Mistreatment in Distributed Caching Groups
Causes and Implications §

Nikolaos Laoutaris†‡ Georgios Smaragdakis† Azer Bestavros† Ioannis Stavrakakis‡
nlaout@cs.bu.edu gsmaragd@cs.bu.edu best@cs.bu.edu istavrak@di.uoa.gr

Abstract— **Although cooperation generally increases the amount of resources available to a community of nodes, thus improving individual and collective performance, it also allows for the appearance of potential mistreatment problems through the exposition of one node's resources to others. We study such concerns by considering a group of independent, rational, self-aware nodes that cooperate using on-line caching algorithms, where the exposed resource is the storage of each node. Motivated by *content networking* applications – including web caching, CDNs, and P2P – this paper extends our previous work on the off-line version of the problem, which was limited to object replication and was conducted under a game-theoretic framework. We identify and investigate two causes of mistreatment: (1) cache *state interactions* (due to the cooperative servicing of requests) and (2) the adoption of a *common scheme* for cache replacement/redirection/admission policies. Using analytic models, numerical solutions of these models, as well as simulation experiments, we show that on-line cooperation schemes using caching are fairly robust to mistreatment caused by state interactions. When this becomes possible, the interaction through the exchange of miss-streams has to be very intense, making it feasible for the mistreated nodes to detect and react to the exploitation. This robustness ceases to exist when nodes fetch and store objects in response to remote requests, *i.e.*, when they operate as Level-2 caches (or proxies) for other nodes. Regarding mistreatment due to a common scheme, we show that this can easily take place when the "outlier" characteristics of some of the nodes get overlooked. This finding underscores the importance of allowing cooperative caching nodes the flexibility of choosing from a diverse set of schemes to fit the peculiarities of individual nodes. To that end, we outline an emulation-based framework for the development of mistreatment-resilient distributed selfish caching schemes.**

I. Introduction

Background, Motivation, and Scope: Network applications often rely on distributed resources available within a cooperative grouping of nodes to ensure scalability and efficiency. Traditionally, such grouping of nodes is dictated by an overarching, common strategic goal. For example, nodes in a CDN such as Akamai or Speedera cooperate to optimize the performance of the overall network, whereas IGP routers in an Autonomous System (AS) cooperate to optimize routing within the AS.

More recently, however, new classes of network applications have emerged for which the grouping of nodes is more "ad hoc" in the sense that it is not dictated by organizational boundaries or strategic goals. Examples include the various overlay protocols [1], [2] and peer-to-peer (P2P) applications. Two distinctive features of such applications are (1) the fact that individual nodes are autonomous, and as such, their membership in a group is motivated solely by the selfish goal of *benefiting* from that group, and (2) group membership is warranted only as long as a node is interested in being part of the application or protocol, and as such, group membership is expected to be fluid. In light of these characteristics, an important question is this: *Are protocols and applications that rely on sharing of distributed resources appropriate for this new breed of ad-hoc node associations?*

In this paper, we answer this question for content networking applications, whereby the distributed resource being shared amongst a group of nodes is *storage*. In particular, we consider a group of nodes that store information objects and make them available to their local users as well as to remote nodes. A user's request is first received by the local node. If the requested object is stored locally, it is returned to the requesting user immediately, thereby incurring a minimal access cost. Otherwise, the requested object is searched for, and fetched from other nodes of the group, at a potentially higher access cost. If the object cannot be located anywhere in the group, it is retrieved from an origin server, which is assumed to be outside the group, thus incurring a maximal access cost.

Under an *object replication* model, once selected for replication at a node, an object is stored permanently at that node (*i.e.*, the object cannot be replaced later). In [3], [4] we established the vulnerability of many *socially optimal* (SO) object replication schemes in the literature to *mistreatment* problems, which make it more attractive for an individual node to break away from the group, opting instead to operate in isolation using a greedy local (GL) replication scheme. A mistreated node is one whose access cost under SO replication is higher than the minimal access cost that the node can guarantee under GL replication. Unlike centrally designed/controlled groups where all constituent nodes have to abide by the ultimate goal of optimizing the social utility of the group, an autonomous, selfish node will not tolerate such a mistreatment. Indeed, the emergence of such mistreatments may cause selfish nodes to secede from the replication group, resulting in severe inefficiencies for both the individual users as well as the entire group.

In [3], [4], we resolved this dilemma by proposing a family of *equilibrium* (EQ) object placement strategies that (a) avoid the mistreatment problems of SO, (b) outperform GL by claiming available "cooperation gain" that the GL algorithm fails to utilize, and (c) are implementable in a distributed manner, requiring the exchange of only a limited amount of information. The EQ strategies were obtained by formulating

† Computer Science Dept, Boston University, Boston, Massachusetts, USA.
‡ Dept of Informatics and Telecommunications, University of Athens, Greece.
§ A. Bestavros is supported in part by NSF grants ANI-9986397 / 0095988 / 0205294 and EIA-0202067. I. Stavrakakis is supported in part by EU IST projects CASCADAS and E-NEXT. N. Laoutaris is supported by a Marie Curie Outgoing International Fellowship of the EU MOIF-CT-2005-007230.

the *Distributed Selfish Replication* (DSR) game and devising a distributed algorithm that is always capable of finding pure Nash equilibrium strategies for this particular game.

Distributed Selfish Caching: Proactive replication strategies are not practical in a highly dynamic content networking setting, which is likely to be the case for most of the Internet overlays and P2P applications we envision for a variety of reasons: (1) Fluid group membership makes it impractical for nodes to decide what to replicate based on what (and where) objects are replicated in the group. (2) Access patterns as well as access costs may be highly dynamic (due to bursty network/server load), necessitating that the selection of replicas and their placement be done continuously, which is not practical. (3) Both the identification of the appropriate re-invocation times [5] and the estimation of the non-stationary demands (or equivalently, the timescale for a stationarity assumption to hold) [6] are non-trivial problems. (4) Content objects may be dynamic and/or may expire, necessitating the use of "pull" (*i.e.*, on-demand caching) as opposed to "push" (*i.e.*, proactive replication) approaches. Using on-demand caching is the most widely acceptable and natural solution to all of these issues because it requires no *a priori* knowledge of local/group demand patterns and, as a consequence, responds dynamically to changes in these patterns over time (*e.g.*, introduction of new objects, reduction in the popularity of older ones, *etc.*)

Therefore, in this paper we consider the problem of *Distributed Selfish Caching* (DSC), which can be seen as the *on-line* equivalent of the DSR problem. In DSC, we adopt an *object caching* model, whereby a node employs demand-driven temporary storing of objects, combined with replacement.

Causes of Mistreatments Under DSC: We begin our examination of DSC by first examining the operational characteristics of a group of nodes involved in a distributed caching solution. This examination will enable us to identify two key culprits for the emergence of mistreatment phenomena.

First, we identify the mutual *state interaction* between replacement algorithms running on different nodes as the prime culprit for the appearance of such phenomena. This interaction takes place through the so called "remote hits". Consider nodes v, u and object o. A request for object o issued by a user of v that cannot be served at v but could be served at u is said to have incurred a *local miss* at v, but a *remote hit* at u. Consider now the implications of the remote hit at u. If u does not discriminate between hits due to local requests and hits due to remote requests, then the remote hit for object o will affect the state of the replacement algorithm in effect at u. If u is employing Least Recently Used (LRU) replacement, then o will be brought to the top of the LRU list. If it employs Least Frequently Used (LFU) replacement, then its frequency will be increased, and so on with other replacement algorithms [7]. If the frequency of remote hits is sufficiently high, *e.g.*, because v has a much higher local request rate and thus sends an intense miss-stream to u, then there could be performance implications for the second: u's cache may get invaded by objects that follow v's demand, thereby depriving the user's of u from valuable storage space for caching their own objects. This can lead to the mistreatment of u, whose cache is effectively "hijacked" by v.

Moving on, we identify a second, less anticipated, culprit for the emergence of mistreatment in DSC. We call it the *common scheme* problem. To understand it, one has first to observe that most of the work on cooperative caching has hinged on the fundamental assumption that all nodes in a cooperating group adopt a common scheme. We use the word "scheme" to refer to the combination of: (i) the employed *replacement algorithm*, (ii) the employed *request redirection algorithm*, and (iii) the employed *object admission algorithm*. Cases (i) and (ii) are more or less self-explanatory. Case (iii) refers to the decision of whether to cache locally an incoming object after a local miss. The problem here is that the adoption of a common scheme can be beneficial to some of the nodes of a group, but harmful to others, particularly to nodes that have special characteristics that make them "outliers". A simple case of an outlier, is a node that is situated further away from the center of the group, where most nodes lie. Here distance may have a topological/affine meaning (*e.g.*, number of hops, or propagation delay), or it may relate to dynamic performance characteristics (*e.g.*, variable throughputs or latencies due to load conditions on network links or server nodes). Such an outlier node cannot rely on the other nodes for fetching objects at a small access cost, and thus prefers to keep local copies of all incoming objects. The rest of the nodes, however, as long as they are close enough to each other, prefer not to cache local copies of incoming object that already exist elsewhere in the group. Since such objects can be fetched from remote nodes at a small access cost, it is better to preserve the local storage for keeping objects that do not exist in the group and, thus, must be fetched from the origin server at a high access cost. Enforcing a common scheme under such a setting is bound to mistreat either the outlier node or the rest of the group.

In addition to the identification of the two causes of mistreatments in a DSC setting, this paper presents a number of concrete results regarding each one of these two causes. We summarize these results next.

Mistreatment Due to Cache State Interaction: Regarding the state interaction problem, our results answer the following basic questions: *"Could and under which schemes do mistreatments arise in a DSC group?"*, and *"What are the possible ways in which a node may react to such mistreatments?"*.

+ We show that state interaction may occur when nodes do not discriminate between local and remote *hits* upon updating the state of their replacement algorithms.

+ To materialize, state interactions require substantial request rate imbalance, *i.e.*, one or more "overactive" nodes must generate disproportionally more requests than the other nodes in the group. Even in this case, mistreatment of less active nodes depends on the amount of storage that they posses: Mistreatment occurs when these nodes have abundant storage, otherwise they are generally immune to, or even benefit from, the existence of overactive nodes.

+ Comparing caching and replication with regard to their relative sensitivity to request rate imbalance, we show that caching is much more robust than replication. This means that the occurrence of mistreatment is much more difficult under caching.

+ Regarding the vulnerability of different replacement algorithms, we show that “noisier” replacement algorithms are more prone to state interactions. In that regard, we show that LRU is more vulnerable than LFU.

+ Even the most vulnerable LRU replacement is quite robust to mistreatment as it requires a very intense miss-stream in order to force a mistreated node to maintain locally unpopular objects in its cache (thus depriving it of cache space for locally popular objects). In particular, the miss-stream has to be strong enough to counter the sharp decline in the popularity of objects in typically skewed workloads.

+ Robustness to mistreatment due to state interaction evaporates when a node operates as a Level-2 cache [8] (L2) for other nodes. L2 caching allows all remote requests (whether they hit or miss) to affect the local state (as opposed to only hits under non-L2 caching), leading to a vulnerability level that approaches the one under replication.

Mistreatment Due to Use of Common Scheme: We classify cooperative schemes into two groups: *Single Copy* (SC) schemes, *i.e.*, schemes where there can be at most one copy of each distinct object in the group – two examples of SC schemes are HASH based caching [9] and LRU-SC [10]; *Multiple Copy* (MC) schemes, *i.e.*, schemes where there can be multiple copies of the same object at different nodes.

+ We show that the relative performance ranking of SC and MC schemes changes with the “tightness” of a cooperative group. SC schemes perform best when the inter-node distances are small compared to the distance to the origin server; in such cases the maintenance of multiple copies of the same object is redundant.[1] MC schemes improve progressively as the inter-node distances increase, and eventually outperform the SC schemes.

+ We demonstrate the possibility of mistreatment due to a common scheme by considering a tight group of nodes that operate under SC and a unique outlier node that has a larger distance to the group. We show that this node is mistreated if it is forced to follow a SC scheme.

Towards Mistreatment-Resilient DSC Schemes: More constructively, we present a framework for the design of mistreatment-resilient DSC schemes, which allow individual nodes to decide autonomously (*i.e.*, without having to trust any other node or service) whether they should stick to, or secede from a DSC caching group, based on whether or not their participation is beneficial to their performance compared to a selfish, greedy scheme. Resilience to mistreatments is achieved by allowing a node to emulate the performance gain possible by switching from one scheme to another, or by adapting some control parameters of its currently deployed DSC scheme.

II. Definitions and Notation

Let o_i, $1 \leq i \leq N$, and v_j, $1 \leq j \leq n$, denote the ith unit-sized object and the jth node, and let $O = \{o_1, \ldots, o_N\}$ and $V = \{v_1, \ldots, v_n\}$ denote the corresponding sets. Node v_j is assumed to have storage capacity for up to C_j unit-sized objects, a total request rate λ_j (total number of requests per unit time, across all objects), and a demand described by a probability distribution over O, $\vec{p}_j = \{p_{1j}, \ldots, p_{Nj}\}$, where p_{ij} denotes the probability of object o_i being request by the local users of node v_j. Successive request are assumed to be independent and identically distributed.[2] Later in this paper, we make the specific assumption that the popularity of objects follows a power-law profile, *i.e.*, the ith most popular object is requested with probability $p_i = K/i^a$. Such popularity distributions occur in many measured workloads [15], [17] and, although used occasionally in our work (*e.g.*, in Section III-A to simplify an analytic argument, or in Section IV for producing numerical results), they do not constitute a basic assumption, in the sense that mistreatment can very well occur with other demand distributions that do not follow such a profile.

Let t_l, t_r, t_s denote the access cost paid for fetching an object locally, remotely, or from the origin server, respectively, where $t_s > t_r > t_l$.[3] User requests are serviced by the closest node that stores the requested object along the following chain: local node, group, origin server. Each node employs a replacement algorithm for managing the content of its cache and employs an object admission algorithm for accepting (or not) incoming objects.

III. Mistreatment Due to State Interaction: Analysis

Our goal in this section is to understand the conditions under which mistreatment may arise as a result of (cache) state interactions. We start in Section III-A with a replacement-agnostic model that focuses on the rate-imbalance (between the local request stream and the remote miss stream)[4] necessary for mistreatment to set in. Next, in Section III-B, we present a more detailed analytic model that allows for the derivation of the average access cost in a distributed caching group composed of n nodes that operate under LRU.

A. General conditions

We would like to determine the level of *request rate imbalance* that is necessary for mistreatment to be feasible. We model this imbalance through the ratio λ_n/λ_j, where λ_j denotes the request rate of any normally-behaving node v_j, while λ_n denotes the request rate of an over-active node, which we use to instigate mistreatment problems. As a convention we assume this overactive node to be the last (nth) node of the group.

We focus on the interaction between v_j and v_n. Fig. 1 shows a particular choice of demand patterns that fosters the occurrence of mistreatment. The initial most popular objects in $\vec{p}_j$ and $\vec{p}_n$ up to the two capacities (C_j for v_j and C_n for v_n) are completely disjoint, while the remaining ones in the middle part of the two distributions are identical; both demands are

[1] We do not consider load balancing and node reliability concerns in this study.

[2] The Independent Reference Model (IRM) [11] is commonly used to characterize cache access patterns [12], [13], [14], [15]. The impact of temporal correlations was shown in [6], [16] to be minuscule, especially under typical, Zipf-like object popularity profiles.

[3] The assumption that the access cost is the same across all node pairs in the group is made only for the sake of simplifying the presentation. Our results can be adapted easily to accommodate arbitrary inter-node distances.

[4] Recall that state-interaction-induced mistreatment occurs when the remote miss stream pollutes the local cache with objects that are not popular to the local users—thereby depriving them from valuable cache space.

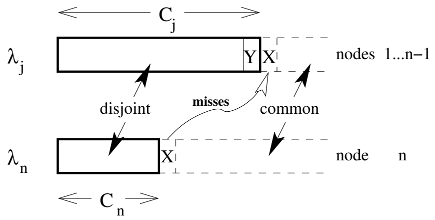

Fig. 1. Reference model for the study of mistreatment due to state interaction. The most popular objects for the two nodes, *i.e.*, those that would fit in the caches under optimal placement, are assumed to be completely disjoint. The subsequent, more popular ones, *i.e.*, those that would be placed in the dashed parts of the figure if the nodes had more storage capacity, are assumed to be identical.

assumed to be power-law with parameter a. Let X denote the most popular object that is common to both distributions. A boundary condition for the occurrence of mistreatment can be obtained by considering the ratio λ_n/λ_j that results in a switch of ranking between X and Y at v_j, where Y denotes the least most popular object that would be kept in the cache of v_j under a perfect ranking of objects according to the local demand, if no miss-stream was received. To derive the condition for the switch we first note that X is the (C_j+1)th most popular object for v_j and the (C_n+1)th most popular one for v_n. Y is the C_jth most popular object for v_j. Let $f(n)$ denote a function that captures the operation of different object location mechanisms in a group of n nodes (used for locating and retrieving objects from remote nodes).

For example, $f(n)=1$ can be used for modeling request flooding (following a local miss, a request is sent to all other nodes in the group); $f(n)=1/(n-1)$ can be used for modeling index-based mechanisms [10] (following a local miss, a request is sent to only one of the nodes that appear to be storing the object according to some index). Then the boundary condition for the occurrence of the switch can be written as follows:

$$\begin{aligned}&\lambda_j p_{C_j} \leq \lambda_j p_{C_j+1} + \lambda_n p_{C_n+1} f(n) \Rightarrow \\ &\lambda_j \frac{1}{(C_j)^a} \leq \lambda_j \frac{1}{(C_j+1)^a} + \lambda_n \frac{1}{(C_n+1)^a} f(n) \Rightarrow \\ &\frac{\lambda_n}{\lambda_j} \geq \frac{(C_n+1)^a}{f(n)} \left(\frac{1}{C_j^a} - \frac{1}{(C_j+1)^a} \right)\end{aligned} \tag{1}$$

Writing a continuous approximation for the rate of change of $1/C^a$ with respect to C, we get:

$$\begin{aligned}&\frac{d\left(\frac{1}{C^a}\right)}{dC} = \frac{\frac{1}{C^a} - \frac{1}{(C+1)^a}}{C-(C+1)} \approx -a \cdot \frac{1}{C^{1+a}} \Rightarrow \\ &\frac{1}{C^a} - \frac{1}{(C+1)^a} \approx a \cdot \frac{1}{C^{1+a}}\end{aligned} \tag{2}$$

Using the approximation from Eq. (2) on Eq. (1) we obtain:

$$\frac{\lambda_n}{\lambda_j} \geq \frac{(C_n+1)^a}{f(n)} \cdot a \frac{1}{(C_j)^{1+a}} \sim \frac{a}{f(n) C_j} \left(\frac{C_n}{C_j} \right)^a \tag{3}$$

Eq. (3) states that the amount of imbalance in request rates ($\frac{\lambda_n}{\lambda_j}$) that is required for the occurrence of mistreatment is: (i) increasing with C_n, (ii) decreasing with C_j, and (iii) increasing when request flooding is employed for locating remote objects (in this case all the nodes get the full miss-stream from v_n, otherwise the miss-stream weakens by being split into $n-1$ parts).

Now assume that as a result of the received miss-stream, k objects of v_j are switched (objects with ids $C_j, \ldots, C_j - k+1$ evicted, objects $C_j+1, \ldots, C_j+k$ inserted); k can be computed from a condition similar to that in (Eq. 1). Define the $Loss$ of v_j as the reduction in the probability mass of the objects that it caches locally.

$$\begin{aligned} Loss &= \sum_{i=C_j-k+1}^{C_j} p_i - \sum_{i=C_j+1}^{C_j+k} p_i \\ &= K \cdot (H_{C_j}^{(a)} - H_{C_j-k}^{(a)} - H_{C_j+k}^{(a)} + H_{C_j}^{(a)}) \\ &= K \cdot (2H_{C_j}^{(a)} - H_{C_j-k}^{(a)} - H_{C_j+k}^{(a)}),\end{aligned} \tag{4}$$

where K is the normalization constant of the power-law distribution $p_i = K/i^a$. The generalized harmonic number $H_C^{(a)}$ can be approximated by its integral expression (see [18]) $H_C^{(a)} = \sum_{i=1}^{C} \frac{1}{i^a} \approx \int_1^C 1/l^a dl = \frac{C^{1-a}-1}{1-a}$. Plugging this into Eq. (4) we obtain:

$$Loss = K \Bigg(2\frac{{C_j}^{1-a}-1}{1-a} - \frac{(C_j-k)^{1-a}-1}{1-a} - \frac{(C_j+k)^{1-a}-1}{1-a} \Bigg) \tag{5}$$

From Eq. (5) it is clear that as C_j increases, both $C_j - k$ and $C_j + k \rightarrow C_j$ thus leading to $Loss \rightarrow 0$. Combining our observations from Eq. (3) and Eq. (5) we conclude that the *occurrence* of mistreatment is fostered by small C_n and large C_j. Its magnitude, however, decreases with C_j. So, practically, it is in intermediate values of C_j that mistreatment can arise in a substantial manner.

B. *Analysis of Mistreatment Under LRU Replacement*

In the remainder of this section, our objective will be to derive the steady-state hit probabilities $\vec{\pi}_j = \{\pi_{1j}, \ldots, \pi_{Nj}\}$, where π_{ij} denotes the steady-state probability of finding object o_i at node v_j upon request.

Let $\vec{\pi} = LRU(\vec{p}, C)$ denote a function that computes the steady-state object hit probabilities for a single LRU cache in isolation, given the cache size and the demand distribution. Due to the combinatorial hardness of analyzing LRU replacement, it is difficult to derive an exact value for $\vec{\pi}$; there are, however, several methods for computing approximate values for it (see for example [19] and references therein). In this paper, we employ the approximate method of Dan and Towsley in [20] that provides an accurate estimation of $\vec{\pi}$ through an iterative algorithm that incurs $O(NC)$ time complexity. Having computed $\vec{\pi}_j$, $\forall v_j \in V$, we can obtain the per node access cost $cost_j$, as well as the social cost of the entire group, $cost_{soc} = \sum_{\forall v_j} cost_j$, by using Eq. (6). In this equation π_{i-j} denotes the probability of finding o_i in any node of the group other than v_j.

We can obtain $\vec{\pi}_j$ by using the $LRU(\cdot,\cdot)$ function for isolated caches as our basic building block and taking into consideration the impact on the local state of the hits caused by remote requests. Deriving an exact expression for these added hits based on the involved cache states is intractable as it leads to state-space explosion. We, therefore, turn to approximate techniques and, in particular, to techniques that consider the expected values of the involved random variables, instead of their exact distributions. The basic idea of our approach is to capture these added hits by properly modifying the input to the $LRU(\cdot,\cdot)$ function. We do so next.

Remote hits can be considered simply as additional request that augment the local demand, thereby creating a new

$$cost_j = \sum_{i=1}^{N} p_{ij} \cdot [\pi_{ij} \cdot t_l + (1-\pi_{ij}) \cdot \pi_{i-j} \cdot t_r + (1-\pi_{ij}) \cdot (1-\pi_{i-j}) \cdot t_s] \quad \text{where} \quad \pi_{i-j} = 1 - \prod_{\forall j' \neq j} (1-\pi_{ij'}) \tag{6}$$

$$p_{ij}^{(k)} = \frac{\lambda_j \cdot p_{ij} + \sum_{j'=1,j'\neq j}^{n} \lambda_{j'} \cdot p_{ij'} \cdot (1-\pi_{ij'}^{(k-1)}) \cdot \left[\frac{\left(\pi_{ij}^{(k-1)}\right)^2}{\sum_{j''=1,j''\neq j'}^{n} \pi_{ij''}^{(k-1)}} \right]^{+}_{\pi_{ij}^{(k-1)}}}{\sum_{i'=1}^{N} \left(\lambda_j \cdot p_{i'j} + \sum_{j'=1,j'\neq j}^{n} \lambda_{j'} \cdot p_{i'j'} \cdot (1-\pi_{i'j'}^{(k-1)}) \cdot \left[\frac{\left(\pi_{i'j}^{(k-1)}\right)^2}{\sum_{j''=1,j''\neq j'}^{n} \pi_{i'j''}^{(k-1)}} \right]^{+}_{\pi_{i'j}^{(k-1)}} \right)} \tag{7}$$

aggregate demand for the $LRU(\cdot,\cdot)$ function as explained later. The idea of modifying the input of a simpler system to capture a policy aspect of a more complex system and then using the modified simpler system to study the more complex one has been employed frequently in the past [21], [22]. Since the remote hits are shaped by the states of the caches, which are coupled due to the exchanges of miss-streams, an iterative procedure is followed for the derivation of the per-node steady-state vectors and access costs. The uncoupled solution (corresponding to nodes operating in isolation) is obtained first, and is refined progressively by taking into account the derived states and the cooperative servicing of the misses. The resulting approximate analytic model for predicting the average access cost in a distributed caching group is described below. In the next section, we show that the results produced from this analytic model match very well the results obtained through simulations of the actual system.

The iterative procedure follows:

(1) For each node v_j compute $\vec{\pi}_j^{(0)} = LRU(\vec{p}_j, C_j)$, *i.e.*, assume no state interaction among the different nodes.

(2) Initiate Iteration: At the kth iteration the aggregate demand distribution for v_j, $\vec{p_j}^{(k)} = \{p_{ij}^{(k)}\}$, $1 \leq i \leq N$, is given by Eq. (7) (see top of page). The function $[x]_y^+$ returns 0 if $y = 0$ and x otherwise.[5] The steady state vector of object hit probabilities for v_j at iteration k can be obtain from:

$$\vec{\pi}_j^{(k)} = LRU(\vec{p_j}^{(k)}, C_j)$$

(3) Convergence Test: if $|\vec{\pi}_j^{(k)} - \vec{\pi}_j^{(k-1)}| < \vec{\epsilon}$ for all v_j, $1 \leq j \leq n$, then set $\vec{\pi}_j = \vec{\pi}_j^{(k)}$ and compute the per node access costs from Eq. (6); $\vec{\epsilon}$ denotes a user-defined tolerance for the convergence of the iterative method. Otherwise, set $\vec{\pi}_j^{(k-1)} = \vec{\pi}_j^{(k)}$ and $\vec{p}_j^{(k-1)} = \vec{p}_j^{(k)}$ and perform another iteration by returning to step 2.

The nominator of Eq. (7) adds the requests generated by the local population of v_j for object o_i, to the requests for the same object due to the $n-1$ miss streams from all other nodes that create hits at v_j. The explanation of the circumstances under which such hits exist, goes as follows (see also Fig. 2): a request for o_i received at the *contributor* node $v_{j'}$ (prob. $p_{ij'}$) affects the *tagged* node v_j, if the request cannot be serviced at the contributor node (prob. $(1-\pi_{ij'}^{(k-1)})$), can be serviced at the tagged node (prob. $\pi_{ij}^{(k-1)}$), and is indeed serviced by the tagged node and not by any other *helper* node $v_{j''}$ that can potentially service it (prob. $\pi_{ij}^{(k-1)} / \sum_{j''=1,j''\neq j'}^{n} \pi_{ij''}^{(k-1)}$, *i.e.*, the model assumes that when more than one helper nodes can offer service, then the request is assigned uniformly to any one of them).

IV. MISTREATMENT DUE TO STATE INTERACTION: EVALUATION

In this section, we use a combination of simulation experiments and numerical solutions of the analytical model developed in the previous section to explore the design space of distributed caching with respect to its vulnerability to the on-set of mistreatment as a result of the state interaction phenomenon. We start by validating the accuracy of the analytical model of Section III-B and follow that with an examination of various dimensions of the design space for distributed caching, including a comparative evaluation of mistreatment in caching versus replication.

It is important to note that throughout this section, we use a number of settings to gain an understanding of state interaction in distributed caches and its consequences on local and group access costs. Some of these settings are *intentionally* very simple (*i.e.*, small "toy" examples) so that they can be possible to track. Others are larger and, therefore, closer to real-world scale, especially when considering the dissemination of voluminous content, *e.g.*, digitized movies or software updates.

Also, it is important to note that the various parameterizations of our analytical and simulation models are not meant to represent particular content networking applications. Examining specific incarnations of the state interaction phenomenon is, after all, not our intention in this paper—which is the first to identify and analyze the problem. Rather, our exploration of the extent of mistreatment is meant to help us gain insights into the fundamental aspects of state interactions in distributed caching, such as its dependence on the request rate imbalance and the relative storage capacity.[6] In most of the following numerical results we assume that nodes follow a common power-law demand distribution with skewness a. Our interest and focus on this type of demand is based on two facts: (i) such distributions have been observed in actual workloads, and (ii)

[5] This function is used to ensure correctness when the denominator $\sum_{j''=1,j''\neq j'}^{n} \pi_{ij''}^{(k-1)}$ becomes zero. Notice that the nominator $\pi_{ij}^{(k-1)}$ is included in the denominator, so when $\pi_{ij}^{(k-1)} > 0$, the denominator is guaranteed to be non zero.

[6] With respect to storage capacities, it is important to note that performance results depend on the relative size of the cache to the object space–*i.e.*, the ratio C/N, but not on the particular values of C and N, *i.e.*, our results are immune to scale.

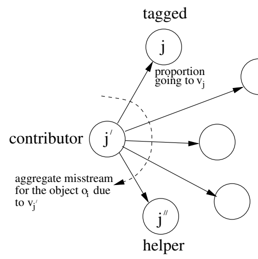

Fig. 2. Graphical illustration for the explanation of Eq. (7).

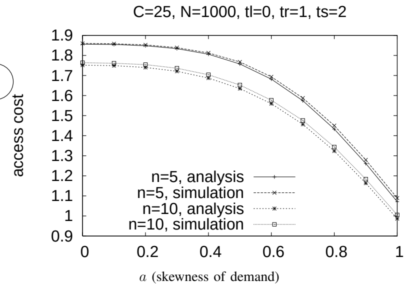

Fig. 3. Validation of the approximate analytic model of Section III-B through comparison with simulation results on the social cost of the group.

cooperative caching is meaningful, and can be effective, only when there is a substantial similarity in the demand patterns of the nodes.

A. Analytic Model Validation

The analytic model presented in Section III-B included a number of approximations—namely: (i) the basic building block, *i.e.*, the $LRU(\cdot, \cdot)$ function, is itself an approximation; (ii) our capturing of the effect of the remote hits on the local state through Eq. (7) is approximate; the solution of the model through the iterative method is approximate.

In this section, we show that despite these approximations, the analytic model presented in Section III-B is fairly accurate. We do so by comparing the model predictions with simulation results in Fig. 3. As evident from these results, the aforementioned approximations have a very limited effect on the accuracy of the model. We have obtained similar results across a wide variety of parameter sets. Thus, in the remainder of this section, we use this model to study several aspects of mistreatment due to state interaction.

B. Understanding State Interaction

Fig. 4 provides a microscopic view of state interaction by showing its effect at the object level. The results are from an illustrative example involving a group of $n = 4$ nodes, each of which has storage for up to $C = 4$ objects, in a universe of $N = 10$ objects (other parameters are shown in the caption and legends of the figure). Nodes $v_1, \ldots, v_3$ have the same fixed request rate $\lambda_1 = 1$, whereas the overactive node v_4 has request rate $\lambda_4 = 1, 10, 100$ (*i.e.*, we have three sets of results that correspond to different λ_4; each set is depicted on a different row of Fig. 4). The three graphs on the left depict the demand and the steady-state vector for node v_1 (which will be used as a representative for all three non-overactive nodes), while the ones on the right depict the corresponding quantities for node v_4. Each graph includes four curves. The bottom two curves indicate the local demand distribution $\vec{p}$ and the aggregate demand distribution $\vec{p}^*$, which includes the effect of the other nodes' miss streams (each of these curves sums up to 1). The top two curves $\vec{\pi}$ and $\vec{\pi}^*$ show the steady-state vectors of a node when the input is $\vec{p}$ (no miss-stream present) and $\vec{p}^*$ (miss-stream present), respectively, as obtained from the analytic method of Section III-B (each of these curves sums up to C).

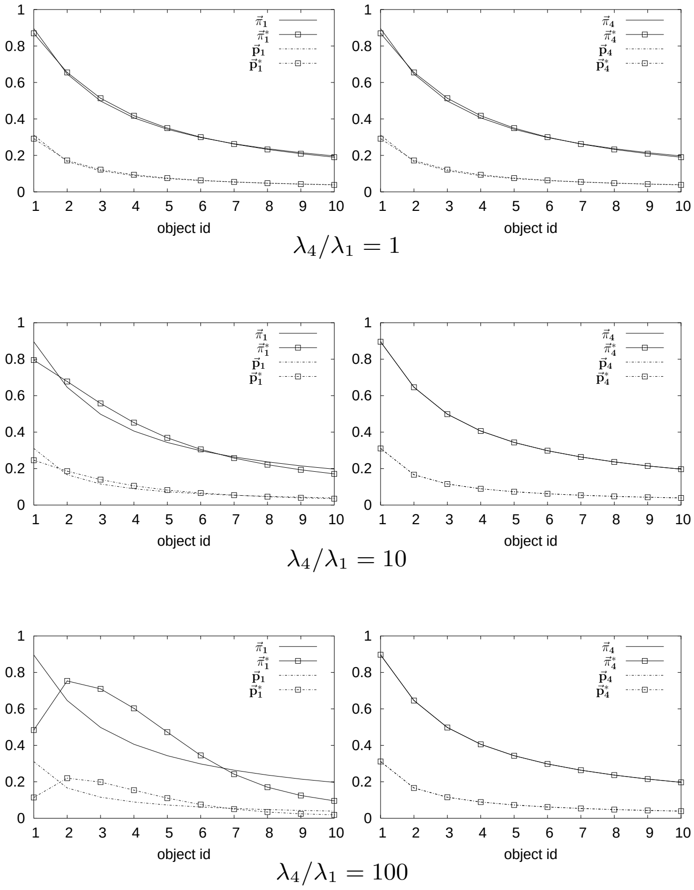

Fig. 4. Analytic results on the effect of request rate imbalance on the per object request and hit probabilities under LRU (values with "*" superscript) and LRU without state interaction. $\vec{p}$ denotes the demand and $\vec{\pi}$ the steady-state hit probabilities. Other parameters: $N = 10,\ n = 4,\ C = 4,\ \alpha = 0.9$.

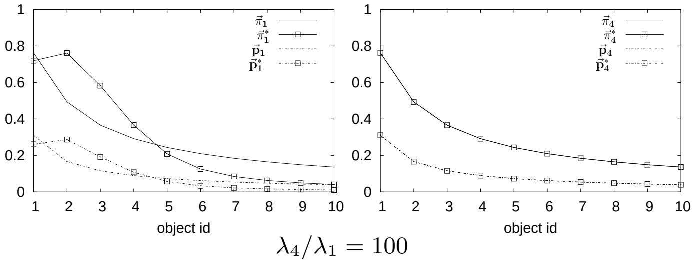

Fig. 5. Analytic results on the repetition of the $\lambda_4/\lambda_1 = 100$ (third row of Fig. 4) - but this time with a smaller storage capacity $C = 3$.

Looking at the three graphs on the right-hand-side of Fig. 4, we see that overactive node v_4 is not affected by the miss streams of the other nodes. For $\lambda_4 = 10$ and 100, its aggregate demand and its steady state vector are identical to the corresponding ones without state interaction, *i.e.*, $\vec{p}_4^* \approx \vec{p}_4$ and $\vec{\pi}_4^* \approx \vec{\pi}_4$. For $\lambda_4 = 1$, there is a very slight effect due to the presence of the miss streams of the other three nodes, but this has almost no effect on the steady-state vector $\vec{\pi}_4^*$.

Looking at the top left graph in Fig. 4, which corresponds to $\lambda_4 = 1$, we see that the same slight effect exists for node v_1 due to the reception of the other three miss streams. The situation, however, changes radically when increasing λ_4. In that case, $\vec{p}_1^*$ and $\vec{p}_1$, and as a consequence $\vec{\pi}_1^*$ and $\vec{\pi}_1$ also become distinctively different. The intense miss stream from

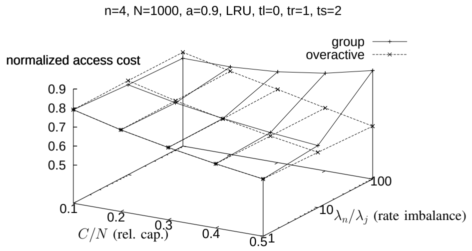

Fig. 6. Analytic results on the effect of the state interaction on the normalized access cost of the overactive node and the remaining nodes of the group under different relative storage capacities and request rate imbalances. No mistreatment occurs in this case.

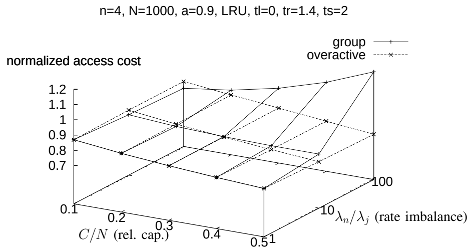

Fig. 7. Analytic results on the state interaction effect on the normalized access cost of the overactive node and the remaining nodes of the group under different relative storage capacities and request rate imbalances. Mistreatment occurs due to larger t_r.

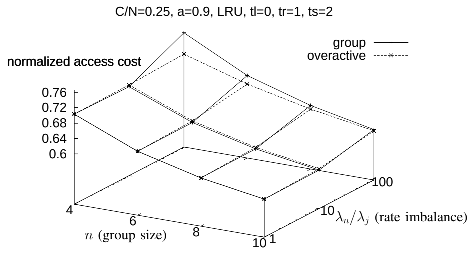

Fig. 8. Analytic results on the effect of the state interaction on the normalized access cost of the overactive node and the remaining nodes of the group under different group sizes and request rate imbalances.

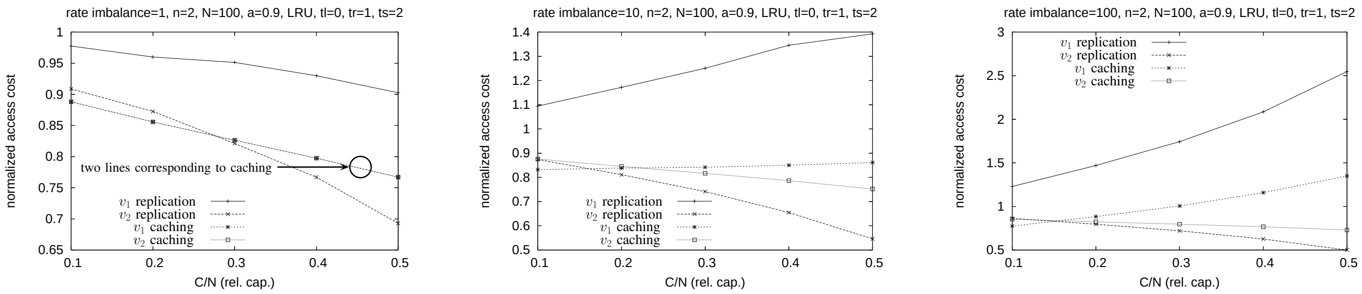

Fig. 9. Analytic results on the comparison of replication and caching under three cases of request imbalance (1,10 and 100).

v_4 increases the popularity of some objects from the middle part of $\vec{p}_1$, thereby making them the most popular objects in $\vec{p}_1^*$. For example, when $\lambda_4 = 100$, objects 2,3 and 4, become more popular than object 1. This change in the profile of $\vec{p}_1^*$ is then reflected in $\vec{\pi}_1^*$, thereby affecting its access cost (Eq. (6)), as we explain below.

C. Effect on Performance

Fig. 6 provides a macroscopic view of state interaction, by considering its effects on the normalized access cost of each node. The normalized cost of node v_j under the aggregate demand $\vec{p}_j^*$ is defined as follows:

$$\hat{cost}_j(\vec{p}_j^*, \vec{p}_j) = \frac{cost_j(\vec{p}_j^*)}{cost_j^{iso}(\vec{p}_j)}, \tag{8}$$

where $cost_j^{iso}(\vec{p}_j) = \sum_{i=1}^{N} p_{ij} \cdot [\pi_{ij} \cdot t_l + (1-\pi_{ij}) \cdot t_s]$ is the cost that would be incurred by v_j if it operated in isolation (outside the group) and received only its local demand $\vec{p}_j$. If $\hat{cost}_j < 1$, the node is benefited from its participation in the group, otherwise, it is being mistreated. When considering two nodes, v_j and $v_{j'}$, then the fact that $1 > \hat{cost}_j > \hat{cost}_{j'}$, means that although both are better off by participating in the group, v_j gets a relatively larger benefit.

There are two main points to be concluded from Fig. 6. First, *it requires a very strong imbalance of request rates in order to create a substantial difference in the incurred normalized access costs*. In the presented example, the overactive node v_4 has a 30% reduction of its normalized cost, only when it produces a 100-fold more intense request stream. Even such a strong imbalance, is not enough to mistreat the other nodes ($v_1, \ldots, v_3$ have normalized access costs < 1). For the occurrence of mistreatment, remote accesses have to be even more expensive (this is shown in Fig. 7, where t_r increases from 1 to 1.4, thereby making the normalized access cost of the group nodes > 1). Second, *the nodes must have large storage capacity to be affected by state interaction related phenomena*. In the presented example, the nodes must have at least 20% relative storage capacity C/N to be affected by the overactive node. Surprisingly, for small C/N, *e.g.*, less than 15%, the group nodes actually benefited more than the overactive node, *i.e.*, they achieve a smaller normalized access cost.

Fig. 5, gives an explanation of this phenomenon by presenting what happens in the toy example of Fig. 4, when decreasing the storage capacity from $C = 4$ to $C = 3$ ($\lambda_4/\lambda_1 = 100$ while keeping all other parameters identical to the ones in Fig. 4). Under $C = 4$, $\hat{cost}_1 = 0.7$ and $\hat{cost}_4 = 0.63$, but under $C = 3$, $\hat{cost}_1 = 0.65$ and $\hat{cost}_4 = 0.69$, *i.e.*, the group nodes do better than the overactive node under $C = 3$, whereas they did worse than it under $C = 4$. To explain the phenomenon, one has to compare the left graph of Fig. 5, with the left graph of the third row of Fig. 4. Comparing the profiles of $\vec{p}_1^*$ and $\vec{\pi}_1^*$ from the two figures, we see that the miss-stream of the overactive node reduces the popularity and the hit probability of object 1 in both cases, but the reduction is much smaller under the smaller storage capacity (this is because under $C = 3$, v_2 will send more requests for o_i to v_1). The next more popular objects, are requested and found in the cache with similar probabilities under both capacities, whereas the least popular objects are more filtered under the smallest capacity. Comparing now, the left and the right graphs of Fig. 5, it becomes obvious the aggregate demand fed to $v_1, \ldots, v_3$ is more skewed (as a result of the state interaction) than the demand going to the overactive node v_4. This does not apply to the $C = 4$ case (third row of Fig. 5), where the hit probability for the (important) object 1 at v_1 is seriously decreased due to the reception of the miss-stream from v_4. Combining the more skewed demand with the help received from fetching some objects from the other nodes, v_1 achieves a lower normalized access cost than the overactive node v_4.

Fig. 8 shows that increasing the size of the group, reduces the effects of the state interaction. This happens because the miss stream of the over-active node(s) (here just one) weakens by being divided among more nodes.

Another important observation relates to the effect of an over-active node on the social cost of the group. By increasing its rate, the over-active node succeeds in hijacking some of the storage of other nodes for caching its own next most popular objects. This lowers its access cost but the gain is inherently limited due to the fast decline of the popularity distribution. The small gain of the over-active node has, nevertheless, a much more serious negative effect on the access cost of the mistreated nodes, which see some of their most popular objects being evicted for much less valuable objects (observe that the angle of increase for the cost of the mistreated nodes is larger than the angle of decrease for the overactive node). Consequently the social cost obtained by adding the individual (unweighed) costs, gets worse with the imbalance of request rates, as is also the case in our previous results regarding object replication [4].

D. Caching versus Replication

In this section we will consider both replication and caching and compare their relative robustness to mistreatment. For replication we will consider the socially optimal (SO) replication algorithm of Leff et al. [23]. For simplicity of exposition, and also to be able to compare with our previous numerical results from [4], we will consider a group with only $n = 2$ nodes and a universe of $N = 100$ objects. The three graphs of Fig. 9 depict the normalized access costs[7] for nodes v_1 and v_2 (overactive), for three cases of request imbalance, 1, 10, and 100. When there is no request imbalance (first graph), no node is mistreated. Caching yields the exact same performance for both nodes (the two curves for v_1, v_2 coinciding), while replication might unintentionally favor one of them (there are several optimal solutions, and the particular one chosen has to do with the specific solution algorithm that is employed, here an LP relaxation of an integer problem solved via Simplex).

The different sensitivity to mistreatment becomes apparent as soon as request imbalance is introduced, *i.e.*, with $\lambda_2/\lambda_1 = 10$ and 100 (second and third graphs of Fig. 9). By observing these figures, we see that the curves for caching are always contained within the angle specified by the curves for replication (except for very small C/N, where we have the peculiar behavior of caching discussed in the previous section). The point to be kept from these results is that *replication is much more sensitive to mistreatment than caching*. Under replication, the slightest imbalance of request intensities is directly reflected in the outcome of the replication algorithm.

[7]For the case of replication, the normalization is conducted by dividing with the performance of the greedy local (GL) replication strategy. See [4] for details.

In contrast, the state interaction that takes place in caching is a much weaker catalyst for mistreatment. This fortunate weakness owes to the stochastic nature of caching, and to the requirement for the concurrent occurrence of two independent events: An unpopular object must first be brought to the cache from the local demand, and then the miss-stream must feed it with requests, if it is to lock it in the cache (and thereafter beat the local request stream that tries to push it out and reclaim the storage space).

E. LRU versus LFU

Fig. 10 shows analytic results under LRU replacement, as well as simulation results under perfect LFU replacement [7] (two group sizes, $n = 2$ and $n = 4$, are considered). We plot the absolute, instead of the normalized access costs, as we are considering different replacement algorithms. Looking first at LRU, we notice the following. The effects of state interaction (reflected in the width of the angle between group and overactive curves, after $\lambda_n/\lambda_j = 10$) decrease as the group grows larger, as also noted in the previous section. Moreover, the absolute access costs for both the group and the overactive node also decrease with the size of the group. The reason is that a bigger group has more aggregate storage capacity and thus succeeds in caching more distinct objects, which in turn benefits all the nodes.

Turning our attention to the LFU curves, we see a completely different behavior. For a given n, both the overactive node and the rest of the group, have the same access cost, *i.e.*, *the request imbalance has no affect on the nodes under LFU*. This happens because once in steady-state, perfect LFU avoids replacement errors, thus does not give any opportunities for locking unpopular objects and losing storage due to the miss-stream of remote overactive nodes. Thus LFU has an advantage over LRU in terms of its immunity to request imbalance. What is even more interesting, however, is that the access cost under LFU remains the same under different n, *i.e.*, increasing the group size does not help in reducing the access costs. This happens because under LFU and common demand patterns, all the nodes end up caching exactly the same sets of objects. In such a group, a local miss is bound to miss also in the group. In other words, *LFU eliminates all the cooperation gain in groups of similar nodes*. This does not occur when the group operates under LRU: the replacement errors committed by the individual nodes in this case, create a healthy amount of noise that increases the distinct objects held in the group, thereby decreasing the access cost of all the nodes. Thus, in large groups under small inter-node distances, LRU is more appropriate than LFU (see for example the access cost for small t_r in Fig. 13 in Section V-A, where $n = 10$). When the inter-node distances increase, then the perfect ranking of objects under LFU becomes more important than the cooperation gain and, thus, LFU becomes better for the group (see Fig. 13 for large t_r).

F. L2 versus Non-L2 Caching

In this section we consider the case in which each node of the group acts as an L2 cache for each other node of the group [8]. This means that a node fetches and maintains a copy from the origin server, for every (remote miss) request that it receives from a remote node. We can capture this behavior by

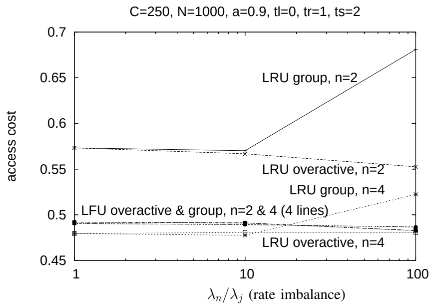

Fig. 10. Analytic results on the comparison LRU vs LFU.

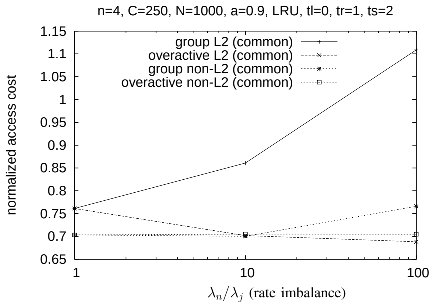

Fig. 11. Analytic results on the comparison L2 vs non-L2 caching - common demand.

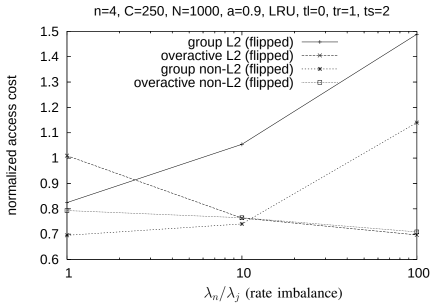

Fig. 12. Analytic results on the comparison L2 vs non-L2 caching - flipped demand.

modifying Eq. (7), so that all the remote requests affect the local state (as opposed to only the remote hits under non-L2 mode):

$$p_{ij}^{(k)} = \frac{\lambda_j \cdot p_{ij} + \sum_{\substack{j'=1 \\ j' \neq j}}^{n} \lambda_{j'} \cdot p_{ij'} \cdot (1 - \pi_{ij'}^{(k-1)})}{\sum_{i'=1}^{N} \left(\lambda_j \cdot p_{i'j} + \sum_{\substack{j'=1 \\ j' \neq j}}^{n} \lambda_{j'} \cdot p_{i'j'} \cdot (1 - \pi_{i'j'}^{(k-1)}) \right)} \qquad (9)$$

Figs 11, 12 compare the sensitivity of L2 (Eq. (9)) and non-L2 (Eq. (7)) caching, under two different scenarios: (i) all the nodes following the same demand, and (ii) when the overactive node has a "flipped" popularity distribution (meaning that its most popular object is object N, the next most popular $N-1$, and so on). Looking at Fig. 11, we see that *L2 caching is much more sensitive to mistreatment than non-L2 caching* (notice that the angle formed by the L2 curves is much wider than the corresponding one for the non-L2 curves, and that the group has normalized cost > 1 in the first case). The reason is that every remote request affects the local state as opposed to only the remote hits under non-L2 caching. Fig. 12 shows that the intensity of the phenomenon increases under the flipped distribution case, for both the L2 and the non-L2 cases (the normalized access cost of the group grows from 1.1 to 1.5 for request imbalance 100 and L2 caching, while it goes from 0.76 to 1.1 for non-L2 caching). The reason for the increase is obvious for L2, the caches of the group are clogged by objects that are completely useless for the local users. The same happens for non-L2, but through a different mechanism. The occasional extremely unpopular objects that are brought in the cache due to the local demand, are easily locked by the miss-stream of the overactive node because these are popular objects for the second. Notice also that when there is no request imbalance (*i.e.*, when $\lambda_n/\lambda_j = 1$), the (no longer) overactive node with the flipped demand actually does worse, because it receives a 3-fold more intense miss-stream from the other nodes (that follow a different demand than its own).

A final observation regarding L2 caching is that both the overactive nodes and the rest of the group incur a higher access cost than in the non-L2 case. This is most probably attributed to the *amplification of replacement error* [19] that occurs under L2 caching (a request for an unpopular objects leads to its caching in all the nodes of the group thereby amplifying the number of replacement errors committed).

V. Mistreatment Due to Use of a Common Scheme

In this section we study cases of mistreatment due to the use of a common scheme vis-a-vis the object admission control algorithm. Specifically, we consider Single Copy (SC) schemes, like, HASH and LRU-SC [8], *i.e.*, schemes that allow for the existence of up to one copy of each object in the group and, Multiple Copy (MC) schemes, *i.e.*, schemes that allow for the existence of multiple copies of the same object at different nodes of the group. All the replacement algorithms when combined with a non-SC object admission control fall into the MC category.

A. Single Versus Multiple Copy Schemes

Fig. 13 depicts simulation results showing the average access cost of a group (social cost) under different SC and MC schemes, and for different values of t_r representing different levels of "tightness" of the group. Three types of demand are considered: lightly (a=0.2), moderately (a=0.6), and highly skewed (a=0.9) demand. The following observations apply. *Single copy schemes, (*i.e.*, HASH and LRU-SC, whose curves overlap almost completely in these figures, as the two have very similar caching behavior) perform better when the access cost between the nodes is small*. In such cases the cost of local and remote accesses is similar, so it pays to eliminate multiple copies of the same object at different nodes and instead make room for storing a larger number of distinct objects. *Multiple copy schemes, (*i.e.*, LRU and LFU) perform better when the access cost between the nodes is high*. In such cases, a much higher cost is incurred when an object is fetched from the group, so it becomes imperative to maintain some of the most popular objects locally (thereby creating multiple copies at different nodes). The threshold value of t_r at which the performance ranking between SC and MC changes depends on the skewness of the demand: the higher the skewness, the lower the value of t_r and the earlier the MC schemes become better.

It is also worthwhile noting that the curves for LFU are parallel to the x-axis, *i.e.*, the access cost is immune

[8]Under HASH, requests are received by the local node which then employs a hash function to identify the node that is responsible for the requested object. The responsible node returns the object immediately if it already caches it, or contacts the origin server, and then returns it, also keeping a local copy in this case. The local node does not keep a local copy, unless it is the one responsible for that object according to the employed hash function. Under LRU-SC (single copy), a local copy is maintained at the local node only for objects that were fetched from the origin server. When an object is fetched from elsewhere in the group, no local copy is kept. In both cases, the number of copies of each object in the group is limited to at most one.

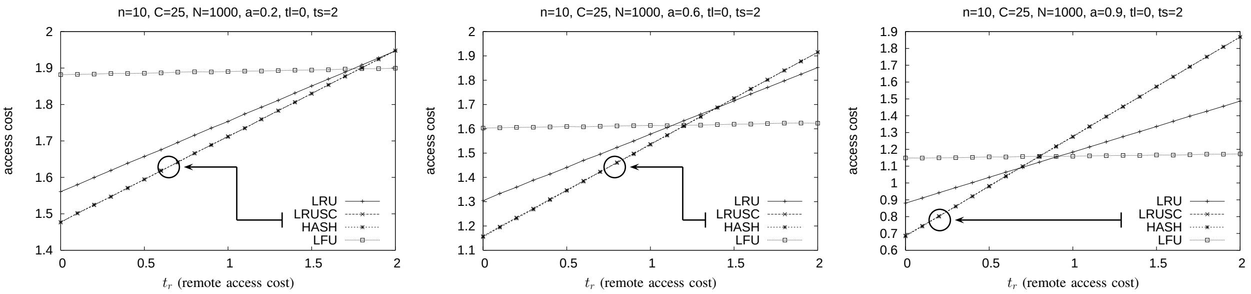

Fig. 13. Simulation results on the effect of the remote access cost t_r on the performance ranking of different SC and MC schemes for three cases of skewness of demand ($a = 0.2$, $a = 0.6$, $a = 0.9$). MC schemes (LRU, LFU) perform better when $t_r \to t_s$.

to the inter-node distance under LFU and identical demand. This happens because, as noted earlier, under LFU all the nodes store the same objects, and this has the consequence of eliminating all remote hits. In that case, the exact value of the remote access cost does not affect the LFU curves, since there are no remote hits. Regarding the relationship between LRU and LFU, the figure shows that LFU is better when the remote access cost is high (see the discussion in Section IV-E for an explanation of this).

The above observations highlight the fact that "fixed schemes" operate efficiently only under specific parameter sets. If these parameter sets are common to all the nodes, then good design choices can be made among the different schemes. When, however, some of the parameters (*e.g.*, inter-node distances) are not common to all nodes, then *it may well be the case that no single scheme is appropriate for all the nodes. Enforcing a common scheme under such conditions is bound to mistreat some of the nodes*. The following section illustrates such an example.

B. Relaxing the Common Scheme Requirement

So far, we have assumed that all group nodes are required to employ the same (common) caching scheme. In this section, we look at the advantages to be begotten from relaxing this constraint.

Consider the group depicted in Fig. 14 in which $n-1$ nodes are clustered together, meaning that they are very close to each other ($t_r \to t_l$), while there's also a single "outlier" node at distance t'_r from the cluster. The $n-1$ nodes would naturally employ the LRU-SC scheme in order to capitalize on their small remote access cost. From the previous discussion it should be clear that the best scheme for the outlier node would depend on t'_r. If $t'_r \to t_r$, the outlier should obviously follow LRU-SC and avoid duplicating objects that already exist elsewhere in the group. If $t'_r \gg t_r$, then the outlier should follow a MC scheme, e.g., LRU.

To permit the outlier to adjust its caching behavior according to its distance from the group, we introduce the LRU(q) scheme, under which, objects that are fetched from the origin server are automatically cached locally, but objects that are fetched from the group are cached locally only with probability q. For $q = 0$, LRU(q) reduces to LRU-SC, while for $q = 1$ it reduces to the multiple copy LRU scheme. One may think of q as a *reliance parameter*, capturing the confidence that a node has in its ability to fetch objects efficiently (*i.e.*, "cheaply")

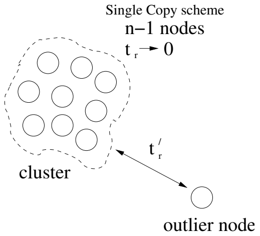

Fig. 14. An example of a group composed of a cluster of $n-1$ nodes and a unique outlier.

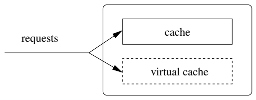

Fig. 15. Block diagram of a node equipped with a virtual cache.

from other members of the group.

Figure 16 presents the performance of LRU(q), for $q = 0, 0.1, 0.5, 1$ under different t'_r. The results are normalized by dividing the access cost of each LRU(q) scheme by the corresponding access cost of the LRU($q = 1$) scheme. The later can be seen as a basis for what a node can achieve by operating greedily, *i.e.*, when it always keeps a copy of each incoming object. Such a behavior corresponds to a node that wants to avoid relying on other nodes for fetching objects. As with the state interaction case, mistreatment is signified by a normalized access cost greater than 1. Figure 16 shows that always keeping local copies of all incoming objects (*i.e.*, employing LRU(1) and incurring a normalized access cost of 1) is a reasonably good choice across most values of t'_r. The only case that LRU(1) performs poor is when t'_r gets very small, which correspond to the case that the node ceases to be an outlier, and actually becomes part of the cluster. As discussed earlier, in this case maintaining multiple object copies within the group becomes wasteful, with the optimal scheme being the single copy LRU(0) scheme.

Another interesting observation from the above results is that there is a noticeable performance differential between the single copy LRU(0) scheme, and any other multiple copy LRU(q) scheme with $q > 0$. A non-zero LRU(q) scheme, even one where q is small, is capable of eventually caching locally the most popular objects, even if this requires several misses. LRU(0), on the other hand, has almost no chance of bringing a globally popular object locally since it is much more likely for such an object to be cached in the cluster before being requested by the outlier node (which means that it won't be cached locally). When this happens for several popular objects, the performance degradation for the outlier node becomes very serious. That is why LRU(0) performs poorly for large values of t'_r.

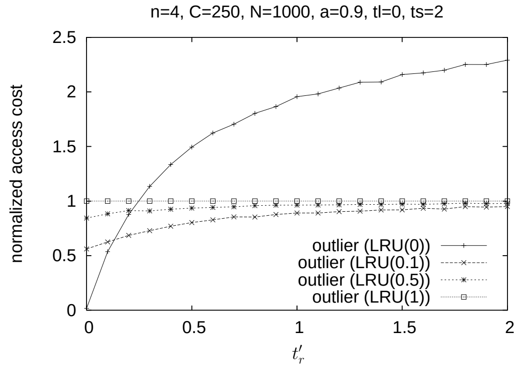

Fig. 16. Simulation results on the effect of the remote access cost t'_r on the normalized access cost of the outlier node under different LRU(q) schemes.

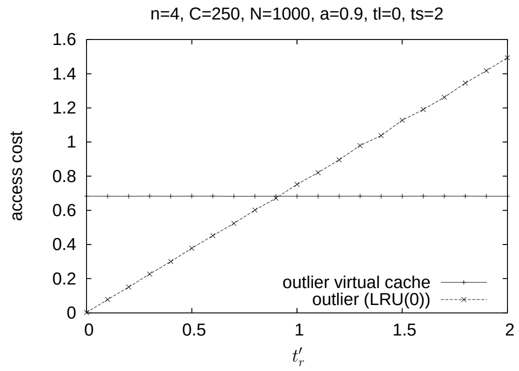

Fig. 17. Simulation results on the effect of the remote access cost t'_r on the access cost of the outlier node under the virtual cache and LRU(0) schemes.

VI. Towards Mistreatment-Resilient Caching

From our exposition so far, it should be clear that there exist situations under which an inappropriate, or enforced, scheme may mistreat some of the nodes. While we have focused on detecting and analyzing two causes of mistreatment which appear to be important (namely, due to cache state interactions and the adoption of a common cache management scheme), it should be evident that mistreatments may well arise through other causes. For example, we have not investigated the possibility of mistreatment due to request re-routing [24], not to mention that there are vastly more parameter sets and combinations of schemes that cannot all be investigated in the context of a single paper.

In an open system, the fact that group settings (*e.g.*, number of nodes, distances, demand patterns) change dynamically, precludes the possibility of addressing the matter with predefined, fixed designs. Instead, we believe that *nodes should adjust their scheme dynamically so as to avoid or respond to mistreatment if and when it emerges*. In this section, we present a framework for designing such mistreatment-resilient schemes. We discuss the use of such a framework to deal with mistreatments due to the use of a common scheme–namely the $LRU(q)$ scheme. Later in this section, we briefly discuss the use of our framework for the development of caching schemes that are resilient to state-interaction-induced mistreatments.

A first requirement for mistreatment resilience is that *a node must be able to realize that it is being mistreated*. In our previous work on replication [4], a node compared its access cost under a given replication scheme with the guaranteed maximal access cost obtained through GL replication. This gave the node a mistreatment test. In that game theoretic framework, it was necessary for each node to know its demand pattern to be able to derive its GL cost threshold. In caching, however, demand patterns (even local ones) are not known *a priori*, nor are they stationary. Thus nodes have to estimate and update their thresholds in an on-line manner. We believe that a promising approach for this is *emulation*.

Fig. 15 depicts a node equipped with an additional *virtual cache*, alongside its "real" cache that holds its objects. The virtual cache does not hold actual objects, but rather object identifiers. It is used for emulating the cache contents and the access cost under a scheme *different from* the one being currently employed by the node to manage its "real" cache under the same request sequence—notice that the input request stream is copied to both caches. The basic idea is that *the virtual cache can be used for emulating the threshold cost that the node can guarantee for itself if it employs a scheme that is different from the one currently in use*.

Resilience to Common-Scheme-Induced Mistreatments: Referring back to the outlier node of Section V-B, the virtual cache could be emulating the LRU(1) scheme, *i.e.*, the scheme in which the reliance parameter q is equal to 1. This emulates the case that the outlier node does not put any trust on the remote nodes for fetching objects and, thus, keeps copies of all incoming objects after local misses. Equipped with such a device, a node is able to calculate a running estimate of its threshold cost based on the objects it emulates as present in the virtual cache.[9] By comparing the payoff from sticking to the current scheme versus the emulated (selfish) one, a node is able to decide which one is more appropriate. Below we give such an example.

Consider the case in which an outlier emulates the selfish LRU(1) scheme, but currently employs the completely unselfish (fully cooperative) LRU(0) scheme, which is equivalent to LRU-SC. This can occur, for example, if the node knows that it is initially close to the cluster, but cannot guarantee that this will remain the case (*e.g.*, due to node movement in an ad-hoc network, or due to changing load conditions in the network). Figure 17 shows that the relative performance ranking of the two schemes depends on the distance from the group t'_r and that there is a value of t'_r in which the ranking changes. Whenever the outlier crosses this point it can compare the emulated cost from the virtual cache with its actual cost and switch schemes (employ the previously emulated one and emulate the previously employed one). One can also design a *smoother* mistreatment-resilient scheme by enabling a node to manipulate the reliance parameter q at a finer scale, in pursuit of an even lower access cost than the one offered by the two extreme schemes. Indeed, there are situations in which intermediate values of q, $0 < q < 1$, may offer superior performance (see the LRU(0.1) and LRU(0.5) curves in Fig. 16). Controlling the parameter q becomes more

[9] The outlier can include in the emulation the remote fetches that would result from misses in the emulated cache contents; this would give it the exact access cost under the emulated scheme. A simpler approach would be to disregard the remote fetches and thus reduce the inter-node query traffic; this would give it an upper bound for the access cost under the emulated scheme.

involved in this case. As part of our ongoing work, we investigate the use of different types of controllers, each of which resulting in different convergence and stability profiles.

Resilience to State-Interaction-Induced Mistreatments: Immunizing a node against mistreatments that emerge from state interactions could be similarly achieved. For instance, one may define an *interaction parameter* p_s, which is used as a control parameter for an LRU(p_s) scheme. Here, a remote hit is allowed to affect the local state with probability p_s, whereas it is denied such access with probability (1-p_s). The interaction parameter p_s could be controlled using schemes similar to those we considered above for the reliance parameter q.

It is important to note that one may argue for *isolationism* (by permanently setting $p_s = 0$) as a simple approach to avoid state-interaction-induced mistreatments. This is not a viable solution. Specifically, by adopting an LRU($p_s = 0$) approach, a node is depriving itself from the opportunity of using miss streams from other nodes to improve the accuracy of LRU-based cache/no-cache decisions (assuming a uniform popularity profile for group members). This was highlighted in the results shown in Fig. 6.

To conclude this section, we note that the approaches we presented above for mistreatment resilience may be viewed as "passive" or "end-to-end" in the sense that a node infers the onset of mistreatment *implicitly* by monitoring its utility function. As we alluded at the outset of this paper, for the emerging class of network applications for which grouping of nodes is "ad hoc" (*i.e.*, not dictated by organizational boundaries or strategic goals), this might be the only realistic solution. In particular, to understand "exactly how and exactly why" mistreatment is taking place would require the use of proactive measures (*e.g.*, monitoring/policing group member behaviors, measuring distances with pings, *etc.*), which would require group members to subscribe to some common services or to trust some common authority—both of which are not consistent with the autonomous nature (and the mutual distrust) of participating nodes.

VII. Related work

Apart from our previous work on distributed selfish replication [3], [4], we are aware of only two additional works on game-theoretic aspects of replication, one due to Chun et al. [25] (distributed selfish replication under infinite storage capacities) and the other due to Erçetin and Tassiulas [26] (market-based resource allocation in content delivery); we are not aware of any previous work on distributed selfish caching. The issue of dynamic adjustment of caching schemes has been raised recently also by Sivasubramanian et al. [27], in a completely different context from ours (consistency control of cached objects). For studies on the social utility of distributed caching we recommend Korupolu and Dahlin [28] and Rodriguez et al. [29].

VIII. Summary and Concluding Remarks

Distributed on-demand caching enables loosely coupled groups of nodes to share their (storage) resources to achieve higher efficiencies and scalability. In addition to its traditional use in content distribution/delivery networks, distributed caching is also used as an important building block of many emerging applications and protocols, including its use in route caching in ad-hoc networks [30] and in P2P content replication [2], [31].

Summary: This paper has uncovered the susceptibility of nodes participating in a distributed on-demand caching group to being *mistreated*. We have identified two causes of mistreatments–namely mistreatment due to cache *state interactions* between various members of the group, and due to the use of a *common scheme* for cache management across all members of the group. We have backed up our findings by analytic models, numerical solutions of these models, as well as simulations in which assumptions (necessary for analysis) have been relaxed.

The results of our analysis and evaluation suggest that on-demand distributed caching is fairly resilient to the onset of mistreatment as long as proxying (a.k.a. L2 caching) is not enabled, and as long as intra-group access costs do not include outliers. More constructively, we have outlined an efficient emulation-based approach that allows individual nodes to decide autonomously (i.e., without having to trust any other node or service) whether they should stick to, or secede from a caching group, based on whether or not their participation is beneficial to their performance compared to a selfish, greedy scheme.

Other Incarnations of Mistreatment in On-Line Distributed Resource Management Problems: In this paper, we focused on distributed caching as an instance of an on-line protocol for the management of a distributed resource—namely the limited storage available at each node. While our exposition has focused on the well-known problem of caching "retrievable" content (*e.g.*, web pages and media objects), it should be evident that our results extend to *any* other type of cached content, including non-retrievable content used as part of the control plane of a distributed protocol or application (*e.g.*, route paths stored in routing tables of group members). Clearly, given the different nature of the workloads that such distributed resources must support, a more specific examination of potential mistreatments in such settings is warranted, and is a current subject of inquiry of ours.

Coincidental versus Adversarial Mistreatment: In this paper we focused on the onset of mistreatment due to benign operating conditions of a caching group. For instance we identified rate imbalance (of local versus remote requests streams) conditions as well as cache sizing conditions that are necessary for mistreatment to occur. As such, the cases of mistreatment we have uncovered could be considered "coincidental". Another possible source of mistreatment, however, could be adversarially motivated, in the sense that one (or more) of the group members collude to negatively impact the performance of other members. While we did not consider adversarial mistreatments *per se*, our results suggest that distributed caching is fairly immune to *high potency exploits* [32] (a.k.a. low rate attacks) by non-clairvoyant adversaries. More work is needed to characterize the vulnerability of distributed caching to more elaborate adversarial exploits, including those from more powerful agents (*e.g.*, those with knowledge of a victim's cache contents).

Towards Intrusion-Resilient On-Line Schemes: More broadly, the results we have put forth in this paper confirm the observations in [32] that distributed on-line algorithms used by a group of autonomous, self-aware agents—of which distributed caching and replications are examples—must be re-examined to assess their vulnerability to parasitic behaviors (whether intentional or not). Game theory provides a good basis for understanding such issues, but it does not have the expressiveness to capture the vast complexities of real mechanisms. The use of tools from other domains—*e.g.*, the use of control theory in [32]—may be necessary to fully understand, and more importantly provide fixes for such problems. This is precisely our current research agenda.

REFERENCES

[1] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 767–780, Oct. 2004.

[2] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," August 2002, proceedings of ACM SIGCOMM'02 Conference.

[3] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, "Local utility aware content replication," in *Proceedings of IFIP Networking 2005*, Waterloo, Canada, May 2005.

[4] ——, "Distributed selfish replication," *IEEE Transactions on Parallel and Distributed Systems*, 2006, [accepted for publication].

[5] T. Loukopoulos, P. Lampsas, and I. Ahmad, "Continuous replica placement schemes in distributed systems," in *Proceedings of the 19th ACM International Conference on Supercomputing (ACM ICS)*, Boston, MA, June 2005.

[6] S. Jin and A. Bestavros, "Sources and Characteristics of Web Temporal Locality," in *Proceedings of Mascots'2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Fransisco, CA, August 2000.

[7] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374–398, 2003.

[8] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative Web proxy caching," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 16–31, 1999.

[9] K. W. Ross, "Hash-routing for collections of shared web caches," *IEEE Network*, vol. 11, no. 6, Nov. 1997.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

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

[12] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: the search for invariants," in *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 1996, pp. 126–137.

[13] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proceedings of USITS*, Monterey, California, United States, December 1997.

[14] N. Young, "The k-server dual and loose competitiveness for paging," *Algorithmica*, vol. 11, pp. 525–541, 1994.

[15] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, Mar. 1999.

[16] K. Psounis, A. Zhu, B. Prabhakar, and R. Motwani, "Modeling correlations in web traces and implications for designing replacement policies," *Computer Networks*, vol. 45, July 2004.

[17] A. Mahanti, C. Williamson, and D. Eager, "Traffic analysis of a web proxy caching hierarchy," *IEEE Network*, vol. 14, no. 3, pp. 16–23, May 2000.

[18] X. Tang and S. T. Chanson, "Adaptive hash routing for a cluster of client-side web proxies," *Journal of Parallel and Distributed Computing*, vol. 64, no. 10, pp. 1168–1184, Oct. 2004.

[19] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Performance Evaluation*, 2006, [to appear].

[20] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *Proceedings of ACM SIGMETRICS*, Boulder, Colorado, United States, 1990, pp. 143–152.

[21] I. Stavrakakis, "Statistical multiplexing under non-i.i.d. packet arrival processes and different priority policies." *Performormance Evaluation*, vol. 12, no. 3, pp. 181–189, 1991.

[22] R. Landry and I. Stavrakakis, "Queueing study of a 3-priority policy with distinct service strategies," *IEEE/ACM Transactions on Networking*, vol. 1, no. 5, pp. 576–589, 1993.

[23] A. Leff, J. L. Wolf, and P. S. Yu, "Replication algorithms in a remote caching architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 11, pp. 1185–1204, Nov. 1993.

[24] J. Pan, Y. T. Hou, and B. Li, "An overview DNS-based server selection in content distribution networks," *Computer Networks*, vol. 43, no. 6, Dec. 2003.

[25] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, "Selfish caching in distributed systems: A game-theoretic analysis," in *Proc. ACM Symposium on Principles of Distributed Computing (ACM PODC)*, Newfoundland, Canada, July 2004.

[26] O. Erçetin and L. Tassiulas, "Market-based resource allocation for content delivery in the internet," *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1573–1585, Dec. 2003.

[27] S. Sivasubramanian, G. Pierre, and M. van Steen, "A case for dynamic selection of replication and caching strategies," in *Proceedings of the 8th International Workshop on Web Caching and Content Distribution (WCW)*, New York, Sept. 2003.

[28] M. R. Korupolu and M. Dahlin, "Coordinated placement and replacement for large-scale distributed caches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 6, pp. 1317–1329, 2002.

[29] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of web caching architectures: Hierarchical and distributed caching," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, Aug. 2001.

[30] M. Marina and S. Das, "Performance of Route Caching Strategies in Dynamic Source Routing," in *Proceedings of the Int'l Workshop on Wireless Networks and Mobile Computing (WNMC) in conjunction with Int'l Conf. on Distributed Computing Systems (ICDCS)*, Washington, DC, USA, 2001.

[31] J. Kangasharju, K. W. Ross, and D. A. Turner, "Optimal Content Replication in P2P Communities. ," Manuscript, 2002.

[32] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the transients of adaptation for RoQ attacks on Internet resources," in *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)*, Berlin, Germany, Oct. 2004.