Adaptive Caching Networks with Optimality Guarantees

Stratis Ioannidis, *Member, IEEE,* and Edmund Yeh, *Senior Member, IEEE*

***Abstract*—We study the optimal placement of content 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 approximation algorithm for placing contents into caches 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, and 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 a $1-1/e$ 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.**

***Index Terms*—Caching, ICN, CCN, pipage rounding, distributed optimization**

I. 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) [2]–[4], core and edge content delivery networks (CDNs) [5], [6], micro/femtocell networks [7], and peer-to-peer networks [8], [9], 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, which 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, we seek algorithms that are adaptive, i.e., (a) discover an optimal item placement without prior knowledge of this demand and (b) adapt to its changes. 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 [8] is a simple, elegant caching algorithm often encountered in the literature of the above networking applications [2], [4], [9]–[12]. 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. Path replication is intuitively appealing in its simplicity, and it clearly attains both of the above desirable properties: it is 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 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 [7], [13].
- We prove that the ubiquitous path replication algorithm, combined with LRU, LFU, or FIFO eviction policies, leads to allocations that are *arbitrarily suboptimal.* Our

This is an extended version of a paper that appeared in the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2016) [1].

Stratis Ioannidis and Edmund Yeh are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, 02115 USA e-mail: ioannidis@ece.neu.edu, eyeh@ece.neu.edu

Manuscript received November 25th, 2016; revised Aug. 21st, 2017.

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 prove that the problem of deterministic caching gain maximization is equivalent to several probabilistic variants. This equivalence has surprising implications. For example, independent caches are as powerful as caches whose contents are coupled, as they attain the same maximal expected caching gain. Moreover, there is no advantage in satisfying relaxed capacity constraints only in expectation, when caches are independent.
- 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.

The remainder of this paper is structured as follows. We review related work in Section II. We formally introduce our problem in Section III. Section IV contains our proof of the suboptimality of path replication with traditional eviction policies. We present an offline, centralized algorithm in Section V, while our distributed, adaptive algorithm with optimality guarantees and our proposed greedy heuristic appear in Sections VI and VII, respectively. The equivalence of deterministic and probabilistic variants of the expected caching gain maximization problem is in Section VIII. Section IX contains our evaluations, and we conclude in Section X.

II. Related Work

Path replication is best known as the de facto caching mechanism in content-centric networking [2], but has a long history in networking literature. In their seminal paper, Cohen and Shenker [8] 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 [14]; our work (c.f. Thm. 1) 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., [4], [10], [11]). In the context of CDNs and ICNs, Rosensweig et al. [3] study conditions under which path replication with LRU, FIFO, and other variants, under fixed paths, lead to an ergodic chain. Che et al. [12] 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 [15]–[18]; alternative analytical models are explored in [19], [20]. None of the above works however study optimality issues or guarantees.

Several papers have studied complexity and optimization issues in offline caching problems [5], [7], [21]–[24]. With the exception of [7], these works model the network as a bipartite graph: nodes generating requests connect directly to caches, and demands are satisfied in a single hop. Beyond content placement, Deghan et al. [6] jointly optimize caching and routing in this bipartite setting. In general, the *pipage rounding* technique of Ageev and Sviridenko [13] (see also [25], [26]) yields again a constant approximation algorithm in the bipartite setting, while approximation algorithms are also known for several variants of this problem [5], [21]–[23].

Among these papers on offline caching, the recent paper by Shanmugam et al. [7] is closest to the problem we tackle here; we rely upon and expand 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 V.

All of the above complexity papers [13], [21]–[23], including [7], 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 solution across nodes; combined, these lead to our distributed, adaptive caching algorithm with provable guarantees (Thm. 3).

Adaptive replication schemes exist for asymptotically large, single-hop CDNs [27]–[29], but these works do not explicitly model a graph structure. The dynamics of the greedy path replication algorithm we propose in Section VII resemble the greedy algorithm used to make caching decisions in [29], 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 [26] and the Frank-Wolfe algorithm [30]; 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 [31] and, more

recently, Yeh et al. [32], 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.

III. 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.

A. 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 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. We denote by $x_{vi} \in \{0, 1\}$ the variable indicating whether $v \in V$ stores item $i \in C$. We denote by $X = [x_{vi}]_{v\in V, i\in C} \in \{0, 1\}^{|V|\times|C|}$ 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 $\sum_{i\in C} x_{vi} = c_v$, for all $v \in V$. We associate each item i in the catalog C with a fixed set of *designated sources* $\mathcal{S}_i \subseteq V$, that always store i. That is, $x_{vi} = 1$, for all $v \in \mathcal{S}_i$. Without loss of generality, we assume that $\mathcal{S}_i$ are feasible, i.e., $\sum_{i:v\in\mathcal{S}_i} x_{vi} \leq c_v$, for all $v \in V$.

B. Content Requests and Routing Costs

The network serves content requests routed over 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 $(p_k, p_{k+1}) \in E$, for every $k \in \{1, \ldots, |p|-1\}$. Given a path p and a $v \in p$, denote by $k_p(v)$ the position of v in p; i.e., $k_p(v)$ equals the $k \in \{1, \ldots, |p|\}$ such that $p_k = v$.

Under this notation, a *request* r is a pair (i, p) where $i \in C$ 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 in $\mathcal{R}$ 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* $w_{uv} \geq 0$ with each edge $(u, v) \in E$, 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 $w_{uv} = w_{vu}$.

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

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

Intuitively, (1) 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.

C. 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.,

$$C_0 = \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k}. \tag{2}$$

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:

$$\begin{aligned} F(X) &\equiv C_0 - \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} C_{(i,p)}(X) \\ &= \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \Big(1 - \prod_{k'=1}^{k} (1 - x_{p_{k'}i})\Big). \end{aligned} \tag{3}$$

In particular, we seek solutions to the following problem:

MaxCG

$$\text{Maximize:} \quad F(X) \tag{4a}$$
$$\text{subj. to:} \quad X \in \mathcal{D}_1 \tag{4b}$$

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

$$\sum_{i\in C} x_{vi} = c_v, \qquad \text{for all } v \in V, \tag{5a}$$
$$x_{vi} \in \{0, 1\}, \qquad \text{for all } v \in V, i \in C, \text{ and} \tag{5b}$$
$$x_{vi} = 1, \qquad \text{for all } v \in \mathcal{S}_i \text{ and all } i \in C. \tag{5c}$$

Problem MaxCG is NP-hard (see [7] for a reduction from the 2-Disjoint Set Cover Problem). Our objective is to solve

TABLE I
NOTATION SUMMARY

$G(V,E)$	Network graph, with nodes V and edges E
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 $i \in C$ and p a path
$k_p(v)$	Position of node $v \in p$ in path p
$\lambda_{(i,p)}$	Rate of request $(i,p) \in \mathcal{R}$
x_{vi}	Variable indicating $v \in V$ stores $i \in C$
y_{vi}	Marginal probability that $v \in V$ stores $i \in C$
X	$\lvert V\rvert \times \lvert C\rvert$ matrix of x_{vi}, for $v \in V$, $i \in C$
Y	$\lvert V\rvert \times \lvert C\rvert$ matrix of marginals y_{vi}, $v \in V$, $i \in C$
$\mathcal{D}_1$	Set of feasible allocations $X \in \{0,1\}^{\lvert V\rvert \times \lvert C\rvert}$
$\mathcal{D}_2$	Convex hull of $\mathcal{D}_1$
F	The expected caching gain (3) in $\mathcal{D}_1$, and its multi-linear relaxation (7) in $\mathcal{D}_2$
L	The concave approximation (11) of F

MAXCG using a *distributed*, *adaptive* algorithm, that produces an allocation within a constant approximation of the optimal, *without* prior knowledge of the network topology, edge weights, or the demand.

IV. PATH REPLICATION SUBOPTIMALITY

Before providing algorithms with guarantees for MAXCG, we begin with a negative result: the simple path replication algorithm described in the introduction, combined with LRU, LFU, FIFO or RR (random replacement) evictions, is *arbitrarily suboptimal*. More specifically, the ratio between the expected caching gain under the optimal policy and that under path replication combined with these eviction policies can be made arbitrarily large.

We prove this using the simple star network illustrated in Figure 1, in which only one file can be cached at the central node v. When serving requests from the bottom node u, 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, FIFO, and RR 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 1. *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, FIFO, or RR policy. Then, for $\alpha = 1/\sqrt{M}$,*

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

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 v 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 $F(X^*) \geq \alpha M + 2 - \alpha - (2-\alpha) = \alpha M$.

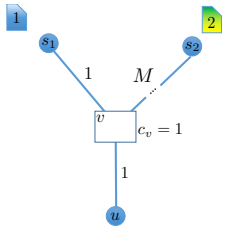


Fig. 1. A simple caching network, with $C = \{1,2\}$, $S_1 = \{s_1\}$, $S_2 = \{s_2\}$. The cache at v has capacity $c_v = 1$, and the cost of the edge between v and s_2 is $M \gg 1$. Node u requests item 1 with rate $1-\alpha$, and item 2 with rate α. For $\alpha = 1/\sqrt{M}$, path replication with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

Consider now the path replication algorithm, in which an item is cached at v whenever it is back-propagated over the reverse path. As $c_v = 1$, all four 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, FIFO or RR evictions to $F(X^*)$ is at most $\alpha + \frac{(1-\alpha)^2}{\alpha M}$; the theorem follows for $\alpha = 1/\sqrt{M}$. □

Taking M to be arbitrarily large makes this ratio arbitrarily small; thus, path replication with these eviction policies is arbitrarily suboptimal. Clearly, the argument in the proof applies to any eviction strategy that coincides with LRU when $c_v = 1$. More broadly speaking, the counterexample should apply to *any eviction strategy that is is insensitive to upstream costs*. Accounting for such upstream costs when caching seems necessary for providing any optimality guarantees; all algorithms we propose below indeed do so.

V. A CENTRALIZED, OFFLINE ALGORITHM

Before presenting our distributed, adaptive algorithm for solving MAXCG, we first discuss how to obtain a polynomial-time approximation 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. [7], improves upon this ratio using a technique called *pipage rounding* [13].

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 ratio 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

[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.

applied to MAXCG below, referring the interested reader to [7], [13] for further details.

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

$$y_{vi} = \mathbf{P}_\mu[x_{vi} = 1] = \mathbb{E}_\mu[x_{vi}]. \tag{6}$$

Denote by $Y = [y_{vi}]_{v\in V, i\in C} \in \mathbb{R}^{|V|\times|C|}$ the matrix comprising the marginal probabilities (6). Then, for F given by (3):

$$\begin{aligned}\mathbb{E}_\mu[F(X)] &= \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \Big(1 - \mathbb{E}_\mu\big[\prod_{k'=1}^{k}(1-x_{p_{k'}i})\big]\Big) \\ &= \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \Big(1 - \prod_{k'=1}^{k}(1-\mathbb{E}_\mu[x_{p_{k'}i}])\Big) \\ &= F(Y). \end{aligned} \tag{7}$$

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 $[0,1]^{|V|\times|C|}$ is known as the *multi-linear relaxation* of F [7], [13], [25], [26]. Consider the problem:

$$\text{Maximize:} \quad F(Y) \tag{8a}$$
$$\text{subject to:} \quad Y \in \mathcal{D}_2, \tag{8b}$$

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

$$\textstyle\sum_{i\in C} y_{vi} = c_v, \qquad \text{for all } v \in V, \tag{9a}$$
$$y_{vi} \in [0,1], \qquad \text{for all } v \in V, i \in C, \text{ and} \tag{9b}$$
$$y_{vi} = 1, \qquad \text{for all } v \in \mathcal{S}_i \text{ and all } i \in C. \tag{9c}$$

Note that an allocation X sampled from a μ 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 (4) and (8), respectively, then

$$F(Y^*) \geq F(X^*), \tag{10}$$

as (8) maximizes the same function over a larger domain.

The multi-linear relaxation (8a) is not concave, so (8) is *not* a convex optimization problem. Nonetheless, (8) can be approximated as follows. Define $L : \mathcal{D}_2 \to \mathbb{R}$ as:

$$L(Y) = \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \min\{1, \sum_{k'=1}^{k} y_{p_{k'}i}\}. \tag{11}$$

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

$$\text{Maximize:} \quad L(Y) \tag{12a}$$
$$\text{Subject to:} \quad Y \in \mathcal{D}_2. \tag{12b}$$

Then, the optimal value of (12) is guaranteed to be within a constant factor from the optimal value of (8)–and, by (10), from the optimal value of (4) as well. In particular:

Theorem 2. *[7], [13] Let Y^*, and Y^{**} be optimal solutions to* (8) *and* (12)*, respectively. Then,*

$$F(Y^*) \geq F(Y^{**}) \geq (1 - \frac{1}{e})F(Y^*). \tag{13}$$

Proof. To begin with, for all $Y \in \mathcal{D}_2$, we have:

$$(1 - \frac{1}{e})L(Y) \leq F(Y) \leq L(Y). \tag{14}$$

To see this, note that

$$\begin{aligned} F(Y) &= \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{E}_\mu[\min\{1, \sum_{k'=1}^{k} x_{p_{k'}i}\}] \\ &\leq \sum_{(i,p)\in\mathcal{R}} \lambda_{(i,p)} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \min\{1, \sum_{k'=1}^{k} \mathbb{E}_\mu[x_{p_{k'}i}]\} \end{aligned}$$

by the concavity of the min operator, so $F(Y) \leq L(Y)$. On the other hand, by Goemans and Williamson [33],

$$1 - \textstyle\prod_{k'=1}^{k}(1 - y_{p_{k'}i}) \geq (1 - (1-1/k)^k) \min\{1, \sum_{k'=1}^{k} y_{p_{k'}i}\},$$

and the first inequality of the statement of the lemma follows as $(1-1/k)^k \leq \frac{1}{e}$. By the optimality of Y^* in $\mathcal{D}_2$, clearly $F(Y^{**}) \leq F(Y^*)$. By (14) and the optimality of Y^{**}, $F(Y^*) \leq L(Y^*) \leq L(Y^{**}) \leq \frac{e}{e-1}F(Y^{**})$. □

Problem (12) is convex; in fact, by introducing auxiliary variables, it can be converted to a linear program and, as such, Y^{**} can be computed in strongly polynomial time (see [13]).

Rounding. To produce an integral solution to MAXCG, the solution Y^{**} of (12) 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 convert it to a $Y' \in \mathcal{D}_2$ with at least one fewer fractional entry than Y, for which $F(Y') \geq F(Y)$. To see this, suppose that Y contains a fractional entry, say $y_{vi} \in (0,1)$, for some $v \in V$, $i \in C$. Then, as capacity constraints are integral, there must exist another entry $y_{vi'} \in (0,1)$–i.e., fractional entries come in pairs. Observe now that F, restricted to *only these two entries*, is a convex function. As such, it is maximized at the extrema of the set of values that the pair $(y_{vi}, y_{vi'})$ may take, presuming all other entries are constant. This implies that we can construct Y' by transferring equal mass between y_{vi} and $y_{vi'}$, increasing one and decreasing the other, so that at least one of them becomes 0 or 1. Pairwise convexity ensures that among the two possible mass transfers one yields a Y' s.t. $F(Y') \geq F(Y)$, while transferring equal mass ensures that $Y' \in \mathcal{D}_2$. Thus, one can construct an integral solution as follows:

1) Start from Y^{**}, an optimal solution to the problem (12).
2) If the solution is fractional, find two variables y_{vi}, $y_{vi'}$ that are fractional.
3) Use the rounding described above 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 $|V| \times |C|$ steps, producing an integral solution $X' \in \mathcal{D}_1$. Since each rounding step can only increase F, X' satisfies:

Algorithm 1 PROJECTED GRADIENT ASCENT

Execute the following at each $v \in V$:
Pick arbitrary state $y_v^{(0)} \in \mathcal{D}_2^v$.
for each period $k \geq 1$ and each $v \in V$ **do**
  Compute the sliding average $\bar{y}_v^{(k)}$ through (17).
  Sample a $x_v^{(k)} \in \mathcal{D}_1^v$ from a μ_v that satisfies (19).
  Place items $x_v^{(k)}$ in cache.
  Collect measurements
  At the end of the period, compute estimate z_v of $\partial_{y_v} L(Y^{(k)})$ through (20).
  Compute new state $y_v^{(k+1)}$ through (16).
end for

$F(X') \geq F(Y^{**}) \overset{(13)}{\geq} (1-\frac{1}{e})F(Y^*) \overset{(10)}{\geq} (1-\frac{1}{e})F(X^*)$, i.e., is a $(1-\frac{1}{e})$-approximate solution to MAXCG.

VI. A DISTRIBUTED, ADAPTIVE ALGORITHM

We now describe our distributed, adaptive algorithm for solving MAXCG. The algorithm performs a *projected gradient ascent* over function L, effectively solving the convex problem (12) in a distributed, adaptive fashion. The concavity of L ensures convergence (in contrast to maximizing non-concave F directly), while Theorem 2 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 information locally available, which has been obtained via messages passing through the node; 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, both the final and the intermediate solutions of convex relaxation (12) during gradient ascent produce fractional values $Y \in \mathcal{D}_2$. To place contents in each cache, discrete allocations $X \in \mathcal{D}_1$ need to be constructed from Y. Our algorithm *cannot* rely on pipage rounding to construct X: pipage rounding is inherently serial, as it is applied to pairs of fractional variables sequentially. It also presumes that, at each rounding step, all other variables are both fixed and globally known. As such, pipage rounding cannot be used to produce a distributed, adaptive algorithm. We address both challenges, proving that both (a) a feasible randomized rounding of any $Y \in \mathcal{D}_2$, and (b) the subgradient of L w.r.t. Y can be computed in a distributed, adaptive fashion.

A. Algorithm Overview and Optimality Guarantee

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* $y_v \in [0,1]^{|C|}$: intuitively, as in (6), each y_{vi} captures the probability that node $v \in V$ stores item $i \in C$. 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.

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

$$\mathbb{E}\left[z_v(Y^{(k)})\right] \in \partial_{y_v} L(Y^{(k)}) \tag{15}$$

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 VI-B. Having these estimates, each node adapts its state as follows: at the conclusion of the k-th period, the new state is computed as

$$y_v^{(k+1)} \leftarrow \mathcal{P}_{\mathcal{D}_2^v}\left(y_v^{(k)} + \gamma_k \cdot z_v(Y^{(k)})\right), \tag{16}$$

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:

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

State Smoothening. Upon performing the state adaptation (16), each node $v \in V$ computes the following "sliding average" of its current and past states:

$$\bar{y}_v^{(k)} = \textstyle\sum_{\ell=\lfloor \frac{k}{2} \rfloor}^{k} \gamma_\ell y_v^{(\ell)} \Big/ \left[\sum_{\ell=\lfloor \frac{k}{2} \rfloor}^{k} \gamma_\ell\right]. \tag{17}$$

This "state smoothening" is necessary precisely because of the non-differentiability of L [34]. 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 $\bar{y}_v^{(k)}$, 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 $x_v^{(k)} \in \{0,1\}^{|C|}$ sampled independently of any other node in V. Put differently, at any point in time, the (global) allocation $X \in \mathcal{D}_1$ is sampled from a joint distribution μ over $\mathcal{D}_1$ that has a *product form*; for every v, we construct appropriate probability distributions μ_v, $v \in V$, such that:

$$\mu(X) = \textstyle\prod_{v\in V} \mu_v(x_{v1}, \ldots, x_{v|C|}). \tag{18}$$

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 a subset of:

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

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

$$\mathbb{E}_{\mu_v}[x_{vi}^{(k)}] = \mathbf{P}_{\mu_v}[x_{vi}^{(k)} = 1] = \bar{y}_{vi}^{(k)}, \text{ for } i \in C. \tag{19}$$

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

The complete projected gradient ascent algorithm is summarized in Algorithm 1. Crucially, the resulting steady-state allocations produced by projected gradient ascent are guaranteed to be within a constant approximation of the optimal, in expectation:

Theorem 3. *Let $X^{(k)} \in \mathcal{D}_1$ be the allocation at the k-th period of Algorithm 1. Then, if $\gamma_k = \Theta(1/\sqrt{k})$,*

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

We prove Theorem 3 below, in Section VI-D. Contrasting this result to Theorem 1, we see that Algorithm 1 (a) is distributed and adaptive, and (b) it attains, in steady state, an expected caching gain with the same ratio to the optimal offline allocation as pipage rounding. Note that 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—we revisit this issue in Section VII. Moreover, 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 (see also Fig. 5). 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 [35]).

Before proving Theorem 3, we present the two pieces missing from our exposition, namely, the subgradient estimate construction that satisfies (15) and the randomized rounding algorithm that satisfies (19).

B. Distributed Sub-Gradient Estimation

We now describe how to compute the estimates z_v of the subgradients $\partial_{y_v} L(Y^{(k)})$ 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.

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 $i \in C$ during a measurement period of duration T. At the end of the measurement period, each node $v \in V$ produces the following estimates:
$$z_{vi} = \textstyle\sum_{t\in\mathcal{T}_{vi}} t/T, \quad i \in C. \tag{20}$$
Note that, in practice, this needs to be computed only for $i \in C$ 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 $z_v = [z_{vi}]_{i\in C} \in \mathbb{R}_+^{|C|}$ the vector constructed through coordinates* (20)*,*

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

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

Proof. First, let:

$$\overline{\partial_{y_{vi}} L}(Y) = \sum_{(i,p)\in\mathcal{R}: v\in p} \lambda_{(i,p)} \sum_{k'=k_p(v)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} y_{p_\ell i} \leq 1}, \tag{21a}$$

$$\underline{\partial_{y_{vi}} L}(Y) = \sum_{(i,p)\in\mathcal{R}: v\in p} \lambda_{(i,p)} \sum_{k'=k_p(v)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} y_{p_\ell i} < 1}. \tag{21b}$$

A vector $z \in \mathbb{R}^{|C|}$ 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. □

C. 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 (19), note that it suffices to consider μ_v such that $\mathbb{E}_{\mu_v}[x_v] = \bar{y}_v$, defined over the set:

$$\bar{\mathcal{D}}_1^v = \{x_v \in \{0,1\}^{|C|} : \textstyle\sum_{i\in C} x_{vi} = c_v\}. \tag{22}$$

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, (19) 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}, $v \in V$, $i \in C$, are independent Bernoulli variables with parameters $\bar{y}_{vi}$ indeed satisfies (19), 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.

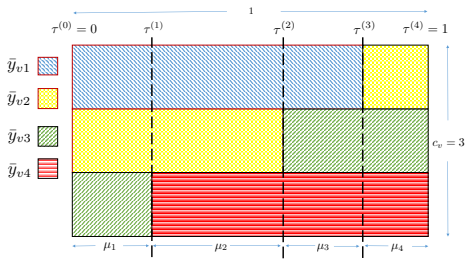


Fig. 2. An allocation that satisfies $\mathbb{E}_\mu[x_{vi}] = \bar{y}_{vi}$, when $\sum_{i\in C} \bar{y}_{vi} = c_v$. After placing the 4 rectangles in a 3×1 grid, assigning probabilities μ_1, μ_2, μ_3, μ_4 to each of the tuples $\{1,2,3\}$, $\{1,2,4\}$, $\{1,3,4\}$, $\{2,3,4\}$, respectively, yields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

Before we formally present our algorithm we first give some intuition behind it, also illustrated in Figure 2. Let $c_v = 3$, $C = \{1,2,3,4\}$, 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 $i \in C$, 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 $\sum_{i=1}^{|C|} \bar{y}_{vi} = c_v$, 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 $|C|-1$ such points. Then, partition the $c_v \times 1$ box vertically, splitting it at these cutting points. This results in at most $|C|$ 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 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 $\bar{y}_{vi} \in C$. 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 cutting points τ 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. (19) holds.*

A proof can be found in Appendix A. 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 $O(c_v|C|\log|C|)$: the sort in line 8 can be implemented in $O(|C|\log|C|)$ time, while a match in 13 can be found in $O(\log|C|)$ time if intervals are stored in a binary search tree, whose construction is also $O(|C|\log|C|)$. Moreover, the support of μ_v has size at most $|C|$, so representing this distribution requires $O(c_v|C|)$ space. Finally, as Algorithm 1 requires only one sample from μ_v, in practice, there is no need to construct the entire distribution: producing a single sample x_v reduces the complexity to $O(|C|\log|C| + c_v\log|C|)$.

Algorithm 2 RANDOMIZED ROUNDING ALGORITHM

1: **Input:** capacity c_v, marginals $\bar{y}_v \in \mathbb{R}^{|C|}$ s.t. $\bar{y}_v \geq \mathbf{0}$, $\sum_{i=1}^{|C|} \bar{y}_{vi} = c_v$
2: **Output:** prob. distr. μ_v over $\{x_v \in \{0,1\}^{|C|} : \sum_{i=1}^{|C|} x_{vi} = c_v\}$ s.t. $\mathbb{E}_\mu[x_{vi}] = \bar{y}_{vi}$, for all $i \in C$.
3: $\texttt{sum} \leftarrow 0$
4: **for all** $i \in C$ **do**
5: $\quad s_i \leftarrow \texttt{sum}$; $t_i \leftarrow \texttt{sum} + \bar{y}_{vi}$; $\tau_i \leftarrow t_i - \lfloor t_i \rfloor$
6: $\quad \texttt{sum} \leftarrow t_i$
7: **end for**
8: Sort all τ_i in increasing order, remove duplicates, and append 1 to the end of the sequence.
9: Let $0 = \tau^{(0)} < \tau^{(1)} < \ldots < \tau^{(K)} = 1$ be the resulting sequence.
10: **for all** $k \in \{0,\ldots,K-1\}$ **do**
11: $\quad$ Create new vector $x_v \in \{0,1\}^{|C|}$; set $x \leftarrow \mathbf{0}$.
12: $\quad$ **for all** $\ell \in \{0,\ldots,c_v-1\}$ **do**
13: $\quad\quad$ Find $i \in C$ such that $(\ell + \tau^{(k)}, \ell + \tau^{(k+1)}) \subset [s_i, t_i]$.
14: $\quad\quad$ Set $x_{vi} \leftarrow 1$
15: $\quad$ **end for**
16: $\quad$ Set $\mu_v(x_{vi}) = \tau^{(k+1)} - \tau^{(k)}$
17: **end for**
18: **return** μ_v

D. Proof of Theorem 3

To prove Theorem 3, we first establish the convergence of the smoothened marginals to a global maximizer of L:

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

$$\varepsilon_k \equiv \mathbb{E}[\max_{Y\in\mathcal{D}_2} L(Y) - L(\bar{Y}^{(k)})] \leq \frac{D^2 + M^2\sum_{\ell=\lfloor k/2\rfloor}^{k}\gamma_\ell^2}{2\sum_{\ell=\lfloor k/2\rfloor}^{k}\gamma_\ell}, \quad (23)$$

where $D \equiv \sqrt{2|V|\max_{v\in V} c_v}$, $M \equiv W|V|\Lambda\sqrt{|V||C|(1+\frac{1}{\Lambda T})}$. In particular, for $\gamma_k = \Theta(1/\sqrt{k})$, $\lim_{k\to\infty}\varepsilon_k = 0$.

Proof. Under dynamics (16) and (17), from Theorem 14.1.1, page 215 of Nemirofski [34], we have that $\varepsilon_k \leq \frac{d^2+m^2\sum_{\ell=\lfloor k/2\rfloor}^{k}\gamma_\ell^2}{2\sum_{\ell=\lfloor k/2\rfloor}^{k}\gamma_\ell}$, where $d = \max_{x,y\in\mathcal{D}_2}\|x-y\|_2 \leq D$, and

$$m = \sup_{Y\in\mathcal{D}_2}\sqrt{\textstyle\sum_{v\in V}\mathbb{E}[\|z_v(Y)\|_2^2]} \leq M,$$

where the last inequality follows from Lemma 1. □

Finally, Thm. 2 and Lemmas 2 and 3 imply that the asymptotic expected caching gain under Algorithm 1 is within a constant factor from the optimal, completing the proof of Theorem 3. In particular, given $\bar{Y}^{(k)}$, by Lemma 2, $X^{(k)}$ is sampled from a distribution μ over $\mathcal{D}_1$ that has product form (18). This product form implies that, conditioned on $\bar{Y}^{(k)}$, Eq. (7) 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 Lemma 3, $\lim_{k\to\infty}\mathbb{E}[L(\bar{Y}^{(k)})] = \max_{Y\in\mathcal{D}_2} L(Y)$. This implies $\lim_{k\to\infty}\mu^{(k)}(\mathcal{D}_2 \setminus \Omega) = 0$ for $\mu^{(k)}$ the distribution of $\bar{Y}^{(k)}$, and $\Omega \equiv \arg\max_{Y\in\mathcal{D}_2} L(Y)$ the set of $Y^{**} \in \mathcal{D}_2$ that are maximizers of L. From Theorem 2, $F(Y^{**}) \geq (1-1/e)\max_{X\in\mathcal{D}_1} F(X)$ for any $Y^{**} \in \Omega$. The theorem therefore follows from the above observations, and the fact that F is bounded in $\mathcal{D}_2 \setminus \Omega$. □

E. Rate of Convergence.

Lemma 3 can be use to characterize the rate of convergence of our proposed algorithm. In particular, when D and M are known, the rate of convergence implied by bound (23) is optimized by setting $\gamma_k = \frac{D}{M\sqrt{k}}$; under this gain, (23) becomes $\varepsilon_k \leq O(1)\frac{MD}{\sqrt{k}}$, where $O(1)$ is an absolute constant. When D and M are not known, setting $\gamma_k = 1/\sqrt{k}$ ensures convergence, albeit at a slower rate. Finally, the relationship between M and T establishes the tradeoff induced by timeslot duration T. Larger values of T reduce M, giving more accurate estimates of the subgradients in each timeslot and reducing the iterations k till convergence. On the other hand, increasing T also makes each timeslot longer; given D and M, (23) can be used to quantify this tradeoff.

F. Extensions

Our model immediately captures a scenario where paths p are random, and sampled independently for each request: if requests for item i are Poisson with aggregate rate λ_i, and the probability a path p is followed is $q_{(i,p)}$, the resulting system behaves exactly as our model, with $\lambda_{(i,p)} = \lambda_i q_{(i,p)}$.

Our algorithms and their analysis readily extend to the case where routing costs (a) are incurred by both request and response messages and (b) are random. In particular, to account for both query and response routing costs, the weight of an edge w_{uv} in (1) can be replaced by: $\alpha w_{uv} + \beta w_{vu}$ where α, β capture, e.g., the lengths of response and query messages, respectively. Moreover, (1) holds for random costs with weights replaced by expected values, as long as cost samples are independent random variables, also independent of the allocation X. Provided random costs have finite means and variances,[2] our analysis extends to this setting.

Our analysis also extends to contents of unequal size: such contents can be partitioned into equally sized "chunks", which would play the role of "items" in our model. Requesting a file amounts to requesting all of its chunks simultaneously. We can thus model this setting by considering "multi-item" requests, whereby $\mathcal{R}$ comprises multiple item/path tuples, e.g., of the form $\{(i_1, p), (i_2, p), \ldots, (i_L, p)\}$, arriving at Poisson epochs. Even though request arrivals per chunk/item are no longer independent, the expected caching gain objective F still has the form (3): each multi-item request contributes multiple summation terms of identical rates, one for each chunk in the requested set. This is true even when requests per chunk are piggy-backed over the same message, that is forwarded until all chunks are found, presuming that routing costs are proportional to the number chunks carried by a response. Our algorithms and analysis directly extend to this setting; note that this extension does not assume, or require, that chunks of the same file are stored in the same cache, though an optimal solution may indeed induce collocation.

VII. Greedy Path Replication

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

In this section, we propose a novel greedy eviction policy, to be used with the path replication algorithm, that has none of the above drawbacks. This algorithm is a heuristic, and we provide no formal guarantees on its performance w.r.t. the optimal. That said, the algorithm has the following advantages. First, it 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). Although we provide no guarantees for this algorithm, our experiments (see Sec. IX) indicate that it converges to an allocation attaining the same caching gain as Algorithm 1.

Algorithm 3 Greedy Path Replication

```
Execute the following at each v ∈ V:
Initialize z_v = 0.
while (true) do
    Wait for new response message.
    Upon receipt of new message, extract counter t_vi and i.
    Update z_v through (25).
    Sort z_vj, j ∈ C, in decreasing order.
    if x_vi = 0 and i is in top c'_v items then
        Set x_vi ← 1; evict c'_v + 1-th item.
    end if
end while
```

A. 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 $x_v \in \{0,1\}^{|C|}$. 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 caching the requested item $i \in C$, 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.

[2] So that the second moment in Lemma 1 remains bounded.

Hence, every node v in the path p that is visited by a response learns the quantity:

$$t_{vi} = \sum_{k'=k_v(p)}^{|p|-1} w_{p_{k'+1}p_{k'}} \mathbb{1}_{\sum_{\ell=1}^{k'} x_{p_\ell i}<1}, \quad (24)$$

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 $z_v \in \mathbb{R}_+^{|C|}$ 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 $j \in C$,

$$z_{vj}(t) = z_{vj}(t') \cdot e^{-\beta(t-t')} + \beta \cdot t_{vi} \cdot \mathbb{1}_{i=j}, \quad (25)$$

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 (25), 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.

B. 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.

Lemma 4. *Let $X(t) = [x_{vi}(t)]_{v\in V, i\in C}$ and $Z(t) = [z_{vi}(t)]_{v\in V, i\in C}$ be the allocation and subgradient estimation matrices at time $t \geq 0$. Their "fluid" trajectories are described by the ODE:*

$$X(t) \in \arg\max_{X\in\mathcal{D}_1} \langle X, Z(t)\rangle \quad (26a)$$

$$\frac{dZ(t)}{dt} = \beta(\underline{\partial L}(X(t)) - Z(t)) \quad (26b)$$

where $\langle A, B\rangle = \mathtt{trace}(AB^\top)$ is the inner product between matrices A, B, and $\underline{\partial L} \in \partial L$ is a subgradient of L at X.

Proof. By the "baby Bernoulli" approximation of a Poisson process (c.f. Chap. 2 p. 37 [36]), 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 (21b); note that the matrix $\underline{\partial L}$ is indeed a subgradient. The fluid dynamics (26) then follow by taking δ to go to zero, and replacing the ϕ_v's by their expectation. □

The dynamics (26) are similar (but not identical) to the "continuous greedy" algorithm for submodular maximization [26] and the Frank-Wolfe algorithm [30]. Eq. (26b) implies that $Z(t)$ indeed "tracks" a subgradient of L at X. On the other hand, the allocation selected by sorting—or, equivalently, by (26a)—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 (26a) 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 (26), provided they exist, must be at maximizers of L.

Lemma 5. *Let $X^* \in \mathcal{D}_2$ and $Z^* \in \mathbb{R}^{|V|\times|C|}$ 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 [37], 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 establish the optimality properties of this set, our numerical evaluations in Section IX show that this greedy heuristic attains a high caching gain in practice, within 95% from the one attained by the maximizer of L.

VIII. Offline Problem Equivalence

MaxCG is a deterministic, offline optimization problem. The algorithms we propose in Section VI produce *randomized* allocations. As randomized allocations are more expressive than deterministic allocations, it is natural to ask whether randomness helps: can we achieve a higher caching gain under randomized allocations than under deterministic ones? The answer is no; surprisingly, a broad array of randomized allocations turn out to be equivalent to deterministic strategies.

We have already seen one probabilistic relaxation of MaxCG, namely, the "independent Bernoulli" relaxation (8) we discussed in Section V. Consider now the variant:

$$\text{Max.:} \quad \mathbb{E}_\mu[F(X)] = \sum_{X\in\mathcal{D}_1} \mu(X)F(X) \quad (27a)$$

$$\text{subj. to:} \quad \mu \text{ is a pr. distr. over } \mathcal{D}_1 \text{ satisfying (18).} \quad (27b)$$

In this optimization, we seek *randomized* cache allocations sampled from a joint distribution μ over $\mathcal{D}_1$ having product form (18). In addition, consider the (more general) problem:

$$\text{Maximize:} \quad \mathbb{E}_\mu[F(X)] = \sum_{X\in\mathcal{D}_1} \mu(X)F(X) \quad (28a)$$

$$\text{subj. to:} \quad \mu \text{ is a pr. distr. over } \mathcal{D}_1. \quad (28b)$$

Our results–including, crucially, Lemma 2–have the following surprising implication: all three relaxations (8), (27), and (28) *are in fact equivalent to* MaxCG.

Theorem 4. *Let X^*, Y^*, μ^*, and μ^{**} be optimal solutions to* (4),(8)*,* (27)*, and* (28)*, 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 $D = \mathrm{supp}(\mu^{**}) \subseteq \mathcal{D}_1$ 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 and non-empty, this exists and is attained). 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 (10), as (8) is a relaxation of (4). 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 (7), 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 (18). □

Theorem 4 has several important implications. First, *deterministic allocations are as powerful as randomized allocations*. Second, the equivalence of (27) to (8) implies that *satisfying capacity constraints in expectation, rather than exactly, does not improve the caching gain.* Third, the equivalence of (27) to (28) implies that considering only distributions in which caches are independent entails no loss of generality: *independent caches are as powerful as fully randomized caches*. Finally, as MAXCG is NP-hard, so are all four problems.

IX. 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 II. 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 [38]. 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 [39] comprising a cycle and 4 randomly selected edges, while `small-world` is the graph by Kleinberg [40], comprising a grid with additional long range edges. The preferential attachment model of Barabási and Albert [41], which yields powerlaw degrees, is used for `barabasi-albert`. Finally, the last 3 graphs represent the Deutche Telekom, Abilene, and GEANT backbone networks [4].

TABLE II
GRAPH TOPOLOGIES AND EXPERIMENT PARAMETERS.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert C\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert Q\rvert$	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

Experiment Setup. We evaluate the performance of different adaptive strategies over the graphs in Table II. Given a graph $G(V, E)$, we generate a catalog C, and assign a cache to each node in the graph. For every item $i \in C$, 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 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 $|C|$, $|\mathcal{R}|$, $|Q|$, and c_v for each experiment are given in Table II. 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. 2, 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 (17) and without (whereby allocations are sampled from marginals $Y^{(k)}$ directly). For brevity, we report 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

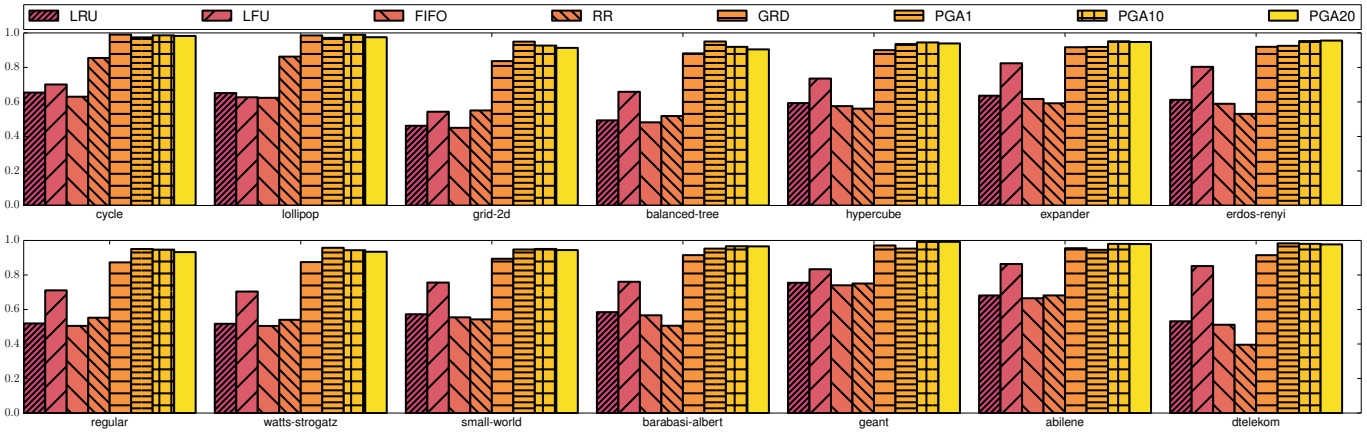


Fig. 3. Ratio of expected caching gain ECG to $F(Y^{**})$, as given in Table II under different networks and caching strategies. The greedy algorithm GRD performs almost as well as PGA in all cases. Both algorithms significantly outperform the remaining eviction policies.

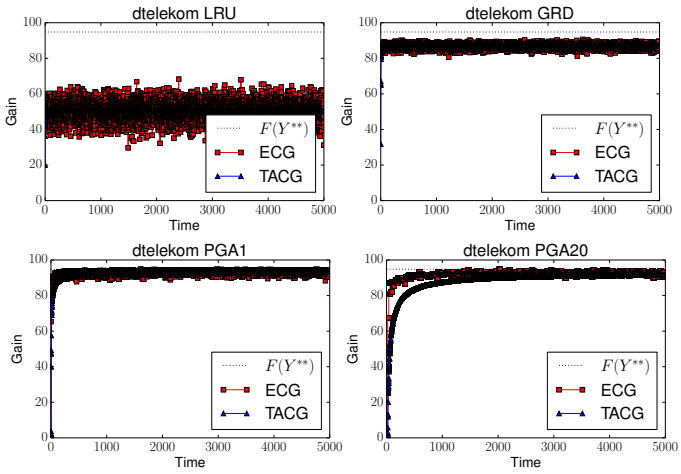


Fig. 4. Trajectories of the expected caching gain ECG and the time average caching gain TACG under the LRU, GRD, and PGA algorithms, the latter with $T = 1$ and $T = 20$. The value $F(Y^{**})$ is shown in a dashed line.

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 4 shows the trajectories of the expected caching gain ECG and the time average caching gain TACG under the LRU, GRD, and PGA1 and PGA20 algorithms. All algorithms converge relatively quickly to a steady state. PGA20 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 PGA1) 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 3, 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.

Finally, to illustrate adaptability, the ECG trajectory of our algorithms under non-stationary demands is shown in Fig. 5. In this experiment, new rates $\lambda_{(i,p)}$, $(i,p) \in \mathcal{R}$, are selected u.a.r. from the range $[0, 100]$ every T_{ch} time units. This leads to a sharp change in the optimal cache allocation. Experiments for PGA1 and GRD are shown. The gain is set to a constant $\gamma = 0.1$ to ensure that adaptation persists, and β is set to 0.1. Both algorithms closely track changes in demand, though GRD adapts much faster.

X. 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" [26] and Frank-Wolfe [30] algorithms, as well as the algorithm in [29], can serve as a basis to characterizing its convergence. Determining the limit points of the fluid dynamics in (26) is key in this task.

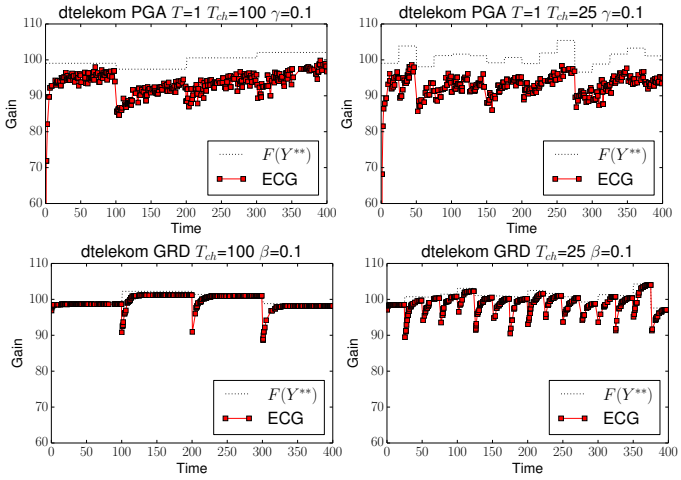


Fig. 5. ECG trajectories under changing demand distributions for PGA with fixed $\gamma = 0.1$ and GRD with $\beta = 0.1$. Demands change every T_{ch} time units, where $T_{ch} = 100$ (*left*) and $T_{ch} = 25$ (*right*). The value $F(Y^{**})$ as demands evolve is shown in a dashed line. Both track changes, but GRD adapts faster.

Jointly optimizing caching *and* routing decisions is another open problem, even more pertinent in the presence of congestion, as in [32]. 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. [7] 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.

APPENDIX A
PROOF OF LEMMA 2

We now prove the correctness of Algorithm 2, by showing that it produces a μ_v over $\bar{\mathcal{D}}_1^v$ with marginals $\bar{y}_v$. For all $i \in C$, denote by $s_i = \sum_{k=0}^{i-1} \bar{y}_{vk}$, $t_i = s_i + \bar{y}_{vi}$, $\tau_i = t_i - \lfloor t_i \rfloor$, the quantities computed at line 5 of the algorithm. Then, $s_0 = 0$, $t_{|C|} = c_v$, and τ_i is the fractional part of t_i. Let $0 = \tau^{(0)} < \tau^{(1)} < \ldots < \tau^{(K)} = 1$ be the sequence of sorted τ_i's, with duplicates removed, as in lines 8-9 of the algorithm. Note that, by construction, K can be at most $|C|$. For $\ell \in \{0, \ldots, c-1\}$, $k \in \{0, \ldots, K-1\}$, let $A_\ell^k = (\ell + \tau^{(k)}, \ell + \tau^{(k+1)})$ be the open intervals in Line 13. Then, the following lemma holds.

Lemma 6. *For every $\ell \in \{1, \ldots, c-1\}$ and $k \in \{0, \ldots, K-1\}$, there exists exactly one $i \in C$ s.t. $A_\ell^k \subset [s_i, t_i]$. Moreover, any such i must have $\bar{y}_{vi} > 0$.*

Proof. By construction $\tau^{(k)} < \tau^{(k+1)}$, so A_ℓ^k is a non-empty interval in $[0, c_v]$. For every $i \in C$, $[s_i, t_i]$ are closed intervals (possibly of length zero, if $\bar{y}_{vi} = 0$) such that $\bigcup_{i \in C} [s_i, t_i] = [0, c_v]$. Hence, there must be at least one $[s_i, t_i]$ s.t. $A_\ell^k \cap [s_i, t_i] \neq \emptyset$. To see that there can be no more than one, suppose that A_ℓ^k intersects more than one sets. Then it must intersect at least two consecutive sets, say $[s_j, t_j]$, $[s_{j+1}, t_{j+1}]$ where, by construction $t_j = s_{j+1}$. This means that $t_j \in A_\ell^k$, which in turn implies that $\tau_j = t_j - \lfloor t_j \rfloor \in (\tau^{(k)}, \tau^{(k+1)})$, which is a contradiction, as the sequence $\tau^{(\cdot)}$ is sorted. Hence, $A_\ell^k \cap [s_i, t_i] \neq \emptyset$ for exactly one $i \in C$. For the same reason as above, t_i cannot belong to A_ℓ^k; if it did, then its fractional part τ_i would belong to $(\tau^{(k)}, \tau^{(k+1)})$, a contradiction. Neither can s_i; if $s_i \in A_\ell^k$, then $s_i > 0$, so $i > 0$. This means that $s_i = t_{i-1}$, and if $t_{i-1} \in A_\ell^k$, we again reach a contradiction. Hence, the only way that $A_\ell^k \cap [s_i, t_i] \neq \emptyset$ is if $A_\ell^k \subset [s_i, t_i]$. Moreover, this implies that $s_i \leq \tau^{(k)}$ and $\tau^{(k+1)} \leq t_i$, which in turn implies that $\bar{y}_{vi} = t_i - s_i > 0$, so the last statement also follows. □

The above lemma implies that Line 13 of the algorithm always finds a unique $i \in C$, for every $\ell \in \{1, \ldots, c_v - 1\}$ and $k \in \{0, \ldots, K-1\}$. Denote this item by $i(k, \ell)$. The next lemma states that all such items obtained for different values of ℓ are distinct:

Lemma 7. *For any two $\ell, \ell' \in \{1, \ldots, c_v - 1\}$ with $\ell \neq \ell'$, $i(k, \ell) \neq i(k, \ell')$.*

Proof. Suppose that $i(k, \ell) = i(k, \ell')$ for some $\ell' > \ell$. From Lemma 6, both $A_\ell^k \subset [s_i, t_i]$ and $A_{\ell'}^k \subset [s_i, t_i]$. Thus, $s_i \leq \ell + \tau^{(k)}$ and $\ell' + \tau^{(k)} < t_i$. On the other hand, $\ell' > \ell$, so $\ell' \geq \ell + 1$. So the above imply that $\bar{y}_{vi} = t_i - s_i > \ell' - \ell \geq 1$, a contradiction, as any feasible $\bar{y}_{vi}$ must be at most 1. □

Observe that μ_v constructed by the algorithm is indeed a probability distribution, as (a) the differences $\tau^{(k+1)} - \tau^{(k)}$ are, by construction, positive, and (b) their sum is 1. Moreover, the above two lemmas imply that the vectors x_v constructed by the algorithm contain exactly c_v non-zero elements, so μ_v is indeed a distribution over $\mathcal{D}_1$. The last lemma establishes that the constructed distribution has the desirable marginals, thereby completing the proof of correctness.

Lemma 8. *If $\bar{y}_{vi} > 0$, then $\sum_{x \in \text{supp}(\mu_v): x_{vi} = 1} \mu_v(x_v) = \bar{y}_{vi}$.*

Proof. Note that the sets A_ℓ^k are disjoint, have non-empty interior, and their union $\mathcal{U} = \bigcup_{\ell,k} A_\ell^k$ has Borel measure (i.e., length) c_v. Consider an $i \in C$ s.t. $\bar{y}_{vi} > 0$. Then, there must be a set A_ℓ^k that intersects $[s_i, t_i]$; if not, then the union $\mathcal{U}$ would have Borel measure at most $1 - \bar{y}_{vi}$, a contradiction. As in the proof of Lemma (6), the set A_ℓ^k that intersects $[s_i, t_i]$ must be a subset of $[s_i, t_i]$. Consider the remainder, i.e., the set difference $[s_i, t_i] \setminus A_\ell^k$. By the same argument as above, if this remainder has non-zero Borel measure, there must be an interval $A_{\ell'}^{k'} \in \mathcal{U}$, different from A_ℓ^k, that intersects it. As before, this set must be included in $[s_i, t_i]$, and as it is disjoint from A_ℓ^k, it will be included in the remainder. We can therefore construct a sequence of such sets $A_\ell^k \in \mathcal{U}$ that are included in $[s_i, t_i]$, so long as their remainder has non-zero Borel measure. Since there are finitely many such sets, this sequence will be finite, and when it terminates the remainder will have Borel measure zero. Hence, the union of these disjoint sets has Borel measure exactly $\bar{y}_{vi}$.

Every interval in this sequence corresponds to an allocation x_v constructed by the algorithm s.t. $x_{vi} = 1$. By Lemma 7, each interval corresponds to a distinct allocation. Moreover, since the remainder of this construction has Borel measure zero, no other set in $\mathcal{U}$ can intersect $[s_i, t_i]$. Finally, the probability of these allocations under μ_v is equal to the sum of the Borel measures of this sets; as they are disjoint, the latter is equal to the Borel measure of their union, which is $\bar{y}_{vi}$. □

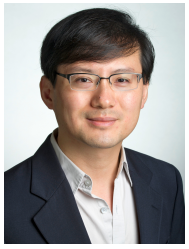
ACKNOWLEDGMENT

The authors gratefully acknowledge support from National Science Foundation grants CNS-1718355, CNS-1423250, OAC-1659403, and a Cisco Systems research grant.

REFERENCES

[1] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," in *SIGMETRICS*, 2016.
[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT*, 2009.
[3] E. J. Rosensweig, D. S. Menasche, and J. Kurose, "On the steady-state of cache networks," in *INFOCOM*, 2013.
[4] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," Telecom ParisTech, Tech. Rep., 2011.
[5] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *INFOCOM*, 2010.
[6] 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.
[7] 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*, vol. 59, no. 12, pp. 8402–8413, 2013.
[8] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *SIGCOMM*, 2002.
[9] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS*, 2002.
[10] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," in *ICPCC*, 2004.
[11] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *Parallel and Distributed Systems*, vol. 15, no. 6, pp. 505–519, 2004.
[12] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, 2002.
[13] A. A. Ageev and M. I. Sviridenko, "Pipage rounding: A new method of constructing algorithms with proven performance guarantee," *Journal of Combinatorial Optimization*, vol. 8, no. 3, pp. 307–328, 2004.
[14] S. Ioannidis and P. Marbach, "Absence of evidence as evidence of absence: A simple mechanism for scalable P2P search," in *INFOCOM*, 2009.
[15] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *ITC*, 2012.
[16] V. Martina, M. Garetto, and E. Leonardi, "A unified approach to the performance analysis of caching systems," in *INFOCOM*, 2014.
[17] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks," *IFIP Performance*, 2014.
[18] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Analysis of TTL-based cache networks," in *VALUETOOLS*, 2012.
[19] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy, "Performance evaluation of the random replacement policy for networks of caches," *Performance Evaluation*, vol. 40, no. 1, pp. 395–396, 2012.
[20] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proceedings of the 23rd international teletraffic congress*. International Teletraffic Congress, 2011, pp. 111–118.
[21] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM Journal on Computing*, vol. 38, no. 4, pp. 1411–1429, 2008.
[22] Y. Bartal, A. Fiat, and Y. Rabani, "Competitive algorithms for distributed data management," *Journal of Computer and System Sciences*, vol. 51, no. 3, pp. 341–358, 1995.
[23] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *SODA*, 2006.
[24] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale VoD system," in *CoNext*, 2010.
[25] 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*. Springer, 2007, pp. 182–196.
[26] J. Vondrák, "Optimal approximation for the submodular welfare problem in the value oracle model," in *STOC*, 2008.
[27] M. Leconte, M. Lelarge, and L. Massoulie, "Designing adaptive replication schemes in distributed content delivery networks," in *ITC*, 2015.
[28] M. Leconte, M. Lelarge, and L. Massoulié, "Bipartite graph structures for efficient balancing of heterogeneous loads," in *SIGMETRICS*, 2012.
[29] S. Ioannidis, L. Massoulié, and A. Chaintreau, "Distributed caching over heterogeneous mobile networks," in *SIGMETRICS*, 2010.
[30] K. L. Clarkson, "Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm," *ACM Transactions on Algorithms*, vol. 6, no. 4, p. 63, 2010.
[31] A. L. Stolyar, "Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm," *Queueing Systems*, vol. 50, no. 4, pp. 401–457, 2005.
[32] 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.
[33] M. X. Goemans and D. P. Williamson, "New 3/4-approximation algorithms for the maximum satisfiability problem," *SIAM Journal on Discrete Mathematics*, vol. 7, no. 4, pp. 656–666, 1994.
[34] A. Nemirovski, *Efficient Methods in Convex Programming*, 2005.
[35] H. J. Kushner and G. Yin, *Stochastic Approximation, Recursive algorithms and Applications*. Springer Science & Business Media, 2003, vol. 35.
[36] R. G. Gallager, *Discrete stochastic processes*. Springer Science & Business Media, 2012, vol. 321.
[37] R. T. Rockafellar, *Convex Analysis*. Princeton University Press, 1970.
[38] O. Gabber and Z. Galil, "Explicit constructions of linear-sized superconcentrators," *Journal of Computer and System Sciences*, vol. 22, no. 3, pp. 407–420, 1981.
[39] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[40] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *STOC*, 2000.
[41] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

Stratis Ioannidis is an Assistant Professor in the Electrical and Computer Engineering department at Northeastern University, in Boston, MA, where he also holds a courtesy appointment with the College of Computer & Information Sciences. He received his B.Sc. (2002) in Electrical and Computer Engineering from the National Technical University of Athens, Greece, and his M.Sc. (2004) and Ph.D. (2009) in Computer Science from the University of Toronto, Canada. Prior to joining Northeastern, he was a research scientist at the Technicolor research centers in Paris, France, and Palo Alto, CA, as well as at Yahoo Labs in Sunnyvale, CA. He is the recipient of an NSF CAREER award, a Google Faculty Research Award, and a Best Paper Award at the 2017 ACM Conference on Information-centric Networking (ICN).



Edmund Yeh is a Professor of Electrical and Computer Engineering at Northeastern University, Boston, USA. He received his B.S. in Electrical Engineering with Distinction and Phi Beta Kappa from Stanford University in 1994. He then studied at Cambridge University on the Winston Churchill Scholarship, obtaining his M.Phil in Engineering in 1995. He received his Ph.D. in Electrical Engineering and Computer Science from MIT under Professor Robert Gallager in 2001. He was previously Assistant and Associate Professor of Electrical Engineering, Computer Science, and Statistics at Yale University. He is the recipient of the Alexander von Humboldt Research Fellowship, the Army Research Office Young Investigator Award, and the Best Paper Award at the 2017 ACM Conference on Information-centric Networking (ICN) and at the 2015 IEEE International Conference on Communications (ICC) Communication Theory Symposium.