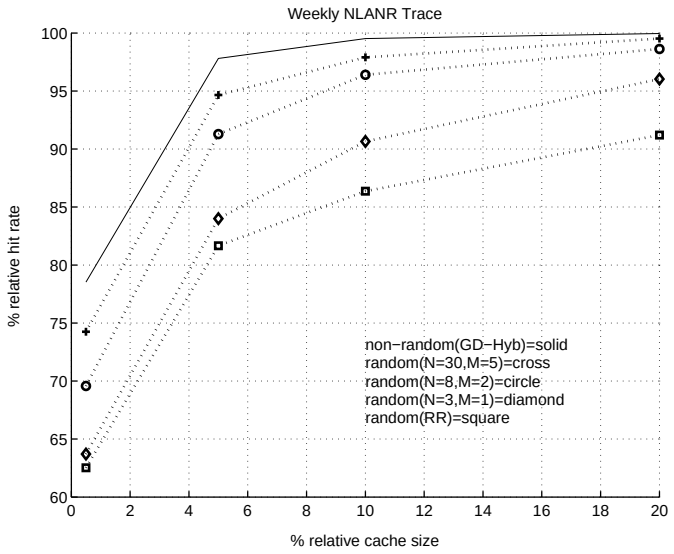

Figure 14: Hit rate comparison between GD-Hyb and RGD-Hyb.

difference in the performance of LRU and GD-Hyb is not significant for a relative cache size of 20%, but for smaller cache sizes like 10% and 5% it is.

Figure 15 and 16 present the ratio of BHR of various schemes over the BHR achieved by an infinite cache. The former figure compares LRU to RLRU, and the later GD-Hyb to RGD-Hyb. The randomized algorithm works well in respect to BHR also. Note that RGD-Hyb performs slightly better than GD-Hyb for smaller cache sizes and more importantly, LRU performs better than GD-Hyb. This somewhat unexpected result is caused because GD-Hyb makes relatively poor choices in terms of BHR by design, since it has a strong bias against large size documents even when these documents are popular. This suboptimal performance of GD-Hyb is inherited from SIZE [18] and Hybrid [19] and could be removed by fine-tuning. All the three schemes trade in HR for BHR.

Recently, an algorithm called GreedyDual* (GD*) has been proposed [7], that achieves superior HR *and* BHR when compared to other web cache replacement policies. This algorithm is a generalization of GD-Size. GD* adjusts the relative worth of long-term popularity versus short-term temporal correlation of references. It achieves that by dynamically adapting to the degree of temporal correlation of the web request streams. GD* requires a priority queue to be efficiently implemented and thus it is a good candidate to be approximated by our randomized algorithm.

From the figures above, it is evident that the randomized versions of the schemes can perform competitively with very small number of samples and memory. One would expect to require more samples and memory to get such good performance. However, since

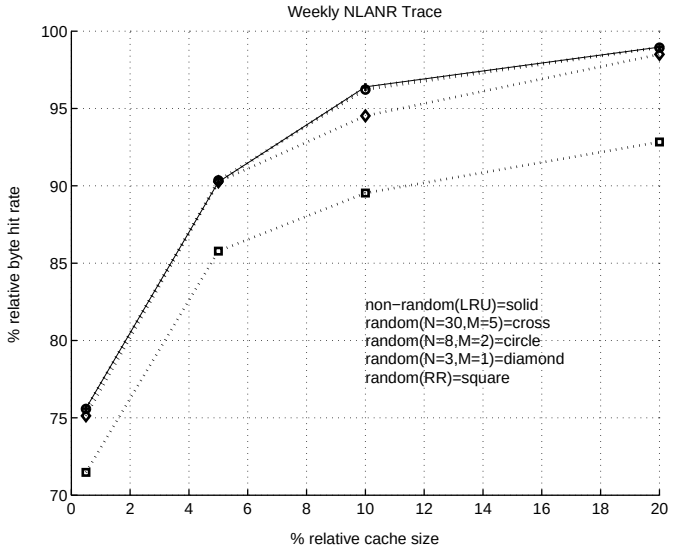

Figure 15: Byte hit rate comparison between LRU and RLRU.

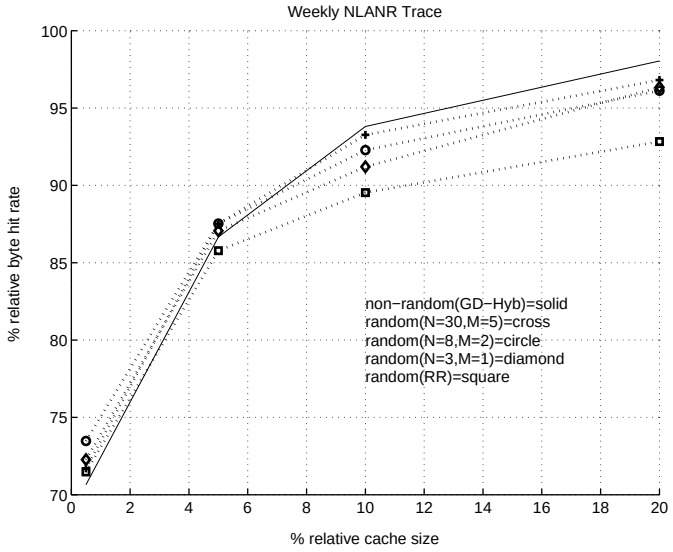

Figure 16: Byte hit rate comparison between GD-Hyb and RGD-Hyb.

all the online cache replacement schemes rely on heuristics to predict future requests, it is not necessary to exactly mimic their behavior in order to achieve high performance. Instead, it usually suffices to evict a document that it is within a reasonable distance from the least useful document.

There are two more issues to be addressed. First, we wish to estimate the probability that documents retained in memory are accessed between iterations. This event very much depends on the request patterns and is hard to analyze exactly. Instead, we use the simulations to estimate the probability of occurring. Thus, we change the eviction value of a document retained in memory whenever it is accessed between iterations, which deteriorates its value as an eviction candidate. Also, we don't obtain a new, potentially better, sample. Despite this, we find that the performance is not degraded. The reason for this is that our policy for retaining samples in memory, deliberately chooses the best eviction candidates. Therefore, the probability that they are accessed is very small. In particular, it is less than 10^{-3} in our simulations.

Second, we wish to verify that the randomized versions of the schemes do not produce dead documents. Due to the sampling procedure, the number of sampling times that a document is not chosen follows a geometric distribution with parameter roughly equal to N over the total number of documents in the cache. This is around 1/100 in our simulations. Hence, the probability that the best ones are never chosen is zero. And the best ones are chosen once every 100 sampling times or so.

6 Implementation Issues

In the previous sections we established that a small number of samples, 6 fresh and 2 carried from one iteration to the next, is sufficient for the randomized algorithm to have good performance. In this section we discuss in detail the implementation savings by using the randomized algorithm.

There are two main operations that a web cache has to support; access to arbitrary cached documents, and eviction of relatively useless documents to make room for new documents.

State-of-the-art web caches access documents through a hash table [20], which has constant time lookups. Also, this hash table is stored in the RAM [20], which ensures that any document can be obtained in a single disk read. Thus, accessing arbitrary cached documents is done very efficiently. This operation is orthogonal to the eviction scheme used.

Different eviction schemes have different implementation requirements. As previously

discussed, LRU can be implemented with a linked list that maintains the temporal order in which the cached documents were accessed. High performance algorithms like GD-Hyb require a priority queue to be implemented efficiently. Thus, for every insertion, they perform $O(logK)$ operations, where K is the number of documents in the cache. The size of a typical web cache today is tens of GBs. Since the average size of web documents is close to 10KB, K is typically many millions [20]. These schemes perform an insertion operation at every access time, since even at hit times the utility value of the document changes and the document should be reinserted at a new position.

The implementation savings due to the randomized eviction scheme are the following: First, the randomized scheme saves memory resources from not maintaining a data structure for eviction purposes. However, the parameters used by the utility function, like the frequency and recency of use, still need to be stored in order to be available when the document is chosen as a sample. Thus, for every document, there will be a corresponding object in RAM holding its utility parameters.

Second, the proposed randomized algorithm draws about 6 fresh samples per eviction time, instead of performing one insertion and one deletion operation in a priority queue per access time. A sample can be easily obtained in constant time by randomly choosing one of the objects that hold the utility parameters of the documents, or one of the entries of the hash table used to access the documents[13]. Thus, drawing random samples is cheaper than updating a priority queue.

Suppose the hit rate is around 50%. Then, there is one miss in every two accesses. Assuming there is no correlation between the size of a document and its popularity [3], every miss causes on average one eviction. Thus, priority queue updates take place twice as often as random sampling. For higher hit rates the CPU savings increase further, for example, for hit rates close to 75%, priority queue updates take place four times as often as random sampling[14].

Finally, the randomized algorithm does not need to recompute the utility value of a cached document every time it is accessed. It performs this operation only with the samples obtained at eviction times if their utility value is not up-to-date. However, as a consequence, whenever RGD-Hyb computes utility values, it uses the minimum utility value among the samples instead of the global minimum among all documents. The simulations in Section 5 take this into account and show that it causes no performance degradation.

This work focuses on implementation savings with respect to CPU and memory resources. In some systems, the disk I/O resources are more important. Eviction schemes in such systems typically aim to reduce the number of and the time taken by disk reads/writes,

[13] Sampling is done without accessing the disk. It is only after the best sample is identified, that the corresponding document is accessed on the disk in order to be evicted.

[14] Hit rates up to 70% are quite common [7], [13].

and ignore parameters like the recency of use of a document. Using the ideas presented in this paper it is possible to take such parameters into account at a minimal overhead, and thus increase the hit rate without trading-off disk I/O resources.

7 Conclusions

Motivated by the need to reduce the complexity of data structures required by good document replacement schemes proposed in the literature, we have introduced a randomized algorithm that requires no data structure. This saves both the memory used up by the data structure and the processing power needed to update it.

Being randomized, the performance of our algorithm depends crucially on the quality of the samples it obtains. We observe that retaining the best unused samples from one iteration to the next improves the quality of the samples and leads to a dramatic improvement in performance. An analysis of the algorithm provides insights about its features and a formula for determining the right parameters for approximating the performance of any deterministic replacement scheme as closely as desired. Trace-driven simulations show that a small number of samples, 6 fresh and 2 carried from one iteration to the next, is sufficient for a good performance.

We believe that the idea of using memory to improve the quality of samples in iterative randomized algorithms is of general interest, possibly applicable in other situations.

8 Acknowledgments

We thank Dawson Engler for conversations about cache replacement schemes, and A.J. Ganesh for helpful discussions regarding the determination of the optimal value of M.

References

[1] M. Abrams, C.R. Standbridge, G. Abdulla, S. Williams and E.A. Fox, "Caching Proxies: Limitations and Potentials", In proceedings of *WWW-4*, Boston, December 1995.

[2] L. Breslau, P. Cao, L. Fan, G. Philips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", In proceedings of *IEEE INFOCOM*, New York, 1999.

[3] M. Busari and C. Williamson, "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics", In proceedings of *IEEE INFOCOM*, Anchorage, Alaska, April 2001.

[4] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", In proceedings of the *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, Dec. 1997.

[5] C.R. Cunba, A. Bestavros and M.E. Crovella, "Characteristics of WWW Client-based Traces", BU-CS-96-010, Boston University.

[6] R. Durrett, *Probability: Theory and Examples*, Duxbury Press, 2nd edition, 1996.

[7] S. Jin and A. Bestavros, "GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams", In Proceedings of the *5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[8] T. Lindvall, *Lectures on the Coupling Method*, Wiley-Interscience, 1992.

[9] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[10] J. Norris, *Markov Chains*, Cambridge University Press, 1997.

[11] K. Psounis, B. Prabhakar and D. Engler, "A Randomized Cache Replacement Scheme Approximating LRU", In Proceedings of the *34th Annual Conference on Information Sciences and Systems*, Princeton University, March 2000.

[12] K. Psounis and B. Prabhakar, "A Randomized Web-Cache Replacement Scheme", In Proceedings of *IEEE INFOCOM*, Anchorage, Alaska, April 2001.

[13] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache", *IEEE/ACM Transactions On Networking*, Vol. 8, No. 2, April 2000.

[14] D. Shah, P. Giaccone and B. Prabhakar, "An Efficient Randomized Algorithm for Input-Queued Switch Scheduling", In Proceedings of *HOT Interconnects 9*, Stanford, CA, August 2001.

[15] D. Shah and B. Prabhakar, "The Use of Memory in Randomized Load Balancing", Unpublished manuscript.

[16] A. Silberschatz and P. Galvin, *Operating System Concepts*, Fifth Edition, Addison Wesley Longman, 1997.

[17] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches", In Proceedings of *IEEE INFOCOM*, , San Fransisco, CA, March 1998.

[18] S. Williams, M. Abrams, C.R. Standbridge, G. Abdulla and E.A. Fox, "Removal Policies in Network Caches for World-Wide Web Documents", In Proceedings of *ACM SIGCOMM*, Stanford University, August 1996.

[19] R. Wooster and M. Abrams, "Proxy Caching that Estimates Edge Load Delays", In Proceedings of the *6th International World Wide Web Conference*, Santa Clara, CA, April 1997.

[20] "Web Caching White Paper", CacheFlow, Inc., http://www.cacheflow.com/technology/whitepapers/web.cfm.

[21] NLANR Cache Access Logs, ftp://ircache.nlanr.net/Traces/.

9 Appendix

Proof of Lemma 1: Convexity follows from showing that the second derivative of the cumulative number of drops is non-negative:

$$D^{''}(t) \stackrel{\triangle}{=} D(t, M-1) - 2D(t, M) + D(t, M+1) \geq 0 \quad \text{for all } t \geq 0. \tag{7}$$

Essentially, this reduces to comparing the cumulative number of drops upto time t, $D(t, M-1)$, $D(t, M)$, and $D(t, M+1)$, from systems with buffer sizes $M-1$, M, and $M+1$, respectively. The three systems have identical arrival processes and start with empty buffers at time 0. In the proof we induce on each arrival, since these are common to all three systems, and hence this is equivalent to inducing over time. Define $d^{''}(a)$ to be the instantaneous difference in drops caused by the common arrival a which occurs at time t_a. Then $D^{''}(t) = \sum_{a:t_a \leq t} d^{''}(a)$.

To determine $d^{''}(a)$, consider the situation depicted in Figure 17. The three buffers are placed next to each other, in increasing order of size. The shaded boxes in the figure represent occupied buffer spaces and the white boxes represent empty buffer spaces. If all three buffers have empty spaces, then no drops will occur. Thus, it suffices to consider cases where at least one of the buffers is full.

We claim that there are only four possible cases to consider where at least one buffer is full, and these are depicted in Figure 17. The following general observation, whose proof is inductive, easily verifies the claim. Consider any two systems A and B with buffer sizes B_A and $B_B = B_A + 1$, respectively. Suppose that A and B have identical arrival processes and there is exactly one departure from each non-empty system in each time slot. Then the buffer occupancies of system A and B, denoted by BO_A and BO_B respectively, will always satisfy $BO_A = BO_B$ or $BO_A = BO_B - 1$. Therefore, the only cases to consider are the ones shown in Figure 17.

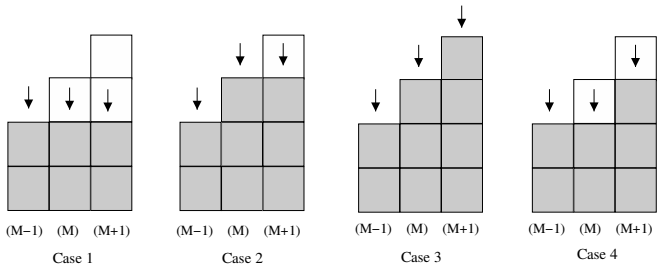

Figure 17: Possible cases of buffer occupancies for Lemma 1.

An inspection of the four cases shows that for each common arrival a the values of $d^{''}(a)$ are given by (1) $d^{''} = 1$, (2) $d^{''} = -1$, (3) $d^{''} = 0$, and (4) $d^{''} = 1$. Thus, under Cases 1, 3 and 4 $d^{''} \geq 0$, and the only troublesome case is 2. Note, however, that every instance of Case 2 is *preceded* by at least one instance of Case 1. Therefore, the negative values of Case 2 are offset by the positive values of Case 1 and it follows that $D^{''}(t) \geq 0$ for all t. ■

Proof of Lemma 2: We need to show

$$\lim_{t\to\infty} \frac{1}{t}(D(t, \lambda(M-1)) - 2D(t, \lambda(M)) + D(t, \lambda(M+1))) \geq 0 \tag{8}$$

by considering three systems with same buffer sizes, and arrival rates that drop linearly with M.

In particular, the arrival processes A_{M-1}, A_M, and A_{M+1} to these systems are Binomially distributed with average rates $\lambda(M-1) = \lambda(M+1) + 2\frac{n}{100}$, $\lambda(M) = \lambda(M+1) + \frac{n}{100}$, and $\lambda(M+1) = (N-M-1)\frac{n}{100}$ respectively. Note that the processes A_{M-1} and A_M stochastically dominate [15] A_{M+1}. Thus, we can use a coupling argument [8] to categorize arrivals as follows:

(a) *Common.* An arrival occurs at all three systems. Common arrivals are Binomially distributed with average rate $(N-M-1)\frac{n}{100}$.

(b) *Single.* An arrival occurs only at the system with buffer size $M-1$. Single arrivals are Bernoulli(n/100).

(c) *Double.* An arrival occurs at the two systems with buffer sizes $M-1$ and M and there is no arrival at the system with buffer size $M+1$. Double arrivals are also Bernoulli(n/100).

[15] A process X stochastically dominates Y iff $P(Y < x) \leq P(X < x)$ for all x.

Define $d''(a)$ to be the instantaneous difference in drops caused by arrival a which occurs at time t_a. Then, $D(t, \lambda(M-1)) - 2D(t, \lambda(M)) + D(t, \lambda(M+1)) = \sum_{a:t_a \leq t} d''(a)$. If all three buffers have empty spaces, then no drops will occur. Thus, it suffices to consider cases where at least one of the buffers is full.

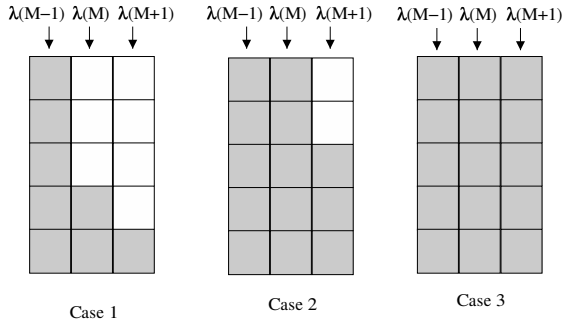

Figure 18: Possible cases when buffer sizes same.

Let $BO(M-1)$, $BO(M)$, and $BO(M+1)$ be the buffer occupancy of a buffer with average arrival rate $\lambda(M-1)$, $\lambda(M)$, and $\lambda(M+1)$ respectively. Note that for any time t, $BO(M-1) \geq BO(M) \geq BO(M+1)$. There are three cases to consider as shown in In Figure 18. Let C be the common buffer size. Case 1 is characterized by $BO(M-1) - BO(M) > 0$, $BO(M-1) = C$, Case 2 by $BO(M) - BO(M+1) > 0$, $BO(M) = C$, and Case 3 by $BO(M+1) = C$.

In Case 1, $d''(a) = 1$ for common, single, and double arrivals. In Case 2, $d''(a)$ equals 1 for single arrivals, and -1 for double and common arrivals. In Case 3, $d''(a)$ equals 1 for single, 0 and common, and -1 for double arrivals. Since single and double arrivals are identically distributed, as $t \to \infty$ their d'''s cancel out in Cases 2 and 3. Thus, it suffices to show that the negative effect of common arrivals in Case 2 is cancelled out by the positive effect of common, single, and double arrivals in Case 1.

We say a single departure takes place if $BO(M-1) = 1$, $BO(M) = 0$, $BO(M+1) = 0$, and a there is a departure. We say a double departure takes place if $BO(M-1) = 1$, $BO(M) = 1$, $BO(M+1) = 0$, and there is a departure.

Assertion 1: $BO(M-1) - BO(M)$ changes at most by one per arrival or departure. $BO(M-1) - BO(M)$ is increased by one, if and only if a single arrival is not dropped. $BO(M-1) - BO(M)$ is decreased by one, if and only if a common or a double arrival comes in Case 1, or there is a single departure.

Assertion 2: $BO(M) - BO(M+1)$ changes at most by one per arrival or departure. $BO(M) - BO(M+1)$ is increased by one, if and only a double arrival is not dropped, or a double arrival comes in Case 1. $BO(M) - BO(M+1)$ is decreased by one, if and

only if a common arrival comes in Case 2, or there is a double departure.

For an arrival or a departure event A, let $\langle A \rangle \stackrel{\triangle}{=} \lim_{t\to\infty} \#(A[0,t])/t$ be the number of times the event takes place in the interval $[0,t]$ divided by t as $t \to \infty$.

Assertion 3: $\langle SD \rangle \stackrel{\triangle}{=} \langle$single departure$\rangle < \langle$double departure$\rangle \stackrel{\triangle}{=} \langle DD \rangle$,
$\langle SAND \rangle \stackrel{\triangle}{=} \langle$single arrival not dropped$\rangle = \langle$double arrival not dropped$\rangle \stackrel{\triangle}{=} \langle DAND \rangle$,
$\langle SA1 \rangle \stackrel{\triangle}{=} \langle$single arrival in Case 1$\rangle = \langle$double arrival in Case 1$\rangle \stackrel{\triangle}{=} \langle DA1 \rangle$.

Using the same notation as above, let $\langle CA1 \rangle \stackrel{\triangle}{=} \langle$common arrival in Case 1$\rangle$ and $\langle CA2 \rangle \stackrel{\triangle}{=} \langle$common arrival in Case 2$\rangle$. Assertion 1 implies

$$\langle SAND \rangle = \langle CA1 \rangle + \langle DA1 \rangle + \langle SD \rangle.$$

Assertion 2 implies

$$\langle DAND \rangle + \langle DA1 \rangle = \langle CA2 \rangle + \langle DD \rangle.$$

Using the above two equations and Assertion 3, we get

$$\langle CA1 \rangle + \langle DA1 \rangle + \langle SA1 \rangle > \langle CA2 \rangle \tag{9}$$

and thus the negative effect of common arrivals in Case 2 is cancelled out by the positive effect of common, single, and double arrivals in Case 1. ■

Proof of Lemma 3: We need to show

$$\lim_{t\to\infty} \frac{1}{t}(D(t, M-1, \lambda(M-1)) - 2D(t, M, \lambda(M)) + D(t, M+1, \lambda(M+1))) \geq 0 \tag{10}$$

by considering three systems of buffer sizes $M-1$, M and $M+1$ and average rates $\lambda(M-1)$, $\lambda(M)$ and $\lambda(M+1)$ respectively. Let $D(M, \lambda(M)) \stackrel{\triangle}{=} \overline{D}(M) = \lim_{t\to\infty} D(t, M, \lambda(M))/t$.

Lemma 2 implies

$$D(M-1, \lambda(M-1)) - 2D(M-1, \lambda(M)) + D(M-1, \lambda(M+1)) \geq 0. \tag{11}$$

We need to show

$$D(M-1, \lambda(M-1)) - 2D(M, \lambda(M)) + D(M+1, \lambda(M+1)) \geq 0. \tag{12}$$

By adding and subtracting $2D(M-1, \lambda(M))$ and $D(M-1, \lambda(M+1))$ in Equation (12) and using Equation (11), it is easy to see that it suffices to show

$$2(D(M-1, \lambda(M)) - D(M, \lambda(M))) \geq D(M-1, \lambda(M+1)) - D(M+1, \lambda(M+1)).$$

By adding and subtracting $D(M, \lambda(M+1))$ in the right hand side of this equation, it is easy to see that is suffices to show

$$D(M-1, \lambda(M)) - D(M, \lambda(M)) \geq D(M-1, \lambda(M+1)) - D(M, \lambda(M+1)) \quad \text{and} \quad (13)$$

$$D(M-1, \lambda(M+1)) - D(M, \lambda(M+1)) \geq D(M, \lambda(M+1)) - D(M+1, \lambda(M+1)). \quad (14)$$

Lemma 1 implies Equation (14) holds. To see why Equation (13) holds, categorize arrivals to basic arrivals with rate $\lambda(M+1)$ and excess arrivals with rate $\lambda(M)-\lambda(M+1)$, and notice that there is no way for excess arrivals to increase $D(M, \lambda(M))$, without also increasing $D(M-1, \lambda(M))$. ■