Adaptive Caching Networks with Optimality Guarantees

Stratis Ioannidis and Edmund Yeh
E.C.E., Northeastern University
360 Huntington Ave, 409DA
Boston, MA, 02115
{ioannidis,eyeh}@ece.neu.edu

ABSTRACT

We study the problem of optimal content placement over a network of caches, a problem naturally arising in several networking applications, including ICNs, CDNs, and P2P systems. Given a demand of content request rates and paths followed, we wish to determine the content placement that maximizes the *expected caching gain*, i.e., the reduction of routing costs due to intermediate caching. The offline version of this problem is NP-hard and, in general, the demand and topology may be a priori unknown. Hence, a distributed, adaptive, constant approximation content placement algorithm is desired. We show that path replication, a simple algorithm frequently encountered in literature, can be arbitrarily suboptimal when combined with traditional eviction policies, like LRU, LFU, or FIFO. We propose a distributed, adaptive algorithm that performs stochastic gradient ascent on a concave relaxation of the expected caching gain, and constructs a probabilistic content placement within $1-1/e$ factor from the optimal, in expectation. Motivated by our analysis, we also propose a novel greedy eviction policy to be used with path replication, and show through numerical evaluations that both algorithms significantly outperform path replication with traditional eviction policies over a broad array of network topologies.

1. INTRODUCTION

We consider a *caching network*, i.e., a network of caches, each capable of storing a constant number of content items. Certain nodes in the network act as designated sources for content, and are guaranteed to always store specific items. Any node can generate a request for an item, which is forwarded over a fixed path toward a designated source. However, requests need not reach the end of this path: forwarding stops upon reaching a node that has cached the requested item. Whenever such a "cache hit" occurs, the item is sent over the reverse path towards the node that requested it.

Our goal is to *allocate items to caches optimally*, i.e., in a way that minimizes the aggregate routing costs due to content transfers across the network. This abstract problem naturally captures—and is directly motivated by—several important real-life networking applications. These include content and information-centric networks (CCNs/ICNs) [22, 32, 33], core and edge content delivery networks (CDNs) [7, 13], micro/femtocell networks [34], and peer-to-peer networks [12, 28], to name a few. For example, in hierarchical CDNs, requests for content can be served by intermediate caches placed at the network's edge, e.g., within the same administrative domain (e.g., AS or ISP) as the originator of the request; if, however, content is not cached locally, the request can be forwarded to a core server, that acts as a cache of last resort. Similarly, in CCNs, named data items are stored at designated sources, and requests for named content are forwarded to these sources. Intermediate routers can cache items carried by responses, and subsequently serve future requests. Both settings directly map to the abstract problem we study here.

In these and many other applications, it is natural to assume that the *demand*, determined by the frequencies of requests and the paths they follow, is dynamic and not a priori known. For this reason adaptive algorithms that (a) discover an optimal item placement without prior knowledge of this demand and (b) adapt to its changes are desired. Moreover, collecting information at a single centralized location may be impractical in large networks consisting of different administrative domains. Distributed algorithms, in which a node's caching decisions rely only on locally available information, allow the network to scale and are thus preferable.

Path replication [12] is a simple, elegant algorithm that attains both properties and is often encountered in the literature of the different applications mentioned above [10, 22, 25, 28, 33, 39]. Cast in the context of our problem, the algorithm roughly proceeds as follows: when an item traverses the reverse path towards a node that requested it, it is cached by *every intermediate node encountered*. When caches are full, evictions are typicaly implemented using traditional policies, like LRU, LFU, FIFO, etc.

This algorithm is intuitively appealing in its simplicity, and it is clearly both distributed and adaptive to demand. Unfortunately, the resulting allocations of items to caches come with no guarantees: we show in this paper that path replication combined with *any* of the above traditional eviction policies is arbitrarily suboptimal. To address this, our main goal is to design a *distributed, adaptive caching algorithm with provable performance guarantees*. To that end, we make the following contributions:

- We set the problem of optimal caching network design

SIGMETRICS '16, June 14–18, 2016, Antibes Juan-Les-Pins, France.

DOI: http://dx.doi.org/10.1145/2896377.2901467

on a formal foundation. We do so by rigorously defining the problem of finding an *allocation*, i.e., a mapping of items to network caches, that maximizes the *expected caching gain*, i.e., the routing cost reduction achieved due to caching at intermediate nodes. The deterministic, combinatorial version of the problem is NP-hard, though it is approximable within a $1 - 1/e$ factor [1, 34].
- We prove that the ubiquitous path replication algorithm, combined with LRU, LFU, or FIFO eviction policies, leads to allocations that are *arbitrarily suboptimal*. Our result extends to any *myopic* strategy, that ignores costs incurred upstream due to cache misses.
- We construct a distributed, adaptive algorithm that converges to a probabilistic allocation of items to caches that is within a $1 - 1/e$ factor from the optimal, *without* prior knowledge of the demand (i.e., items requested and routes followed) or the network's topology. The algorithm performs a projected gradient ascent over a concave objective approximating the expected caching gain.
- Motivated by this construction, we also propose a new eviction policy to be used with path replication: whenever an item is back-propagated over a path, the nodes on the path have the opportunity to store it and evict an existing content, according to a greedy policy we design.
- We conduct extensive simulations over a broad array of both synthetic and real-life topologies. We show that both algorithms significantly outperform path replication combined with traditional eviction policies. In all cases studied, the greedy heuristic performs exceptionally well, achieving at least 95% of the gain achievable by the projected gradient ascent algorithm, that comes with provable guarantees.

Our analysis requires overcoming several technical hurdles. To design our distributed algorithm, we show that it is always possible to construct a probabilistic allocation, mapping items to caches, that satisfies capacity constraints *exactly*, from a probabilistic allocation that satisfies capacity constraints only in *expectation*. This construction is simple and intuitive and can be performed in polynomial time. The concave relaxation we study is non-differentiable; this introduces additional technical difficulties when performing projected gradient ascent, which we address.

The remainder of this paper is structured as follows. We review related work in Section 2. We formally introduce our problem in Section 3, and discuss offline algorithms for it solution in Section 4. Our main results on distributed, adaptive algorithms in Section 5; we also prove an equivalence theorem on several variants of the expected caching gain maximization problem in Section 6. Finally, Section 7 contains our evaluations, and we conclude in Section 8.

2. RELATED WORK

Path replication is best known as the de facto caching mechanism in content-centric networking [22], but has a long history in networking literature. In their seminal paper, Cohen and Shenker [12] show that path replication, combined with constant rate of evictions leads to an allocation that is optimal, in equilibrium, when nodes are visited through uniform sampling. This is one of the few results on path replication's optimality—see also [19]; our work (c.f. Thm. 2) proves that, unfortunately, this result does not generalize to routing over arbitrary topologies. Many studies provide numerical evaluations of path replication combined with simple eviction policies, like LRU, LFU, etc., over different topologies (see, e.g., [25, 33, 39]). In the context of CDNs and ICNs, Rosensweig et al. [32] study conditions under which path replication with LRU, FIFO, and other variants, under fixed paths, lead to an ergodic chain. Che et al. [10] approximate the LRU policy hit probability through a TTL-based eviction scheme; this approach has been refined and extended in several recent works to model many traditional eviction policies [6, 15, 16, 29]; alternative analytical models are explored in [9, 18]. None of the above works however study optimality issues or guarantees.

Several papers have studied complexity and optimization issues in offline caching problems [2, 3, 5, 7, 14, 34]. With the exception of [34], these works model the network as a bipartite graph: nodes generating requests connect directly to caches, and demands are satisfied a single hop (if at all). Beyond content placement, Deghan et al. [13] jointly optimize caching and routing in this bipartite setting. In general, the *pipage rounding* technique of Ageev and Sviridenko [1] (see also [8,36]) yields again a constant approximation algorithm in the bipartite setting, while approximation algorithms are also known for several variants of this problem [3, 5, 7, 14].

Among these papers on offline caching, the recent paper by Shanmugam et al. [34] is closest to the problem we tackle here; we rely and expand upon this work. Shanmugam et al. consider wireless nodes that download items from (micro/femtocel) base stations in their vicinity. Base stations are visited in a predefined order (e.g., in decreasing order of connection quality), with the wireless service acting as a "cache of last resort". This can be cast as an instance of our problem, with paths defined by the traversal sequence of base-stations, and the network graph defined as their union. The authors show that determining the optimal allocation is NP-hard, and that an $1 - 1/e$ approximation algorithm can be obtained through pipage rounding; we review these results, framed in the context of our problem, in Section 4.

All of the above complexity papers [1, 3, 5, 14], including [34], study *offline*, *centralized* versions of their respective caching problems. Instead, we focus on providing *adaptive, distributed* algorithms, that operate without any prior knowledge of the demand or topology. In doing so, we we produce distributed algorithms for (a) performing projected gradient ascent over the concave objective used in pipage rounding, and (b) rounding the objective across nodes; combined, these lead to our distributed, adaptive caching algorithm with provable guarantees (Thm. 4).

Adaptive replication schemes exist for asymptotically large, single-hop CDNs [20, 26, 27], but these works do not explicitly model a graph structure. The dynamics of the greedy path replication algorithm we propose in Section 5.3 resemble the greedy algorithm used to make caching decisions in [20], though our objective is different, and we cannot rely on a mean-field approximation in our argument. These dynamics are also similar (but not identical) to the dynamics of the "continuous-greedy" algorithm used for submodular maximization [36] and the Frank-Wolfe algorithm [11]; these can potentially serve as a basis for formally establishing its convergence, which we leave as future work.

The path replication eviction policy we propose also relates to greedy maximization techniques used in throughput-optimal backpressure algorithms—see, e.g., Stolyar [35] and, more recently, Yeh et al. [38], for an application to throughput-

optimal caching in ICN networks. We minimize routing costs and ignore throughput issues, as we do not model congestion. Investigating how to combine these two research directions, capitalizing on commonalities between these greedy algorithms, is an interesting open problem.

3. MODEL

We consider a network of caches, each capable of storing at most a constant number of content items. Item requests are routed over given (i.e., fixed) routes, and are satisfied upon hitting the first cache that contains the requested item. Our goal is to determine an item allocation (or, equivalently, the contents of each cache), that minimizes the aggregate routing cost. We describe our model in detail below.

3.1 Cache Contents and Designated Sources

We represent a network as a directed graph $G(V, E)$. Content items (e.g., files, or file chunks) of equal size are to be distributed across network nodes. In particular, each node is associated with a cache, that can store a finite number of items. We denote by $\mathcal{C}$ the set of content items available, i.e., the *catalog*, and assume that G is symmetric, i.e., $(i, j) \in E$ if and only if $(j, i) \in E$.

We denote by $c_v \in \mathbb{N}$ the cache capacity at node $v \in V$: exactly c_v content items are stored in this node. For each node $v \in V$, we denote by

$$x_{vi} \in \{0, 1\}, \quad \text{for } v \in V, i \in \mathcal{C},$$

the variable indicating whether v stores content item i. We denote by $X = [x_{vi}]_{v \in V, i \in \mathcal{C}} \in \{0, 1\}^{|V| \times |\mathcal{C}|}$ the matrix whose rows comprise the indicator variables of each node. We refer to X as the *global allocation strategy* or, simply, *allocation*. Note that the capacity constraints imply that

$$\textstyle\sum_{i \in \mathcal{C}} x_{vi} = c_v, \quad \text{for all } v \in V.$$

We associate each item i in the catalog $\mathcal{C}$ with a fixed set of *designated sources* $\mathcal{S}_i \subseteq V$, that always store i. That is:

$$x_{vi} = 1, \text{ for all } v \in \mathcal{S}_i.$$

Without loss of generality, we assume that the sets $\mathcal{S}_i$ are feasible, i.e., $\sum_{i : v \in \mathcal{S}_i} x_{vi} \leq c_v$, for all $v \in V$.

3.2 Content Requests and Routing Costs

The network serves content requests routed over the graph G. In short, a request is determined by (a) the item requested, and (b) the path that the request follows. Formally, a *path* p of length $|p| = K$ is a sequence

$$\{p_1, p_2, \ldots, p_K\}$$

of nodes $p_k \in V$ such that edge (p_k, p_{k+1}) is in E, for every $k \in \{1, \ldots, |p|-1\}$. Under this notation, a *request* r is a pair (i, p) where $i \in \mathcal{C}$ is the item requested, and p is the path traversed to serve this request. We say that a request (i, p) is *well-routed* if the following natural assumptions hold:

(a) The path p is simple, i.e., it contains no loops.
(b) The terminal node in the path is a designated source node for i, i.e., if $|p| = K$, $p_K \in S_i$.
(c) No other node in the path is a designated source node for i, i.e., if $|p| = K$, $p_k \notin S_i$, for $k = 1, \ldots, K-1$.

We denote by $\mathcal{R}$ the set of all requests. Without loss of generality, we henceforth assume that all requests in $\mathcal{R}$ are well-routed. Moreover, requests for each element $\mathcal{R}$ arrive according to independent Poisson processes; we denote by $\lambda_{(i,p)} > 0$ the arrival rate of a request $(i, p) \in \mathcal{R}$.

An incoming request (i, p) is routed over the network G following path p, until it reaches a cache that stores i. At that point, a *response* message is generated, carrying the item requested. The response is propagated over p in the reverse direction, i.e., from the node where the "cache hit" occurred, back to the first node in p, from which the request originated. To capture costs (e.g., delay, money, etc.), we associate a *weight* $w_{ij} \geq 0$ with each edge $(i, j) \in E$, representing the cost of transferring an item across this edge. We assume that (a) costs are solely due to response messages that carry an item, while request forwarding costs are negligible, and (b) requests and downloads are instantaneous (or, occur at a smaller timescale compared to the request arrival process). We do not assume that $w_{ij} = w_{ji}$.

When a request $(i, p) \in \mathcal{R}$ is well-routed, the cost for serving it can be written concisely in terms of the allocation:

$$C_{(i,p)} = C_{(i,p)}(X) = \sum_{k=1}^{|p|-1} w_{p_{k+1} p_k} \prod_{k'=1}^{k} (1 - x_{p_{k'} i}). \quad (1)$$

Intuitively, this formula states that $C_{(i,p)}$ includes the cost of an edge (p_{k+1}, p_k) in the path p if *all* caches preceding this edge in p do not store i. If the request is well-routed, no edge (or cache) appears twice in (1). Moreover, the last cache in p stores the item, so the request is always served.

3.3 Maximizing the Caching Gain

As usual, we seek an allocation that minimizes the aggregate expected cost. In particular, let C_0 be the expected cost per request, when requests are served by the designated sources at the end of each path, i.e.,

$$C_0 = \textstyle\sum_{(i,p) \in \mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1} p_k}.$$

Since requests are well-routed, C_0 is an upper bound on the expected routing cost. Our objective is to determine a feasible allocation X that maximizes the *caching gain*, i.e., the expected cost reduction attained due to caching at intermediate nodes, defined as:

$$\begin{aligned} F(X) &= C_0 - \textstyle\sum_{(i,p) \in \mathcal{R}} \lambda_{(i,p)} C_{(i,p)}(X) \\ &= \sum_{(i,p) \in \mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1} p_k} \left(1 - \prod_{k'=1}^{k} (1 - x_{p_{k'} i}) \right). \end{aligned} \quad (2)$$

In particular, we seek solutions to the following problem:

MaxCG

$$\text{Maximize:} \quad F(X) \quad (3a)$$

$$\text{subj. to:} \quad X \in \mathcal{D}_1 \quad (3b)$$

where $\mathcal{D}_1$ is the set of matrices $X \in \mathbb{R}^{|V| \times |\mathcal{C}|}$ satisfying the capacity, integrality, and source constraints, i.e.:

$$\textstyle\sum_{i \in \mathcal{C}} x_{vi} = c_v, \quad \text{for all } v \in V, \quad (4a)$$

$$x_{vi} \in \{0, 1\}, \quad \text{for all } v \in V, i \in \mathcal{C}, \text{ and} \quad (4b)$$

$$x_{vi} = 1, \quad \text{for all } v \in \mathcal{S}_i \text{ and all } i \in \mathcal{C}. \quad (4c)$$

Problem MaxCG is NP-hard (see Shanmugam et al. [34] for a reduction from the 2-Disjoint Set Cover Problem). Our objective is to solve MaxCG using a *distributed*, *adaptive* algorithm, that produces an allocation within a constant

Table 1: Notation Summary

$G(V, E)$	Network graph, with nodes V and edges E
$\mathcal{C}$	Item catalog
c_v	Cache capacity at node $c \in V$
w_{uv}	Weight of edge $(u, v) \in E$
$\mathcal{R}$	Set of requests (i, p), with $i \in \mathcal{C}$ and p a path
$\lambda_{(i,p)}$	Rate of request $(i, p) \in \mathcal{R}$
x_{vi}	Variable indicating $v \in V$ stores $i \in \mathcal{C}$
y_{vi}	Marginal probability that $v \in V$ stores $i \in \mathcal{C}$
X	$\lvert V\rvert \times \lvert\mathcal{C}\rvert$ matrix of x_{vi}, for $v \in V$, $i \in \mathcal{C}$
Y	$\lvert V\rvert \times \lvert\mathcal{C}\rvert$ matrix of marginals y_{vi}, $v \in V$, $i \in \mathcal{C}$
$\mathcal{D}_1$	Set of feasible allocations $X \in \{0, 1\}^{\lvert V\rvert \times \lvert\mathcal{C}\rvert}$
$\mathcal{D}_2$	Convex hull of $\mathcal{D}_1$
F	The expected caching gain (2) in $\mathcal{D}_1$, and its multi-linear relaxation (6) in $\mathcal{D}_2$
L	The concave approximation (10) of F

approximation of the optimal, *without* prior knowledge of the network topology, edge weights, or the demand.

4. PIPAGE ROUNDING

Before presenting our distributed, adaptive algorithm for solving MaxCG, we first discuss how to obtain a constant approximation solution in polynomial time in a *centralized*, *offline* fashion. To begin with, MaxCG is a submodular maximization problem under matroid constraints: hence, a solution within a 1/2 approximation from the optimal can be constructed by a greedy algorithm.[1] The solution we present below, due to Shanmugam et al. [34], improves upon this ratio using a technique called *pipage rounding* [1].

In short, the resulting approximation algorithm consists of two steps: (a) a *convex relaxation* step, that relaxes the integer program to a convex optimization problem, whose solution is within a constant approximation from the optimal, and (b) a *rounding* step, in which the (possibly) fractional solution is rounded to produce a solution to the original integer program. The convex relaxation plays an important role in our distributed, adaptive algorithm; as a result, we briefly overview pipage rounding as applied to MaxCG below, referring the interested reader to [1, 34] for further details.

Convex Relaxation. To construct a convex relaxation of MaxCG, suppose that variables x_{vi}, $v \in V$, $i \in \mathcal{C}$, are independent Bernoulli random variables. Let ν be the corresponding joint probability distribution defined over matrices in $\{0, 1\}^{\lvert V\rvert \times \lvert\mathcal{C}\rvert}$, and denote by $\mathbf{P}_\nu[\cdot]$, $\mathbb{E}_\nu[\cdot]$ the probability and expectation w.r.t. ν, respectively. Let y_{vi}, $v \in V$, $i \in \mathcal{C}$, be the (marginal) probability that v stores i, i.e.,

$$y_{vi} = \mathbf{P}_\nu[x_{vi} = 1] = \mathbb{E}_\nu[x_{vi}]. \tag{5}$$

Denote by $Y = [y_{vi}]_{v \in V, i \in \mathcal{C}} \in \mathbb{R}^{\lvert V\rvert \times \lvert\mathcal{C}\rvert}$ the matrix comprising the marginal probabilities (5). Then, for F given by (2):

$$\begin{aligned}\mathbb{E}_\nu[F(X)] &= \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{\lvert p\rvert-1} w_{p_{k+1}p_k} \left(1 - \mathbb{E}_\nu\left[\prod_{k'=1}^{k} (1 - x_{p_{k'}i})\right]\right) \\ &= \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{\lvert p\rvert-1} w_{p_{k+1}p_k} \left(1 - \prod_{k'=1}^{k} \left(1 - \mathbb{E}_\nu\left[x_{p_{k'}i}\right]\right)\right) \\ &= F(Y). \end{aligned} \tag{6}$$

[1]Starting from items placed only at designated sources, this algorithm iteratively adds items to caches, selecting at each step a feasible assignment $x_{vi} = 1$ that leads to the largest increase in the caching gain.

Note that the second equality holds by independence, and the fact that path p is simple (no node appears twice). This extension of F to the domain $[0, 1]^{\lvert V\rvert \times \lvert\mathcal{C}\rvert}$ is known as the *multi-linear relaxation* of F. Consider now the problem:

$$\text{Maximize:} \quad F(Y) \tag{7a}$$
$$\text{subject to:} \quad Y \in \mathcal{D}_2, \tag{7b}$$

where $\mathcal{D}_2$ is the set of matrices $Y = [y_{vi}]_{v \in V, i \in \mathcal{C}} \in \mathbb{R}^{\lvert V\rvert \times \lvert\mathcal{C}\rvert}$ satisfying the capacity and source constraints, with the integrality constraints relaxed, i.e.:

$$\textstyle\sum_{i \in \mathcal{C}} y_{vi} = c_v, \qquad \text{for all } v \in V, \tag{8a}$$
$$y_{vi} \in [0, 1], \qquad \text{for all } v \in V, i \in \mathcal{C}, \text{ and} \tag{8b}$$
$$y_{vi} = 1, \qquad \text{for all } v \in \mathcal{S}_i \text{ and all } i \in \mathcal{C}. \tag{8c}$$

Note that an allocation X sampled from a ν with marginals $Y \in \mathcal{D}_2$ only satisfies the capacity constraints *in expectation*; hence, X may not be in $\mathcal{D}_1$. Moreover, if X^* and Y^* are optimal solutions to (3) and (7), respectively, then

$$F(Y^*) \geq F(X^*), \tag{9}$$

as (7) maximizes the same function over a larger domain.

The multi-linear relaxation (7a) is not concave, so (7) is *not* a convex optimization problem. Nonetheless, (7) can be approximated as follows. Define $L : \mathcal{D}_2 \to \mathbb{R}$ as:

$$L(Y) = \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{\lvert p\rvert-1} w_{p_{k+1}p_k} \min\{1, \sum_{k'=1}^{k} y_{p_{k'}i}\}. \tag{10}$$

Note that L is concave, and consider now the problem:

$$\text{Maximize:} \quad L(Y) \tag{11a}$$
$$\text{Subject to:} \quad Y \in \mathcal{D}_2. \tag{11b}$$

Then, the optimal value of (11) is guaranteed to be within a constant factor from the optimal value of (7)–and, by (9), from the optimal value of (3) as well. In particular:

THEOREM 1. *[1,34] Let Y^*, and Y^{**} be optimal solutions to* (7) *and* (11)*, respectively. Then,*

$$F(Y^*) \geq F(Y^{**}) \geq (1 - \frac{1}{e}) F(Y^*). \tag{12}$$

A proof can be found in [21]. Problem (11) is convex; in fact, by introducing auxiliary variables, it can be converted to a linear program and, as such, the minimizer Y^{**} can be computed in strongly polynomial time [1].

Rounding. To produce a constant approximation solution to MaxCG, the solution Y^{**} of (11) is rounded. The rounding scheme is based on the following property of F: given a fractional solution $Y \in \mathcal{D}_2$, there is always a way to transfer mass between any two fractional variables y_{vi}, $y_{vi'}$ so that[2] (a) at least one of them becomes 0 or 1, (b) the resulting Y' under this transformation is feasible, i.e., $Y' \in \mathcal{D}_2$, and (c) the caching gain at Y' is at least as good as at Y, i.e., $F(Y') \geq F(Y)$.

This suggests the following iterative algorithm for producing an integral solution.

1. Start from Y^{**}, an optimal solution to the problem (11).

[2]Properties (a)-(c) are a direct consequence of the convexity of F when restricted to any two such variables, a property that Ageev and Sviridenko refer to as ϵ-*convexity* [1].

2. If the solution is fractional, find two variables y_{vi}, $y_{vi'}$ that are fractional: as capacities are integral, if a fractional variable exists, then there must be at least two.
3. Use the rounding described by properties (a)-(c) to transform (at least) one of these two variables to either 0 or 1, while increasing the caching gain F.
4. Repeat steps 2-3 until there are no fractional variables.

As each rounding step reduces the number of fractional variables by at least 1, the above algorithm concludes in at most $|V|\times|\mathcal{C}|$ steps, producing an integral solution $X' \in \mathcal{D}_1$. Since each rounding step can only increase F, X' satisfies:

$$F(X') \geq F(Y^{**}) \overset{(12)}{\geq} (1-\frac{1}{e})F(Y^*) \overset{(9)}{\geq} (1-\frac{1}{e})F(X^*),$$

i.e., is a $(1-\frac{1}{e})$-approximate solution to MaxCG.

5. DISTRIBUTED ADAPTIVE CACHING

5.1 Path Replication Suboptimality

Having discussed how to sove MaxCG offline, we turn our attention to distributed, adaptive algorithms. We begin with a negative result: the simple path replication algorithm described in the introduction, combined with LRU, LFU, or FIFO evictions, is arbitrarily suboptimal.

We prove this below using the simple star network illustrated in Figure 1, in which only one file can be cached at the central node. Intuitively, when serving requests from the bottom node, path replication with, e.g., LRU evictions, alternates between storing either of the two files. However, the optimal allocation is to permanently store file 2, (i.e., $x_{v2} = 1$): for large M, this allocation leads to a caching gain arbitrarily larger than the one under path replication and LRU. As $c_v = 1$, LFU and FIFO coincide with LRU, so the result extends to these policies as well.

Formally, assume that request traffic is generated only by node u: requests for items 1 and 2 are routed through paths $p_1 = \{u, v, s_1\}$ and $p_2 = \{u, v, s_2\}$, respectively. Let $\lambda_{(1,p_1)} = 1-\alpha$, $\lambda_{(2,p_2)} = \alpha$, for $\alpha \in (0,1)$. As illustrated in Figure 1, the routing cost is 1 over both (s_1, v), (v, u) and $M \gg 1$ over (s_2, v). Then, the following holds:

Theorem 2. *Let $X(t) \in \{0,1\}^2$ be the allocation of the network in Figure 1 at time t, under path replication with an LRU, LFU, or FIFO policy. Then, for $\alpha = \frac{1}{\sqrt{M}}$,*

$$\lim_{t\to\infty} \mathbb{E}[F(X(t))] / \max_{X\in\mathcal{D}_1} F(X) = O(1/\sqrt{M}).$$

Proof. The worst case cost, when cache v is empty, is: $C_0 = \alpha\times(M+1)+(1-\alpha)\times 2 = \alpha M + 2 - \alpha$. Suppose that u permanently caches item 2. This results in an expected routing cost of $\alpha\times 1+(1-\alpha)\times 2 = 2-\alpha$. Hence, an optimal allocation X^* necessarily has a caching gain $F(X^*) > \alpha M + 2 - \alpha - (2-\alpha) = \alpha M$. Consider now the path replication algorithm, in which either item is cached at v whenever it is back-propagated over the reverse path. As $c_v = 1$, the LRU, FIFO and LFU policies coincide, and yield exactly the same eviction decision. Moreover, as request arrivals are independent Poisson, the steady state probabilities that v stores item 1 or 2 are $1-\alpha$ and α, respectively. Hence, the expected routing cost in steady state is $\alpha^2 + \alpha(1-\alpha)\times(M+1) + (1-\alpha)\alpha\times 2 + (1-\alpha)^2 = 1 + \alpha M - \alpha^2 M + \alpha - \alpha^2)$, leading to a caching gain of $\alpha M + 2 - \alpha - (1 + \alpha M - \alpha^2 M + \alpha - \alpha^2) = \alpha^2 M + 1 - 2\alpha + \alpha^2$. Hence, the ratio of the expected caching gain under path replication with LRU, LFU, or FIFO evictions to $F(X^*)$ is at most $\alpha + \frac{(1-\alpha)^2}{\alpha M}$, and the theorem follows for $\alpha = 1/\sqrt{M}$. □

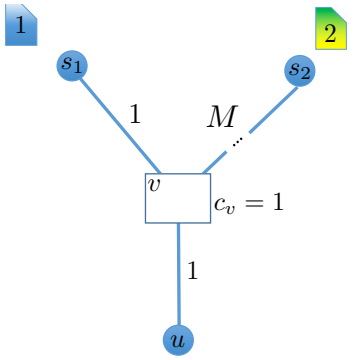

Figure 1: A simple caching network, with $\mathcal{C} = \{1, 2\}$, $S_1 = \{s_1\}$, $S_2 = \{s_2\}$. The cache at v has capacity $c_v = 1$, and the cost of the edge between v and s_2 is $M \gg 1$. Node u requests item 1 with rate $1-\alpha$, and item 2 with rate α. For $\alpha = \frac{1}{\sqrt{M}}$, path replication with LRU, LFU, or FIFO leads to an arbitrarily suboptimal caching allocation, in steady state.

Taking M to be arbitrarily large therefore makes this ratio arbitrarily small. Clearly, this argument applies to any eviction strategy that is *myopic*, *i.e.*, is insensitive to upstream costs. Accounting for the cost of an item's retrieval when caching seems necessary to provide any optimality guarantees; the algorithms we propose below indeed do so.

5.2 Projected Gradient Ascent

Given the negative result of Theorem 2, we now describe our distributed, adaptive algorithm for solving MaxCG. Intuitively, the algorithm performs a *projected gradient ascent* over function L, effectively solving the convex problem (11) in a distributed, adaptive fashion. The concavity of L ensures convergence (in contrast to minimizing F directly), while Theorem 1 ensures that the caching gain attained in steady state is within an $1-\frac{1}{e} \approx 0.62$ factor from the optimal. We describe the algorithm in detail below.

We deal with the following two challenges. First, in each step of gradient ascent, a node must estimate the contribution of its own caching allocation to the gradient of the (global) function L. The estimation should rely only on local information; additional care needs to be taken as L is not differentiable in the entire domain $\mathcal{D}_2$, so a *subgradient* needs to be estimated instead. Second, the final value of the convex relaxation (11), as well as intermediate solutions during gradient ascent, produce fractional values $Y \in \mathcal{D}_2$. To determine what to place in each cache, discrete allocations $X \in \mathcal{D}_1$ need to be determined from Y. Our algorithm *cannot* rely on pipage rounding to construct such cache allocations: each node must determine its cache contents in a distributed way, without explicitly computing F or L.

We address both challenges, by proving that (a) a *feasible* randomized rounding of any $Y \in \mathcal{D}_2$, and (b) the subgradient of L w.r.t. Y can both be computed in a distributed fashion, using only information locally available at each node.

Algorithm 1 PROJECTED GRADIENT ASCENT

1: Execute the following at each $v \in V$:
2: Pick arbitrary state $y_v^{(0)} \in \mathcal{D}_2^v$.
3: **for** each period $k \geq 1$ and each $v \in V$ **do**
4: Compute the sliding average $\bar{y}_v^{(k)}$ through (16).
5: Sample a $x_v^{(k)} \in \mathcal{D}_1^v$ from a μ_v that satisfies (17).
6: Place items $x_v^{(k)}$ in cache.
7: Collect measurements
8: At the end of the period, compute estimate z_v of $\partial_{y_v} L(Y^{(k)})$ through (18).
9: Compute new state $y_v^{(k+1)}$ through (15).
10: **end for**

5.2.1 Algorithm Overview

We begin by giving an overview of our distributed, adaptive algorithm. We partition time into periods of equal length $T > 0$, during which each node $v \in V$ collects measurements from messages routed through it. Each node keeps track of *its own marginals* $y_v \in [0,1]^{|\mathcal{C}|}$: intuitively, as in (5), each y_{vi} captures the probability that node $v \in V$ stores item $i \in \mathcal{C}$. We refer to y_v as the *state* at node v; these values, as well as the cache contents of a node, remain constant during a measurement period. When the period ends, each node (a) adapts its state vector y_v, and (b) reshuffles the contents of its cache, in a manner we describe below.

In short, at any point in time, the (global) allocation $X \in \mathcal{D}_1$ is sampled from a joint distribution μ that has a *product form*; for every v, there exist appropriate probability distributions μ_v, $v \in V$, such that:

$$\mu(X) = \textstyle\prod_{v \in V} \mu_v(x_{v1}, \ldots, x_{v|\mathcal{C}|}). \tag{13}$$

Moreover, each marginal probability $\mathbf{P}_\mu[x_{vi} = 1]$ (i.e., the probability that node v stores i) is determined as a "smoothened" version of the current state variable y_{vi}.

State Adaptation. A node $v \in V$ uses local measurements collected from messages it receives during a period to produce a random vector $z_v \in \mathbb{R}_+^{|\mathcal{C}|}$ that is an unbiased estimator of a subgradient of L w.r.t. to y_v. That is, if $Y^{(k)} \in \mathbb{R}^{|V| \times |\mathcal{C}|}$ is the (global) matrix of marginals at the k-th measurement period, $z_{vi} = z_{vi}(Y^{(k)})$ is a random variable satisfying:

$$\mathbb{E}\left[z_v(Y^{(k)})\right] \in \partial_{y_v} L(Y^{(k)}) \tag{14}$$

where $\partial_{y_v} L(Y)$ is the set of subgradients of L w.r.t y_v. We specify how to produce such estimates in a distributed fashion below, in Section 5.2.2. Having these estimates, each node adapts its state as follows: at the conclusion of the k-th period, the new state is computed as

$$y_v^{(k+1)} \leftarrow \mathcal{P}_{\mathcal{D}_2^v}\left(y_v^{(k)} + \gamma_k \cdot z_v(Y^{(k)})\right), \tag{15}$$

where $\gamma_k > 0$ is a gain factor and $\mathcal{P}_{\mathcal{D}_2^v}$ is the projection to v's set of relaxed constraints:

$$\mathcal{D}_2^v = \{y_v \in [0,1]^{|\mathcal{C}|} : \sum_{i \in \mathcal{C}} y_{vi} = c_v, y_{vi} = 1, \text{for } i \text{ s.t } v \in \mathcal{S}_i\}.$$

State Smoothening. Upon performing the state adaptation (15), each node $v \in V$ computes the following "sliding average" of its current and past states:

$$\bar{y}_v^{(k)} = \textstyle\sum_{\ell=\lfloor \frac{k}{2} \rfloor}^{k} \gamma_\ell y_v^{(\ell)} \Big/ \left[\sum_{\ell=\lfloor \frac{k}{2} \rfloor}^{k} \gamma_\ell\right]. \tag{16}$$

This "state smoothening" is necessary precisely because of the non-differentiability of L [30]. Note that $\bar{y}_v^{(k)} \in \mathcal{D}_2^v$, as a convex combination of points in $\mathcal{D}_2^v$.

Cache reshuffling. Finally, given $\bar{y}_v$, each node $v \in V$ reshuffles its contents, placing items in its cache *independently of all other nodes*: that is, node v selects a random allocation $x_v^{(k)} \in \{0,1\}^{|\mathcal{C}|}$ sampled independently of any other node in V, so that the joint distribution satisfies (13).

In particular, $x_v^{(k)}$ is sampled from a distribution μ_v that has the following two properties:

1. μ_v is a distribution over *feasible* allocations, satisfying v's capacity and source constraints, i.e., μ_v's support is:

$$\mathcal{D}_1^v = \{x_v \in \{0,1\}^{|\mathcal{C}|} : \sum_{j \in \mathcal{C}} x_{vj} = c_v, x_{vi} = 1, \text{for } i \text{ s.t } v \in \mathcal{S}_i\}.$$

2. μ_v is *consistent* with the marginals $\bar{y}_v^{(k)} \in \mathcal{D}_2^v$, i.e.,

$$\mathbb{E}_{\mu_v}[x_{vi}^{(k)}] = \bar{y}_{vi}^{(k)}, \text{ for } i \in \mathcal{C}. \tag{17}$$

It is not obvious that a μ_v that satisfies these properties exists. We show below, in Section 5.2.3, that such a μ_v exists, it has $O(|\mathcal{C}|)$ support, and can be computed in $O(c_v|\mathcal{C}| \log |\mathcal{C}|)$ time. As a result, having $\bar{y}_v$, each node v can sample a feasible allocation x_v from μ_v in $O(c_v|\mathcal{C}| \log |\mathcal{C}|)$ time.

The complete projected gradient ascent algorithm is summarized in Algorithm 1. A node may need to retrieve new items, not presently in its cache, to implement the sampled allocation $x_v^{(k)}$. This incurs additional routing costs but, if T is large, this traffic is small compared to regular response message traffic. Before we state its convergence properties, we present the two missing pieces, namely, the subradient estimation and randomized rounding we use.

5.2.2 Distributed Sub-Gradient Estimation

We now describe how to compute the estimates z_v of the subgradients $\partial_{y_v} L(Y)$ in a measurement period, dropping the superscript $\cdot^{(k)}$ for brevity. These estimates are computed in a distributed fashion at each node, using only information available from messages traversing the node. This computation requires additional "control" messages to be exchanged between nodes, beyond the usual request and response traffic.

The estimation proceeds as follows. Given a path p and a $v \in p$, denote by $k_p(v)$ is the position of v in p; i.e., $k_p(v)$ equals the $k \in \{1, \ldots, |p|\}$ such that $p_k = v$. Then:

1. Every time a node generates a new request (i, p), it also generates an additional control message to be propagated over p, in parallel to the request. This message is propagated until a node $u \in p$ s.t. $\sum_{\ell=1}^{k_p(u)} y_{p_\ell i} > 1$ is found, or the end of the path is reached. This can be detected by summing the state variables y_{vi} as the control message traverses nodes $v \in p$ up to u.
2. Upon reaching either such a node or the end of the path, the control message is sent down in the reverse direction. Every time it traverses an edge in this reverse direction, it adds the weight of this edge into a weight counter.
3. Every node on the reverse path "sniffs" the weight counter field of the control message. Hence, every node visited learns the sum of weights of all edges connecting it to visited nodes further upstream towards u; i.e., visited

node $v \in p$ learns the quantity:

$$t_{vi} = \sum_{k'=k_v(p)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} y_{p_\ell i} \leq 1}.$$

4. Let $\mathcal{T}_{vi}$ be the set of quantities collected in this way at node v regarding item $i \in \mathcal{C}$ during a measurement period of duration T. At the end of the measurement period, each node $v \in V$ produces the following estimates:

$$z_{vi} = \frac{1}{T} \sum_{t \in \mathcal{T}_{vi}} t, \quad i \in \mathcal{C}. \tag{18}$$

Note that, in practice, this needs to be computed only for $i \in \mathcal{C}$ for which v has received a control message.

Note that the control messages in the above construction are "free" under our model, as they do not carry an item. Moreover, they can be piggy-backed on/merged with request/response messages, wherever the corresponding traversed subpaths of p overlap. It is easy to show that the above estimate is an unbiased estimator of the subgradient:

LEMMA 1. *For $z_v = [z_{vi}]_{i \in \mathcal{C}} \in \mathbb{R}_+^{|\mathcal{C}|}$ the vector constructed through coordinates* (18)*,*

$$\mathbb{E}[z_v(Y)] \in \partial_{y_v} L(Y) \text{ and } \mathbb{E}[\|z_v\|_2^2] < W^2 |V|^2 |\mathcal{C}| (\Lambda^2 + \frac{\Lambda}{T}),$$

where $W = \max_{(i,j) \in E} w_{ij}$ *and* $\Lambda = \max_{v \in V, i \in \mathcal{C}} \sum_{(i,p) \in \mathcal{R}: v \in p} \lambda_{(i,p)}$.

PROOF. First, let:

$$\overline{\partial_{y_{vi}} L}(Y) = \sum_{(i,p) \in \mathcal{R}: v \in p} \lambda_{(i,p)} \sum_{k'=k_p(v)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} y_{p_\ell i} \leq 1} \tag{19a}$$

$$\underline{\partial_{y_{vi}} L}(Y) = \sum_{(i,p) \in \mathcal{R}: v \in p} \lambda_{(i,p)} \sum_{k'=k_p(v)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} y_{p_\ell i} < 1} \tag{19b}$$

where $k_p(v)$ is the position of v in p, i.e., it is the $k \in \{1, \ldots, |p|\}$ such that $p_k = v$.

A vector $z \in \mathbb{R}^{|\mathcal{C}|}$ belongs to the subgradient set $\partial_{y_v} L(Y)$ if and only if $z_i \in [\underline{\partial_{y_{vi}} L}(Y), \overline{\partial_{y_{vi}} L}(Y)]$. If L is differentiable at Y w.r.t y_{vi}, the two limits coincide and are equal to $\frac{\partial L}{\partial y_{vi}}$. It immediately follows from the fact that requests are Poisson that $\mathbb{E}[z_{vi}(Y)] = \overline{\partial_{y_{vi}} L}(Y)$, so indeed $\mathbb{E}[z_v(Y)] \in \partial_{y_v} L(Y)$. To prove the bound on the second moment, note that $\mathbb{E}[z_{vi}^2] = \frac{1}{T^2} \mathbb{E}[(\sum_{t \in \mathcal{T}_{vi}} t)^2] \leq \frac{W^2 |V|^2}{T^2} \mathbb{E}\left[|\mathcal{T}_{vi}|^2\right]$ as $t \leq W|V|$. On the other hand, $|\mathcal{T}_{vi}|$ is Poisson distributed with expectation $\sum_{(i,p) \in \mathcal{R}: v \in p} \lambda_{(i,p)} T$, and the lemma follows. □

5.2.3 Distributed Randomized Rounding

We now turn our attention to the distributed, randomized rounding scheme executed each node $v \in V$. To produce a μ_v over $\mathcal{D}_1^v$ that satisfies (17), note that it suffices to consider μ_v such that $\mathbb{E}_{\mu_v}[x_v] = \bar{y}_v$, defined over the set:

$$\bar{\mathcal{D}}_1^v = \{x_v \in \{0,1\}^{|\mathcal{C}|} : \sum_{i \in \mathcal{C}} x_{vi} = c_v\}. \tag{20}$$

That is, subject to attaining correct marginals, one can ignore the source constraints: to see this, note that if $v \in S_i$, $\bar{y}_{vi} = 1$ for any $Y \in \mathcal{D}_2$. Hence, (17) ensures that v stores item i w.p. 1. We thus focus on constructing a distribution μ_v over $\bar{\mathcal{D}}_1^v$, under a given set of marginals $\bar{y}_v$. Note that a "naïve" construction in which x_{vi}, $v \in V$, $i \in \mathcal{C}$, are independent Bernoulli variables with parameters $\bar{y}_{vi}$ indeed satisfies (17), but *does not* yield vectors $x_v \in \bar{\mathcal{D}}_1^v$: indeed, such vectors only satisfy the capacity constraint in expectation, and may contain fewer or more items than c_v.

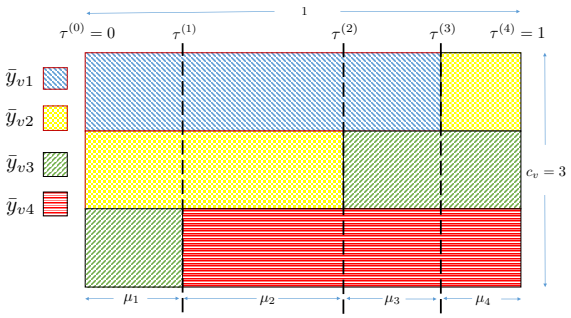

Figure 2: An allocation that satisfies $\mathbb{E}_\mu[x_{vi}] = \bar{y}_{vi}$, when $\sum_{i \in \mathcal{C}} \bar{y}_{vi} = c_v$. After placing the 4 rectangles in a 3×1 grid, assigning probabilities μ_1, μ_2, μ_3, μ_4 to each of the tuples $\{1,2,3\}$, $\{1,2,4\}$, $\{1,3,4\}$, $\{2,3,4\}$, respectively, yields the desired marginals.

Before we formally present our algorithm we first give some intuition behind it, also illustrated in Figure 2. Let $c_v = 3$, $\mathcal{C} = \{1,2,3,4\}$, and consider a $\bar{y}_v \in \mathcal{D}_2^v$. To construct an allocation with the desired marginal distribution, consider a rectangle box of area $c_v \times 1$. For each $i \in \mathcal{C}$, place a rectangle of length $\bar{y}_{vi}$ and height 1 inside the box, starting from the top left corner. If a rectangle does not fit in a row, cut it, and place the remainder in the row immediately below, starting again from the left. As $\sum_{i=1}^{|\mathcal{C}|} \bar{y}_{vi} = c_v$, this space-filling method completely fills (i.e., tessellates) the $c_v \times 1$ box.

Consider now, for each row, all fractional values $\tau \in [0,1]$ at which two horizontal rectangles meet. We call these values the *cutting points*. Notice that there can be at most $|\mathcal{C}| - 1$ such points. Then, partition the $c_v \times 1$ box vertically, splitting it at these cutting points. This results in at most $|\mathcal{C}|$ vertical partitions (also rectangles), with c_v rows each. Note that each one of these vertical partitions correspond to tuples comprising c_v distinct items of $\mathcal{C}$. Each row of a vertical partition must contain some portion of a horizontal rectangle, as the latter tessellate the entire box. Moreover, no vertical partition can contain the same horizontal rectangle in two rows or more (i.e., a horizontal rectangle cannot "overlap" with itself), because $\bar{y}_{vi} \leq 1$, for all $\bar{y}_{vi} \in \mathcal{C}$. The desired probability distribution μ_v can then be constructed by setting (a) its support to be the c_v-tuples defined by each vertical partition, and (b) the probability of each c-tuple to be the length of the partition (i.e., the difference of the two consecutive τ cutting points that define it). The marginal probability of an item will then be exactly the length of its horizontal rectangle, i.e., $\bar{y}_{vi}$, as desired.

The above process is described formally in Algorithm 2. The following lemma establishes its correctness:

LEMMA 2. *Alg. 2 produces a μ_v over $\bar{\mathcal{D}}_1^v$ s.t.* (17) *holds.*

A proof can be found in [21]. Note that, contrary to the "naïve" Bernoulli solution, the resulting variables x_{vi}, x_{vj}, where $i \neq j$, *may not be independent* (even though allocations are independent across caches). The algorithm's complexity is $O(c_v |\mathcal{C}| \log |\mathcal{C}|)$, as the sort in line 8 can be implemented in $O(c_v |\mathcal{C}| \log |\mathcal{C}|)$ time, while a match in 13 can be

Algorithm 2 PLACEMENT ALGORITHM

Input: capacity c_v, marginals $\bar{y}_v \in \mathbb{R}^{|\mathcal{C}|}$ s.t. $\bar{y}_v \geq \mathbf{0}$, $\sum_{i=1}^{|\mathcal{C}|} \bar{y}_{vi} = c_v$
Output: prob. distr. μ_v over $\{x \in \{0,1\}^{|\mathcal{C}|} : \sum_{i=1}^{|\mathcal{C}|} = c_v\}$ s.t. $\mathbb{E}_\mu[x_i] = \bar{y}_{vi}$, for all $i \in \mathcal{C}$.
$\texttt{sum} \leftarrow 0$
for all $i \in \mathcal{C}$ **do**
  $s_i \leftarrow \texttt{sum}$; $t_i \leftarrow \texttt{sum} + \bar{y}_{vi}$; $\tau_i \leftarrow t_i - \lfloor t_i \rfloor$
  $\texttt{sum} \leftarrow t_i$
end for
Sort all τ_i in increasing order, remove duplicates, and append 1 to the end of the sequence.
Let $0 = \tau^{(0)} < \tau^{(1)} < \ldots < \tau^{(K)} = 1$ be the resulting sequence.
for all $k \in \{0, \ldots, K-1\}$ **do**
  Create new vector $x \in \{0,1\}^{|\mathcal{C}|}$; set $x \leftarrow \mathbf{0}$.
  for all $\ell \in \{0, \ldots, c-1\}$ **do**
    Find $i \in \mathcal{C}$ such that $(\ell + \tau^{(k)}, \ell + \tau^{(k+1)}) \subset [s_i, t_i]$.
    Set $x_i \leftarrow 1$
  end for
  Set $\mu_v(x) = \tau^{(k+1)} - \tau^{(k)}$
end for
return μ_v

found in $O(\log |\mathcal{C}|)$ time if intervals are stored in a binary search tree. Moreover, the support of μ_v has size at most $|\mathcal{C}|$, so representing this distribution requires $O(c_v|\mathcal{C}|)$ space.

5.2.4 Convergence

We now establish the convergence of the smoothened marginals to a global maximizer of L:

THEOREM 3. *Let $\bar{Y}^{(k)} \in \mathcal{D}_2$ be the smoothened marginals at the k-th period of Algorithm 1. Then,*

$$\varepsilon_k \equiv \mathbb{E}[\max_{Y \in \mathcal{D}_2} L(Y) - L(\bar{Y}^{(k)})] \leq \frac{D^2 + M^2 \sum_{\ell=\lfloor k/2 \rfloor}^{k} \gamma_\ell^2}{\sum_{\ell=\lfloor k/2 \rfloor}^{k} \gamma_\ell},$$

where $D = \sqrt{2|V| \max_{v \in V} c_v}$, $M = W|V|\Lambda\sqrt{|V||\mathcal{C}|(1 + \frac{1}{\Lambda T})}$. *In particular, for* $\gamma_k = \frac{D}{M\sqrt{k}}$, *we have* $\varepsilon_k \leq O(1)\frac{MD}{\sqrt{k}}$, *where* $O(1)$ *is an absolute constant.*

PROOF. Under dynamics (15) and (16), from Theorem 14.1.1, page 215 of Nemirofski [30], we have that

$$\mathbb{E}[\max_{Y \in \mathcal{D}_2} L(Y) - L(\bar{Y}^{(k)})] \leq \frac{D^2 + M^2 \sum_{\ell=\lfloor k/2 \rfloor}^{k} \gamma_\ell^2}{\sum_{\ell=\lfloor k/2 \rfloor}^{k} \gamma_\ell}$$

where $D \equiv \max_{x,y \in \mathcal{D}_2} \|x - y\|_2 = \sqrt{\max_v 2|V|c_v}$ is the diameter of $\mathcal{D}_2$, and

$$M \equiv \sup_{Y \in \mathcal{D}_2} \sqrt{\sum_{v \in V} \mathbb{E}[\|z_v(Y)\|_2^2]} \leq W|V|\sqrt{|V||\mathcal{C}|(\Lambda^2 + \frac{\Lambda}{T})},$$

where the last equality follows from Lemma 1. □

The $O(1)\frac{MD}{\sqrt{k}}$ upper bound presumes knowledge of D and M when setting the gains γ_k, $k \geq 1$. Nonetheless, even when these are not apriori known, taking $\gamma_k = 1/\sqrt{k}$ suffices to ensure the algorithm converges with rate $1/\sqrt{k}$, up to (larger) constants, that depend on D and M. Moreover, the relationship between M and T captures the tradeoff induced by T: larger Ts give more accurate estimates of the subgradients, reducing the overall number of steps till convergence, but increase the length of each individual period. Finally, Thms. 1 and 3 imply that the asymptotic expected caching gain under Algorithm 1 is within a constant factor from the optimal:

THEOREM 4. *Let $X^{(k)} \in \mathcal{D}_1$ be the allocation at the k-th period of Algorithm 1. Then, if $\gamma_k = \Theta(1/\sqrt{k})$,*

$$\lim_{k \to \infty} \mathbb{E}[F(X^{(k)})] \geq \left(1 - \frac{1}{e}\right) \max_{X \in \mathcal{D}_1} F(X).$$

PROOF. Given $\bar{Y}^{(k)}$, by Lemma 2, $X^{(k)}$ is sampled from a distribution μ over $\mathcal{D}_1$ that has product form (13). This product form implies that, conditioned on $\bar{Y}^{(k)}$, Eq. (6) holds; thus, $\mathbb{E}[F(X^{(k)}) \mid \bar{Y}^{(k)}] = F(\bar{Y}^{(k)})$, so $\lim_{k\to\infty} \mathbb{E}[F(X^{(k)})] = \lim_{k\to\infty} \mathbb{E}[F(\bar{Y}^{(k)})]$. From Thm. 3, $\lim_{k\to\infty} \mathbb{E}[L(\bar{Y}^{(k)})] = \max_{Y \in \mathcal{D}_2} L(Y)$. This implies that, for $\nu^{(k)}$ the distribution of $\bar{Y}^{(k)}$, and Ω the set of $Y \in \mathcal{D}_2$ that are maximizers of L, $\lim_{k\to\infty} \nu^{(k)}(\mathcal{D}_2 \setminus \Omega) = 0$. From Theorem 1, $F(Y) \geq (1 - 1/e) \max_{X \in \mathcal{D}_1} F(X)$ for any $Y \in \Omega$. The theorem therefore follows from the above observations, and the fact that F is bounded in $\mathcal{D}_2 \setminus \Omega$. □

We state these results under stationary demands but, in practice, we would prefer that caches adapt to demand fluctuations. To achieve this, one would fix γ to a constant positive value, ensuring that Algorithm 1 tracks demand changes. Though convergence to a minimizer is not guaranteed in this case, the algorithm is nonetheless guaranteed to reach states concentrated around an optimal allocation (see, e.g., Chapter 8 of Kushner & Yin [24]).

5.3 Greedy Path Replication

Algorithm 1 has certain drawbacks. First, to implement an allocation at the end of a measurement period, nodes may need to retrieve new items, which itself incurs additional traffic costs. Second, there is a timescale separation between how often requests arrive and when adaptations happen; an algorithm adapting at the request timescale may converge faster. Third, caches are synchronized, and avoiding such coordination is preferable. Finally, beyond request and response messages, the exchange of additional control messages are required to implement it.

In this section, we propose a greedy eviction policy, to be used with the path replication algorithm, that has none of the above drawbacks. This algorithm does not require any control traffic beyond the traffic generated by message exchanges. It is asynchronous, and its adaptations happen at the same timescale as requests. Each node makes caching decisions *only* when it receives a response message carrying an item: that is, a node decides whether to store an item exactly when it passes through, and requires no additional traffic to retrieve it. Finally, the eviction heuristic is very simple (though harder to analyze than Algorithm 1).

5.3.1 Algorithm Overview

In short, every $v \in V$ maintains an estimate z_v of $\partial_{x_v} L(X)$, i.e., a subgradient of L w.r.t. its allocation $x_v \in \{0,1\}^{|\mathcal{C}|}$. At any point in time, v stores all i s.t. $v \in S_i$; the remaining slots are occupied with items of *steepest gradient value*, namely, the items i that correspond to highest estimates z_{vi}. Crucially, a gradient estimate z_{vi} increases *only when a packet carrying i passes through v*. This ensures that an item entering the cache is always available. In more detail:

1. As in classic path replication, for each $(i,p) \in \mathcal{R}$, request messages are propagated until they reach a node u

Algorithm 3 GREEDY PATH REPLICATION

Execute the following at each $v \in V$:

Initialize $z_v = \mathbf{0}$.
while (`true`) **do**
  Wait for new response message.
  Upon receipt of new message, extract counter t_{vi} and i.
  Update z_v through (22).
  Sort z_{vj}, $j \in \mathcal{C}$, in decreasing order.
  if $x_{vi} = 0$ and i is in top c'_v items **then**
    Set $x_{vi} \leftarrow 1$; evict $c'_v + 1$-th item.
  end if
end while

caching the requested item $i \in \mathcal{C}$, i.e., for which $x_{ui} = 1$. Upon reaching such a node, a response message carrying the item is backpropagated over p.

2. The response message for a request (i, p) contains a weight counter that is initialized to zero by u. Whenever the response traverses an edge in the reverse path, the edge weight is added to the counter. The counter is "sniffed" by every node in p that receives the response. Hence, every node v in the path p that is visited by a response learns the quantity:
$$t_{vi} = \sum_{k'=k_v(p)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} x_{p_\ell i} < 1}, \qquad (21)$$
where, as before, $k_v(p)$ is the position of v in path p.
3. For each item i, each node v maintains again an estimate $z_v \in \mathbb{R}_+^{|\mathcal{C}|}$ of a subgradient in $\partial_{x_{vi}} L(X)$. This estimate is maintained through an *exponentially weighted moving average* (EWMA) of the quantities t_{vi} collected above. These are adapted each time v receives a response message. If v receives a response message for i at time t, then it adapts its estimates as follows: for all $j \in \mathcal{C}$,
$$z_{vj}(t) = z_{vj}(t') \cdot e^{-\beta(t-t')} + \beta \cdot t_{vi} \cdot \mathbb{1}_{i=j}, \qquad (22)$$
where $\beta > 0$ is the EWMA gain, and $t' < t$ is the last time node v it received a response message prior to t. Thus, all estimates z_{vj} decay exponentially between responses, while only z_{vi}, corresponding to the requested item i, contains an additional increment βt_{vi}.
4. After receiving a response and adapting z_v, the node (a) sorts z_{vi}, $i \in C$, in a decreasing order, and (b) stores the top c'_v items, where $c'_v = c_v - |\{i : v \in S_i\}|$ is v's capacity excluding permanent items.

The above steps are summarized in Algorithm 3. Note that, upon the arrival of a response carrying item i, there are only two possible outcomes after the new z_v values are sorted: either (a) the cache contents remain unaltered, or (b) item i, which was previously not in the cache, is now placed in the cache, and another item is evicted. These are the only possibilities because, under (22), all items $j \neq i$ *preserve their relative order*: the only item whose relative position may change is i. As i is piggy-backed in the response, no additional traffic is needed to acquire it.

5.3.2 Formal Properties

Though simpler to describe and implement, Algorithm 3 is harder to analyze than Algorithm 1. Nonetheless, some intuition on its performance can be gained by looking into its fluid dynamics. In particular, let $X(t) = [x_{vi}(t)]_{v\in V, i\in\mathcal{C}}$ and $Z(t) = [z_{vi}(t)]_{v\in V, i\in\mathcal{C}}$ be the allocation and subgradient estimation matrices at time $t \geq 0$. Ignoring stochasticity, the "fluid" trajectories of these matrices are described by the following ODE:
$$X(t) \in \arg\max_{X\in\mathcal{D}_1} \langle X, Z(t)\rangle \qquad (23a)$$
$$\frac{dZ(t)}{dt} = \beta\big(\underline{\partial L}(X(t)) - Z(t)\big) \qquad (23b)$$
where $\langle A, B\rangle = \mathtt{trace}(AB^\top) = \sum_{v,i} A_{vi}B_{vi}$ is the inner product between two matrices, and $\underline{\partial L} \in \partial L$ is a subgradient of L at X.

PROOF. By the "baby Bernoulli" approximation of a Poisson process, and the fact that $e^x = 1 + x + o(x)$, for small $\delta > 0$ the EWMA adaptations have the form:
$$z_{vi}(t+\delta) = (1 - \beta\delta)z_{v_i}(t) + \beta\delta\phi_{vi} + o(\delta)$$
where $\mathbb{E}[\phi_{vi}] = \partial_{x_{vi}} L(X)$, and $\underline{\partial_{x_{vi}} L}$ is given by (19b); note that the matrix $\underline{\partial L}$ is indeed a subgradient. The fluid dynamics (23) then follow by taking δ to go to zero, and replacing the ϕ_v's by their expectation. □

The dynamics (23) are similar (but not identical) to the "continuous greedy" algorithm for submodular maximization [36] and the Frank-Wolfe algorithm [11]. Eq. (23b) implies that $Z(t)$ indeed "tracks" a subgradient of L at X. On the other hand, the allocation selected by sorting—or, equivalently, by (23a)—identifies the most valuable items at each cache w.r.t. the present estimate of the subgradient. Hence, even if z_v is an inaccurate estimate of the subgradient at v, the algorithm treats it as correct and places the "most valuable" items in its cache. This is why we refer to this algorithm as "greedy". Note that (23a) also implies that
$$X(t) \in \arg\max_{Y\in\mathcal{D}_2} \langle Y, Z(t)\rangle.$$
This is because, subject to the capacity and source constraints, $\langle\cdot, Z\rangle$ is maximized by taking any set of top c'_v items, so an integral solution indeed always exists. The following lemma states that fixed points of the ODE (23), provided they exist, must be at maximizers of L.

LEMMA 3. *Let $X^* \in \mathcal{D}_2$ and $Z^* \in \mathbb{R}^{|V|\times|\mathcal{C}|}$ be such that $X^* \in \arg\max_{X\in\mathcal{D}_2}\langle X, Z^*\rangle$ and $Z^* \in \partial L(X^*)$. Then, $X^* \in \arg\max_{X\in\mathcal{D}_2} L(X)$.*

The lemma holds by the concavity of L, and is stated as Theorem 27.4 of Rockafellar [31], so we omit its proof. Though the conditions stated in the lemma are both necessary and sufficient in our case, the lemma does not imply that a *integral* solution (i.e., one in which $X^* \in \mathcal{D}_1$) need exist. In practice, the algorithm may converge to a chain-recurrent set of integral solutions. Though we do not study the optimality properties of this set formally, our numerical evaluations in Section 7 show that this greedy heuristic has excellent performance in practice, very close to the one attained by the maximizer of L.

6. OFFLINE PROBLEM EQUIVALENCE

Each iteration of the projected gradient ascent algorithm of Section 5 constructs probabilistic allocations that are (a) feasible, and (b) independent across nodes. This motivates us to study the following probabilistic relaxations of MAXCG,

beyond the "independent Bernoulli" relaxation (7) we discussed in Section 4. First, consider the variant:

$$\text{Max.:} \quad \mathbb{E}_{\mu}[F(X)] = \sum_{X \in \mathcal{D}_1} \mu(X) F(X) \tag{24a}$$

$$\text{subj. to:} \quad \mu \text{ is a pr. distr. over } \mathcal{D}_1 \text{ satisfying (13).} \tag{24b}$$

I.e., we seek random cache allocations sampled from a joint distribution μ having product form (13). In addition, consider the following (more general) variant of MaxCG:

$$\text{Maximize:} \quad \mathbb{E}_{\mu}[F(X)] = \sum_{X \in \mathcal{D}_1} \mu(X) F(X) \tag{25a}$$

$$\text{subj. to:} \quad \mu \text{ is a pr. distr. over } \mathcal{D}_1. \tag{25b}$$

Our results and, in particular, Lemma 2, have the following surprising implication: all three relaxations (7), (24), and (25) *are in fact equivalent to* MaxCG.

Theorem 5. *Let X^*, Y^*, μ^*, and μ^{**} be optimal solutions to* (3),(7), (24)*, and* (25)*, respectively. Then,*

$$F(X^*) = F(Y^*) = \mathbb{E}_{\mu^*}[F(X)] = \mathbb{E}_{\mu^{**}}[F(X)].$$

Proof. We establish the following inequalities:

$$\mathbb{E}_{\mu^{**}}[F(X)] \leq F(X^*) \leq F(Y^*) \leq \mathbb{E}_{\mu^*}[F(x)] \leq \mathbb{E}_{\mu^{**}}[F(X)]$$

To see that $\mathbb{E}_{\mu^{**}}[F(X)] \leq F(X^*)$, let $D = \texttt{supp}(\mu^{**}) \subseteq \mathcal{D}_1$ be the support of μ^{**}. Let $X' \in \arg\max_{X \in D} F(X)$ be an allocation maximizing F over D (as D is finite, this exists). Then, by construction, $\mathbb{E}_{\mu^{**}}[F(X)] \leq F(X') \leq F(X^*)$, as $X' \in \mathcal{D}_1$. $F(X^*) \leq F(Y^*)$ by (9), as (7) is a relaxation of (3). To see that $F(Y^*) \leq \mathbb{E}_{\mu^*}[F(X)]$, note that, by Lemma 2, since $Y^* \in \mathcal{D}_2$, there exists a measure μ' that has a product form and whose marginals are Y^*. Since μ' has a product form, it satisfies (6), and $F(Y^*) = \mathbb{E}_{\mu'}[F(X)] \leq \mathbb{E}_{\mu^*}[F(X)]$. Finally, $\mathbb{E}_{\mu^*}[F(X)] \leq \mathbb{E}_{\mu^{**}}[F(X)]$, as the former is the expected cost under a restricted class of distributions μ, namely, ones that have the product form (13). □

Theorem 5 is specific to MaxCG: e.g., Ineq. (9) can be strict in other problems solvable through the pipage rounding method. The theorem has some non-obvious, interesting implications. First, equivalence of (24) to (7) implies that satisfying capacity constraints in expectation, rather than exactly, *does not* improve the caching gain. Similarly, the equivalence of (24) to (25) implies that considering only distributions that describe *independent* caches does not restrict the caching gain attainable: independent caches are as powerful as fully randomized (or deterministic) caches. Finally, as MaxCG is NP-hard, so are all three other problems.

7. NUMERICAL EVALUATION

We simulate Algorithms 1 and 3 over synthetic and real networks, and compare their performance to path replication combined with `LRU`, `LFU`, `FIFO`, and random replacement (`RR`) caches. Across the board, greedy path replication performs exceptionally well, attaining at least 95% of the expected caching gain attained by Algorithm 1, while both significantly outperform traditional eviction policies.

Topologies. The networks we consider are summarized in Table 2. The first six graphs are deterministic. Graph `cycle` is a simple cyclic graph, and `lollipop` is a clique (i.e., complete graph), connected to a path graph of equal size. Graph `grid_2d` is a two-dimensional square grid, `balanced_tree` is a complete binary tree of depth 6, and `hypercube` is a 7-dimensional hypercube. Graph `expander` is a Margulies-Gabber-Galil expander [17]. The next 5 graphs are random, i.e., were sampled from a probability distribution. Graph `erdos_renyi` is an Erdős-Rényi graph with parameter $p = 0.1$, and `regular` is a 3-regular graph sampled uniformly at random (u.a.r.). The `watts_strogatz` graph is a graph generated according to the Watts-Strogatz model of a small-world network [37] comprising a cycle and 4 randomly selected edges, while `small_world` is the navigable small-world graph by Kleinberg [23], comprising a grid with additional long range edges. The preferential attachment model of Barabási and Albert [4], which yields powerlaw degrees, is used for `barabasi_albert`. Finally, the last 3 graphs represent the Deutche Telekom, Abilene, and GEANT backbone networks [33].

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert Q\rvert$	c'_v	$F(Y^*)$
`cycle`	30	60	10	100	10	2	847.94
`lollipop`	30	240	10	100	10	2	735.78
`grid_2d`	100	360	300	1K	20	3	381.09
`balanced_tree`	127	252	300	1K	20	3	487.39
`hypercube`	128	896	300	1K	20	3	186.29
`expander`	100	716	300	1K	20	3	156.16
`erdos_renyi`	100	1042	300	1K	20	3	120.70
`regular`	100	300	300	1K	20	3	321.63
`watts_strogatz`	100	400	300	1K	20	3	322.49
`small_world`	100	491	300	1K	20	3	218.19
`barabasi_albert`	100	768	300	1K	20	3	113.52
`geant`	22	66	10	100	10	2	203.76
`abilene`	9	26	10	100	10	2	121.25
`dtelekom`	68	546	300	1K	20	3	94.79

Table 2: Graph Topologies and Experiment Parameters.

Experiment Setup. We evaluate the performance of different adaptive strategies over the graphs in Table 2. Given a graph $G(V, E)$, we generate a catalog $\mathcal{C}$, and assign a cache to each node in the graph. For every item $i \in \mathcal{C}$, we designate a node selected u.a.r. from V as a source for this item. We set the capacity c_v of every node v so that $c'_v = c_v - |\{i : v \in S_i\}|$ is constant among all nodes in V. We assign a weight to each edge in E selected u.a.r. from the interval $[1, 100]$. We then generate a set of requests $\mathcal{R}$ as follows. First, to ensure path overlaps, we select $|Q|$ nodes in V u.a.r., that are the only nodes that generate requests; let Q be the set of such query nodes. We generate a set of requests starting from a random node in Q, with the item requested selected from $\mathcal{C}$ according to a Zipf distribution with parameter 1.2. The request is then routed over the shortest path between the node in Q and the designated source for the requested item. We assign a rate $\lambda_{(i,p)} = 1$ to every request $(i, p) \in \mathcal{R}$.

The values of $|\mathcal{C}|$, $|\mathcal{R}|$ $|Q|$, and c_v for each experiment are given in Table 2. For each experiment, we also provide in the last column the quantity $F(Y^*)$, for $Y^* \in \arg\max_{Y \in \mathcal{D}_2} L(Y)$, i.e., the expected caching gain under a product form distribution that maximizes the relaxation L. By Thm. 1, this is within $1 - 1/e$ from the optimal expected caching gain.

Caching Algorithms and Measurements. We evaluate the performance of Algorithms 1 and 3, denoted by `PGA` (for Projected Gradient Ascent) and `GRD` (for Greedy), respectively. In the case of `PGA`, we tried different measurement periods $T = 1.0, 10.0, 20.0$, termed `PGA1`, `PGA10`, and `PGA20`, respectively. We implemented the algorithm both with state smoothening (16) and without (whereby allocations are sampled from marginals $Y^{(k)}$ directly). For brevity, we report

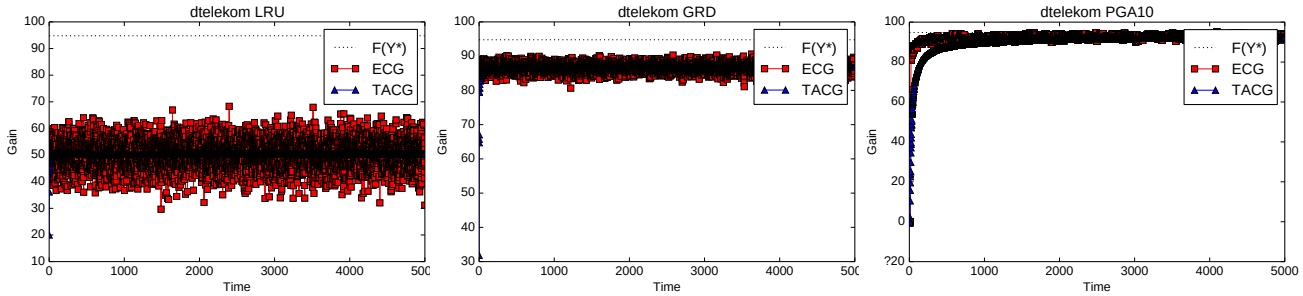

Figure 3: Trajectories of the expected caching gain ECG and the time average caching gain TACG under the LRU, GRD, and PGA algorithms, the latter with $T = 10$. The value $F(Y^*)$ for this experiment is shown in a dashed line.

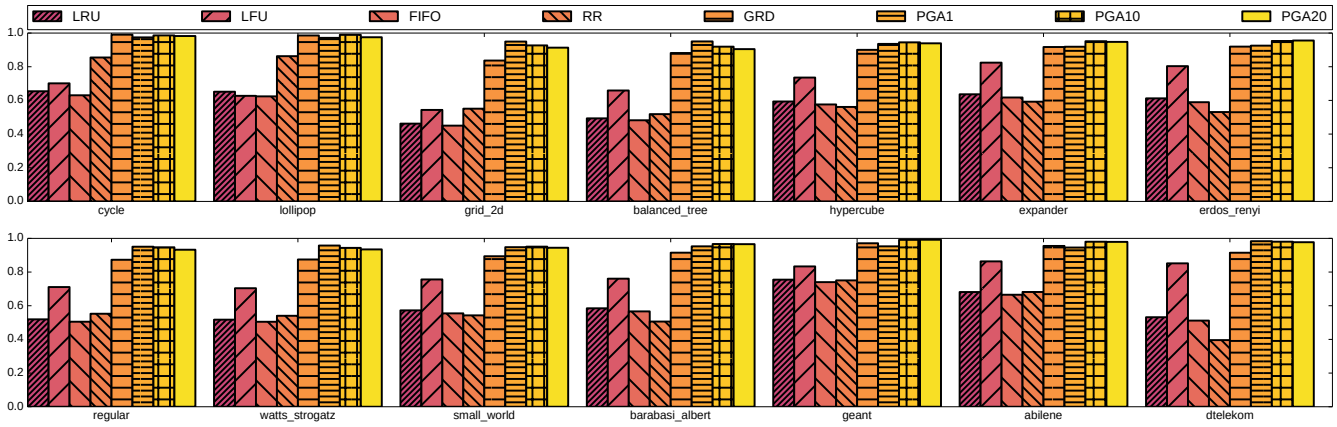

Figure 4: Ratio of expected caching gain ECG to $F(Y^*)$, as given in Table 2 under different networks and caching strategies. The greedy algorithm GRD performs almost as well as PGA in all cases. Both algorithms significantly outperform the remaining eviction policies.

only the non-smoothened versions, as time-average performance was nearly identical for both versions.

We also compare to path replication with LRU, LFU, FIFO, and RR eviction policies. In all cases, we simulate the network for 5000 time units. We collect measurements at epochs of a Poisson process with rate 1.0 (to leverage the PASTA property). In particular, at each measurement epoch, we extract the current allocation X, and compute the expected caching gain (ECG) as $F(X)$. In addition, we keep track of the actual cost of each request routed through the network, and compute the time average caching gain (TACG), measured as the difference of the cost of routing till the item source, minus the time average cost per request.

Results. Figure 3 shows the trajectories of the expected caching gain ECG and the time average caching gain TACG under the LRU, GRD, and PGA10 algorithms. All three algorithms converge relatively quickly to a steady state. PGA10 converges the slowest but it indeed reaches $F(Y^*)$, as expected. In addition, the greedy heuristic GRD performs exceptionally well, converging very quickly (faster than PGA) to a value close to $F(Y^*)$. In contrast, the allocations reached in steady state by LRU are highly suboptimal, close to 50% of $F(Y^*)$. Moreover, LRU exhibits high variability, spending considerable time in states with as low as 35% of $F(Y^*)$. We note that the relatively low variability of both GRD and PGA is a desirable feature in practice, as relatively stable caches are preferred by network administrators.

The above observations hold across network topologies and caching algorithms. In Figure 4, we plot the relative performance w.r.t ECG of all eight algorithms, normalized to $F(Y^*)$. We compute this as follows: to remove the effect of initial conditions, we focus on the interval $[1000, 5000]$, and average the ECG values in this interval.

We see that, in all cases, PGA attains $F(Y^*)$ for all three values of the measurement period T. Moreover, the simple heuristic GRD has excellent performance: across the board, it attains more than 95% of $F(Y^*)$, sometimes even outperforming PGA. Both algorithms consistently outperform all other eviction policies. We observe that RR and LFU perform quite well in several cases, and that "hard" instances for one appear to be "easy" for the other.

The differentiating instances, where performance is reversed, are the cycle and lollipop graphs: though small, these graphs contain long paths, in contrast to the remaining graphs that have a relatively low diameter. Intuitively, the long-path setting is precisely the scenario where local/myopic strategies like LRU, LFU, and FIFO make suboptimal decisions, while RR's randomization helps.

8. CONCLUSIONS

The main intuition from our analysis is that caching decisions *must account for upstream costs* to attain optimality guarantees. Establishing guarantees for Algorithm 3 remains an open question, of great interest especially in light of its performance in simulations. We believe that its relationship to the "continuous greedy" [36] and Frank-Wolfe [11] algorithms, as well as the algorithm in [20], can serve as a basis to characterizing its convergence. Determining the limit points of the fluid dynamics in (23) is key in this task.

Jointly optimizing caching *and* routing decisions is another open problem, even more pertinent in the presence of congestion, as in [38]. Ideally, one would reason about congestion within the context of MaxCG to produce joint routing and caching algorithms with delay *and* throughput optimality guarantees. Finally, Shanmugam et al. [34] show that caching fountain-coded content has interesting connections to the relaxation L; studying adaptive caching schemes in this setting also an important open question.

9. ACKNOWLEDGMENTS

E. Yeh gratefully acknowledges support from National Science Foundation grant CNS-1423250 and a Cisco Systems research grant.

10. REFERENCES

[1] A. A. Ageev and M. I. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[2] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan. Optimal content placement for a large-scale VoD system. In *CoNext*, 2010.

[3] I. Baev, R. Rajaraman, and C. Swamy. Approximation algorithms for data placement problems. *SIAM Journal on Computing*, 38(4):1411–1429, 2008.

[4] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.

[6] D. S. Berger, P. Gland, S. Singla, and F. Ciucu. Exact analysis of TTL cache networks. *IFIP Performance*, 2014.

[7] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM*, 2010.

[8] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *Integer programming and combinatorial optimization*, pages 182–196. Springer, 2007.

[9] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling data transfer in content-centric networking. In *Proceedings of the 23rd international teletraffic congress*, pages 111–118. International Teletraffic Congress, 2011.

[10] H. Che, Y. Tung, and Z. Wang. Hierarchical web caching systems: Modeling, design and experimental results. *Selected Areas in Communications*, 20(7):1305–1314, 2002.

[11] K. L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms*, 6(4):63, 2010.

[12] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM*, 2002.

[13] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman. On the complexity of optimal routing and content caching in heterogeneous networks. In *INFOCOM*, 2014.

[14] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *SODA*, 2006.

[15] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley. Analysis of TTL-based cache networks. In *VALUETOOLS*, 2012.

[16] C. Fricker, P. Robert, and J. Roberts. A versatile and accurate approximation for LRU cache performance. In *ITC*, 2012.

[17] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[18] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy. Performance evaluation of the random replacement policy for networks of caches. *Performance Evaluation*, 40(1):395–396, 2012.

[19] S. Ioannidis and P. Marbach. Absence of evidence as evidence of absence: A simple mechanism for scalable p2p search. In *INFOCOM*, 2009.

[20] S. Ioannidis, L. Massoulié, and A. Chaintreau. Distributed caching over heterogeneous mobile networks. In *SIGMETRICS*, 2010.

[21] S. Ioannidis and E. Yeh. Adaptive caching networks with optimality guarantees, 2016. arXiv:1604.03175.

[22] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT*, 2009.

[23] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *STOC*, 2000.

[24] H. J. Kushner and G. Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

[25] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *ICPCC*, 2004.

[26] M. Leconte, M. Lelarge, and L. Massoulié. Bipartite graph structures for efficient balancing of heterogeneous loads. In *SIGMETRICS*, 2012.

[27] M. Leconte, M. Lelarge, and L. Massoulie. Designing adaptive replication schemes in distributed content delivery networks. In *ITC*, 2015.

[28] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS*, 2002.

[29] V. Martina, M. Garetto, and E. Leonardi. A unified approach to the performance analysis of caching systems. In *INFOCOM*, 2014.

[30] A. Nemirovski. *Efficient methods in convex programming*. 2005.

[31] R. T. Rockafellar. *Convex analysis*. Princeton University Press, 1970.

[32] E. J. Rosensweig, D. S. Menasche, and J. Kurose. On the steady-state of cache networks. In *INFOCOM*, 2013.

[33] D. Rossi and G. Rossini. Caching performance of content centric networks under multi-path routing (and more). Technical report, Telecom ParisTech, 2011.

[34] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire. Femtocaching: Wireless content delivery through distributed caching helpers. *Transactions on Information Theory*, 59(12):8402–8413, 2013.

[35] A. L. Stolyar. Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm. *Queueing Systems*, 50(4):401–457, 2005.

[36] J. Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, 2008.

[37] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[38] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong. Vip: A framework for joint dynamic forwarding and caching in named data networks. In *ICN*, 2014.

[39] Y. Zhou, Z. Chen, and K. Li. Second-level buffer cache management. *Parallel and Distributed Systems*, 15(6):505–519, 2004.