GreedyDual* Web Caching Algorithm

*Exploiting the Two Sources of Temporal Locality in Web Request Streams**

Shudong Jin and Azer Bestavros

Computer Science Department
Boston University
Boston, MA 02215

{jins,best}@cs.bu.edu

Abstract

The relative importance of long-term popularity and short-term temporal correlation of references for Web cache replacement policies has not been studied thoroughly. This is partially due to the lack of accurate characterization of temporal locality that enables the identification of the relative strengths of these two sources of temporal locality in a reference stream. In [21], we have proposed such a metric and have shown that Web reference streams differ significantly in the the prevelance of these two sources of temporal locality. These findings underscore the importance of a Web caching strategy that can adapt in a dynamic fashion to the prevelance of these two sources of temporal locality. In this paper, we propose a novel cache replacement algorithm, GreedyDual*, which is a generalization of GreedyDual-Size. GreedyDual* uses the metrics proposed in [21] to adjust the relative worth of long-term popularity versus short-term temporal correlation of references. Our trace-driven simulation experiments show the superior performance of GreedyDual* when compared to other Web cache replacement policies proposed in the literature.

1 Introduction

Web caching aims to reduce network traffic, server load, and user-perceived retrieval delay by replicating "popular" content on proxy caches [2, 19] that are strategically placed within the network—at organizational boundaries or at major AS exchanges, for example.

One might argue that the ever decreasing prices of RAM and disks renders the optimization or fine tuning of cache replacement policies a "moot point". Such a conclusion is ill-guided for several reasons. First, recent studies have shown that Web cache hit ratio (HR) and byte hit ratio (BHR) grow in a *log-like* fashion as a function of cache size [3, 10, 14, 11]. Thus, a better algorithm that increases hit ratios by only several percentage points would be equivalent to a several-fold increase in cache size. Second, the growth rate of Web content is much higher than the rate with which memory sizes for Web caches are likely to grow. The only way to bridge this widening gap is through efficient cache management. Finally, the benefit of even a slight improvement in cache performance may have an appreciable effect on network traffic, especially when such gains are compounded through a hierarchy of caches.

One difficulty of Web caching is that there are many factors affecting the performance of a given cache replacement policy. Among others, these factors include object size, miss penalty, temporary locality, and long-term access frequency.

(1) Unlike traditional caching in memory systems, Web caches manage objects of *variable* sizes. Caching smaller objects usually results in higher hit ratios, especially given the preference for small objects [14]—though this preference seems to be weakening [6].
(2) The miss penalty (i.e. retrieval cost of missed objects from server to proxy) varies significantly. Thus, giving a preference to objects with a high retrieval latency can achieve high latency saving [31].
(3) Web traffic patterns were found to exhibit temporal locality [3, 11, 24], i.e., recently accessed objects are more likely to be accessed again in the near future. This has led to the use of LRU cache replacement policy and generalizations thereof [11].
(4) The popularity of Web objects was found to be highly variable (i.e. bursty) over short times scales, but much smoother over long time scales [5, 17], suggesting the significance of *long-term* measurements of access frequency in cache replacement algorithms.

Exploiting Temporal Locality Properties: Locality of reference properties have been exploited in a number of Web caching, replication, and prefetching protocols and systems. Generally speaking, such protocols can be classified as: (1) Server-based (i.e. at the server or a proxy thereof) [7, 8, 13], (2) Client-based (i.e. at the client or a proxy thereof) [1, 9], or (3) Network-based (i.e. in the network, transparent to both the server and the client) [18, 11, 27]. An important reason for this classification is that Web reference characteristics are likely to be different in each of the above categories. To understand this, it suffices to note that at each of the above categories, the request streams being multiplexed through a cache (or replica) are significantly different. For client-based caches, the request streams are *from* a limited and possibly homogeneous community of users (e.g. students in a University,

*This work was partially supported by a NSF research grant CCR-9706685 and by a NSF ANIR grant.

or subscribers to a local ISP, etc.). For server-based caches (or replicas), the request streams are *to* the limited set of objects offered by the server. For network-based caches, neither of these constraints could be assumed. Furthermore, misses in client-based caches are hits in network-based caches, and misses in those are hits in server-based caches (or replicas).

If the underlying determinant of temporal locality is different for each of the above categories, it follows that an effective cache replacement strategy must be able to delineate between the *different causes* of temporal locality and to quantify their relative significance (in order to effectively capitalize thereon). In the Section 3 of this paper, we present a metric that enables such a capability. Then we sketch and evaluate a cache replacement strategy that effectively uses this metric.

2 Related Work

Table 1 classifies sixteen replacement policies according to whether they exploit access recency and access frequency, and whether they are sensitive to the variable $cost/size$ of objects, which is one of the salient aspects of Web caching (namely, unlike traditional memory systems, object sizes and miss penalties are highly variable).

Recency-Based Policies: Cache replacement algorithms in traditional memory systems deal with uniform $cost/size$ objects. LRU is the most widely used cache replacement algorithm, as it captures recency and is superior to other policies, e.g. FIFO and Random. Since Web traffic also exhibits locality, LRU is adopted widely in Web servers, client applications, and proxy servers. An example is the the variance implemented in the Squid Internet object cache [28].

A disadvantage of LRU is that it does not consider variable-size or variable-cost objects. The GreedyDual-Size or GDS algorithm [11] enables the incorporation this variability through the use of a $cost/size$ weighting. GDS is a generalization of the GreedyDual or GD algorithm [32], which deals with uniform-size variable-cost objects. It was shown to be online optimal in terms of its competitive ratio and superior to LFF, which evicts the largest file and does not capture recency. A common drawback of LRU and GreedyDual-Size is that they do not take into account the frequency of resource use.

Frequency-Based Policies: The basic frequency-aware replacement algorithm is LFU. It always evicts the object with the lowest reference count. LFU is online-optimal under a purely independent reference model. There are two subtle problems with LFU. First, there are different versions of LFU algorithm, e.g., Perfect LFU and In-Cache LFU, according to whether the reference count is also discarded when an object is evicted. Second, in an LFU replacement algorithm, when two objects have the same reference count, a *tiebreaker* is necessary.

A straightforward generalization of LFU for dealing with variable $cost/size$ objects is the normalized-cost LFU, which uses $\frac{frequency \times cost}{size}$ as the key. The Hybrid policy in [31] is a variant of this version of LFU. It estimates the load delay as the *cost* function. Experiments indicate that this Hybrid policy outperforms the simpler Latency policy, which evicts the objects with minimum load delay but does not take into account frequency and size factor.

Recency/Frequency-based Policies: Several studies have considered both recency and frequency information under a fixed cost/fixed size assumption. The LRU-K [26] algorithm maintains the last K reference times to each object to compute the average reference rate. The LFU-DA algorithm [4] is a frequency-based algorithm with dynamic aging. On a fetch or a hit, the object value is set to the reference count plus the minimum reference count in the cache. Simulations with large traces indicate LFU-DA obtains the highest byte-hit-ratio. Another work [23] generalizes LRU and LFU to a hybrid policy LRFU, which subsumes a range of algorithms, depending on the different weights given to recency and frequency.

To deal with variable $cost/size$, generalizations of the above techniques have also been proposed. In [29], the LNC-W3 algorithm is proposed as a generalization of LRU-K to deal with variable-cost and variable-size Web objects. It computes the average reference rate and uses that to estimate the *profit* of caching an object. Simulations indicated that LNC-W3 obtains higher delay saving ratios than those achived through LRU and LRU-K. In another direction, several studies [4, 22] proposed generalizations of the GreedyDual-Size algorithm to incorporate frequency. These algorithms include GDSF [4] and GD-LFU [22]. In [20], we proposed the GDSP algorithm, which incorporates frequency into GDS and maintains an accurate popularity profile. Finally, another important work is the LRV algorithm [24]. LRV uses the cost, size, and last reference time of an object in calculating its *utility*. The calculation is based on extensive empirical analysis of trace data. Their work indicates that exploiting the regularities in Web traffic can enhance caching performance.

3 Temporal Locality

Denning and Schwartz [15] established the fundamental properties that characterize the phenomenon of *locality*. Such properties are the catalyst for well-established practices in the design of caching systems in hierarchical memory structures [30]. In order to apply these practices to the design of Web caching and prefetching systems, it's important to characterize the degree of locality present in typical Web request streams.

Early characterizations of Web access patterns suggested the presence of temporal locality of reference [3, 5, 11, 17, 24]. However, more recent studies have concluded that this temporal locality is weakening [6]. One reason for this trend was attributed to effective client caching. To understand this, it suffices to note that the request stream generated by a client using an efficient caching policy is precisely the set of requests that missed in the client cache. Such a request stream is likely to exhibit weak temporal locality of reference—in particular, a recently accessed object is *unlikely* to be accessed again in the future. [1]

[1] Assuming very efficient client caching, it follows that repeated

	Aspect of Temporal Locality Being Exploited			
Cost/Size	**None**	**Recency-based**	**Frequency-based**	**Both**
Fixed	FIFO,Random	LRU	LFU	LRU-K,LFU-DA,LRFU
Variable	LFF,Latency	GD,GDS	Hybrid	LNC-W3,GDSF,GDSP,LRV

Table 1: Taxonomy of existing Web cache replacement policies

Using an independent reference model [12], Breslau et al [10] showed that the Zipf-like popularity distribution of objects in Web request streams can asymptotically explain other properties (namely, cache efficiency and temporal locality). In particular, they showed that the probability of referencing an object t units of time after it has been last referenced is roughly proportional to $1/t$. Thus, the probability distribution of reference interarrival times (or inter-request time) could be used to model temporal locality. In this paper, we call this the reference interarrival model.

However, the skewed popularity profile of objects in Web request streams is not the only source of temporal locality. To explain this point, consider the following two request streams: ``XAXBXCXDXEXF...'' and ``GGHHIIJJKKLL...''. Obviously, both streams exhibit temporal locality properties. In the first stream, the locality is due to the popularity of X, whereas in the second stream it is due to the correlation in time of the 1st and 2nd requests for G, H, I, J, K, and L. Temporal locality due to popularity is preserved under reordering. Thus, in a random permutation of the first stream, reference interarrival time is still proportional to the probability of access. Temporal locality due to correlation in time is *not* preserved under reordering.

Given the above two dimensions of temporal locality, an important question is whether the characterization of temporal locality using the reference interarrival model (used in many studies including [10, 11]) is capable of quantifying the effects of popularity and temporal correlation, independently. In [21], we have shown that this is not the case. In particular, we have shown that the distribution of reference inter-arrivals is predominantly affected by the popularity profile of objects in the Web request streams. The relationship between these two aspects is formalized in the following Theorem, which is reproduced from [21].

Theorem 1 *If the distribution of document popularity in a request stream asymptotically follows a power law with parameter α, where $0.5 \ll \alpha \leq 1$, then the distribution of reference interarrivals in a random permutation of that request stream can be characterized asymptotically using a power law with parameter $(2 - \frac{1}{\alpha})$.* **Proof:** *See [21].*

This strong relationship between popularity and temporal locality often disguises another important aspect of temporal locality, namely the temporal correlation of repeated references to the same objects. This has lead to (for example) the inadequate conclusion in [10] that a Zipf-like popularity distribution, together with an independent reference model, is enough to explain temporal locality.

To characterize the level of temporal correlation in a request stream, we must "equalize" the effect of popularity. Thus, we consider the probability distribution of reference interarrivals for *equally popular* objects. For such a distribution, we expect the effect of popularity to be eliminated. In [21], we have shown that such a distribution is a good estimator of the degree of temporal correlation in the request stream. Also, we have shown that the degree of temporal correlation can be quantified by β, the slope of log-log scaled distribution of reference interarrivals for equally popular objects. For the objects of same popularity, the probability that the reference interarrival time equals t is roughly proportional to $t^{-\beta}$ in short term. Table 2 gives the ranges of β for the traces in our study (described in section 5). It indicates that the value of β is rather stable for different values of k's, but that it varies significantly across traces.[2]

Traces	**DEC**	**RTP**	**SD**	**UC**
$k = 1$	0.61	0.51	0.39	0.50
$k = 2$	0.63	0.49	0.41	0.50
$k = 4$	0.63	0.47	0.40	0.46
$k = 8$	0.65	0.46	0.41	0.43

Table 2: The values of β identify the degree of short-term temporal correlation in the request streams. β is estimated with a least-square fit from the plots in [21] for the various traces considered.

4 GreadyDual* Replacement

Let p be a document. Let $s(p)$ be the size of p, and $c(p)$ be the cost to fetch it. Each document has a relative frequency value, denoted by $f(p)$. Let the *utility* value of $u(p)$ represent the normalized value of p. The replacement algorithm tries to maximize the cost saving brought by caching under the restriction of total cache size. In addition, let β be the parameter of the power law $T \sim t^{-\beta}$ characterizing reference correlation. Recall that reference correlation is measured by the distribution of reference interarrivals for equally-popular objects as explained in Section 3.

requests to an object at a caching proxy are likely to be from multiple clients, and thus are reflective of popularity and *not* of temporal correlation of references. This was termed *geographical locality of reference* in [8]. This observation is supported by our findings in [21] regarding the relative contributions of popularity and temporal correlation to locality properties.

[2] Recall from our earlier discussion that proxy cache traces represent different client and server populations and thus is expected to exhibit different temporal locality characteristics.

4.1 From GDS to GD*

In the absence of any reference correlations, a greedy algorithm can compute $u(p)$ and sort the objects in decreasing order, then keeps as many objects as possible in this order. With reference correlation, the GDS algorithm [11] takes $c(p)/s(p)$ as $u(p)$, and uses an inflation value L to age the objects. On retrieval or on a hit, the key of a document $H(p)$ is set to $L + u(p)$; on each eviction, L is set to the value $H(p)$ of the evicted document p. GreedyDual* (GD*) redefines the utility value $u(p)$ and the dynamic aging mechanism to reflect the strength or weakness of locality due to temporal correlation versus that due to popularity.

Utility Value: In GD*, the utility value $u(p)$ reflects the normalized expected cost saving if the document stays in the cache. Obviously $u(p)$ is proportional to $c(p)/s(p)$. Moreover, it is also proportional to the long-term frequency if the reference pattern is stable. Since the past reference count is a good approximation of the frequency, $u(p)$ should be roughly proportional to $\frac{f(p)\times c(p)}{s(p)}$, where $f(p)$ is approximated by the reference count so far.

Dynamic Aging: Since Web traffic exhibits reference correlation GD* uses a dynamic aging mechanism similar to that used in GDS. Namely, each document p has an $H(p)$ value, meanwhile the algorithm keeps an inflation value L to age the objects in the cache. When the L value catches up with $H(p)$, then p will be the candidate for eviction. On each hit or when fetching from the server, the algorithm resets the H value of a document to its *base value* plus L.

Now the only remaining problem is setting the base value. The base value for a document must reflect both the document utility and the degree of reference correlation. Since reference interarrivals for equally popular objects follow $T \sim t^{-\beta}$, the maximal length of time for p to stay in the cache should be proportional to $u(p)^{1/\beta}$. We assume the cache is in a steady state, when L increases steadily, and the time for a document to stay in the cache is roughly proportional to its base value. Therefore, in GD*, the base value is set to $u(p)^{1/\beta}$. We illustrate how it works through an example. Assume $\beta = 0.5$. Assume $u(p_1)$ is twice $u(p_2)$ due to differences in p_1 and p_2's retrieval costs, their sizes, or their relative frequencies. Given the power law $T \sim t^{-\beta}$, it follows that document p_1 at time $4t$ after the last reference to it is as competitive as document p_2 at time t after the last reference to it. Thus, GD* obtains equal marginal cost saving from caching either.

4.2 GreedyDual* Algorithm

Our GreedyDual* (GD*) algorithm captures *both* popularity and temporal correlation. The frequency in the base value formula captures long-term popularity, while β controls the rate of aging. A smaller β means weaker reference correlation, therefore a larger base value, which means objects are aged more slowly. The complete GD* algorithm is described in Figure 1.

GreedyDual* subsumes a family of algorithms, each with a different level of dependency on long-term document popularity and short-term reference correlations. We discuss some of these cases below.

```
Algorithm GreedyDual*

L ← 0.0
for each request for document p do
    if p is in cache
        then H(p) ← L + (f(p)×c(p)/s(p))^(1/β)
        else fetch p
                while there is not enough free cache for p
                    do L ← min{H(q)|q is in cache}
                        Evict the minimum q
                H(p) ← L + (f(p)×c(p)/s(p))^(1/β)
```

Figure 1: The GreedyDual* Algorithm

$\beta =$ **1:** When β is close to unity, then the special algorithm GD^1 approximates a simple generalization of the GreedyDual-Size algorithm with frequency. Such an algorithm is called GDSF in Table 1. LFU-DA is a special case of GDSF. Its $cost/size$ is a constant, which means it maximizes byte hit ratio. GD* does not exclude the possibility of the β values being larger than unity. However, our analysis of typical workloads suggest that such a possibility is not likely. As discussed earlier, we found that typical values of β are between 0.3 and 0.7.

$\beta =$ **0:** When β is close to 0, GD^0 degenarates to a variance of the LFU algorithm, i.e., a normalized cost LFU which uses $\frac{f(p)\times c(p)}{s(p)}$ as the key to sort the objects and evict the document with minimum index (in this case, the dynamic aging should not be used). More specifically, when the cost is a constant, GD$^0(1)$ is a simple frequency-aware size-based algorithm. These special cases fall in column 3 of row 2 in Table 1. They are good choices when correlation does not exist or when it is very weak. Since Web traffic still exhibits reference correlations, these algorithms are not the best choices for Web proxy caches.

Constant cost/size: Another member of the GD* family of algorithm is one for which the ratio $cost/size$ is a constant. Such an algorithm would fall between columns 3 and 4 in Table 1. Such an algorithm would be the algorithm of choice for memory systems with a fixed block size and fixed miss penalty.

4.3 Implementation Details

Our implementation of GreedyDual* maintains a priority queue with key $H(p)$. Handling either a hit or a replacement requires $O(\log n)$ time. Thus, it has the same overhead as GreedyDual-Size.

In our implementation, the constant β is estimated in an online fashion. This is done by keeping the number of references at different intervals for equally popular objects

and computing β using a least-square fit. The value of β for a given proxy cache was found to be stable over time. For example, the β value for the DEC traces is close to 2/3, and for the NLANR traces it is less than 0.5.

All frequency-aware replacement policies face a common problem. They must keep track of reference counts, in order to guarantee the accuracy. Obviously, it is unrealistic for an algorithm to keep all the reference counters of evicted objects. However, it is possible to only keep a subset of the the reference counters. In our simulation, the space for reference counters was subject to the following constraints: (1) less than 1% of the cache keeps the reference counters of evicted objects, and (2) the total number of counters is less than 512K. Counter replacement can be done in $O(1)$ if the lowest counters are linked together and the victims are choosen in a LRU fashion.

5 Performance Evaluation

We used trace-driven simulations to evaluate GD* against a number of cache replacement strategies.

5.1 Traces Used in our Experiments

In this paper we use traces from DEC [16] and NLANR [25]. Some of the characteristics of these traces are shown in Table 3. HR_∞ and BHR_∞ are the hit ratios when cache size is infinite. The temporal correlation parameter β values are different, as shown in Table 2.

Preprocessing of DEC Traces: Our preprocessing of the DEC traces follows the same procedures described in [10, 11]. In particular, we have excluded non-cache-able requests, including cgi-bin requests and queries. In addition, in our experiments, we count a request as a hit if the last modification times of the cached object and the actual reply to users are the same when both are known, or if the object size has not changed when both last modification times are unknown. In this paper we only present the results we obtained from the first week of the DEC trace (results from the other weeks are similar).

Preprocessing of NLANR Traces: Our preprocessing of the NLANR traces is more elaborate. The NLANR traces include many IMS (If-Modified-Since) and REFRESH requests with a reply code of "304" (Not Modified). In order to include such requests in the workload, we have to first find the sizes of the objects of such requests. We do so through a 2-pass scanning of the entire trace.[3] In addition to this preprocessing, we have also excluded non-cache-able requests, including cgi-bin requests and queries. In this paper we only present the results we obtained from the three two-week NLANR traces from sites RTP, SD, and UC (hereafter called the RTP, SD, and UC traces).

[3]This process is 96%-successful in identifying cache-able requests. The remaining 4% are IMS and Refresh requests for which we were unable to identify the document sizes.

5.2 Experimental Setup and Metrics

We compare GD* with LRU, GDS, and LFU-DA (LFU with dynamic aging) examined in [4]). Other policies such as LFF, Hybrid [31], and LRV [24] are not considered since they have been shown to under-perform GDS [11].

Both GDS and GD* describe a family of algorithms. To complete the specification of a member of this family, we need to define what constitutes the *cost* of a miss (i.e. miss penalty). To that end, we adopt two models. Under the *constant cost* model, we assume that objects have the same retrieval cost. The resulting algorithms are termed GDS(1) and GD*(1). Under the *packet cost* model, we assume that document retrieval costs are proportional to document size.[4] The resulting algorithms are termed GDS(packets) and GD*(packets).

In our experiments, we consider two main performance metrics—Hit Rate (HR) and Byte Hit Rate (BHR). The constant cost model aims at optimizing HR, whereas the packet cost model aims at optimizing BHR. The use of the constant cost model is appropriate if the purpose of caching is to improve the performance as perceived by the clients of the cache. This would be the case if the cache is deployed close to a client population (e.g. an organizational caching proxy) to reduce response times. The use of the packet cost model is appropriate if the purpose of caching is to improve the utility of the cache or to reduce the overall traffic between the cache and Web servers. This would be the case if the cache is deployed by an ISP to optimize the performance of its networks.

In our experiments, we varied the cache size from a size of less than 1% to a size of about 20% of the total number of unique bytes in the trace.

5.3 Performance Under Constant Cost

Figure 2 shows HR and BHR for the different traces. In each plot, the x-axis (in logarithmic scale) represents the cache size and the y-axis represents the HR (left plots) or BHR (right plots). To establish how each of the algorithms approaches the upper bounds for HR and BHR, the range of the y-axis is set to [0-HR_∞] or [0-BHR_∞].

The results in Figure 2 show that LRU and LFU-DA perform significantly worse than GDS(1) and GD*(1). The poor performance of LRU and LFU-DA can be explained by noting that these policies do not consider *size* as a factor in evaluating the utility of a document. The results in Figure 2 also indicate that the performance of GD*(1) is consistently better than that of GDS(1), especially when the cache size is small. This suggests that accurately capturing both long-term popularity and short-term temporal correlation is more crucial for small caches.

In terms of BHR, the performance of GDS(1) is always the worst. This is not surprising since GDS(1) favors small objects and sacrifices BHR [6]. Interestingly, even though the main objective of GD*(1) is the optimization of HR, the BHR of GD*(1) remains competitive with those of LFU and LFU-DA. This can be explained by noting that the

[4]Namely, the number of *packets* transferred $= 2 + \frac{size}{536}$.

Trace	All requests	Unique requests	HR_∞	BHR_∞
DEC: 29/8-4/9, 1996	3,543,968(44.9G)	1,354,996(21.9G)	48.7%	35.8%
NLANR site RTP: 4/6-17/6, 1999	9,113,027(91.4G)	3,249,549(45.2G)	64.3%	50.7%
NLANR site SD: 4/6-17/6, 1999	9,082,461(129.7G)	3,549,609(61.5G)	60.9%	52.6%
NLANR site UC: 4/6-17/6, 1999	8,983,585(113.1G)	2,459,366(47.3G)	72.6%	58.2%

Table 3: Traces used in this paper

incorporation of reference frequency into GD* allows it to retain frequently referenced objects, including large ones.

There are two conclusions. First, frequency-based policies consistently outperform recency-based policies, e.g., LFU-DA outperforms LRU and GD*(1) outperforms GDS(1). Second, when HR is the main objective, GD*(1) obtains higher HR than GDS(1), without significantly compromising BHR.

5.4 Performance Under Packets Cost

Figure 3 shows the hit ratios of LRU, LFU-DA, GDS(packets), and GD*(packets) for the different traces.

The results in Figure 3 indicate that the hit ratios achieved by GDS(packets) and LRU are very close. This can be explained by noting that when the cost is proportional to document size, GDS(packets) is nearly equivalent to LRU. The slight advantage of GDS(packets) is due to its sensitivity to document size. Figure 3 also shows that for both HR and BHR, LFU-DA outperforms GDS(packets) and LRU. This is consistent with other studies, which confirmed the advantage of frequency-based policies over recency-based policies [4, 10, 24, 31].

The results in Figure 3 indicate that GD*(packets) consistently outperforms all other policies with respect to both HR and BHR. Table 4 illustrates an instance of the improvements and savings achieved through the use of GD* under the UC trace when cache size is 1GB.[5]

5.5 Sensitivity to Temporal Correlations

To evaluate the effect of β on the efficiency of GD*, we varied the value of β used in GD*(packets) from 0.125 to 2. Figure 4 shows the resulting hit ratios. Both HR and BHR are maximized when β is close to 0.5, which is very close to the actual value of β (between 0.46 and 0.51) for the RTP traces as shown in Table 2. This confirms that $u(p)^{1/\beta}$ is an appropriate quantification of the base value for a document. The incorporation of both frequency and the degree of reference correlation achieves the maximum performance gain.

There are two more observations in Figure 4. (1) When cache is small, the effect of β is more obvious. When cache is large, the hit ratios are close to the upper bound, so the effect of β is less obvious. This might suggest that capturing short-term correlations are more crucial for small ccahes. (2) A small β harms hit ratios less than a large β, suggesting that correlation is not important.

To gauge the relative importance of request popularity and temporal correlation, we examined the performance gain of incorporating only the degree of correlation but not the frequency into GDS, i.e., the base value is $(\frac{c(p)}{s(p)})^{1/\beta}$. Figure 5 shows the changes of hit ratios with different value settings of β. The decrease of β results in an increase in hit ratios, but not byte hit ratios. Moreover, in both cases, the ratios are below those achieved through GD*(packets).

6 Summary

In this paper, we have established the importance of temporal temporal locality of references for Web proxy cache replacement algorithms. We proposed a novel cache replacement strategy, GreedyDual*, which capitalizes on and adapts to the relative strengths of *both* long-term popularity and short-term temporal correlation. Simulation results show the superiority of this approach and are in line with our characterization of temporal locality in these traces [21]. For example, we found Web proxy cache replacment policies should capture access frequency, consistent with the predominance of long-term popularity in Web request streams, and replacement policies should capture access recency when cache size is small, consistent with the existence of weak temporal correlations in short term.

Vitae

Azer Bestavros: Dr. Azer Bestavros received his S.M. and Ph.D. degrees in Computer Science from Harvard University in 1988 and 1991, respectively. Since 1991, he has been on the Faculty of the Computer Science Department at Boston University, where he is currently an Associate Professor. Dr. Bestavros' research interests are in the areas of scalable Web server architectures, (Inter)networking services, and real-time embedded computation and communication.

Dr. Bestavros has an extensive list of publications, including three edited books, dozens of book chapters, and over 90 technical papers in refereed, archived journals and conference proceedings. He has delivered over 40 presentations and speeches at universities, industrial laboratories, panels, and national standards committees. His research is supported through substantial grants from government agencies and industrial labs, including the National Science Foundation, the US Army Research Office, GTE and Microsoft.

Dr. Bestavros has extensive consulting and professional experience. He is the co-founder and Principal Technical Officer of Commonwealth Network Technology, Inc. a startup

[5] For the UC trace, 1 GB of cache represents approximately 2.5% of the total size of the unique objects.

	Performance Improvement Achieved through GD*					
	HR	(Gain%)	BHR	(Gain%)	Packets	(Gain%)
LRU → GD*	33.3% → 50.1%	(50.4%)	31.1% → 37.6%	(20.9%)	68.8M → 86.1M	(25.1%)
GDS → GD*	36.5% → 50.1%	(37.3%)	31.4% → 37.6%	(19.9%)	71.1M → 86.1M	(19.7%)
LFU-DA → GD*	39.0% → 50.1%	(28.4%)	33.7% → 37.6%	(11.6%)	79.0M → 86.1M	(8.9%)

Table 4: Example of performance gains achieved through the use of GD* under packet cost assumption.

company based on one of Dr. Bestavros' three patent-pending technologies. CNT was acquired in October 1999 by WebManage Technologies Inc. Dr. Bestavros consulting experience includes engagements with a number of major organizations, including Bull, Ericsson, and the World Health Organization, as well as with a number of startup technology comapny, including Allaire, Bowne Internet Solutions, WebManage Technologies, Quarry Technologies, and Adero.

Shudong Jin: Shudong Jin is a PhD candidate in Computer Science at Boston University, where he conducts research in Web traffic characterization, caching and replication algorithms, Internet measurement and modeling. Mr. Jin obtained his BS and MS degrees in Computer Science in 1991 and 1994 (respectively), from the Department of Computer Science and Engineering, Huazhong University of Science and Technology, China.

References

[1] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of WWW Conferences*, December 1995.

[2] Akamai Technologies. Freeflow content delivery system. Available at http://www.akamai.com.

[3] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of 1996 International Conference on Parallel and Distributed Information Systems (PDIS'96)*, December 1996.

[4] Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating content management techniques for Web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance*, May 1999.

[5] Martin Arlitt and Carey Williamson. Web server workload characteristics: The search for invariants. In *Proceedings of ACM SIGMETRICS'96*, May 1996.

[6] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web*, 2(1): 15-28, 1999.

[7] Azer Bestavros. WWW traffic reduction and load balancing through server-based caching. *IEEE Concurreny: Special Issue on Parallel and Distributed Technology*, 5(1):56–67, Jan-Mar 1997. IEEE Press.

[8] Azer Bestavros and Carlos Cunha. Server-initated document dissemination for the WWW. *IEEE Data Engineering Bulletin*, 19(3): 3-11, September 1996.

[9] Azer Bestavros, Robert Carter, Mark Crovella, Carlos Cunha, Abdelsalam Heddaya, and Sulaiman Mirdad. Application level document caching in the Internet. In *IEEE SDNE'96: The Second International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, June 1995.

[10] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of Infocom'99*, April 1999.

[11] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[12] E. G. Coffman and P. J. Denning. Operating systems theory. Prentice-Hall, 1973.

[13] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Evaluating server-assisted cache replacement in the Web. In *Proceedings of ESA'98*, 1998.

[14] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of WWW client-based traces. Technical Report BUCS95-010, April 1995.

[15] P. Denning and S. Schwartz. Properties of the working set model. *Communications of the ACM*, 15(3):191–198, 1972.

[16] Digital Equipment Corporation. ftp://ftp.digital.com/pub/DEC/traces/proxy/.

[17] Steven D. Gribble and Eric A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[18] A. Heddaya and S. Mirdad. WebWave: Globally load balanced fully distributed caching of hot published Documents. In *Proc. 17th IEEE Intl. Conference on Distributed Computing Systems, Baltimore, Maryland, USA*, May 1997.

[19] Infolibria Inc. Dynacache and mediamall caching solutions. Available at http://www.infolibria.com

[20] Shudong Jin and Azer Bestavros. Popularity-aware GreedyDual-Size algorithm for Web access. In *Proceedings of IEEE ICDCS'00*, April, 2000. Computer Science Technical Report BUCS1999-009, Boston University.

[21] Shudong Jin and Azer Bestavros. Temporal locality in Web request streams. To appear as a short paper in *ACM Sigmetrics'00*, June, 2000. Computer Science Technical Report BUCS1999-014, Boston University.

[22] Balachander Krishnamurthy and Craig E. Wills. Proxy cache coherency and replacement–towards a more complete picture. In *Proceedings of the 19th IEEE ICDCS'99*, June 1999.

[23] Donghee Lee, Jongmoo Choi, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. On the existence of a spectrum of policies that subsumes the LRU and LFU policies In *Proceeding of the 1999 ACM SIGMETRICS Conference*, May 1999.

[24] P. Lorenzetti, L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. Technical Report LR-960731, Univ. di Pisa.

[25] National Laboratory for Applied Network Research. ftp://ircache.nlanr.net/Traces/.

[26] Elizabeth J. O'Neil, Patrick E. O'Neil, Gerhard Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of ACM SIGMOD*, 1993.

[27] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predicative prefetching to improve World Wide Web latency. In *Proceedings of ACM SIGCOMM*, 1996.

[28] Squid Internet Object Cache. http://squid.nlanr.net/Squid.

[29] Peter Scheuermann, Junho Shim, and Radek Vingralek. A case for delay-conscious caching of Web Documents. In *Proceedings of WWW Conference*, 1997.

[30] Alan Jay Smith. Cache memories. *Computing Surveys*, 14(3):473–530, September 1982.

[31] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proceedings of the 6th International WWW Conference*, 1997.

[32] Neal E. Young. On-line caching as cache size varies. In *Proceedings of Symposium on Discrete Algorithms*, 1991.

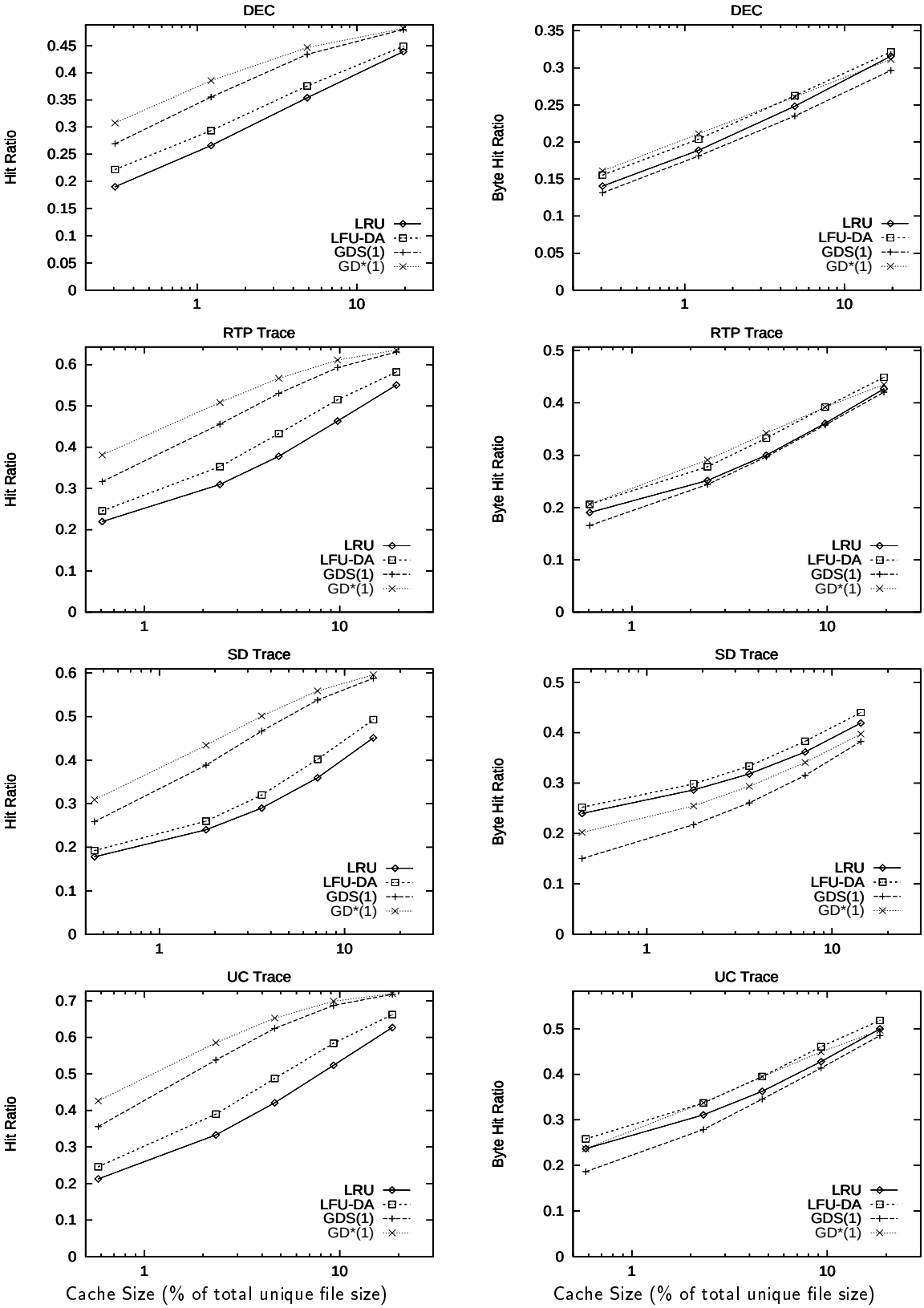

Figure 2: Hit ratios when *cost* of document retrieval is fixed

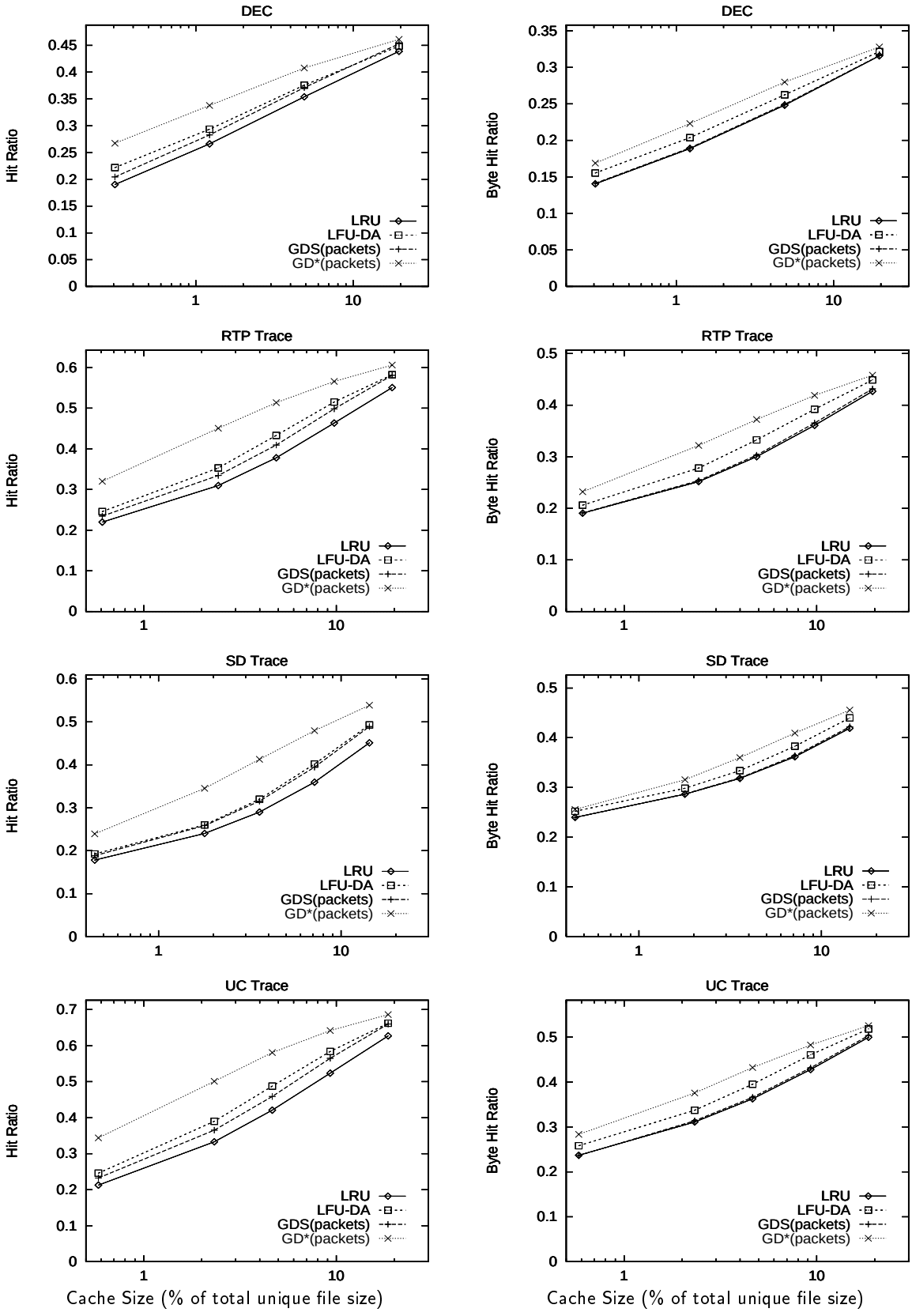

Figure 3: Hit ratios when *cost* of document retrieval is equal to the number of packets transferred

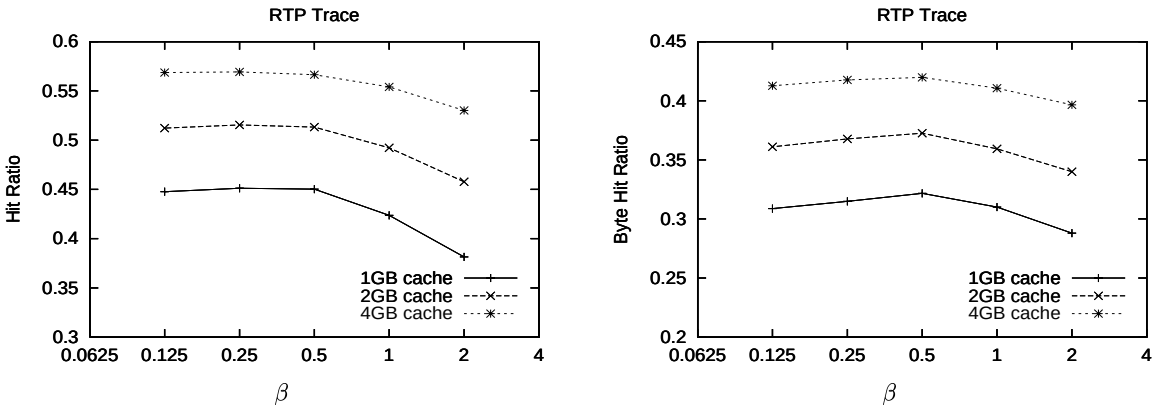

Figure 4: Hit ratios of GD*(packets) with different settings of β

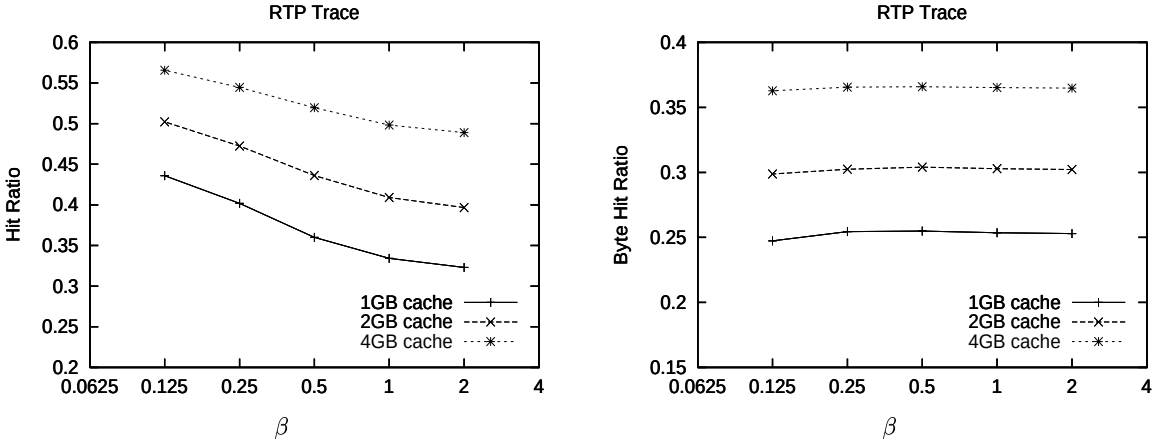

Figure 5: The effects of solely incorporating β into GDS(packets)