SECOND QUARTER 2004, VOLUME 6, NO. 2

IEEE COMMUNICATIONS SURVEYS

The Electronic Magazine of Original Peer-Reviewed Survey Articles

www.comsoc.org/pubs/surveys

AN OVERVIEW OF WEB CACHING REPLACEMENT ALGORITHMS

ABDULLAH BALAMASH AND MARWAN KRUNZ, UNIVERSITY OF ARIZONA

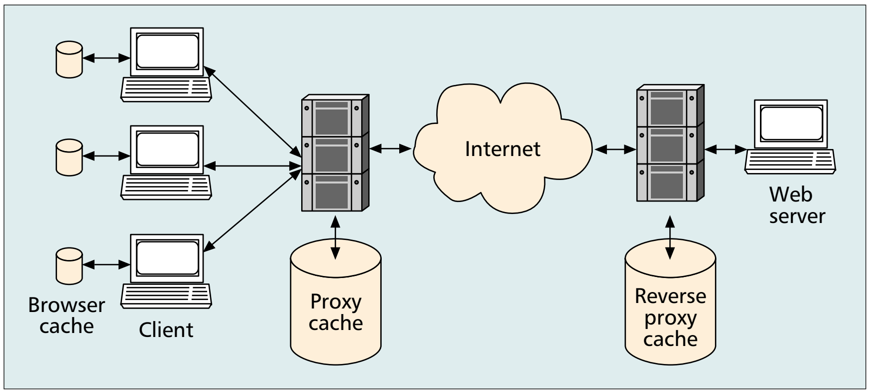
ABSTRACT

The increasing demand for World Wide Web (WWW) services has made document caching a necessity to decrease download times and reduce Internet traffic. To make effective use of caching, an informative decision has to be made as to which documents are to be evicted from the cache in case of cache saturation. This is particularly important in a wireless network, where the size of the client cache at the mobile terminal (MT) is small. Several types of caching are used over the Internet, including client caching, server caching, and more recently, proxy caching. In this article we review some of the well known proxy-caching policies for the Web. We describe these policies, show how they operate, and discuss the main traffic properties they incorporate in their design. We argue that a good caching policy adapts itself to changes in Web workload characteristics. We make a qualitative comparison between these policies after classifying them according to the traffic properties they consider in their designs. Furthermore, we compare a selected subset of these policies using trace-driven simulations.

In recent years, the WWW has become an essential tool for interaction among people and for providing a wide range of Internet-based services, including shopping, banking, entertainment, etc. As a consequence, the volume of transported WWW traffic has been increasing at a fast rate. Such rapid growth has made the network prone to congestion and has increased the load on servers, resulting in an increase in the access times of WWW documents.

WWW caching provides an efficient remedy to the latency problem by bringing documents closer to clients. Caching can be deployed at various points in the Internet: within the client browser, at or near the server (reverse proxy) to reduce the server load, or at a proxy server. A proxy server is a computer that is often placed near a gateway to the Internet (Fig. 1) and that provides a shared cache to a set of clients. Client requests arrive at the proxy regardless of the Web servers that host the required documents. The proxy either serves these requests using previously cached responses or obtains the required documents from the original Web servers on behalf of the clients. It optionally stores the responses in its cache for future use. Hence, the goals of proxy caching are twofold: first, proxy caching reduces the access latency for a document; second, it reduces the amount of "external" traffic that is transported over the wide-area network (primarily from servers to clients), which also reduces the user's perceived latency. A proxy cache may have limited storage in which it stores "popular" documents (documents that users tend to request more frequently than other documents). For example, in a wireless ad hoc network, each mobile terminal (MT) is equipped with a small storage space that enables the MT to act as a proxy server for a group of neighboring devices [1]. Whenever the cache is full and the proxy needs to cache a new document, it has to decide which document to evict from the cache to accommodate the new document. The policy used for the eviction decision is referred to as the *replacement policy*. The topic of WWW caching for wireless users is rather new, and not much work has been done in this area. It can be argued, however, that the extensive research on caching for wireline networks can be adapted for the wireless environment with modifications to account for MT limitations and the dynamics of the wireless channel. These limitations include the MT's limited battery life and its small cache size. To the best of our knowledge, no work has dealt specifically with caching policies for the wireless environment. In fact, while much work has been done on caching for wireless networks (e.g., [2–5]), only a few papers have been published specifically on WWW caching. In [1] the authors investigated the design of a cooperative WWW caching for ad hoc networks, in which neighboring MTs exchange information about their cache contents. Although the work suggested a caching policy that considers the energy cost associated with getting the document from a remote server in its replacement decision, the details of the policy were left for a future work.

Caching policies for traditional memory systems do not necessarily perform well when applied to WWW traffic for the following reasons:



■ FIGURE 1. *Possible locations for deploying WWW caching.*

- In memory systems, caches deal mostly with fixed-size pages, so the size of the page does not play any role in the replacement policy. In contrast, WWW documents are of variable size, and document size can affect the performance of the policy.
- The cost of retrieving missed WWW documents from their original servers depends on several factors, including the distance between the proxy and the original servers, the size of the document, and the bandwidth between the proxy and the original servers. Such dependence does not exist in traditional memory systems.
- WWW documents are frequently updated, which means that it is very important to consider the document expiration date at replacement instances. In memory systems, pages are not generally associated with expiration dates.
- The popularity of WWW documents generally follows a Zipf-like law (i.e., the relative access frequency for a document is inversely proportional to the "rank" of that document) [6]. This essentially says that popular WWW documents are very popular and a few popular documents account for a high percentage of the overall traffic. Accordingly, document popularity needs to be considered in any Web caching policy to optimize a desired performance metric. A Zipf-like law has not been noticed in memory systems.[1]

Several WWW replacement policies have been proposed in the literature. Such policies attempt to optimize various performance metrics, including the file hit ratio, the byte hit ratio, the average download time, and the "delay saving ratio" [7, 8]. Table 1 describes some of these metrics.

Replacement policies rely on key metrics (parameters) to achieve their goals. Many of them use the recency or frequency information of past references; both properties are well exhibited in WWW traffic [9–11]. For example, the well known least recently used (LRU) caching policy employs the time since last access as its only parameter. Some policies combine both recency and frequency information, along with some other parameters such as the size of the document and the cost associated with each document. Since WWW documents are of variable size, two documents with different sizes and with the same likelihood of being referenced can have different costs. The cost of a document includes the time and processing overhead associated with retrieving the document from the original server. The lifetime of the document and the cache space overhead associated with the document size are also considered as cost factors. Table 2 summarizes some of the parameters used in cache replacement policies.

In [12] Wang provided a good survey of Web caching schemes. However, the survey does not discuss the replacement policies in detail. Rather, it addresses several topics related to Web caching, including cache architectures, protocols, replacement policies, prefetching, cache coherency, proxy placement, user access prediction, and dynamic objects caching.

In this article we review a representative set of WWW proxy cache replacement policies. There are other policies that are not discussed in this article because they represent variations to the policies that we survey. Information about these policies can be found in the cited references. We provide a general comparison between these policies based on the criteria used for evicting documents. We divide these policies into two main groups: deterministic policies and randomized policies. Randomized algorithms use some sort of randomness in the selection criteria to reduce the running-time complexity. Such randomness is usually based on some parameters, including the time since last access, the number of references, and the document size. On the other hand, deterministic algorithms use a deterministic manipulation of such parameters in the decision making. We describe each category and survey some of the representative policies from each category. Moreover, we make a qualitative comparison between these policies, and support our discussion with simulation results and citations from the literature.

DETERMINISTIC POLICIES

One popular group of deterministic replacement policies are key-based. In these policies, one or more keys are used in the decision-making in a prioritized fashion. A primary key (or

Metric	Definition
File hit ratio	$\dfrac{\sum_{i\in R} h_i}{\sum_{i\in R} f_i}$
Byte hit ratio	$\dfrac{\sum_{i\in R} s_i \cdot h_i}{\sum_{i\in R} s_i \cdot f_i}$
Saved bandwidth	Directly related to byte hit ratio
Delay saving ratio	$\dfrac{\sum_{i\in R} d_i \cdot h_i}{\sum_{i\in R} d_i \cdot f_i}$ (validation delay is ignored)
Average download time	$\dfrac{\sum_{i\in R} d_i \cdot (1-h_i/f_i)}{\lVert R\rVert}$
Notation: s_i = size of document i; f_i = total number of requests for document i; h_i = total number of hits for document i; d_i = mean fetch delay from server for document i; R = set of all accessed documents; $\lVert R\rVert$ = size of R	

■ **Table 1.** *Examples of performance metrics used in cache replacement policies.*

[1] *While memory systems are known to exhibit strong temporal locality, this concept is different from document popularity.*

Parameter	Rationale
Last access time	Web traffic exhibits strong temporal locality.
Number of previous accesses	Frequently accessed documents are likely to be accessed in the near future.
Average retrieval time	Caching documents with high retrieval times can reduce the average access latency.
Document size	Caching small documents can increase the hit ratio.
"Expires" or "last modified" HTTP header values	Caching an expired document wastes cache space and results in a miss when the document is accessed.

Table 2. *Examples of commonly used parameters (keys) in cache replacement policies.*

parameter) is used to decide which document to evict from the cache in case of cache saturation. Additional keys are used to break ties that may arise during the selection process. Classical replacement policies, such as the LRU and the least frequently used (LFU) policies, fall under this category. LRU evicts the least recently accessed document first, on the basis that the traffic exhibits temporal locality. Intuitively, the farther in time a document has last been requested, the less likely it will be requested in the near future. LFU evicts the least frequently accessed document first, on the basis that a popular document tends to have a *long-term* popularity profile. Other key-based policies (e.g., SIZE [13] and LOG2-SIZE [14]) consider document size as the primary key (large documents are evicted first), assuming that users are less likely to re-access large documents because of the high access delay associated with such documents. SIZE considers the document size as the only key, while LOG2-SIZE breaks ties according to $\lfloor \log_2 (\text{document size}) \rfloor$, using the last access time as a secondary key. Note that LOG2-SIZE is less sensitive than SIZE to small variations in document size (e.g., $\lfloor \log_2 1024 \rfloor = \lfloor \log_2 2040 \rfloor = 10$). The LRU-threshold and the LRU-MIN [14] policies are variations of the LRU policy. LRU-threshold works the same way as LRU except that documents that are larger than a given threshold are never cached. This policy tries to prevent the replacement of several small documents with a large document by enforcing a maximum size on all cached documents. Moreover, it implicitly assumes that a user tends not to re-access documents greater than a certain size. This is particularly true for users with low-bandwidth connections. LRU-MIN gives preference to small-size documents to stay in the cache. This policy tries to minimize the number of replaced documents, but in a way that is less discriminating against large documents. In other words, large documents can stay in the cache when replacement is required as long as they are smaller than the incoming one. If an incoming document with size S does not fit in the cache, the policy considers documents whose sizes are no less than S for eviction using the LRU policy. If there is no document with such size, the process is repeated for documents whose sizes are at least $S/2$, then documents whose sizes are at least $S/4$, and so on. Effectively, LRU-MIN uses $\lfloor \log_2 (\text{document size}) \rfloor$ as its primary key and the time since last access as the secondary key, in the sense that the cache is partitioned into several size ranges and document removal starts from the group with the largest size range. The difference between LOG2-SIZE and LRU-MIN is that cache partitioning in LRU-MIN depends on the incoming document size and LOG2-SIZE tends to discard larger documents more often than LRU-MIN. Hyper-G [13] is an extension of the LFU policy, where ties are broken according to the last access time. Note that under the LFU policy, ties are very likely to happen. Pitkow/Recker [13] is an LRU policy that operates on a diurnal cycle; documents accessed on the same day are assumed to have the same recency. For same-day accesses, the largest document is evicted first. This policy seems to implicitly assume that client interests change from one day to another, and on a given day the client tends to concentrate his/her requests on a given set of documents.

The main goal of the previously discussed policies is to increase the file (or byte) hit ratio. In contrast, the lowest-latency-first policy [15] considers the document download time as its primary and only key (the document with the lowest download time is evicted first), so as to minimize the average download latency.

Function-based policies are another type of deterministic policy. These policies are also key-based, but with multiple keys used combinedly in a balanced manner, that is, there is no sequential ordering of these keys. Instead, the keys can have different weights in the cost function. All function-based policies aim at retaining the most valuable documents in the caches, but may differ in the way they define the cost function. Weights given to different keys are based on their relative importance and the optimized performance metric. Since the relative importance of these keys can vary from one WWW stream of requests to another or even within the same stream, some policies adjust the weights dynamically to achieve the best performance.

The GreedyDual algorithms [16] constitute a broad class of algorithms that include a generalization of LRU (GreedyDual-LRU). GreedyDual-LRU is concerned with the case in which different costs are associated with fetching documents from their servers. Several function-based policies are designed based on GreedyDual-LRU. They include the GreedyDualSize (GDS) [17], the Popularity-Aware Greedy DualSize (PGDS) [11], and the GreedyDual* (GD*) [10] policies. Other function-based policies are based on "classical" algorithms (e.g., LRU). These policies include the Size Adjusted LRU (SLRU) policy [18]. In the following sections we present some of the well known function-based policies. Throughout the sections we use S_p and C_p to indicate the size and cost of a document p, respectively. The capacity of the cache (in bytes) is indicated by S.

GreedyDual-Based Policies

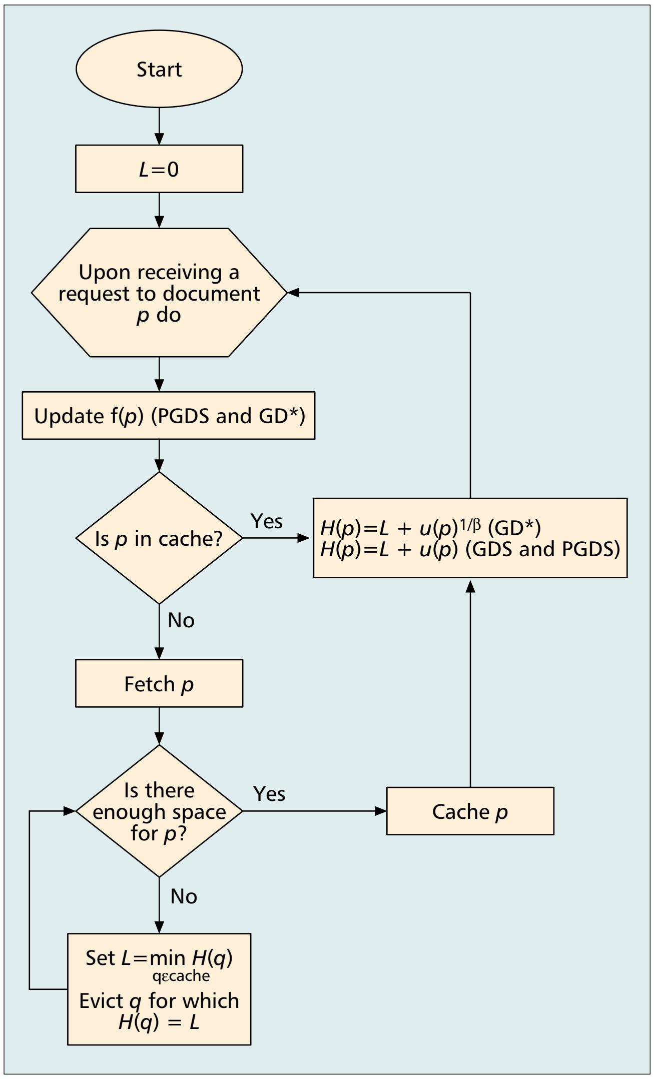
Cao and Irani extended GreedyDual-LRU to incorporate varying WWW document sizes. In GreedyDual-LRU, each cached document, p, is assigned a cost value $H(p)$. Initially, $H(p)$ is the cost of bringing document p to the cache. Every time a document is accessed, its H value is set to the cost of bringing that document to the cache. Once there is a need to evict a document from the cache, the document with the lowest H value, denoted by H_{min}, is evicted and the H values of all other documents are reduced by H_{min}. Thus, the H values of recently accessed documents retain higher fractions of their initial costs than those that have not been accessed for a longer time (i.e., the document value reflects the recency information plus the document cost).

GDS extends GreedyDual-LRU by incorporating in the H value the size of a document. Initially, $H(p)$ is assigned the utility value $u(p) \overset{\Delta}{=} C_p/S_p$. So a document A is less valuable than a document B that has the same cost if A is larger in size than B. The cost C_p is defined according to the goal of the caching policy. It is set to 1 if the goal is to maximize the hit ratio, or it can be set to the document download latency if the goal is to minimize the average download latency. Setting $C_p = 1$ enables the policy to discriminate against large documents, allowing for smaller documents to be cached. Size discrimination combined with the adoption of an aging mechanism (through the retained recency information) can maximize the hit ratio. Without the aging mechanism, the cache can be polluted by small documents that are never requested, as can happen in the SIZE policy [19, 20]. To reduce the complexity of the GDS algorithm (the need to subtract H_{min} from the H values of all the documents that are retained in the cache during the eviction process), an inflation value L is defined at the replacement decision time and is initially set to H_{min}. Once a cached document p is accessed, its H value is reset to its initial value C_p/S_p plus L. The H values of the other cached documents are not changed.

The popularity profile of WWW accesses is known to approximately follow a Zipf-like distribution [6], whereby the access frequency of a document is proportional to the rank of that document. This means that a popular document is very popular. Such a characteristic suggests the use of the *long-term* frequency information (in contrast, the temporal locality in a request stream [21] suggests the use of *short-term* recency information). Jin and Bestavros took the GDS policy one step further by incorporating into it the popularity profile of WWW documents. The result of this extension is the PGDS algorithm. In PGDS, a utility value $u(p) \overset{\Delta}{=} f(p)C_p/S_p$ is assigned to each document p, where $f(p)$ is the access frequency (i.e., popularity) of document p. Access recency is captured through an inflation variable, L, as in the GreedyDualSize policy. Each document p in the cache is initially assigned the value $H(p) = u(p)$. In case of cache saturation, the document with the lowest H value (H_{min}) is evicted, and the inflation variable is assigned the value H_{min}. Each time a document is accessed, its H value is set to its utility value plus L. To compute $f(p)$, preference is given to more recent references to avoid polluting the cache with previously popular documents that are no longer popular. This is done by de-emphasizing the significance of old accesses by scaling down the count of old references by half every two days (experimental results showed that the relative significance of past accesses should not be down-weighted too quickly [11]).

In [9, 11] the authors studied the temporal locality in WWW traffic and concluded that such a phenomenon is induced by both temporal correlations and long-term popularity. More specifically, references to long-term popular documents tend to be close to each other in time. Moreover, references to certain unpopular documents (in the long term) exhibit strong temporal correlations, as these references appear "clustered" in time (e.g., a document is repetitively requested but only during a short period of time). It is important to differentiate between the two sources of temporal locality since this can help the policy adapt itself based on the dominant source of temporal locality. Both sources are incorporated in the GD* policy. Temporal correlations are captured by modeling the probability distribution of the reference interarrival time for equally popular documents (to equalize the effect of popularity). For equally popular documents, the probability that the reference interarrival time equals t is roughly proportional to $t^{-\beta}$, where $0 < \beta < 1$.

In essence, GD* is an extension of the PGDS policy, in which a utility value u is defined for every accessed document. The utility value is adjusted to reflect the degree of temporal correlations found in the traffic history (old references). Based on the distribution for the reference interarrival times for equally popular documents, which is proportional to $t^{-\beta}$, the authors argued that the maximum time a document p


Start
L=0
Upon receiving a request to document p do
Update f(p) (PGDS and GD*)
Is p in cache?
Yes
No
H(p)=L + u(p)^1/β (GD*)
H(p)=L + u(p) (GDS and PGDS)
Fetch p
Is there enough space for p?
Yes
No
Cache p
Set L=min H(q) q∈cache
Evict q for which H(q) = L

■ **FIGURE 2.** *GreedyDualSize, PGDS, and GD* policies.*

stays in the cache is proportional to $u^{1/\beta}$ (the higher the value of β, the weaker are the temporal correlations). As in PGDS, every accessed document p is assigned a value $H(p)$, which is equal to the utility value plus an inflation value L. Documents are replaced according to their H values (the document with the lowest H value is evicted first). The smaller the H value, the higher is the probability that the document will be evicted in case of cache saturation. Once a document is evicted, its H value is assigned to the inflation value L. The parameter β is computed online using a least-square fit. Figure 2 depicts the general operation of GDS, PGDS, and GD*.

LEAST RELATIVE VALUE (LRV) POLICY

Rizzo and Vicisano [22] performed a comprehensive statistical analysis of several WWW traffic traces, looking for the suitable parameters that can be used to design an adaptive replacement policy. They came up with the LRV policy. As in other function-based policies, LRV assigns a value $V(p)$ for each document p. Initially, $V(p)$ is set to $C_p P_r(p)/G(p)$, where $P_r(p)$ is the probability that document p will be accessed again in the future starting from the current replacement time and $G(p)$ is a quantity that reflects the gain obtained from evicting document p from the cache ($G(p)$ is related to the size S_p). As a result of this choice, the value of any document is weighted by its access probability, meaning that a valuable document (from the cache point of view) that is unlikely to be re-accessed is actually not valuable. It was assumed that the propagation delay is the same for all documents, irrespective of the location of the server. Accordingly, C_p depends only on S_p, and thus $V(p)$ is mainly dependent on $P_r(p)$. In turn, $P_r(p)$ depends on the time since the last access to document p (T_p), the number of previous accesses to document p (n_p), and S_p, that is, $P_r(p) = P_r(n_p, T_p, S_p)$. Let I be the inter-access time for an *arbitrary document* given that this document will be re-accessed, and let f_I and F_I be the pdf and CDF of I, respectively. The computation of $P_r(n_p, T_p, S_p)$ is performed as follows. First, based on extensive trace analysis, F_I is approximated as

$$F_I(x) = c * \log\left(\frac{f(x)+\tau_1}{\tau_2}\right),$$

where $f(x) \overset{\Delta}{=} \tau_2(1 - e^{-x/\tau_2})$, τ_1 is a parameter that reflects the overall periodicity of references to popular documents, τ_2 is a parameter that reflects the long-term decay in the distribution F_I (the tail of the distribution), and c is a constant. These parameters are computed dynamically. The size of a document is found to play a role in accessing a document for the second time, but once a document has been accessed more than once, its size has almost no effect on the number of future accesses. The authors argued that $P_r(n_p, T_p, S_p)$ can be computed as follows:

$$P_r(n_p, T_p, S_p) = \begin{cases} \Psi(1, S_p)(1 - F_I(T_p)), & n_p = 1 \\ \Theta(n_p)(1 - F_I(T_p)), & \text{otherwise.} \end{cases}$$

where $\Theta(n_p)$ is the probability of re-accessing document p after it has been accessed n_p times in the past, where $n_p > 1$, and $\Psi(1, S_p)$ is the probability of re-accessing a document of size S_p after its first access. The ratio of the number of documents seen so far that have each been requested at least $n_p + 1$ times to the number of documents that have each been requested at least n_p times is equal to $\Theta(n_p)$. As for $\Psi(1, S_p)$, it is computed as the ratio of the number of documents of size S_p seen so far that have each been accessed at least two times to the number of documents of size S_p that have each been accessed at least one time. The expression for P_r says that documents with the same number of accesses that is greater than 1 are LRU ordered, and so are the documents with exactly one access and that have the same size. This is clear from the above equation where $1 - F_I(T_p)$ is a monotonically decreasing function of the inter-access time.

The above computations need to be performed for each document p in the cache at each replacement time. To reduce the complexity of the algorithm, documents that have been accessed only once are classified into k groups based on their sizes, and documents with more than one access are classified into 10 groups according to the number of previous accesses. Because $\Theta(n_p)$ was found to saturate quickly for $n_p > 10$, the last group contains all documents with more than 10 accesses. Documents in each group are ordered using a separate LRU stack. Any document at the bottom of an LRU stack has a smaller $1 - F_I(T_p)$ value than any other document in the same stack, which means that it has a lower P_r value than the other documents in its respective stack. Since the number of groups ($k + 10$) is small and fixed, the replacement decision requires a fixed amount of computational time. In case of cache saturation, the P_r values for documents at the bottom of each LRU stack are recomputed, and the one with the lowest value is replaced first. Once a cached document is accessed, it is moved to the top of the LRU stack of the next higher group.

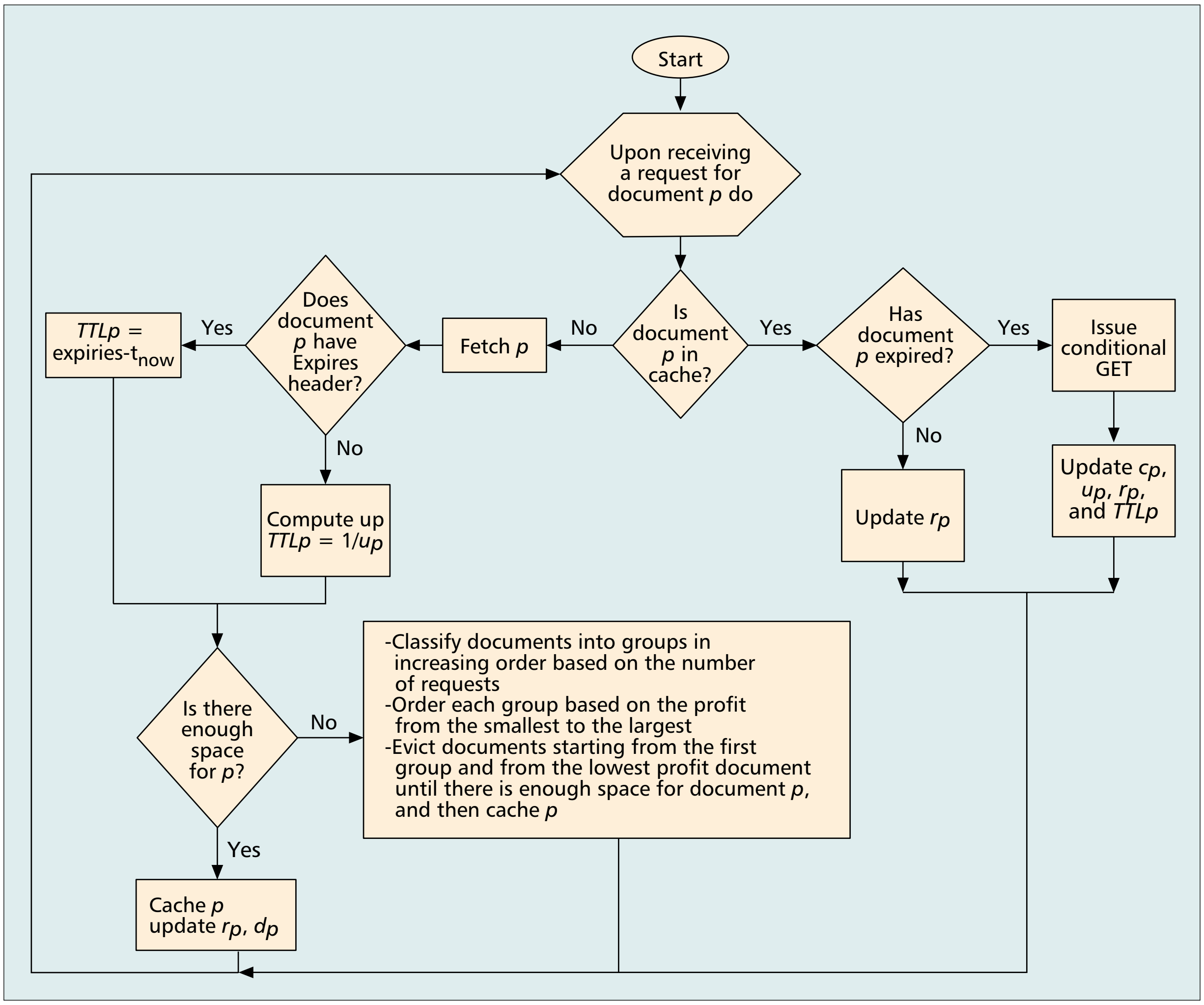
LNC-R-W3-U Policy

Shim *et al.* [8] designed and implemented a cache policy called the *least normalized cost replacement for the Web with updates* (LNC-R-W3-U). This policy is based on the LRU-K buffer caching policy [23], integrating both cache replacement and consistency (the two were treated independently in previous policies). Consistency mechanisms try to keep cached documents up to date, which is important for frequently updated WWW documents. LNC-R-W3-U gives preference to documents that are infrequently updated and that have high download latencies. The replacement policy is based on the following optimization problem, in which the cache retains documents that account for a large fraction of the communication delay: $\max \Sigma_{p \in Cache}(r_p d_p - u_p c_p)$ subject to $\Sigma_{p \in Cache} S_p \leq S$, where r_p and u_p are the mean reference rate and the mean validation rate for document p, respectively; and d_p and c_p are the delays to fetch document p from the original server and to validate document p, respectively. The mean reference rate for a given document is the average number of references to that document over a fixed period of time, while the mean validation rate is the average number of validation requests over a fixed period of time. The above optimization problem is equivalent to the well known knapsack problem, in which the goal is to retain documents in the cache that maximize the total cost. The knapsack problem is NP-hard (i.e., it cannot be solved in polynomial time). One of the well known heuristics to address this problem is to keep in the cache documents with higher cost per size. The intuition behind this heuristic is that each document can be viewed as consisting of several small pieces, each of a fixed size and a cost that is equal to the cost of the document divided by its size. Using this heuristic, a profit value is defined for each document p: $profit_p \overset{\Delta}{=} (r_p d_p - u_p c_p)/S_p$. At the replacement time, the document with the lowest profit is replaced first.

The mean delay to fetch a document (d_p) and the mean delay to validate a document (c_p) are computed using a moving average of recently measured values. The mean reference rate r_p is computed using a sliding window: $r_p = k/(t - t_{k,p})$, where t is the current time and $t_{k,p}$ is the time of the kth most recent reference to document p. To improve the precision of computing r_p, the authors used a result from [24], in which r_p was set to $c/(S_p^b)$, where b and c are constants. This estimation assumes that the document size and the likelihood of re-accessing this document are correlated. The two above estimates of r_p are combined as $r_p = k/((t - t_{k,p})S_p^b)$. The validation rate u_p is computed from the Expires header (date at which a document expires) that is provided by the HTTP server, using a sliding window: $u_p = k/(t_{r,p} - tu_{k,p})$, where $t_{r,p}$ is the most recent Expires and $tu_{k,p}$ is the kth most recent Expires for document p. If the Expires information is not available (not every HTTP server response has this header), the Last-Modified time-stamp is used instead (all HTTP server responses have this header). In this case, $t_{r,p}$ is the time when the last version of the document was brought to the cache and $tu_{k,p}$ is the kth most recent Last-Modified time-stamp. Since this policy incorporates cache consistency, it sets the time to live (TTL) variable for each document in the cache based on the Expiries information, if available; otherwise, it sets the TTL based on the estimated mean invalidation rate u_p. Figure 3 depicts how the policy works and how the replacement decision is taken.

Size-Adjusted LRU (SLRU) Policy

In [18] Aggarwal *et al.* designed a size/cost aware LRU policy called SLRU. To better explain this policy, we first revisit the standard LRU policy. The underlying principle of the LRU


Start
Upon receiving a request for document p do
Is document p in cache?
No
Fetch p
Does document p have Expires header?
Yes
TTLp = expiries-t_now
No
Compute up TTLp = 1/up
Yes
Has document p expired?
Yes
Issue conditional GET
Update cp, up, rp, and TTLp
No
Update rp
Is there enough space for p?
No
-Classify documents into groups in increasing order based on the number of requests
-Order each group based on the profit from the smallest to the largest
-Evict documents starting from the first group and from the lowest profit document until there is enough space for document p, and then cache p
Yes
Cache p update rp, dp

■ **FIGURE 3.** *LNC-R-W3-U policy.*

policy is that a *dynamic frequency* $1/\Delta T_{p,k}$ is defined for every document that has been accessed more than once, where $\Delta T_{p,k}$ is the number of requested documents since the last access to document p at time k. The LRU policy removes objects starting from the one that has the smallest dynamic frequency. This dynamic frequency can be considered as the cost of purging a document from the cache. The goal is to reduce this cost. The LRU caching policy solves the following optimization problem: Minimize $\Sigma_{p\in\Omega(k)} Y_p/\Delta T_{p,k}$ such that $\Sigma_{p\in\Omega(k)} S_p Y_p \geq S$, where $\Omega(k)$ is the set of documents in the cache at time k and Y_p is a decision variable set to 1 if we decide to purge document p, and set to 0 if we decide to keep the document in the cache. The standard LRU policy works as if all documents have the same size (S_p is constant), making the optimization problem trivial (it can be solved by ordering the documents in an increasing order of their dynamic frequencies). In the case of SLRU, this optimization problem becomes a knapsack problem. As we indicated before, a fast heuristic for solving this problem is to order documents according to the cost/size ratio, $1/(\Delta T_{p,k} S_p)$, and evict the document with the lowest ratio first to accommodate a newly arrived document.

SLRU is a complex algorithm. At each replacement point, the $1/(\Delta T_{p,k} S_p)$ values need to be computed for every document in the cache, and then the documents need to be sorted based on these values. A simpler implementation can be used, in which documents are classified into groups based on their sizes. Documents in group i have sizes in the range $[2^{i-1}, 2^i - 1]$. Documents in each group are LRU ordered. The $1/(\Delta T_{p,k} S_p)$ values of the least recently used documents in each group are compared to decide on which document to evict. Note that this classification of documents is the same as the one used in LOG2-SIZE, but with the difference that LOG2-SIZE always chooses the least recently accessed document from the group that has the largest size range. The simplified implementation of SLRU was found to select the document with the lowest $1/(\Delta T_{p,k} S_p)$ value within a factor of 1/2, even in the worst case.

LRU can be further extended to include the document access cost and the document expiration time by modifying the decision index $1/(\Delta T_{p,k} S_p)$ to become $C_p(1 - \gamma_p)/(\Delta T_{p,k} S_p)$, where γ_p is called the refresh overhead factor and is defined as $\gamma_p = \min\{1, \delta t_{p1}/\delta t_{p2}\}$, where δt_{p1} is the difference between the time when the document enters the cache (t) and the last access time of the document, and δt_{p2} is the difference between t and the document expiration time. The value of γ_p approximately represents the reciprocal of the number of expected accesses to document p before document p needs to be refreshed. The remaining TTL could be used instead, but this value is dynamic and needs to be computed for all documents in the cache at every replacement instance.

Both LRU and SLRU always cache a newly incoming document. Sometimes it is not worth doing so in the first place. SLRU can be extended to incorporate some admission criteria for the newly incoming document. The names and timestamps of the last few accessed documents, the access cost, the expiration time data, and the number of accesses to these documents are stored in an auxiliary cache. Once a document is accessed for the first time, it is cached if there is enough space to accommodate it. If the cache is full, then the auxiliary cache is checked, and if the name of the document is not in the auxiliary cache, the document does not enter the cache and its information is added to the auxiliary cache. If the name of the document is in the auxiliary cache and the document has been accessed for at least k times, then it is considered as a candidate for caching according to the SLRU replacement policy. Such an admission control policy ensures that an incoming document is popular enough to offset the loss due to the replaced document.

LEAST UNIFIED VALUE (LUV) POLICY

In [7] Bahn *et al.* tried to get the benefit of both LRU and LFU in one unified scheme. In their scheme, each document p in the cache is assigned a value

$$V_p(k) \triangleq \frac{C_p}{S_p} \sum_{i=1}^{k} F(\Delta_{k,p}),$$

where $\Delta_{k,p}$ is the time since the kth reference to document p and F is a function that gives different weights to past references to document p. This definition includes both LRU and LFU policies as special cases, where in LRU the weights are 0 for all references other than the last reference, and in LFU the weights associated with all references are equal.

The two special cases described above can be combined by taking

$$F(\Delta_{k,p}) = \left(\frac{1}{p}\right)^{\lambda\Delta_{k,p}}, \quad p \geq 2.$$

If $\lambda = 0$, then $F(\Delta_{k,p})$ is constant, and if $\lambda = 1$, then

$$F(\Delta_{k,p}) > \sum_{y=\Delta_{1,p}}^{\Delta_{k-1,p}} F(y).$$

The range of values that λ can take $(0 \leq \lambda \leq 1)$ allows LUV to represent a wide range of caching policies that consider both the *short-term* recency information and the *long-term* frequency information of the accessed documents, with different emphasis based on the value of λ. If λ is close to 1, more emphasis is placed on the recency information; if λ is close to 0, the emphasis is shifted toward the frequency information. The value of a document p at the kth reference can be defined recursively as a function of its value at the $(k - 1)$th reference as follows: $Vp(k) = F(\text{time since last access})V_p(k-1) + C_p/S_p$. The manner in which the function F is defined guarantees that the relative ordering of documents does not change unless a document is accessed. So the value of a given document is updated only when the document is accessed. In the case of cache saturation, the document with the lowest value is evicted first. The parameter λ is computed offline in a way to maximize the desired performance metric.

HYBRID POLICY

Wooster and Abrams introduced the Hybrid policy [15], with the aim of reducing the access latency. This policy uses a utility function that depends on the originating server through the round trip delay (rtt_s) and the bandwidth (b_s) between the proxy cache and server s. For a document p, the utility function also depends on the number of times p has been requested since it was brought to the cache ($nref_p$) and the size of the document. More specifically, this utility function is defined as: $u(p) = (rtt_s + W_b/b_s) * (nref_p)^{W_n}/S_p$, where W_b and W_n are constants. The value of W_b is selected based on the importance of the connection time relative to the connection bandwidth, while the value of W_n quantifies the importance of the frequency information relative to the size of the document. The values of rtt_s and b_s are estimated based on the time to fetch documents from server s. Hybrid evicts the document with the smallest utility value first.

MIX POLICY

The Mix policy [25] is an extension of the Hybrid policy, with one more parameter added, namely, the time elapsed since the last reference to document p ($tref_p$). The utility function in Mix is defined as:

$$\frac{(lat_p)^{r_1} * (nref_p)^{r_2}}{(tref_p)^{r_3} * (S_p)^{r_4}},$$

where lat_p is the download latency of the last access to document p. The exponents r_1, r_2, r_3, and r_4 are determined experimentally.

LOGISTIC REGRESSION ALGORITHM (LR) POLICY

In [26] Foong *et al.* used a logistic regression model to acquire knowledge about WWW documents in a given trace and predict the expected distance to the next access for any document in the cache. Logistic regression can be used to model the probability of an event as a function of some factors (predictors). In the case of WWW traffic, the factors that were considered are the document size, its type (i.e., HTML, image, etc.), the number of accesses in a backward window W_b, and the elapsed time since the last access. The logistic regression probability is given by:

$$\begin{aligned} P_{LR} &= P(event\ Y \mid predictors = \{X_1, X_2, \ldots, X_k\}) \\ &= \frac{1}{1+\exp(-z)} \end{aligned}$$

where $z = \Sigma_{i=0}^{k} \beta_i X_i$ and β_i is X_i's respective coefficient. We can think of β_i as a weight given to the predictor X_i. The optimal offline replacement algorithm for fixed-size objects is the well-known Longest Forward Distance (LFD) algorithm [27], which replaces the document whose next access is farthest in the future. To generalize this algorithm to the non-uniform size case, the expected distance to the next access is predicted and is weighted by the size of the document. The document whose weighted expected distance to the next access is the highest is replaced first.

The probability P_{LR} is taken as the probability of accessing a document p at least once in the next N requests. It is computed by the regression model. The probability that document p is not accessed in the next N references is given by $(1 - q)^N = 1 - P_{LR}$, where q is the probability of accessing document p next. The number of references until the next reference to a given document is a geometric random variable of mean $E[L(p)] = 1/q = 1/(1 - (1 - P_{LR})^{1/N})$, and the weighted expected distance is $E[L(p)]/S_p$. The algorithm runs in two phases: a learning phase and a prediction phase. Experimentally, N is selected to be 50. Each time the cache gets saturated, the policy simply works by evicting the document with the highest weighted distance. The parameters of the algorithm are updated periodically (in the learning phase).

RANDOMIZED POLICIES

In general, function-based deterministic policies require complex data structures and incur high computational overhead, which limits the scalability of these policies. An alternative approach is to rely on randomized algorithms. A simple example of a randomized replacement policy is to evict a document drawn randomly from the cache. Starobinski *et al.* [28] designed a group of simple randomized algorithms that extend two key-based algorithms, LRU and CLIMB, by incorporating the size, cost, or both in a randomized fashion. Both LRU and CLIMB replace the least recently accessed document first. However, when a cached document is accessed, LRU brings it to the top of the cache while CLIMB exchanges the position of that document with the one ahead of it. To incorporate the document size, the randomized LRU-SIZE algorithm [28] brings any requested document p to the top of the stack with probability min(1, S_1/S_p). For the randomized CLIMB-SIZE algorithm [28], once a document p is accessed, its position is exchanged with the one ahead of it with probability min(1, S_{p-1}/S_p). The cost can be incorporated in a similar way. To incorporate both size and cost, a value $\beta_p = C_p/S_p$ is defined for every document p. In the randomized version of LRU, any accessed document p is brought to the top of the stack with probability min(1, β_p/β_1), while in the randomized version of CLIMB, the accessed document is exchanged with the one ahead of it with probability min(1, β_p/β_{i-1}). Eviction is done the same way as in the standard LRU and CLIMB policies (evict the document at the bottom of the stack first), but with the difference that the incoming document can be evicted in the first place (i.e., not admitted).

The randomized algorithm by Posounis *et al.* [29] tries to approximate any existing function-based algorithm by employing randomization. We will refer to this algorithm as RANDOM. RANDOM works by randomly selecting N documents from the cache at the time of replacement, and then selecting the least useful document among them for replacement based on the given policy. The next M least useful documents out of N are kept for the next iteration. The iterations are continued until there is enough space in the cache to accommodate the remaining documents. The optimal value of M was found analytically to be approximately

$$N - \sqrt{(N+1)100/n}$$

where N is the length of the sample and n is a value that says an error has occurred if the evicted document does not belong to the least useful n percent of all documents. For example, for $N = 40$ and $n = 6$, the approximate value of M is 13.8. An algorithm somewhat similar to RANDOM was previously used in the Squid caching system [30].

Qualitative Assessment and Comparison

Replacement algorithms can, in principle, be analyzed by means of *competitive analysis* [16, 31, 32], where the performance of the replacement algorithm is compared with the performance of the optimal offline replacement algorithm. The difference between an offline algorithm and an online algorithm is that in the offline algorithm future requests are known beforehand. The comparison is done based on the cost of cache misses (a cache miss incurs a unit cost). An algorithm is said to be k-competitive if the cost of cache misses is at most k times that under the optimal offline algorithm. Cao and Irani [17] showed that the GreedyDualSize policy is k-competitive, where k is the ratio of the cache size and the size of the smallest document. It was shown in [33] that deterministic online paging algorithms (fixed page size) can at best achieve a competitive ratio of k, where k is the number of pages the cache can hold. Furthermore, LRU was found to be optimal in the sense of being k-competitive. Since it is difficult to compute how competitive a given online algorithm is, especially in the case of non-uniform document sizes, almost all replacement policies proposed in the literature are evaluated using trace-driven simulation. Some of the works (e.g., [11, 17, 26]) relied on different traces to come up with an average assessment of the policy's performance over different workloads.

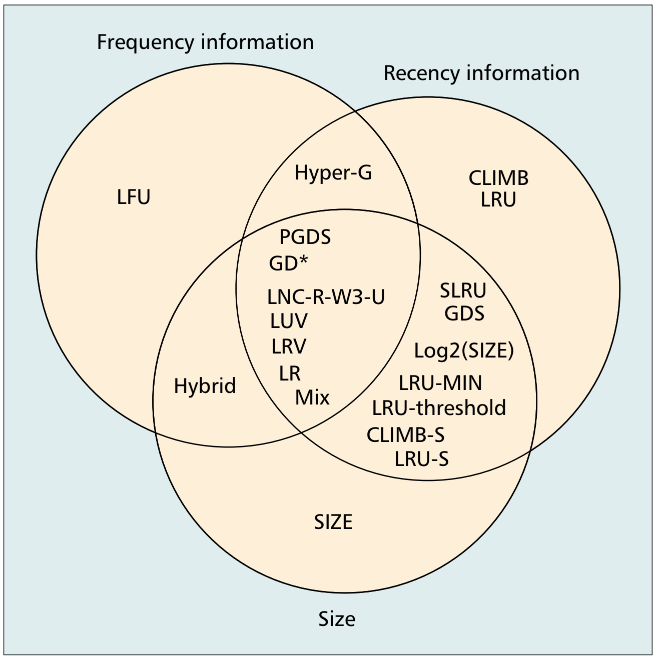
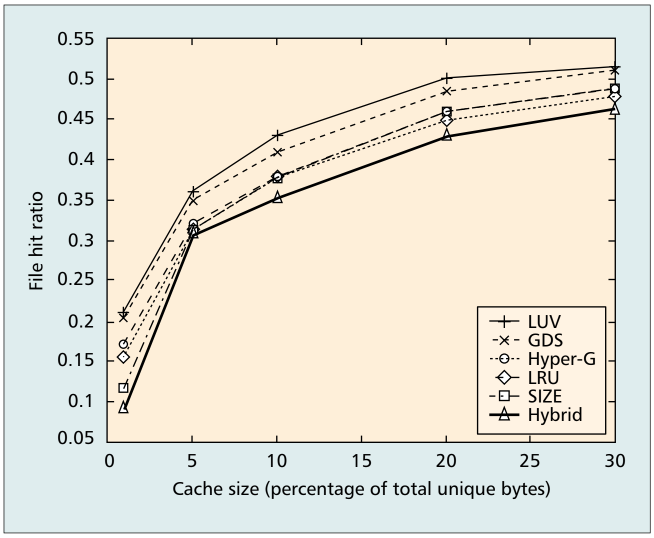


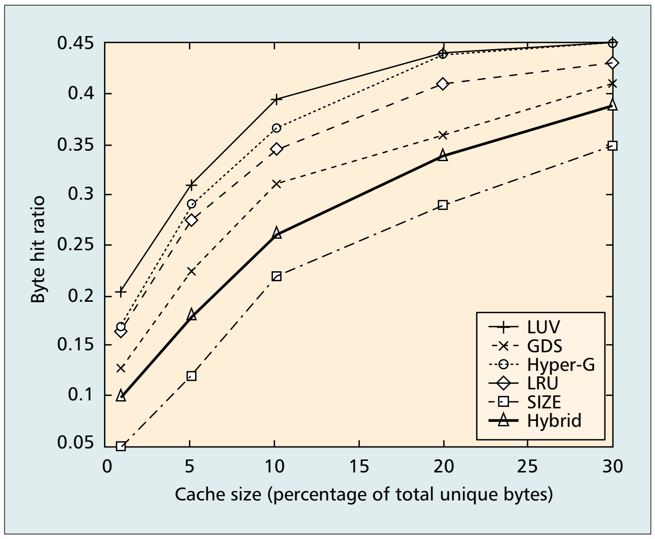
Figure 4. *Classification of caching policies according to the traffic information considered in their designs.*

Given the lack of competitive analysis for most of the surveyed works, an alternative would be to provide a quantitative comparison between them and rank them according to some performance metric. However, such a task is made difficult because these policies, in general, differ in their design objectives (i.e., they optimize different cost functions). Furthermore, their performance depends to varying degrees on the properties of the underlying WWW traffic, so the relative performance is likely to vary from one workload to another. This may be the main reason why in the literature, such policies are often contrasted against "classic" policies (i.e., LRU, LFU, and SIZE) and not against each other. One point of commonality, though, is the significance of the document size and the need to incorporate it in the design of the replacement policy [34].

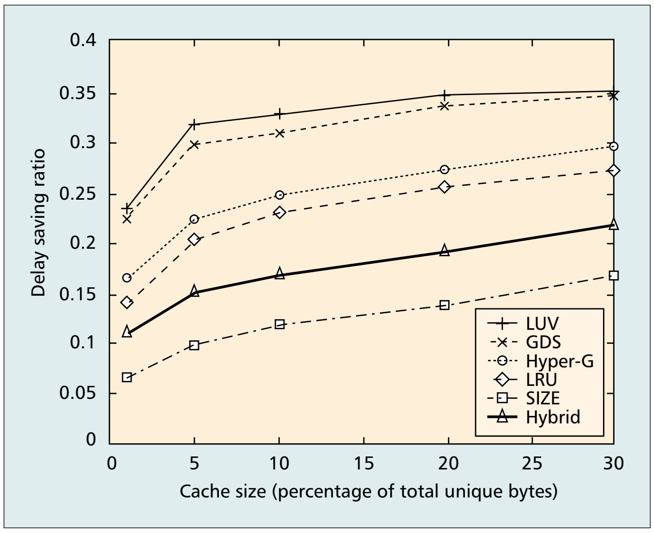
In this section, we first classify various caching policies according to the traffic properties these policies consider in their designs (irrespective of the performance metric used). We then try to compare policies within a given class, supporting our assessment, when possible, with references from the literature. Finally, we select one of the best policies in each class and compare the performance of the selected policies using trace-driven simulation. Figure 4 depicts our traffic-based classification. Most of the surveyed policies consider the recency (temporal locality) or frequency (popularity) information of past references to decide which documents to keep in the cache, on the basis that a recently or frequently referenced document is more likely to be referenced again. Some policies consider both properties simultaneously. In this case, the relative importance of the two properties can be static (fixed), making the performance vary with the workload, or it can be dynamic, allowing the policy to adapt to changes in the traffic properties. GD* is an example of a policy in which the relative significance of the recency and frequency information



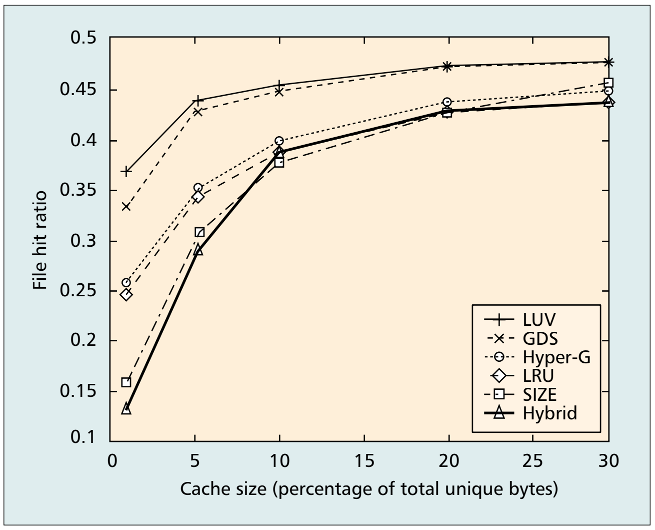
■ **FIGURE 5.** *File hit ratio for the NLANR trace.*



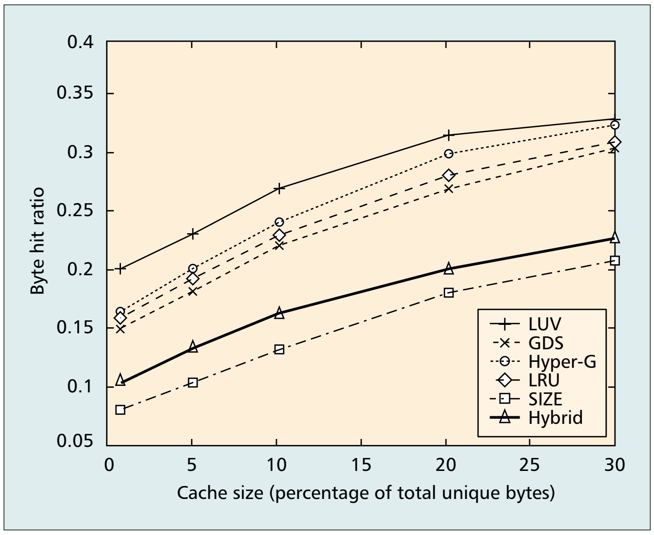
■ **FIGURE 6.** *Byte hit ratio for the NLANR trace.*



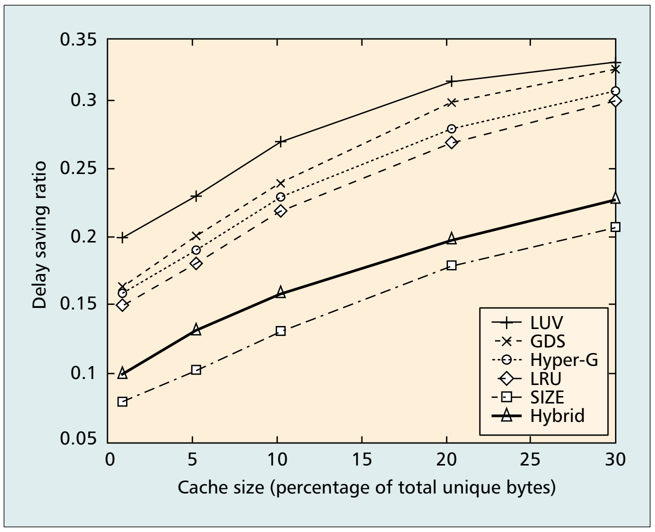
■ **FIGURE 7.** *Delay saving ratio for the NLANR trace.*



■ **FIGURE 8.** *File hit ratio for the DEC trace.*



■ **FIGURE 9.** *Byte hit ratio for the DEC trace.*



■ **FIGURE 10.** *Delay saving ratio for the DEC trace.*

vary with time. In principle, such adaptiveness improves the performance of the replacement policy.

Caching policies that rely on both the recency information and the document size include LOG2-SIZE, LRU-MIN, LRU-threshold, GDS (with uniform document cost), and SLRU. SLRU and GDS are expected to give the best file (or byte) hit ratio within this group, since the other policies first give preference to the document size and then consider the recency information (i.e., they are size-biased). In contrast, SLRU and GDS consider both properties in a more balanced manner [9, 11, 18]. However, the performance of SLRU and GDS will likely degrade if there is a strong negative correlation between the object size and the likelihood of it being referenced. This correlation, which is implicitly assumed in LRU-MIN, LRU-threshold, and LOG2-SIZE, was found to be very weak [6, 35], which supports our judgment that these policies are inferior to GDS and SLRU. Moreover, the authors in [18] showed that SLRU is superior to LRU-MIN, which itself was found to outperform LRU-threshold in most cases [14]. The LRU-MIN policy is less biased toward the document size compared with LRU-threshold, so in the absence of the above correlation LRU-

Trace	Trace period	All requests (all bytes)	Unique files (unique bytes)
DEC	9/1–9/10, 1996	6,056,025 (51.7GB)	3,070,404 (33.0GB)
NLANR	3/20–3/26, 2002	3,810,537 (30.6GB)	1,733,794 (15.6GB)

Table 3. *Characteristics of traces used in the simulations.*

Policy	Key parameters	Complexity	Eviction	Admission
LRU	• Time since last access	$O(1)$	The least recently accessed first	All
LFU	• Number of references	$O(\log(n))$	The least frequently accessed first	All
SIZE	• Document size	$O(\log(n))$	The largest first	All
Hyper-G	• Time since last access • Number of references	$O(\log(n))$	The least frequently accessed first, and then the least recently accessed first among equally accessed documents	All
Log2(SIZE)	• Document size • Time since last access	$O(\log(n))$	The largest first, and then the least recently accessed first among equal-sized documents	All
LRU-threshold	• Document size • Time since last access	$O(1)$	The least recently accessed first	Documents smaller than a certain size
LRU-MIN	• Document size • Time since last access	$O(n)$	The least recently accessed with size greater than S, the least recently accessed with size greater than $S/2$, the least recently accessed with size greater than $S/4$, and so on, where S is the size of incoming document	All
GDS	• Document size S_p • Document cost C_p • An inflation value L	$O(\log(n))$	Least valuable first according to $value_p = C_p/S_p + L$	All
PGDS	• Document size S_p • Document cost C_p • Number of non-aged references f_p • Time since last access • An inflation value L	$O(\log(n))$	Least valuable first according to $value_p = C_p \cdot f_p/S_p + L$	All
GD*	• Document size S_p • Document cost C_p • Number of non-aged references f_p • Time since last access • Temporal correlation measure β • Interaccess time for some of the equally popular documents • An inflation value L	$O(\log(n))$	Least valuable first according to $value_p = (C_p \cdot f_p/S_p)^{\beta} + L$	All
Hybrid	• Document size S_p • Number of references to document p since brought to cache • Bandwidth between proxy and server b_s • Round trip delay rtt_s • Some constant values W_b and W_n	$O(\log(n))$	Least valuable first according to $value_p = (rtt_p + W_b/b_s) \cdot (nref_p)^{W_n}/S_p$	All

Table 4. *Summary of cache replacement policies.*

MIN is expected to outperform LRU-threshold. LRU-threshold can outperform LRU-MIN for small cache sizes [14]. LOG2-SIZE falls in between LRU-MIN and LRU-threshold in terms of discriminating against large documents, so it is expected to perform better than LRU-threshold and to perform worse than LRU-MIN. LRU-threshold has a threshold parameter that is selected offline, which can highly affect the performance of the policy. Although both SLRU and GDS utilize the recency information and document size, GDS is expected to outperform SLRU because of the approximation done in the implementation of SLRU, as was described before.

PGDS, GD*, LNC-R-W3-U (with uniform access delay), LRV, LR, and LUV are comparable policies, since they all utilize recency, frequency, and size information. They differ in that some of them are adaptive while others are not. GD* is an adaptive policy that tries to dynamically balance the two sources of temporal locality of referenced documents (temporal correlations and long-term popularity [11, 36]), which allows it to outperform PGDS [10, 11]. LUV and LNC-R-

Policy	Key parameters	Complexity	Eviction	Admission
Mix	• Document size S_p • Time since last access $tref_p$ • Download latency of last access to p, lat_p • Constant weights r_1, r_2, r_3, and r_4 • Number of references to document p since brought to cache	$O(\log(n))$	Least valuable first according to $value_p = (lat_p^{r_1} \cdot nref_p^{r_2})/(tref_p^{r_3} \cdot S_p^{r_4})$	All
LNC-R-W3-U	• Document size S_p • Time since the kth most recent reference • Document access rate r_p • Document mean fetch delay d_p • Document update rate u_p • Document mean validation delay c_p • Document expiration time or last modified time	$O(n)$	Least valuable first according to $value_p = (r_p \cdot d_p - u_p \cdot c_p)/S_p$	All
SLRU	• Number of accesses since last reference $\Delta_{p,k}$ • Document size S_p • Document cost C_p	$O(\log(n))$	Least valuable first according to $value_p = (C_p \cdot (1 - \gamma_p))/S_p\Delta_{p,k}$, where γ_p is document p refresh overhead factor	Based on the past
LUV	• Time since last access t_l • Document cost C_p • Document size S_p • Constant value λ	$O(\log(n))$	Least valuable first according to $V_p(k) = F(t_l)V_p(k-1) + C_p/S_p$, where k is the kth reference to document p and $F(x) = (1/2)^{\lambda x}$	All
LRV	• Document size S_p • Time since last access T_p • Number of previous accesses n_p • Other tuning parameters τ_1 andτ_2	$O(1)$	Evict the least likely to be accessed first based on documents access probabilities $Pr(n_p, T_p, S_p)$	All
LR	• Document size S_p • Document type • Number of references in a backward window W_b • Time since last access	$O(\log(n))$	Evict the document with the least weighted expected distance to next access, which is equal to $S_p/(1-(1-P_{LR})^{(1/N)})$, where P_{LR} is the probability of accessing the document in the next $N = 50$ requests	All
LRU-S	• Document size S_p	$O(1)$	Document at the bottom of the LRU stack first (assuming that a newly arrived document is admitted)	Random
CLIMB-S	• Document size S_p	$O(1)$	Document at the bottom of the LRU stack first (assuming that a newly arrived document is admitted)	Random
RANDOM	• Number of samples N • Number of samples to keep from a previous iteration M • Error control parameter n	Variable	Open	Open

■ **Table 5.** *Summary of cache replacement policies.*

W3-U are less adaptive than GD*, since they involve some parameters that need to be tuned offline. Accordingly, we expect GD* to outperform them. LUV was shown to outperform LNC-R-W3-U and LRV [7] with offline tuning of its λ parameter with respect to the cache size and the optimized performance metric, but it is not clear if this holds under all types of traffic.

Next, we compare policies from different groups using trace-driven simulation. We select one representative policy from each group. A representative policy does not reflect the average performance of its group, but rather the performance of the better policies in that group. We select the standard policies LRU, LFU, and SIZE to represent three different groups. From the group that utilizes the recency, frequency, and size information, we choose LUV. GDS (with unit cost) is selected to represent the group that utilizes the recency and size information. Hybrid and Hyper-G are the only policies of the remaining groups, so they are also considered in the comparison. Since Hyper-G is an LFU policy that breaks ties using the time since the last access, it is expected to perform better than LFU, so we do not simulate LFU. Moreover, in implementing LFU we need to adopt a policy to break ties, which are very likely to happen. Our comparison is made based on three performance metrics: the file hit ratio, the byte hit ratio, and the delay saving ratio.

In our simulations, we use two different traces: the first trace is from the public proxy server of Digital Equipment Corporation (DEC); the second trace is from the National Laboratory for Applied Network Research (NLANR). We manipulated the two traces to extract the needed information. Primarily, we simulated only those requests that are cacheable (i.e., we excluded CGI requests). Some of the characteristics of these traces are shown in Table 3. Figures 5 through 10 show the simulation results. As can be seen, LUV policy outperforms the other policies for all performance metrics [7]. This can be attributed in part to the offline tuning of λ, for both the cache size and the performance measures. For the file hit ratio, it is clear that the GDS policy (with unit cost) outperforms LRU and Hyper-G, and comes second after LUV for different cache sizes. Hyper-G is shown to be superior to LRU [10, 11, 34]. For a small cache size, SIZE outperforms Hybrid, but they tend to perform similarly when the cache size is reasonably large [7]. For a small cache size, these policies are inferior to the other policies [17, 26]. For a large cache size, SIZE and Hybrid tend to outperform Hyper-G and LRU [7, 17, 22, 26]. For the byte hit ratio, the policies are ranked irrespective of the cache size. In this case, Hyper-G comes next to LUV and comes very close to LUV as the cache size increases [7]. LRU is next to Hyper-G and outperforms GDS [9, 10, 17, 34]. GDS outperforms Hybrid, which outperforms SIZE [10, 15, 34]. With respect to the delay saving ratio, the simulated policies are clearly distinct from each other, having the same ranking as that for the byte hit ratio, with the exception that the GDS policy jumps to second place after the LUV policy.

Since randomized algorithms (i.e., RANDOM) generally try to approximate the performance of other policies with the benefit of less complexity, they exhibit similar performance to the approximated policies [29].

The computational complexity of the surveyed replacement policies ranges from $\mathcal{O}(1)$ to $\mathcal{O}(n)$, where n is the number of cached documents. Most of the policies that achieve good performance have $\mathcal{O}(\log(n))$ complexities. According to the above classification, policies in a given group have, in general, comparable complexities (both computation and space). Tables 4 and 5 summarize the main aspects of the algorithms described in this article.

Conclusions

In this article we reviewed some of the well known proxy caching policies for WWW traffic. We classified these policies into deterministic and randomized policies. We discussed the parameters they use in decision making and made a qualitative comparison between them after regrouping them based on the traffic information they utilize. We compared policies in the same group irrespective of the performance metric. We then selected representative policies from each group and ranked them according to three performance metrics using results reported in the literature as well as our own simulations. The goodness of any policy lies in the adaptiveness of its algorithm, which adjusts dynamically to changes in the workload characteristics.

Acknowledgment

This work was supported by the National Science Foundation through Grants ANI-0095626, ANI-0313234, and ANI-0325979, by a scholarship from the government of Saudi Arabia through King Abdul-aziz University, and by the Center for Low Power Electronics (CLPE) at the University of Arizona. CLPE is supported by the NSF (grant # EEC-9523338), the State of Arizona, and a consortium of industrial partners.

References

[1] F. Sailhan and V. Issarny, "Energy-Aware Web Caching for Mobile Terminal," *Proc. IEEE 22nd Int'l. Conf. Distributed Computing Systems*, July 2002, pp. 820–25.
[2] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Trans. Knowledge and Data Engineering*, vol. 15, no. 5, Sept. 2003, pp. 1251–65.
[3] S. Gitzenis and N. Bambos, "Power-Controlled Data Prefetching/Caching in Wireless Packet Networks," *Proc. IEEE INFOCOM Conf.*, vol. 3, June 2002, pp. 1405–14.
[4] G. Cao, "Proactive Power-Aware Cache Management for Mobile Computing Systems," *IEEE Trans. Comp.*, vol. 51, no. 6, June 2002, pp. 608–21.
[5] P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini, "Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks," *Proc. 4th ACM Int'l. Symp. Mobile Ad Hoc Net. and Comp.*, June 2003, pp. 25–34.
[6] L. Breslau *et al.*, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM Conf.*, 1999, pp. 126–34.
[7] H. Bahn *et al.*, "Efficient Replacement of Nonuniform Objects in Web Caches," *IEEE Comp.*, June 2002, pp. 65–73.
[8] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation, and Performance," *IEEE Trans. Knowledge and Data Engineering*, vol. 11, no. 4, July/Aug. 1999, pp. 549–62.
[9] S. Jin and A. Bestavros, "Sources and Characteristics of Web Temporal Locality," *Proc. IEEE/ACM Int'l. Symp. Modeling, Analysis and Simulation of Comp. and Telecommun. Sys.*, San Francisco, CA, Aug. 2000, pp. 28–35.
[10] S. Jin and A. Bestavros, "Greedy-dual* Web Caching Algorithm," *Int'l. J. Comp. Commun.*, vol. 24, no. 2, Feb. 2001, pp. 174–83.
[11] S. Jin and A. Bestavros, "Popularity-Aware Greedy-dual Size Web Proxy Caching Algorithms," *Proc. IEEE Int'l. Conf. Distributed Computing Systems (ICDCS)*, Taiwan, May 2000, pp. 254–61.
[12] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Comp. Commun. Review*, vol. 29, no. 5, Oct. 1999, pp. 36–46.
[13] S. Williams *et al.*, "Removal Policies in Network Caches for World Wide Web Documents," *Proc. ACM SIGCOMM Conf., Stanford University*, Aug. 1996, pp. 293–305.
[14] M. Abrams *et al.*, "Caching Proxies: Limitations and Potentials," *Proc. 4th Int'l. World Wide Web Conf.*, Boston University, Dec. 1995.

[15] R. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delays," *Proc. 6th Int'l. World Wide Web Conf.*, Santa Clara, CA, Apr. 1997, pp. 325–34.
[16] N. Young, "The k-server Dual and Loose Competitiveness for Paging," *Algorithmica*, vol. 11, no. 6, June 1994, pp. 525–41.
[17] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. 1997 USENIX Symp. Internet Tech. and Sys.*, 1997, pp. 193–206.
[18] C. Aggarwal, J. Wolf, and P. Fellow, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, Jan./Feb. 1999, pp. 94–107.
[19] M. Arlitt and C. Williamson, Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers," *Simulation Journal*, vol. 68, no. 1, Nov.–Dec. 1997, pp. 44–50.
[20] J. Dilley and M. Arlitt, "Improving Proxy Cache Performance," *IEEE Internet Computing*, vol. 3, no.6, Nov.–Dec. 1999, pp. 44–50.
[21] V. Almeida *et al.*, "Characterizing Reference Locality in the WWW," *Proc. 4th Int'l. Conf. Parallel and Distributed Info. Sys. (PDIS)*, 1996, pp. 92–103.
[22] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache," *IEEE/ACM Trans. Net.*, vol. 8, no. 2, Apr. 2000, pp. 158–70.
[23] E. O'Neil, P. O'Neil, and G. Weikum, "The lruk Page Replacement Algorithm for Database Disk Buffering," *Proc. ACM SIGMOD Int'l. Conf. Management of Data*, Washington, D.C., USA, May 1993, pp. 297–306.
[24] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW Client-Based Traces," *IEEE/ACM Trans. Net.*, vol. 1, no. 3, Jan 1999, pp. 134–233.
[25] N. Niclausse, Z. Liu, and P. Nain, "A New Efficient Caching Policy for the World Wide Web," *Proc. Internet Server Perf. Wksp. (WISP '98)*, Madison, WI, USA, June 1998, pp. 119–28.
[26] A. Foong, Y.-H. Hu, and D. Heisey, "Adaptive Web Caching Using Logistic Regression," *Proc. 1999 IEEE Signal Processing Society Wksp.*, Madison, WI, Aug. 1999, pp. 515–24.
[27] A. Tanenbaum, *Modern Operating Systems*, Prentice Hall, Inc, 1992.
[28] D. Starobinski and D. Tse, "Probabilistics Methods for Web Caching," *Performance Evaluation*, vol. 46, no. 2–3, Oct. 2001, pp. 125–37.
[29] K. Psounis and Balaji Prabhakar, "A Randomized Web-Cache Replacement Scheme," *Proc. IEEE INFOCOM Conf.*, vol. 3, Apr. 2001, pp. 1407–15.
[30] http://www.squid-cache.org/doc/faq/faq-12.html.
[31] S. Irani, "Page Replacement with Multi-Size Pages and Applications to Web Caching," *Algorithmica*, vol. 33, no. 3, July 2002, pp. 384–409.
[32] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, May 1998.
[33] D. D. Sleator and R. Tarjan, "Amortized Efficiency of List Update and Paging Rules," *Commun. ACM*, vol. 28, Feb. 1985, pp. 202–08.
[34] M. Arlitt, R. Friedrich, and T. Jin, "Performance Evaluation of Web Proxy Cache Replacement Policies," *Performance Tools*, Palma de Mallora, Spain, Sept. 1998.
[35] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. ACM SIGMETRICS Conf.*, 1998, pp. 151–60.
[36] L. Cherkasova and G. Ciardo, "Characterizing Temporal Locality and its Impact on Web Server Performance," *Proc. 9th Int'l. Conf. Comp. Commun. and Net. (ICCCN)*, 2000, pp. 434–41.

BIOGRAPHIES

ABDULLAH BALAMASH (balamash@ece.arizona.edu) received the B.S. degree in electrical and computer engineering from King Abdulaziz University, Jeddah, Saudi Arabia, in 1991. From 1991 to 1994 he worked for the Saudi Consolidated Electricity Company as a system engineer. In January 1995 he joined the Department of Electrical and Computer Engineering of King Abdul-aziz University as a teaching assistant. In August 1996 he received a scholarship from the Saudi government to pursue his M.S. and Ph.D. degrees. He received his M.S. degree in electrical and computer engineering from Syracuse University in 1999. He is currently working toward his Ph.D. degree at the University of Arizona. His research interests are in traffic analysis, network systems modeling, and performance evaluation.

MARWAN KRUNZ (krunz@ece.arizona.edu) is an associate professor of electrical and computer engineering at the University of Arizona. He received his Ph.D. degree in electrical engineering from Michigan State University in 1995. From 1995 to 1997 he was a postdoctoral research associate with the Department of Computer Science and the Institute for Advanced Computer Studies (UMIACS) at the University of Maryland. His research interests lie in the field of computer networks, especially in its performance and traffic control aspects. His recent work has focused on power control for mobile ad hoc networks, quality of service over wireless links, routing (path selection, state aggregation), WWW traffic modeling, and video streaming. He has published more than 70 journal papers and refereed conference papers in these areas. He is a recipient of the National Science Foundation CAREER Award (1998–2002). He currently serves on the editorial board for the *IEEE/ACM Transactions on Networking* and the *Computer Communications Journal*. He was a guest co-editor for special issues in *IEEE Micro* and *IEEE Communications Magazine*. He is the technical program chair for the IEEE INFOCOM 2004 Conference (to be held in Hong Kong) and was the technical program co-chair for the 9th Hot Interconnects Symposium (Stanford University, August 2001). He has served and continues to serve on the executive and technical program committees of many international conferences. He consults for a number of corporations in the telecommunications industry.