Kelly Cache Networks

Milad Mahdian, Armin Moharrer, Stratis Ioannidis, and Edmund Yeh

***Abstract*— We study networks of M/M/1 queues in which nodes act as caches that store objects. Exogenous requests for objects are routed towards nodes that store them; as a result, object traffic in the network is determined not only by demand but, crucially, by where objects are cached. We determine how to place objects in caches to attain a certain design objective, such as, e.g., minimizing network congestion or retrieval delays. We show that for a broad class of objectives, including minimizing both the expected network delay and the sum of network queue lengths, this optimization problem can be cast as an NP-hard submodular maximization problem. We show that so-called *continuous greedy* algorithm attains a ratio arbitrarily close to $1 - 1/e \approx 0.63$ using a deterministic estimation via a power series; this drastically reduces execution time over prior art, which resorts to sampling. Finally, we show that our results generalize, beyond M/M/1 queues, to networks of M/M/k and symmetric M/D/1 queues.**

***Index Terms*— Kelly networks, cache networks, ICN.**

I. Introduction

KELLY networks [3] are multi-class networks of queues capturing a broad array of queue service disciplines, including FIFO, LIFO, and processor sharing. Both Kelly networks and their generalizations (including networks of quasi-reversible and symmetric queues) are well-studied, classic topics [3]–[6]. One of their most appealing properties is that their steady-state distributions have a product-form: as a result, steady state properties such as expected queue sizes, packet delays, and server occupancy rates have closed-form formulas as functions of, e.g., routing and scheduling policies.

In this paper, we consider Kelly networks in which nodes are equipped with caches, i.e., storage devices of finite capacity, which can be used to store objects. Exogenous requests for objects are routed towards nodes that store them; upon reaching a node that stores the requested object, a response packet containing the object is routed towards the request source. As a result, object traffic in the network is determined not only by the demand but, crucially, by where objects are cached. This abstract setting is motivated by–and can be used to model–various networking applications involving the placement and transmission of content. This includes information centric networks (ICNs) [7]–[9], content delivery networks (CDNs) [10], [11], web-caches [12]–[14], wireless/femtocell networks [15]–[17], and peer-to-peer networks [18], [19], to name a few.

Manuscript received April 16, 2019; revised January 3, 2020; accepted February 23, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Joo. Date of publication April 14, 2020; date of current version June 18, 2020. This work was supported in part by the National Science Foundation under Grant NeTS-1718355, in part by Intel Corporation, and in part by Cisco Systems. This is an extended version of an article that appeared in the IEEE International Conference on Computer Communications (INFOCOM 2019). *(Corresponding author: Armin Moharrer.)*

The authors are with the Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: mmahdian@ece.neu.edu; amoharrer@ece.neu.edu; ioannidis@ece.neu.edu; eyeh@ece.neu.edu).

Digital Object Identifier 10.1109/TNET.2020.2982863

In many of these applications, determining the *object placement*, i.e., how to place objects in network caches, is a decision that can be made by the network designer in response to object popularity and demand. To that end, we are interested in determining how to place objects in caches so that traffic attains a design objective such as minimizing delay.

We make the following contributions. First, we study the problem of optimizing the placement of objects in caches in Kelly cache networks of M/M/1 queues, with the objective of minimizing a cost function of the system state. We show that, for a broad class of cost functions, including packet delay, system size, and server occupancy rate, *this optimization amounts to a submodular maximization problem with matroid constraints*. This result applies to general Kelly networks with fixed service rates; in particular, it holds for FIFO, LIFO, and processor sharing disciplines at each queue.

The so-called continuous greedy algorithm [1] attains a $1 - 1/e$ approximation for this NP-hard problem. However, it does so by computing an expectation over a random variable with exponential support via randomized sampling. The number of samples required to attain the $1 - 1/e$ approximation guarantee can be prohibitively large in realistic settings. Our second contribution is to show that, for Kelly networks of M/M/1 queues, *this randomization can be entirely avoided*: a closed-form solution can be computed using the Taylor expansion of our problem's objective. To the best of our knowledge, we are the first to identify a submodular maximization problem that exhibits this structure, and to exploit it to eschew sampling. Finally, we extend our results to *networks of M/M/k and symmetric M/D/1 queues*, and prove a negative result: submodularity does *not* arise in networks of M/M/1/k queues. We extensively evaluate our proposed algorithms over several synthetic and real-life topologies.

The remainder of our paper is organized as follows. We review related work in Sec. II. We present our mathematical model of a Kelly cache network in Sec. III, and our results on submodularity and the continuous-greedy algorithm in networks of M/M/1 queues in Sections IV and V, respectively. Our extensions are described in Sec. VI; our numerical evaluation is in Sec. VII. Finally, we conclude in Sec. VIII.

II. Related Work

Our approach is closest to, and inspired by, recent work by Shanmugam *et al.* [15] and Ioannidis and Yeh [9]. Ioannidis and Yeh consider a setting very similar to ours but without queuing: edges are assigned a fixed weight, and the objective is a linear function of incoming traffic scaled by these weights. This can be seen as a special case of our model, namely, one where edge costs are linear (see also Eq. (15) in Sec. III-B). Shanmugam *et al.* [15] study a similar optimization problem, restricted to the context of femtocaching. The authors show that this is an NP-hard, submodular maximization problem

with matroid constraints. They provide a $1-1/e$ approximation algorithm based on a technique by Ageev and Sviridenko [20]: this involves maximizing a concave relaxation of the original objective, and rounding via pipage-rounding [20]. Ioannidis and Yeh show that the same approximation technique applies to more general cache networks with linear edge costs. They also provide a distributed, adaptive algorithm that attains an $1-1/e$ approximation. The same authors extend this framework to jointly optimize both caching and routing decisions [21].

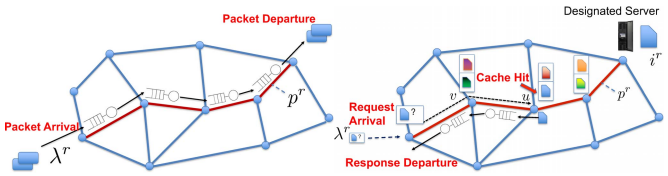
Our work can be seen as an extension of [9], [15], in that it incorporates queuing in the cache network. In contrast to both [9] and [15] however, costs like delay or queue sizes are highly non-linear in the presence of queuing. From a technical standpoint, this departure from linearity requires us to employ significantly different optimization methods than the ones in [9], [15]. In particular, our objective *does not admit a concave relaxation* and, consequently, the technique by Ageev and Sviridenko [20] used in [9], [15] *does not apply*. Instead, we must solve a non-convex optimization problem directly (c.f. Eq. (23)) using the so-called continuous-greedy algorithm.

Several papers have studied the cache optimization problems under restricted topologies [10], [22]–[25]. These works model the network as a bipartite graph: nodes generating requests connect directly to caches in a single hop. The resulting algorithms do not readily generalize to arbitrary topologies. In general, the approximation technique of Ageev and Sviridenko [20] applies to this bipartite setting, and additional approximation algorithms have been devised for several variants [10], [22]–[24]. We differ by (a) considering a multi-hop setting, and (b) introducing queuing, which none of the above works considers.

Submodular function maximization subject to matroid constraints appears in many important problems in combinatorial optimization; for a brief review of the topic and applications, see [26], respectively. Nemhauser *et al.* [27] show that the greedy algorithm produces a solution within 1/2 of the optimal. Vondrák [28] and Calinescu *et al.* [1] show that the continuous-greedy algorithm produces a solution within $(1-1/e)$ of the optimal in polynomial time, which cannot be further improved [29]. In the general case, the continuous-greedy algorithm requires sampling to estimate the gradient of the so-called multilinear relaxation of the objective (see Sec. V). One of our main contributions is to show that MAXCG, the optimization problem we study here, exhibits additional structure: we use this to construct a sampling-free estimator of the gradient via a power-series or Taylor expansion. To the best of our knowledge, we are the first to use such an expansion to eschew sampling; this technique may apply to submodular maximization problems beyond MAXCG.

III. MODEL

Motivated by applications such as ICNs [7], CDNs [10], [11], and peer-to-peer networks [18], we introduce Kelly cache networks. In contrast to classic Kelly networks, each node is associated with a cache of finite storage capacity. Exogenous traffic consisting of *requests* is routed towards nodes that store objects; upon reaching a node that stores the requested object, a *response* packet containing the object is routed towards the node that generated the request. As a result, content traffic in the network is determined not only by demand but, crucially, by how contents are cached. An illustration highlighting the differences between Kelly cache networks, introduced below, and classic Kelly networks, can be found in Fig. 1.



(a) Kelly Network (b) Kelly Cache Network

Fig. 1. (a) Example of a Kelly network. Packets of class r enter the network with rate λ^r, are routed through consecutive queues over path p^r, and subsequently exit the network. (b) Example of a Kelly cache network. Each node $v \in V$ is equipped with a cache of capacity c_v. Exogenous requests of type r for object i^r enter the network and are routed over a predetermined path p^r towards the designated server storing i^r. Upon reaching an intermediate node u storing the requested object i^r, a response packet containing the object is generated. The response is then forwarded towards the request's source in the reverse direction on path p^r. Request packets are of negligible size compared to response messages; as a result, we ignore request traffic and focus on queuing due to response traffic alone.

Although we describe Kelly cache networks in terms of FIFO M/M/1 queues, the product form distribution (c.f. (6)) arises for many different service principles beyond FIFO (c.f. Section 3.1 of [3]) including Last-In First-Out (LIFO) and processor sharing. All results we present extend to these service disciplines; we discuss more extensions in Sec. VI.

A. Kelly Cache Networks

Graphs and Paths: We use the notation $G(V,E)$ for a directed graph G with nodes V and edges $E \subseteq V \times V$. A directed graph is called *symmetric* or *bidirectional* if $(u,v) \in E$ if and only if $(v,u) \in E$. A *path* p is a sequence of adjacent nodes, i.e., $p = p_1, p_2, \ldots, p_K$ where $(p_k, p_{k+1}) \in E$, for all $1 \leq k < K \equiv |p|$. A path is *simple* if it contains no loops (i.e., each node appears once). We use the notation $v \in p$, where $v \in V$, to indicate that node v appears in the path, and $e \in p$, where $e = (u,v) \in E$, to indicate that nodes u,v are two consecutive (and, therefore, adjacent) nodes in p. For $v \in p$, where p is simple, we denote by $k_p(v) \in \{1, \ldots, |p|\}$ the position of node $v \in V$ in p, i.e., $k_p(v) = k$ if $p_k = v$.

Network Definition: Formally, we consider a Kelly network of M/M/1 FIFO queues, represented by a symmetric directed graph $G(V,E)$. As in classic Kelly networks, each edge $e \in E$ is associated with an M/M/1 queue with service rate μ_e.[1] In addition, each node has a cache that stores objects of equal size from a set $\mathcal{C}$, the *object catalog*. Each node $v \in V$ may store at most $c_v \in \mathbb{N}$ objects from $\mathcal{C}$ in its cache. Hence, if $x_{vi} \in \{0,1\}$ is a binary variable indicating whether node $v \in V$ is storing object $i \in \mathcal{C}$, then $\sum_{i \in \mathcal{C}} x_{vi} \leq c_v$, for all $v \in V$. We refer to $\mathbf{x} = [x_{vi}]_{v \in V,\ i \in \mathcal{C}} \in \{0,1\}^{|V||\mathcal{C}|}$ as the *global placement* or, simply, *placement* vector. We denote by

$$\mathcal{D} = \left\{ \mathbf{x} \in \{0,1\}^{|V||\mathcal{C}|} : \textstyle\sum_{i \in \mathcal{C}} x_{vi} \leq c_v, \forall v \in V \right\}, \quad (1)$$

[1]We associate queues with edges for concreteness. Alternatively, queues can be associated with nodes, or both nodes and edges; all such representations lead to product form distributions (6), and all our results extend to these cases.

the set of *feasible* placements that satisfy the storage capacity constraints. We assume that for every object $i \in \mathcal{C}$, there exists a set of nodes $\mathcal{S}_i \subseteq V$ that *permanently store* i. We refer to nodes in $\mathcal{S}_i$ as *designated servers* for $i \in \mathcal{C}$. We assume that designated servers store i in permanent storage *outside* their cache. Put differently, the aggregate storage capacity of a node is $c'_v = c_v + |\{i : v \in \mathcal{S}_i\}|$, but only the non-designated slots c_v are part of the system's design.

Object Requests and Responses: Traffic in the cache network consists of two types of packets: *requests* and *responses*, as shown in Fig. 1(b). Requests for an object are always routed towards one of its designated servers, ensuring that every request is satisfied. However, requests may terminate early: upon reaching any node that caches the requested object, the latter generates a response carrying the object. This is forwarded towards the request's source, following the same path as the request, in reverse. Consistent with prior literature [9], [21], we treat request traffic as negligible when compared to response traffic, which carries objects, and henceforth focus only on queues bearing response traffic.

Formally, a request and its corresponding response are fully characterized by (a) the object being requested, and (b) the path that the request follows. That is, for the set of requests $\mathcal{R}$, a request $r \in \mathcal{R}$ is determined by a pair (i^r, p^r), where $i^r \in \mathcal{C}$ is the object being requested and p^r is the path the request follows. Each request r is associated with a corresponding Poisson arrival process with rate $\lambda^r \geq 0$, independent of other arrivals and service times. We denote the vector of arrival rates by $\boldsymbol{\lambda} = [\lambda^r]_{r\in\mathcal{R}} \in \mathbb{R}_+^{|\mathcal{R}|}$. For all $r \in \mathcal{R}$, we assume that the path p^r is well-routed [9], that is: (a) path p^r is simple, (b) the terminal node of the path is a designated server, i.e., a node in $\mathcal{S}_{i^r}$, and (c) no other intermediate node in p^r is a designated server. As a result, requests are always served, and response packets (carrying objects) always follow a sub-path of p^r in reverse towards the request source (namely, p^r_1).

Steady State Distribution: Given an object placement $\mathbf{x} \in \mathcal{D}$, the resulting system is a multi-class Kelly network, with packet classes determined by the request set $\mathcal{R}$. This is a Markov process over the state space determined by queue contents. In particular, let n^r_e be the number of packets of class $r \in \mathcal{R}$ in queue $e \in E$, and

$$n_e = \sum_{r\in\mathcal{R}} n^r_e \tag{2}$$

be the total queue size. The state of a queue $\mathbf{n}_e \in \mathcal{R}^{n_e}$, $e \in E$, is the vector of length n_e representing the class of each packet in each position of the queue. The *system state* is then given by $\mathbf{n} = [\mathbf{n}_e]_{e\in E}$; we denote by Ω the state space of this Markov process.

In contrast to classic Kelly networks, network traffic and, in particular, the load on each queue, depend on placement $\mathbf{x}$. Indeed, if $(v, u) \in p^r$ for $r \in \mathcal{R}$, the arrival rate of responses of class $r \in \mathcal{R}$ in queue $(u, v) \in E$ is:

$$\lambda^r_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) = \lambda^r \prod_{k'=1}^{k_{p^r}(v)} (1 - x_{p^r_{k'} i^r}), \quad \text{for } (v, u) \in p^r, \tag{3}$$

i.e., responses to requests of class r pass through edge $(u, v) \in E$ if and only if no node preceding u in the path p^r stores object i^r–see also Fig. 1(b). Note that (3) presumes queues on p^r are stable. As $\mu_{(u,v)}$ is the service rate of the queue in $(u, v) \in E$, the load on edge $(u, v) \in E$ is:

$$\rho_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) = \tfrac{\lambda_{(u,v)}(\mathbf{x},\boldsymbol{\lambda})}{\mu_{(u,v)}}, \tag{4}$$

where

$$\lambda_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) = \textstyle\sum_{r\in\mathcal{R}:(v,u)\in p^r} \lambda^r_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) \tag{5}$$

is the total arrival rate of responses in queue $(u, v) \in E$. The Markov process $\{\mathbf{n}(t); t \geq 0\}$ is positive recurrent when $\rho_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) < 1$, for all $(u, v) \in E$ [3], [30]. Then, the steady-state distribution has a *product form*, i.e.:

$$\pi(\mathbf{n}) = \textstyle\prod_{e\in E} \pi_e(\mathbf{n}_e), \quad \mathbf{n} \in \Omega, \tag{6}$$

where

$$\pi_e(\mathbf{n}_e) = (1 - \rho_e(\mathbf{x}, \boldsymbol{\lambda})) \textstyle\prod_{r\in\mathcal{R}:e\in p^r} \left(\frac{\lambda^r_e(\mathbf{x},\boldsymbol{\lambda})}{\mu_e}\right)^{n^r_e}, \tag{7}$$

and $\lambda^r_e(\mathbf{x}, \boldsymbol{\lambda})$, $\rho_e(\mathbf{x}, \boldsymbol{\lambda})$ are given by (3), (4), respectively. As a consequence, the queue sizes n_e, $e \in E$, also have a product form distribution in steady state, and their marginals are given by:

$$\mathbf{P}(n_e = k) = (1 - \rho_e(\mathbf{x}, \boldsymbol{\lambda}))\rho^k_e(\mathbf{x}, \boldsymbol{\lambda}), \quad k \in \mathbb{N}. \tag{8}$$

Stability Region: Given a placement $\mathbf{x} \in \mathcal{D}$, a vector of arrival rates $\boldsymbol{\lambda} = [\lambda^r]_{r\in\mathcal{R}}$ yields a stable (i.e., positive recurrent) system if and only if $\boldsymbol{\lambda} \in \Lambda_{\mathbf{x}}$, for

$$\Lambda_{\mathbf{x}} := \{\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq 0, \rho_e(\mathbf{x}, \boldsymbol{\lambda}) < 1, \forall e \in E\} \subset \mathbb{R}_+^{|\mathcal{R}|}, \tag{9}$$

where loads ρ_e, $e \in E$, are given by (4). Conversely, given a vector $\boldsymbol{\lambda} \in \mathbb{R}_+^{|\mathcal{R}|}$,

$$\mathcal{D}_{\boldsymbol{\lambda}} = \{\mathbf{x} \in \mathcal{D} : \rho_e(\mathbf{x}, \boldsymbol{\lambda}) < 1, \forall e \in E\} \subseteq \mathcal{D} \tag{10}$$

is the set of feasible placements under which the system is stable. It is easy to confirm that, by the monotonicity of ρ_e w.r.t. $\mathbf{x}$, if $\mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}$ and $\mathbf{x}' \geq \mathbf{x}$, then $\mathbf{x}' \in \mathcal{D}_{\boldsymbol{\lambda}}$, where the vector inequality $\mathbf{x}' \geq \mathbf{x}$ is component-wise. In particular, if $\mathbf{0} \in D_{\boldsymbol{\lambda}}$ (i.e., the system is stable without caching), then $\mathcal{D}_{\boldsymbol{\lambda}} = \mathcal{D}$.

B. Cache Optimization

Given a Kelly cache network represented by graph $G(V, E)$, service rates μ_e, $e \in E$, storage capacities c_v, $v \in V$, a set of requests $\mathcal{R}$, and arrival rates λ_r, for $r \in \mathcal{R}$, we wish to determine placements $\mathbf{x} \in \mathcal{D}$ that optimize a certain design objective. In particular, we seek placements that are solutions to optimization problems of the following form: MINCOST

$$\text{Minimize: } C(\mathbf{x}) = \textstyle\sum_{e\in E} C_e(\rho_e(\mathbf{x}, \boldsymbol{\lambda})), \tag{11a}$$

$$\text{subj. to: } \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}, \tag{11b}$$

where $C_e : [0, 1) \to \mathbb{R}_+$, $e \in E$, are positive *cost* functions, $\rho_e : \mathcal{D} \times \mathbb{R}_+^{|\mathcal{R}|} \to \mathbb{R}_+$ is the load on edge e, given by (4), and $\mathcal{D}_{\boldsymbol{\lambda}}$ is the set of feasible placements that ensure stability, given by (10). We make the following standing assumption on the cost functions appearing in MINCOST:

Assumption 1: For all $e \in E$, functions $C_e : [0, 1) \to \mathbb{R}_+$ are convex and non-decreasing on $[0, 1)$.

Assumption 1 is natural; indeed it holds for many cost functions that often arise in practice. We list several examples:

Example 1 (Queue Size): Under steady-state distribution (6), the expected number of packets in queue $e \in E$ is given by

$$\mathbb{E}[n_e] = C_e(\rho_e) = \frac{\rho_e}{1-\rho_e}, \tag{12}$$

which is indeed convex and non-decreasing for $\rho_e \in [0,1)$. Note that, for C_e given by (12), objective (11a) captures the expected total number of packets in the system in steady state.

Example 2 (Delay): From Little's Theorem [30], the expected delay experienced by a packet in the system is

$$\mathbb{E}[T] = \frac{1}{\|\boldsymbol{\lambda}\|_1} \sum_{e \in E} \mathbb{E}[n_e], \tag{13}$$

where $\|\boldsymbol{\lambda}\|_1 = \sum_{r \in \mathcal{R}} \lambda^r$ is the total arrival rate, and $\mathbb{E}[n_e]$ is the expected size of each queue. Thus, the expected delay can also be written as the sum of functions that satisfy Assumption 1. We note that the same is true for the sum of the expected delays per queue $e \in E$, as the latter are given by

$$\mathbb{E}[T_e] = \frac{1}{\lambda_e}\mathbb{E}[n_e] = \frac{1}{\mu_e(1-\rho_e)}, \tag{14}$$

which are also convex and non-decreasing in ρ_e.

Example 3 (Queuing Probability/Load per Edge): In a FIFO queue, the *queuing probability* is the probability of arriving in a system where the server is busy; by (8), this is:

$$C_e(\rho_e) = \rho_e = \lambda_e/\mu_e, \tag{15}$$

which is again non-decreasing and convex. This is also, of course, the load per edge. By treating $1/\mu_e$ as the weight of edge $e \in E$, this setting recovers the objectives of Shanmugam *et al.* [15] and Ioannidis and Yeh [9] as a special case of our model.

Example 4 (Monotone Separable Costs): More generally, Assumption 1 holds for *arbitrary monotone separable costs*, i.e., costs that (1) are summed across queues, (2) depend only on queue sizes n_e, and (3) are non-decreasing. Formally:

Lemma 1: Consider a state-dependent cost function $c : \Omega \to \mathbb{R}_+$ such that:

$$c(\mathbf{n}) = \sum_{e \in E} c_e(n_e),$$

where $c_e : \mathbb{N} \to \mathbb{R}_+$, $e \in E$, are non-decreasing functions of the queue sizes n_e, $e \in E$. Then, the steady state cost under distribution (6) *takes the form* (11a)*, i.e.,*

$$\mathbb{E}[c(\mathbf{n})] = \sum_{e \in E} C_e(\rho_e)$$

where $C_e : [0,1) \to \mathbb{R}_+$ satisfy Assumption 1.

Proof: As the cost at state $\mathbf{n} \in \Omega$ can be written as $c(\mathbf{n}) = \sum_{e \in E} c_e(n_e)$, we have that $\mathbb{E}[c(\mathbf{n})] = \sum_{e \in E} \mathbb{E}[c_e(n_e)]$. On the other hand, as $c_e(n_e) \geq 0$,

$$\begin{aligned}
\mathbb{E}[c_e(n_e)] &= \sum_{n=0}^{\infty} c_e(n)\mathbf{P}(n_e = n) \\
&= c_e(0) + \sum_{n=0}^{\infty} (c_e(n+1) - c_e(n))\mathbf{P}(n_e > n) \\
&\overset{(8)}{=} c_e(0) + \sum_{n=0}^{\infty} (c_e(n+1) - c_e(n))\rho_e^n
\end{aligned} \tag{16}$$

TABLE I
NOTATION SUMMARY

Kelly Cache Networks	
$G(V,E)$	Network graph, with nodes V and edges E
$k_p(v)$	position of node v in path p
$\mu_{(u,v)}$	Service rate of edge $(u,v) \in E$
$\mathcal{R}$	Set of classes/types of requests
λ^r	Arrival rate of class $r \in \mathcal{R}$
p^r	Path followed by class $r \in \mathcal{R}$
i^r	Object requested by class $r \in \mathcal{R}$
$\mathcal{C}$	Item catalog
$\mathcal{S}_i$	Set of designated servers of $i \in \mathcal{C}$
c_v	Cache capacity at node $v \in V$
x_{vi}	Variable indicating whether $v \in V$ stores $i \in \mathcal{C}$
$\mathbf{x}$	Placement vector of x_{vi}s, in $\{0,1\}^{\lvert V\rvert\lvert\mathcal{C}\rvert}$
$\boldsymbol{\lambda}$	Vector of arrival rates λ^r, $r \in \mathcal{R}$
λ_e^r	Arrival rate of class r responses over edge $e \in E$
ρ_e	Load on edge $e \in E$
Ω	State space
$\mathbf{n}$	Global state vector in Ω
$\pi(\mathbf{n})$	Steady-state distribution of $\mathbf{n} \in \Omega$
$\mathbf{n}_e$	State vector of queue at edge $e \in E$
$\pi_e(\mathbf{n}_e)$	Marginal of steady-state distribution of queue $\mathbf{n}_e$
n_e	Size of queue at edge $e \in E$
Q	Set of request sources
Cache Optimization	
C	Global Cost function
C_e	Cost function of edge $e \in E$
$\mathcal{D}$	Set of placements $\mathbf{x}$ satisfying capacity constraints
$\mathbf{x}_0$	A feasible placement in $\mathcal{D}$
$F(\mathbf{x})$	Caching gain of placement $\mathbf{x}$ over $\mathbf{x}_0$
y_{vi}	Probability that $v \in V$ stores $i \in \mathcal{C}$
$\mathbf{y}$	Vector of marginal probabilities y_{vi}, in $\{0,1\}^{\lvert V\rvert\lvert\mathcal{C}\rvert}$
$G(\mathbf{y})$	Multilinear extension under marginals $\mathbf{y}$
$\mathcal{D}_{\boldsymbol{\lambda}}$	Set of placements under which system is stable under arrivals $\boldsymbol{\lambda}$
$\tilde{\mathcal{D}}$	Convex hull of constraints of MAXCG
Conventions	
$\mathbb{N}, \mathbb{R}$	Natural numbers (including 0), real numbers
$\mathbb{R}_+$	non-negative reals
$\texttt{supp}(\cdot)$	Support of a vector
$\texttt{conv}(\cdot)$	Convex hull of a set
$[\mathbf{x}]_{+i}$	Vector equal to $\mathbf{x}$ with i-th coordinate set to 1
$[\mathbf{x}]_{-i}$	Vector equal to $\mathbf{x}$ with i-th coordinate set to 0
$\mathbf{0}$	Vector of zeros

As c_e is non-decreasing, $c_e(n+1) - c_e(n) \geq 0$ for all $n \in \mathbb{N}$. On the other hand, for all $n \in \mathbb{N}$, ρ^n is a convex non-decreasing function of ρ in $[0,1)$, so $\mathbb{E}[c_e(n_e)]$ is a convex function of ρ_e as a positively weighted sum of convex non-decreasing functions. □

In summary, MINCOST captures many natural cost objectives, while Assumption 1 holds for *any* non-decreasing cost function that depends only on queue sizes.

C. Set Functions and Submodularity

Given a finite set $\mathcal{X}$, a set function $f : 2^{\mathcal{X}} \to \mathbb{R}$ is called *non-decreasing* if $f(S) \leq f(S')$ for all $S \subseteq S' \subseteq \mathcal{X}$, and *non-increasing* if $-f$ is non-decreasing. Function f is called *submodular* if it satisfies the following *diminishing returns* property: for all $S \subseteq S' \subseteq \mathcal{X}$, and all $x \in \mathcal{X}$,

$$f(S' \cup \{x\}) - f(S') \leq f(S \cup \{x\}) - f(S), \tag{17}$$

A function is called *supermodular* if $-f$ is submodular (or, equivalently, (17) holds with the inequality reversed).

IV. SUBMODULARITY AND THE GREEDY ALGORITHM

Problem MINCOST is NP-hard; this is true even when cost functions c_e are linear, and the objective is to minimize the sum of the loads per edge [9], [15]. In what follows, we outline

our methodology for solving this problem; it relies on the fact that the objective of MINCOST is a *supermodular set function*; our first main contribution is to show that this property is a direct consequence of Assumption 1.

Cost Supermodularity and Caching Gain. First, observe that the cost function C in MINCOST can be naturally expressed as a set function. Indeed, for $S \subset V \times \mathcal{C}$, let $\mathbf{x}_S \in \{0,1\}^{|V||\mathcal{C}|}$ be the binary vector whose support is S (i.e., its non-zero elements are indexed by S). As there is a 1-1 correspondence between a binary vector $\mathbf{x}$ and its support $\mathtt{supp}(\mathbf{x})$, we can interpret $C : \{0,1\}^{|V||\mathcal{C}|} \to \mathbb{R}_+$ as a set function $C : V \times \mathcal{C} \to \mathbb{R}_+$ via $C(S) \triangleq C(\mathbf{x}_S)$. Then, the following theorem holds:

Theorem 1: Under Assumption 1, $C(S) \triangleq C(\mathbf{x}_S)$ is non-increasing and supermodular over $\{\mathtt{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

Proof: We use the following auxiliary lemma (see, e.g., [26]); we prove this here for completeness.

Lemma 2: Let $f : \mathbb{R} \to \mathbb{R}$ be a convex and non-decreasing function. Also, let $g : \mathcal{X} \to \mathbb{R}$ be a non-increasing supermodular set function. Then $h \triangleq f \circ g$ is also supermodular.

Proof: Since g is non-increasing, for any $\mathbf{x}, \mathbf{x}' \subseteq \mathcal{X}$ we have $g(\mathbf{x} \cap \mathbf{x}') \geq g(\mathbf{x}) \geq g(\mathbf{x} \cup \mathbf{x}')$, and $g(\mathbf{x} \cap \mathbf{x}') \geq g(\mathbf{x}') \geq g(\mathbf{x} \cup \mathbf{x}')$. Due to supermodularity of g, we can find $\alpha, \alpha' \in [0,1]$, $\alpha + \alpha' \geq 1$ such that $g(\mathbf{x}) = (1-\alpha)g(\mathbf{x} \cap \mathbf{x}') + \alpha g(\mathbf{x} \cup \mathbf{x}')$, and $g(\mathbf{x}') = (1-\alpha')g(\mathbf{x} \cap \mathbf{x}') + \alpha' g(\mathbf{x} \cup \mathbf{x}')$. Then, we have

$$\begin{aligned}
&f(g(\mathbf{x})) + f(g(\mathbf{x}')) \\
&\quad \leq (1-\alpha) f(g(\mathbf{x} \cap \mathbf{x}')) + \alpha f(g(\mathbf{x} \cup \mathbf{x}')) \\
&\qquad + (1-\alpha') f(g(\mathbf{x} \cap \mathbf{x}')) + \alpha' f(g(\mathbf{x} \cup \mathbf{x}')) \\
&\quad = f(g(\mathbf{x} \cap \mathbf{x}')) + f(g(\mathbf{x} \cup \mathbf{x}')) \\
&\qquad + (1-\alpha-\alpha')(f(g(\mathbf{x} \cap \mathbf{x}')) - f(g(\mathbf{x} \cup \mathbf{x}'))) \\
&\quad \leq f(g(\mathbf{x} \cap \mathbf{x}')) + f(g(\mathbf{x} \cup \mathbf{x}')),
\end{aligned}$$

where the first inequality is due to convexity of f, and the second one is because $\alpha + \alpha' \geq 1$ and $f(g(.))$ is non-increasing. This proves $h(\mathbf{x}) \triangleq f(g(\mathbf{x}))$ is supermodular. □

To conclude the proof of Thm. 1, observe that it is easy to verify that $\rho_e, \forall e \in E$, is supermodular and non-increasing in S (see also [9]). Since, by Assumption 1, C_e is a non-decreasing function, then, $C_e(S) \triangleq C_e(\rho_{u,v}(S))$ is non-increasing. By Lemma 2, $C_e(S)$ is also supermodular. Hence, the cost function is non-increasing and supermodular as the sum of non-increasing and supermodular functions. □

In light of Lemma 1, Thm. 1 implies that supermodularity arises *for a broad array of natural cost objectives*, including expected delay and system size; it also applies under the *full generality of Kelly networks* under which a product form arises, including FIFO, LIFO, and round robin service disciplines. Armed with this theorem, we turn our attention to converting MINCOST to a *submodular maximization* problem. In doing so, we face the problem that the domain $\mathcal{D}_{\boldsymbol{\lambda}}$, determined not only by storage capacity constraints, but also by stability, may be difficult to characterize. Nevertheless, we show that a problem that is amenable to approximation can be constructed, provided that a placement $\mathbf{x}_0 \in \mathcal{D}_{\boldsymbol{\lambda}}$ is known.

In particular, suppose that we have access to a single $\mathbf{x}_0 \in \mathcal{D}_{\boldsymbol{\lambda}}$. We define the *caching gain* $F : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}_+$ as $F(\mathbf{x}) = C(\mathbf{x}_0) - C(\mathbf{x})$. Note that, for $\mathbf{x} \geq \mathbf{x}_0$, $F(\mathbf{x})$ is the relative decrease in the cost compared to the cost under $\mathbf{x}_0$.

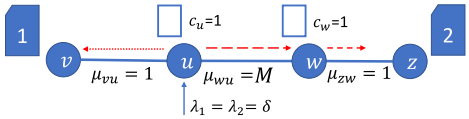


Fig. 2. Path graph, illustrating that the $1/2$-approximation ratio of greedy is tight. Greedy caches item 2 in node u, while the optimal decision is to cache item 1 in u and item 2 in node w. For M large enough, the approximation ratio can be made arbitrarily close to 1/2. In our experiments in Sec. VII, we set $\delta = 0.5$ and $M = 200$.

We consider the following optimization problem: MAXCG

$$\text{Maximize: } F(\mathbf{x}) = C(\mathbf{x}_0) - C(\mathbf{x}) \tag{18a}$$

$$\text{subj. to: } \mathbf{x} \in \mathcal{D}, \mathbf{x} \geq \mathbf{x}_0 \tag{18b}$$

Recall that, if $\mathbf{0} \in \mathcal{D}_{\boldsymbol{\lambda}}$, then $\mathcal{D}_{\boldsymbol{\lambda}} = \mathcal{D}$; in this case, taking $\mathbf{x}_0 = \mathbf{0}$ ensures that problems MINCOST and MaxCG are equivalent. If $\mathbf{x}_0 \neq \mathbf{0}$, the above formulation attempts to maximize the gain restricted to placements $\mathbf{x} \in \mathcal{D}$ that dominate $\mathbf{x}_0$: such placements necessarily satisfy $\mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}$, as $\mathbf{x}_0 \in \mathcal{D}_{\boldsymbol{\lambda}}$. Thm. 1 has the following immediate implication:

Corollary 1: The caching gain $F(S) \triangleq F(\mathbf{x}_S)$ is non-decreasing and submodular over $\{\mathtt{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

Greedy Algorithm: Constraints (18b) define a (partition) matroid [1], [15]. This, along with the submodularity and monotonicity of F imply that we can produce a solution within $\frac{1}{2}$-approximation from the optimal via the *greedy* algorithm [31]. The algorithm, summarized in Alg. 1, iteratively allocates items to caches that yield the largest marginal gain.

Algorithm 1 Greedy

Input: $F : \mathcal{D} \to \mathbb{R}_+, \mathbf{x}_0$

$\mathbf{x} \leftarrow \mathbf{x}_0$
while $A(\mathbf{x}) := \{(v,i) \in V \times \mathcal{C} : \mathbf{x} + \mathbf{e}_{vi} \in \mathcal{D}\}$ is not empty **do**
  $(v^*, i^*) \leftarrow \arg\max_{(v,i) \in A(\mathbf{x})} (F(\mathbf{x} + \mathbf{e}_{vi}) - F(\mathbf{x}))$
  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_{v^* i^*}$
end while
return $\mathbf{x}$

The solution produced by Algorithm 1 is guaranteed to be within a $\frac{1}{2}$-approximation ratio of the optimal solution of MAXCG [27]. The approximation guarantee of $\frac{1}{2}$ is tight:

Lemma 3: For any $\varepsilon > 0$, there exists a cache network such that the greedy algorithm solution is within $\frac{1}{2} + \varepsilon$ from the optimal, when the objective is the sum of expected delays per edge.

Proof: Consider the path topology illustrated in Fig. 2. Assume that requests for files 1 and 2 are generated at node u with rates $\lambda_1 = \lambda_2 = \delta$, for some $\delta \in (0,1)$. Files 1 and 2 are stored permanently at v and z, respectively. Caches exist only on u and w, and have capacity $c_u = c_w = 1$. Edges (u,v), (w,z) have bandwidth $\mu_{(u,v)} = \mu_{(w,z)} = 1$, while edge (u,w) is a high bandwidth link, having capacity $M \gg 1$. Let $\mathbf{x}_0 = \mathbf{0}$. The greedy algorithm starts from empty caches and adds item 2 at cache u. This is because the caching gain from this placement is $c_{(u,w)} + c_{(w,z)} = \frac{1}{M-\delta} + \frac{1}{1-\delta}$, while the caching gain of all other decisions is at most $\frac{1}{1-\delta}$. Any subsequent caching decisions do not change the caching gain.

The optimal solution is to cache item 1 at u and item 2 at w, yielding a caching gain of $2/(1-\delta)$. Hence, the greedy solution attains an approximation ratio $0.5 \cdot (1 + \frac{1-\delta}{M-\delta})$. By appropriately choosing M and δ, this can be made arbitrarily close to 0.5. □

As we discuss in Sec. VII, the greedy algorithm performs well in practice for some topologies; however, Lemma 3 motivates us to seek alternative algorithms, that attain improved approximation guarantees. We note that it is easy to extend Lemma 3 to other objectives, including, e.g., expected delay, queue size, etc. We note also that tight instances can be constructed using caches with capacities larger than 1 (see, e.g., Fig. 3).

V. Continuous-Greedy Algorithm

The *continuous-greedy* algorithm by Calinescu *et al.* [1] attains a tighter guarantee than the greedy algorithm, raising the approximation ratio from 0.5 to $1 - 1/e \approx 0.63$. The algorithm maximizes the so-called *multilinear extension* of objective F, thereby obtaining a fractional solution Y in the convex hull of the constraint space. The resulting solution is then rounded to produce an integral solution. The algorithm requires estimating the gradient of the multilinear extension via sampling; interestingly, we prove that MaxCG exhibits additional structure, which can be used to construct a polynomial-time estimator of this gradient that *eschews sampling altogether*, by using a Taylor expansion.

A. Algorithm Overview

Formally, the multilinear extension of the caching gain F is defined as follows. Define the convex hull of the set defined by the constraints (18b) in MaxCG as:

$$\tilde{\mathcal{D}} = \texttt{conv}(\{\mathbf{x} : \mathbf{x} \in \mathcal{D}, \mathbf{x} \geq \mathbf{x}_0\}) \subseteq [0,1]^{|V||\mathcal{C}|} \tag{19}$$

Intuitively, $\mathbf{y} \in \tilde{\mathcal{D}}$ is a *fractional* vector in $\mathbb{R}^{|V||\mathcal{C}|}$ satisfying the capacity constraints, and the bound $\mathbf{y} \geq \mathbf{x}_0$.

Given a $\mathbf{y} \in \tilde{\mathcal{D}}$, consider a random vector $\mathbf{x}$ in $\{0,1\}^{|V||\mathcal{C}|}$ generated as follows: for all $v \in V$ and $i \in \mathcal{C}$, the coordinates $x_{vi} \in \{0,1\}$ are independent Bernoulli variables such that $\mathbf{P}(x_{vi} = 1) = y_{vi}$. The multilinear extension $G : \tilde{\mathcal{D}} \to \mathbb{R}_+$ of $F : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}_+$ is defined via expectation

$$G(\mathbf{y}) = \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})], \tag{20}$$

parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$. That is:

$$G(\mathbf{y}) = \sum_{\mathbf{x} \in \{0,1\}^{|V||\mathcal{C}|}} F(\mathbf{x}) \times \prod_{(v,i) \in V \times \mathcal{C}} y_{vi}^{x_{vi}} (1 - y_{vi})^{1 - x_{vi}}, \tag{21}$$

The continuous-greedy algorithm, summarized in Alg. 2, proceeds by first producing a fractional vector $\mathbf{y} \in \tilde{\mathcal{D}}$. Starting from $\mathbf{y}_0 = \mathbf{x}_0$, the algorithm iterates over:

$$\mathbf{m}_k \in \arg\max_{\mathbf{m} \in \tilde{\mathcal{D}}} \langle \mathbf{m}, \nabla G(\mathbf{y}_k) \rangle, \tag{22a}$$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \gamma_k \mathbf{m}_k, \tag{22b}$$

for an appropriately selected step size $\gamma_k \in [0,1]$. Intuitively, this yields an approximate solution to the non-convex problem:

$$\text{Maximize: } G(\mathbf{y}) \tag{23a}$$

$$\text{subj. to: } \mathbf{y} \in \tilde{\mathcal{D}}. \tag{23b}$$

Algorithm 2 Continuous-Greedy

Input: $G : \tilde{\mathcal{D}} \to \mathbb{R}_+$, $\mathbf{x}_0$, stepsize $0 < \gamma \leq 1$

$t \leftarrow 0, k \leftarrow 0$ $\mathbf{y}_0 \leftarrow \mathbf{x}_0$

while $t < 1$ **do**

  $\mathbf{m}_k \leftarrow \arg\max_{\mathbf{m} \in \tilde{\mathcal{D}}} \langle \mathbf{m}, \nabla G(\mathbf{y}_k) \rangle$

  $\gamma_k \leftarrow \min\{\gamma, 1 - t\}$

  $\mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + \gamma_k \mathbf{m}_k, t \leftarrow t + \gamma_k, k \leftarrow k + 1$

end while

return $\mathbf{y}_k$

Even though (23) is not convex, the output of Alg. 2 is within a $1 - 1/e$ factor from the optimal solution $\mathbf{y}^* \in \tilde{\mathcal{D}}$ of (23). This fractional solution can be rounded to produce a solution to MaxCG with the same approximation guarantee using either the pipage rounding [20] or the swap rounding [1], [32] schemes: for completeness, we review both in Appendix A.

Note that the maximization in (22a) is a Linear Program (LP): it involves maximizing a linear objective subject to a set of linear constraints, and can thus be computed in polynomial time. However, this presumes access to the gradient ∇G. On the other hand, the expectation $G(\mathbf{y}) = \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})]$ alone, given by (21), involves a summation over $2^{|V||\mathcal{C}|}$ terms, and it may not be easily computed in polynomial time. To address this, the customary approach is to first generate random samples $\mathbf{x}$ and then use these to produce an unbiased estimate of the gradient (see, e.g., [1]); this estimate can be used in Alg. 2 instead of the gradient. Before presenting our estimator tailored to MaxCG, we first describe this sampling-based estimator.

A Sampling-Based Estimator: Function G is linear when restricted to each coordinate y_{vi}, for some $v \in V$, $i \in \mathcal{C}$ (i.e., when all inputs except y_{vi} are fixed). As a result, the partial derivative of G w.r.t. y_{vi} can be written as:

$$\frac{\partial G(\mathbf{y})}{\partial y_{vi}} = \mathbb{E}_{\mathbf{y}}[F(\mathbf{x}) | x_{vi} = 1] - \mathbb{E}_{\mathbf{y}}[F(\mathbf{x}) | x_{vi} = 0] \geq 0, \tag{24}$$

where the last inequality is due to monotonicity of F. One can thus estimate the gradient by (a) producing T random samples $\mathbf{x}^{(\ell)}$, $\ell = 1, \ldots, T$ of the random vector $\mathbf{x}$, consisting of independent Bernoulli coordinates, and (b) computing, for each pair $(v,i) \in V \times \mathcal{C}$, the average

$$\widehat{\frac{\partial G(\mathbf{y})}{\partial y_{vi}}} = \frac{1}{T} \sum_{\ell=1}^{T} \left(F([\mathbf{x}^{\ell}]_{+(v,i)}) - F([\mathbf{x}^{\ell}]_{-(v,i)}) \right), \tag{25}$$

where $[\mathbf{x}]_{+(v,i)}$,$[\mathbf{x}]_{-(v,i)}$ are equal to vector $\mathbf{x}$ with the (v,i)-th coordinate set to 1 and 0, respectively. Using this estimate, Alg. 2 attains an approximation ratio arbitrarily close to $1 - 1/e$ for appropriately chosen T. In particular, the following theorem holds:

Theorem 2([Calinescu et al. [1]]): Consider Alg. 2, with $\nabla G(\mathbf{y}_k)$ replaced by the sampling-based estimate $\widehat{\nabla G(\mathbf{y}^k)}$, given by (25). Set $T = \frac{10}{\delta^2}(1 + \ln(|\mathcal{C}||V|))$, and $\gamma = \delta$, where $\delta = \frac{1}{40|\mathcal{C}||V| \cdot (\sum_{v \in V} c_v)^2}$. Then, the algorithm terminates after $K = 1/\gamma = 1/\delta$ steps and, with high probability,

$$G(\mathbf{y}^K) \geq (1 - (1 - \delta)^{1/\delta}) G(\mathbf{y}^*) \geq (1 - 1/e) G(\mathbf{y}^*),$$

where $\mathbf{y}^*$ is an optimal solution to (23).

The proof of the theorem can be found in Appendix A of Calinescu *et al.* [1] for general submodular functions over

arbitrary matroid constraints; we state Thm. V-A here with constants T and γ set specifically for our objective G and our set of constraints $\tilde{D}$.

Complexity: Under this parametrization of T and γ, Alg. 2 runs in polynomial time. More specifically, note that $1/\delta = O(|\mathcal{C}||V| \cdot (\sum_{v\in V} c_v)^2)$ is polynomial in the input size. Moreover, the algorithm runs for $K = 1/\delta$ iterations in total. Each iteration requires $T = O(\frac{1}{\delta^2}(1 + \ln(|\mathcal{C}||V|)$ samples, each involving a polynomial computation (as F can be evaluated in polynomial time). LP (22a) can be solved in polynomial time in the number of constraints and variables, which are $O(|V||\mathcal{C}|)$. Finally, the rounding schemes presented in Appendix VIII are also poly-time, both requiring at most $O(|V||\mathcal{C}|)$ steps.

B. A Novel Estimator via Taylor Expansion

The classic approach to estimate the gradient via sampling has certain drawbacks. The number of samples T required to attain the $1 - 1/e$ ratio is quadratic in $|V||\mathcal{C}|$. In practice, even for networks and catalogs of moderate size (say, $|V| = |\mathcal{C}| = 100$), the number of samples becomes prohibitive (of the order of 10^8). Producing an estimate for ∇G via a closed form computation that eschews sampling thus has significant computational advantages. In this section, we show that the multilinear relaxation of the caching gain F admits such a closed-form characterization.

We say that a polynomial $f : \mathbb{R}^d \to \mathbb{R}$ is in *Weighted Disjunctive Normal Form* (*W-DNF*) if it can be written as

$$f(\mathbf{x}) = \textstyle\sum_{s\in\mathcal{S}} \beta_s \cdot \prod_{j\in\mathcal{I}(s)}(1 - x_j), \tag{26}$$

for some index set $\mathcal{S}$, positive coefficients $\beta_s > 0$, and index sets $I(s) \subseteq \{1, \ldots, d\}$. Intuitively, treating binary variables $x_j \in \{0, 1\}$ as boolean values, each W-DNF polynomial can be seen as a weighted sum (disjunction) among products (conjunctions) of negative literals. These polynomials arise naturally in the context of our problem; in particular:

Lemma 4: For all $k \geq 1$, $\mathbf{x} \in \mathcal{D}$, and $e \in E$, $\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})$ is a W-DNF polynomial whose coefficients depend on $\boldsymbol{\lambda}$.

Proof (Sketch): The lemma holds for $k = 1$ by (3) and (4). The lemma follows by induction, as W-DNF polynomials over binary $\mathbf{x} \in \mathcal{D}$ are closed under multiplication; this is because $(1 - x)^\ell = (1 - x)$ for all $\ell \geq 1$ when $x \in \{0, 1\}$. A detailed proof can be found in Appendix B.1. □

Hence, *all load powers are W-DNF polynomials.* Expectations of W-DNF polynomials have a remarkable property:

Lemma 5: Let $f : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$ be a W-DNF polynomial, and let $\mathbf{x} \in \mathcal{D}$ be a random vector of independent Bernoulli coordinates parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$. Then $\mathbb{E}_{\mathbf{y}}[f(\mathbf{x})] = f(\mathbf{y})$, where $f(\mathbf{y})$ is the evaluation of the W-DNF polynomial representing f over the real vector $\mathbf{y}$.

Proof: As f is W-DNF, it can be written as

$$f(\mathbf{x}) = \sum_{s\in\mathcal{S}} \beta_s \prod_{t\in\mathcal{I}(s)} (1 - x_t)$$

for appropriate $\mathcal{S}$, and appropriate $\beta_s, \mathcal{I}(s)$, where $s \in \mathcal{S}$. Hence,

$$\begin{aligned}\mathbb{E}_{\mathbf{y}}[f(\mathbf{x})] &= \sum_{s\in\mathcal{S}} \beta_s \mathbb{E}_{\mathbf{y}}\left[\prod_{t\in\mathcal{I}(s)} (1 - x_t)\right] \\ &= \sum_{s\in\mathcal{S}} \beta_s \prod_{t\in\mathcal{I}(s)} (1 - \mathbb{E}_{\mathbf{y}}[x_t]), \text{ by independence} \\ &= \sum_{s\in\mathcal{S}} \beta_s \prod_{t\in\mathcal{I}(s)} (1 - y_t).\end{aligned}$$

□

Lemma 5 states that, to compute the expectation of a W-DNF polynomial f over i.i.d. Bernoulli variables with expectations $\mathbf{y}$, it suffices to *evaluate f over input $\mathbf{y}$*. Expectations computed this way therefore do not require sampling.

We leverage this property to approximate $\nabla G(\boldsymbol{y})$ by taking the Taylor expansion of the cost functions C_e at each edge $e \in E$. This allows us to write C_e as a power series w.r.t. ρ_e^k, $k \geq 1$; from Lemmas 4 and 5, we can compute the expectation of this series in a closed form. In particular, by expanding the series and rearranging terms it is easy to show the following lemma:

Lemma 6: Consider a cost function $C_e : [0, 1) \to \mathbb{R}_+$ which satisfies Assumption 1 and for which the Taylor expansion exists at some $\rho^* \in [0, 1)$. Then, for $\mathbf{x} \in \mathcal{D}$ a random Bernoulli vector parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$,

$$\frac{\partial G(\mathbf{y})}{\partial y_{vi}} \approx \sum_{e\in E}\sum_{k=1}^{L} \alpha_e^{(k)}\left[\rho_e^k\left([\mathbf{y}]_{-(v,i)}, \boldsymbol{\lambda}\right) - \rho_e^k\left([\mathbf{y}]_{+(v,i)}, \boldsymbol{\lambda}\right)\right] \tag{27}$$

where, $\alpha_e^{(k)} = \sum_{j=k}^{L} \frac{(-1)^{j-k}\binom{j}{k}}{j!} C_e^{(j)}(\rho^*)(\rho^*)^{j-k}$, for $k = 0, 1, \cdots, L$, and the error of the approximation is: $\frac{1}{(L+1)!}\sum_{e\in E} C_e^{(L+1)}(\rho')\Big[\mathbb{E}_{[\mathbf{y}]_{-(v,i)}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}] - \mathbb{E}_{[\mathbf{y}]_{+(v,i)}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}]\Big]$, where $\rho' \in [\rho^*, \rho]$.

Proof: The Taylor expansion of C_e at ρ^* is given by:

$$\begin{aligned}C_e(\rho) = C_e(\rho^*) + \sum_{k=1}^{L} \frac{1}{k!} C_e^{(k)}(\rho^*)(\rho - \rho^*)^k + \\ + \frac{1}{(L+1)!} C_e^{(L+1)}(\rho')(\rho - \rho^*)^{L+1},\end{aligned}$$

where $\rho' \in [\rho^*, \rho]$ and $C_e^{(k)}$ is the k-th order derivative of C_e. By expanding this polynomial and reorganizing the terms, we get

$$C_e(\rho) = \sum_{k=0}^{L} \alpha_e^{(k)} \rho^k + \frac{1}{(L+1)!} C_e^{(L+1)}(\rho')(\rho - \rho^*)^{L+1},$$

where

$$\alpha_e^{(k)} = \textstyle\sum_{j=k}^{L} \frac{(-1)^{j-k}\binom{j}{k}}{j!} C_e^{(j)}(\rho^*)(\rho^*)^{j-k},$$

for $k = 0, 1, \cdots, L$. Consider now the L-th order Taylor approximation of C_e, given by

$$\hat{C}_e(\rho) = \sum_{k=0}^{L} \alpha_e^{(k)} \rho^k.$$

Clearly, this is an estimator of C_e, with an error of the order $|C_e(\rho) - \hat{C}_e(\rho)| = o\left((\rho - \rho_*)^L\right)$. Thus, for $\mathbf{x} \in \mathcal{D}$ a random Bernoulli vector parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$,

$$\mathbb{E}_{\mathbf{y}}[C_e(\rho_e(\mathbf{x}, \boldsymbol{\lambda}))] \approx \mathbb{E}_{\mathbf{y}}[\hat{C}_e(\rho_e(\mathbf{x}, \boldsymbol{\lambda}))] = \sum_{k=0}^{L} \alpha_e^{(k)} \mathbb{E}_{\mathbf{y}}[\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})] \tag{28}$$

On the other hand, for all $v \in V$ and $i \in \mathcal{C}$:

$$\begin{aligned} \frac{\partial G(\mathbf{y})}{\partial y_{vi}} &\overset{(24)}{=} \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})|x_{vi} = 1] - \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})|x_{vi} = 0] \\ &\overset{(18a)}{=} \mathbb{E}_{\mathbf{y}}[C(\mathbf{x})|x_{vi} = 0] - \mathbb{E}_{\mathbf{y}}[C(\mathbf{x})|x_{vi} = 1] \\ &\overset{(11a),(28)}{\approx} \sum_{e \in E} \sum_{k=1}^{L} \alpha_e^{(k)} \Big(\mathbb{E}_{\mathbf{y}}[\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})|x_{vi} = 0] \\ &\quad - \mathbb{E}_{\mathbf{y}}[\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})|x_{vi} = 1] \Big), \end{aligned} \tag{29}$$

where the error of the approximation is given by

$$\frac{1}{(L+1)!} \sum_{e \in E} C_e^{(L+1)}(\rho') \Big[\mathbb{E}_{\mathbf{y}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}|x_{vi} = 0] - \mathbb{E}_{\mathbf{y}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}|x_{vi} = 1] \Big]$$

The lemma thus follows from Lemmas 4 and 5. □

Estimator (27) is *deterministic*: no random sampling is required. Moreover, Taylor's theorem allows us to characterize the error (i.e., the *bias*) of this estimate. We use this to characterize the final fractional solution $\mathbf{y}$ produced by Alg. 2:

Theorem 2: Assume that all C_e, $e \in E$, satisfy Assumption 1, are $L+1$-differentiable, and that all their $L+1$ derivatives are bounded by $W \geq 0$. Then, consider Alg. 2, in which $\nabla G(\mathbf{y}_k)$ is estimated via the Taylor estimator (27)*, where each edge cost function is approximated at $\rho_e^* = \mathbb{E}_{\mathbf{y}_k}[\rho_e(\mathbf{x}, \boldsymbol{\lambda})] = \rho_e(\mathbf{y}_k, \boldsymbol{\lambda})$. Then,*

$$G(\mathbf{y}_K) \geq (1 - \tfrac{1}{e}) G(\mathbf{y}^*) - 2DB - \tfrac{P}{2K}, \tag{30}$$

where $K = \frac{1}{\gamma}$ is the number of iterations, $\mathbf{y}^*$ is an optimal solution to (23), $D = \max_{\mathbf{y} \in \tilde{\mathcal{D}}} \|\mathbf{y}\|_2 \leq |V| \cdot \max_{v \in V} c_v$, is the diameter of $\tilde{\mathcal{D}}$, $B \leq \frac{W|E|}{(L+1)!}$ is the bias of the estimator (27)*, and $P = 2C(\mathbf{x}_0)(|\mathcal{C}||V|)^2$, is a Lipschitz constant of ∇G.*

The proof can be found in Appendix VIII-B. The theorem immediately implies that we can replace (27) as an estimator in Alg. 2, and attain an approximation arbitrarily close to $1 - 1/e$. Note that the computational complexity of the estimator depends on the number of terms in the W-DNF form; this, in turn, depends on L. In practice, as shown in Section VII, this estimator significantly outperforms sampling in both execution time and caching gain attained.

Estimation via Power Series: For arbitrary $L+1$-differentiable cost functions C_e, the estimator (27) can be leveraged by replacing C_e with its Taylor expansion. In the case of queue-dependent cost functions, as described in Example 4 of Section III-B, the power-series (16) can be used instead. For example, the expected queue size (Example 1, Sec. III-B), is given by $C_e(\rho_e) = \frac{\rho_e}{1-\rho_e} = \sum_{k=1}^{\infty} \rho_e^k$. In contrast to the Taylor expansion, this power series does not depend on a point ρ_e^* around which the function C_e is approximated.

TABLE II
GRAPH TOPOLOGIES AND EXPERIMENT PARAMETERS

Graph	$\|V\|$	$\|E\|$	$\|\mathcal{C}\|$	$\|\mathcal{R}\|$	$\|Q\|$	c_v	$F_{\text{PL}}(\mathbf{x}_{\text{RND}})$	$F_{\text{UNI}}(\mathbf{x}_{\text{RND}})$
ER	100	1042	300	1K	4	3	2.75	2.98
ER-20Q	100	1042	300	1K	20	3	3.1	2.88
HC	128	896	300	1K	4	3	2.25	5.23
HC-20Q	128	896	300	1K	20	3	2.52	5.99
star	100	198	300	1K	4	3	6.08	8.3
path	4	3	2	2	1	1	1.2	1.2
dtelekom	68	546	300	1K	4	3	2.57	3.66
abilene	11	28	4	4	2	1/2	4.39	4.39
geant	22	66	10	40	4	2	19.68	17.22

VI. BEYOND M/M/1 QUEUES

As discussed in Section III, the classes of M/M/1 queues for which the supermodularity of the cost functions arises are quite broad, and include FIFO, LIFO, and processor sharing queues. In this section, we discuss how our results extend to even broader families of queuing networks. Chapter 3 of Kelly [3] provides a general framework for a set of queues for which service times are exponentially distributed. A large class of networks can be modeled by this framework, including networks of M/M/k queues; all such networks maintain the property that steady-state distributions have a product form. This allows us to extend our results to M/M/k queues for two cost functions C_e:

Lemma 7: For a network of M/M/k queues, both the queuing probability[2] *and the expected queue size are non-increasing and supermodular over sets* $\{\texttt{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

The proof can be found in Appendix VIII-C. We note that, as an immediate consequence of Lemma 7 and Little's theorem, both the sum of the expected delays per queue, but also the expected delay of an arriving packet, are also supermodular and non-decreasing.

Product-form steady-state distributions arise also in settings where service times are not exponentially distributed. A large class of quasi-reversible queues, named *symmetric queues* exhibit this property (c.f. Section 3.3 of [3] and Chapter 10 of [5]). In the following lemma, whose proof is in Appendix VIII-D, we leverage the product form of symmetric queues to extend our results to M/D/1 symmetric queues [30]:

Lemma 8: For a network of M/D/1 symmetric queues, the expected queue size is non-increasing and supermodular over sets $\{\texttt{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

Again, Lemma 8 and Little's theorem imply that this property also extends to network delays. It is worth noting that conclusions similar to these in Lemmas 7 and 8 are not possible for all general queues with product form distributions. In particular, we prove in Appendix VIII-E the following negative result:

Lemma 9: There exists a network of M/M/1/k queues, containing a queue e, for which no strictly monotone function C_e of the load ρ_e at a queue e is non-increasing and supermodular over sets $\{\texttt{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$. *In particular, the expected size of queue e is neither monotone nor supermodular.*

VII. NUMERICAL EVALUATION

Networks: We execute Algorithms 1 and 2 over 9 network topologies, summarized in Table II. Graphs ER and ER-20Q

[2]This is given by the so-called Erlang C formula [30].

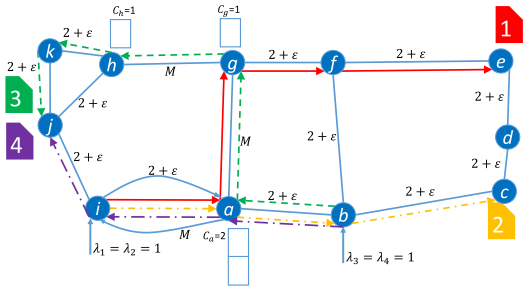


Fig. 3. The abilene topology. We consider a catalog size of $|\mathcal{C}| = 4$ and 4 requests ($|\mathcal{R}| = 4$). Requests originate from $|Q| = 2$ nodes, b and i. Three edges have a high service rate $M \gg 1$, and the rest have a low service rate $2 + \varepsilon$. Only nodes a, g, and h can cache items, and have capacities 2, 1, and 1, respectively. We set $M = 200$ and $\varepsilon = 0.05$ in our experiments. Greedy is 0.5-approximate in this instance.

are the same 100-node Erdős-Rényi graph with parameter $p = 0.1$. Graphs HC and HC-20Q are the same hypercube graph with 128 nodes, and graph star is a star graph with 100 nodes. The graph path is the topology shown in Fig. 2. The last 3 topologies, namely, dtelekom, geant, and abilene represent the Deutsche Telekom, GEANT, and Abilene backbone networks, respectively. The latter is also shown in Fig. 3.

Experimental Setup: For path and abilene, we set demands, storage capacities, and service rates as illustrated in Figures 2 and 3, respectively. Both of these settings induce an approximation ratio close to $1/2$ for greedy. For all remaining topologies, we consider a catalog of size $|\mathcal{C}|$ objects; for each object, we select 1 node uniformly at random (u.a.r.) from V to serve as the designated server for this object. To induce traffic overlaps, we also select $|Q|$ nodes u.a.r. that serve as sources for requests; all requests originate from these sources. All caches are set to the same storage capacity, i.e., $c_v = c$ for all $v \in V$.

We generate a set of $|\mathcal{R}|$ possible types of requests. For each request type $r \in \mathcal{R}$, $\lambda^r = 1$ request per second, and path p^r is generated by selecting a source among the $|Q|$ sources u.a.r., and routing towards the designated server of object i^r using a shortest path algorithm. We consider two ways of selecting objects $i^r \in \mathcal{C}$: in the *uniform* regime, i^r is selected u.a.r. from the catalog $\mathcal{C}$; in the *power-law* regime, i^r is selected from the catalog $\mathcal{C}$ via a power law distribution with exponent 1.2. All the parameter values, e.g., catalog size $|\mathcal{C}|$, number of requests $|\mathcal{R}|$, number of query sources $|Q|$, and caching capacities c_v are presented in Table II. We evaluate the caching gain $F(x)$ with queue size as the cost function, as defined in (12).

We construct heterogeneous service rates as follows. Every queue service rate is either set to a low value $\mu_e = \mu_{\text{low}}$ or a high value $\mu_e = \mu_{\text{high}}$, for all $e \in E$. We select μ_{low} and μ_{high} as follows. Given the demands $r \in \mathcal{R}$ and the corresponding arrival rates λ^r, we compute the highest load under no caching ($\mathbf{x} = \mathbf{0}$), i.e., we find $\lambda_{\max} = \max_{e \in E} \sum_{r: e \in p^r} \lambda^r$. We then set $\mu_{\text{low}} = \lambda_{\max} \times 1.05$ and $\mu_{\text{high}} = \lambda_{\max} \times 200$. We set the service rate to μ_{low} for all congested edges, i.e., edges e s.t. $\lambda_e = \lambda_{\max}$. We set the service rate for each remaining edge $e \in E$ to μ_{low} independently with probability 0.7, and to μ_{high} otherwise. Note that, as a result $\mathbf{0} \in \mathcal{D}_{\boldsymbol{\lambda}} = \mathcal{D}$, i.e., the system is stable even in the absence of caching and, on average, 30 percent of the edges have a high service rate.

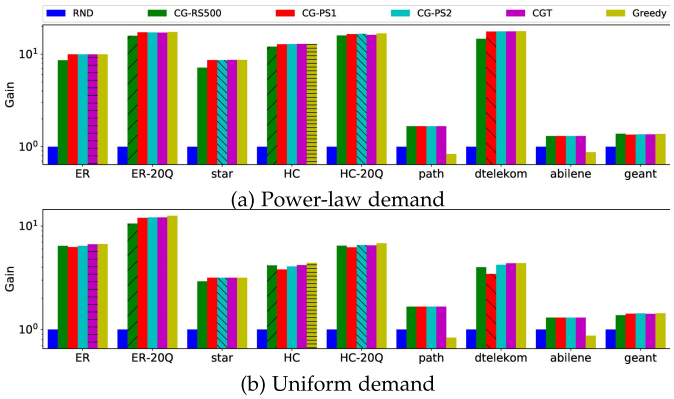


Fig. 4. Caching gains for different topologies and different arrival distributions, normalized by the gains corresponding to RND, reported in Table. II. Greedy performs comparatively well. However, it attains sub-optimal solutions for path and abilene; these solutions are worse than RND. CG-RS500 has a poor performance compared to other variations of the continuous-greedy algorithm.

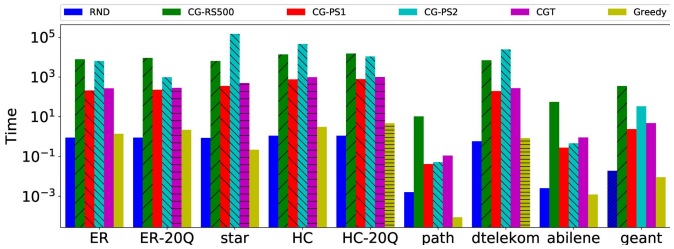
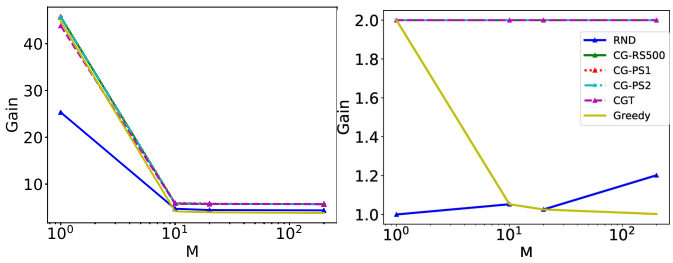


Fig. 5. Running time for different topologies and power-law arrival distribution, in seconds. CG-RS500 is slower than power series estimation CG-PS1 and CGT, sometimes exceeding CG-PS2 as well.

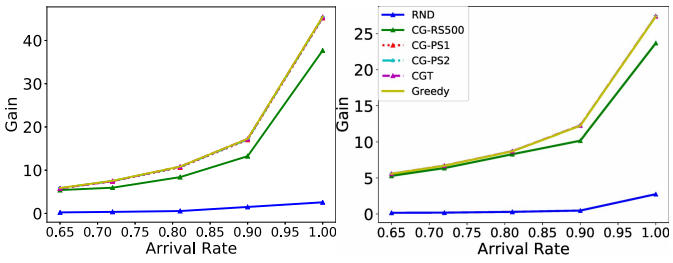
Placement Algorithms: We implement several placement algorithms: (a) *Greedy*, i.e., the greedy algorithm (Alg. 1), (b) *Continuous-Greedy with Random Sampling* (CG-RS), i.e., Algorithm 2 with a gradient estimator based on sampling, as described in Sec. V-A, (c) *Continuous-Greedy with Taylor approximation* (CGT), i.e., Algorithm 2 with a gradient estimator based on the Taylor expansion, as described in Sec. V-B, and (d) *Continuous-Greedy with Power Series approximation* (CG-PS), i.e., Algorithm 2 with a gradient estimator based on the power series expansion, described also in Sec. V-B. In the case of CG-RS, we collect 500 samples, i.e., $T = 500$. In the case of CG-PS we tried the first and second order expansions of the power series as CG-PS1 and CG-PS2, respectively. In the case of CGT, we tried the first-order expansion ($L = 1$). In both cases, subsequent to the execution of Alg. 2 we produce an integral solution in $\mathcal{D}$ by rounding via the swap rounding method [32]. All continuous-greedy algorithms use $\gamma = 0.001$. We also implement a random selection algorithm (RND), which caches c_v items at each node $v \in V$, selected uniformly at random, from the catalog $\mathcal{C}$. We repeat RND 10 times, and report the average running time and caching gain.

Caching Gain Across Different Topologies: The caching gain $F(\mathbf{x})$ for $\mathbf{x}$ generated by different placement algorithms, is shown for power-law arrival distribution and uniform arrival distribution in Figures 4a and 4b, respectively. The values are normalized by the gains obtained by RND, reported in Table II. Also, the running times of the algorithms for power-law arrival distribution are reported in Fig. 5. As we



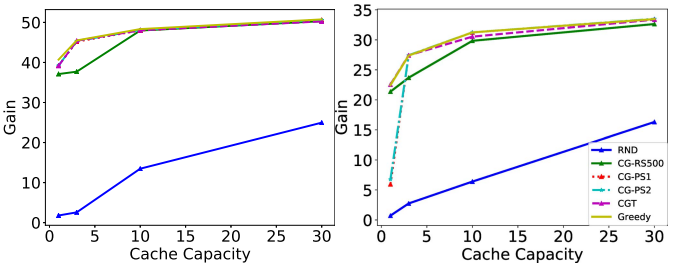
(a) abilene (b) path

Fig. 6. Caching gain vs. M. As the discrepancy between the service rate of low-bandwidth and high-bandwidth links increases, the performance of Greedy deteriorates.



(a) dtelekom (b) ER

Fig. 7. Caching gain vs. arrival rate. As the arrival rate increases caching gains get larger.



(a) dtelekom (b) ER

Fig. 8. Caching gain vs. cache capacity. As caching capacities increase, caching gains rise.

see in Fig. 4, Greedy is comparable to other algorithms in most topologies. However, for topologies path and abilene Greedy obtains a sub-optimal solution, in comparison to the continuous-greedy algorithm. In fact, for path and abilene Greedy performs even worse than RND. This is precisely because it comes with a worse guarantee $(1/2)$ over these topologies. Note that the 500 samples in CG-RS500 are significantly lower than the value, stated in Theorem V-A, needed to attain the theoretical guarantees of the continuous-greedy algorithm. This is quadratic in $|V||\mathcal{C}|$ ($\sim 10^8$ for, e.g., ER). Because of this, in Fig. 4, we see that the continuous-greedy algorithms with gradient estimators based on Taylor and Power series expansion, i.e., CG-PS1, CG-PS2, and CGT outperform CG-RS500 in most topologies. Despite this, from Fig. 5, we see that CG-RS500 runs 100 times slower than the continuous-greedy algorithms with first-order gradient estimators, i.e., CG-PS1 and CGT.

Varying Service Rates: For topologies path and abilene, the approximation ratio of Greedy is ≈ 0.5. This ratio is a function of service rate of the high-bandwidth link M. In this experiment, we explore the effect of varying M on the performance of the algorithms in more detail. We plot the caching gain obtained by different algorithms for path and abilene topologies, using different values of $M \in \{M_{\min}, 10, 20, 200\}$, where $M_{\min}$ is the value that puts the system on the brink of instability, i.e., 1 and $2+\epsilon$ for path and abilene, respectively. Thus, we gradually increase the discrepancy between the service rate of low-bandwidth and high-bandwidth links. The corresponding caching gains are plotted in Fig. 6, as a function of M. We see that as M increases the gain attained by Greedy worsens in both topologies: when $M = M_{\min}$ Greedy matches the performance of the continuous-greedy algorithms, in both cases. However, for higher values of M it is beaten not only by all variations of the continuous-greedy algorithm, but by RND as well.

Effect of Congestion on Caching Gain: In this experiment, we study the effect of varying arrival rates λ^r on caching gain F. We report results only for the dtelekom and ER topologies and power-law arrival distribution. We obtain the cache placements $\mathbf{x}$ using the parameters presented in Table II and different arrival rates: $\lambda^r \in \{0.65, 0.72, 0.81, 0.9, 1.0\}$, for $r \in \mathcal{R}$. Fig. 7 shows the caching gain attained by the placement algorithms as a function of arrival rates. We observe that as we increase the arrival rates, the caching gain attained by almost all algorithms, except RND, increases significantly. Moreover, CG-PS1, CG-PS2, CGT, and Greedy have a similar performance, while CG-RS500 achieves lower caching gains.

Varying Caching Capacity: In this experiment, we study the effect of increasing cache capacity c_v on the acquired caching gains. Again, we report the results only for the dtelekom and ER topologies and power-law arrival distribution. We evaluate the caching gain obtained by different placement algorithms using the parameters of Table II and different caching capacities: $c_v \in \{1, 3, 10, 30\}$ for $v \in V$. The caching gain is plotted in Fig. 8. As we see, in all cases the obtained gain increases, as we increase the caching capacities. This is expected: caching more items reduces traffic and delay, increasing the gain.

VIII. Conclusions

Our analysis suggests feasible object placements targeting many design objectives of interest, including system size and delay, can be determined using combinatorial techniques. Our work leaves the exact characterization of approximable objectives for certain classes of queues, including M/M/1/k queues, open. Our work also leaves open problems relating to stability. This includes the characterization of the stability region of arrival rates $\Lambda = \cup_{\mathbf{x}\in\mathcal{D}}\Lambda(\mathbf{x})$. It is not clear whether determining membership in this set (or, equivalently, given λ, determining whether there exists a $\mathbf{x} \in \mathcal{D}$ under which the system is stable) is NP-hard or not, and whether this region can be somehow approximated. Finally, all algorithms presented in this paper are offline: identifying how to determine placements in an online, distributed fashion, in a manner that attains a design objective (as in [9], [21]), or even stabilizes the system (as in [8]), remains an important open problem.

Appendix A
Rounding

Several poly-time algorithms can be used to round the fractional solution that is produced by Alg. 2 to an integral $\mathbf{x} \in \mathcal{D}$. We briefly review two such rounding algorithms: pipage rounding [20], which is deterministic, and

swap-rounding [32], which is randomized. For a more rigorous treatment, we refer the reader to [9], [20] for pipage rounding, and [32] for swap rounding.

Pipage rounding uses the following property of G: given a fractional solution $\mathbf{y} \in \tilde{\mathcal{D}}$, there are at least two fractional variables y_{vi} and $y_{v'i'}$, such that transferring mass from one to the other, 1) makes at least one of them 0 or 1, 2) the new $\hat{\mathbf{y}}$ remains feasible in $\tilde{\mathcal{D}}$, and 3) $G(\hat{\mathbf{y}}) \geq G(\mathbf{y}(1))$, that is, the expected caching gain at $\hat{\mathbf{y}}$ is at least as good as $\mathbf{y}$. This process is repeated until $\hat{\mathbf{y}}$ does not have any fractional element, at which point pipage rounding terminates and return $\hat{\mathbf{y}}$. This procedure has a run-time of $O(|V||\mathcal{C}|)$ [9], and since (a) the starting solution $\mathbf{y}$ is such that

$$G(\mathbf{y}) \geq (1 - \frac{1}{e})G(\mathbf{y}^*),$$

where $\mathbf{y}^*$ is an optimizer of G in $\tilde{\mathcal{D}}$, and (b) each rounding step can only increase G, it follows that the final integral $\hat{\mathbf{y}} \in \mathcal{D}$ must satisfy

$$F(\hat{\mathbf{y}}) = G(\hat{\mathbf{y}}) \geq G(\mathbf{y}) \geq (1 - \frac{1}{e})G(\mathbf{y}^*) \geq (1 - \frac{1}{e})F(\mathbf{x}^*),$$

where $\mathbf{x}^*$ is an optimal solution to MAXCG. Here, the first equality holds because F and G are equal when their arguments are integral, while the last inequality holds because (23) is a relaxation of MAXCG, maximizing the same objective over a larger domain.

In swap rounding, given a fractional solution $\mathbf{y} \in \tilde{\mathcal{D}}$ produced by Alg. 2 observe that it can be written as a convex combination of integral vectors in $\mathcal{D}$, i.e., $\mathbf{y} = \sum_{k=1}^{K} \gamma_k \mathbf{m}_k$, where $\gamma_k \in [0,1], \sum_{k=1}^{K} \gamma_k = 1$, and $\mathbf{m}_k \in \mathcal{D}$. Moreover, by construction, each such vector $\mathbf{m}_k$ is maximal, in that all capacity constraints are tight.

Swap rounding iteratively merges these constituent integral vectors, producing an integral solution. At each iteration i, the present integral vector $\mathbf{c}_k$ is merged with $\mathbf{m}_{k+1} \in \mathcal{D}$ into a new integral solution $\mathbf{c}_{k+1} \in \mathcal{D}$ as follows: if the two solutions $\mathbf{c}_k$, $\mathbf{m}_{k+1}$ differ at a cache $v \in V$, items in this cache are swapped to reduce the set difference: either an item i in a cache in $\mathbf{c}_k$ replaces an item j in $\mathbf{m}_{k+1}$, or an item j in $\mathbf{m}_{k+1}$ replaces an item i in $\mathbf{c}_k$; the former occurs with probability proportional to $\sum_{\ell=1}^{k} \gamma_\ell$, and the latter with probability proportional to γ_{k+1}. The swapping is repeated until the two integer solutions become identical; this merged solution becomes $\mathbf{c}_{k+1}$. This process terminates after $K-1$ steps, after which all the points $\mathbf{m}_k$ are merged into a single integral vector $\mathbf{c}_K \in \mathcal{D}$.

Observe that, in contrast to pipage rounding, swap rounding does not require any evaluation of the objective F during rounding. This makes swap rounding significantly faster to implement; this comes at the expense of the approximation ratio, however, as the resulting guarantee $1 - 1/e$ is in expectation.

APPENDIX B
CONTINUOUS GREEDY WITH TAYLOR-EXPANSION GRADIENT ESTIMATION

A. Proof of Lemma 4

We prove this by induction on $k \geq 1$. Observe first that, by (4), the load on each edge $e = (u,v) \in E$ can be written as a polynomial of the following form:

$$\rho_e(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{r \in \mathcal{R}_e} \beta_r(\boldsymbol{\lambda}) \cdot \prod_{j \in \mathcal{I}_e(r)} (1 - x_j), \tag{31}$$

for appropriately defined

$$\mathcal{R}_e = \mathcal{R}_{(u,v)} = \{r \in \mathcal{R} : (v,u) \in p^r\},$$
$$\mathcal{I}_e(r) = \{(w, i^r) \in V \times \mathcal{C} : w \in p^r, k_{p^r}(w) \leq k_{p^r}(v)\}, \text{ and}$$
$$\beta_r(\boldsymbol{\lambda}) = \lambda^r / \mu_e.$$

In other words, $\rho_e : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$ is indeed a *W-DNF* polynomial. For the induction step, observe that W-DNF polynomials, seen as functions over the integral domain $\mathcal{D}_{\boldsymbol{\lambda}}$, are *closed under multiplication*. In particular, the following lemma holds:

Lemma 10: Given two W-DNF polynomials $f_1 : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$ and $f_2 : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$, given by

$$f_1(\mathbf{x}) = \sum_{r \in \mathcal{R}_1} \beta_r \prod_{t \in \mathcal{I}_1(r)} (1 - x_t), \quad \textit{and}$$
$$f_2(\mathbf{x}) = \sum_{r \in \mathcal{R}_2} \beta_r \prod_{t \in \mathcal{I}_2(r)} (1 - x_t),$$

their product $f_1 \cdot f_2$ is also a W-DNF polynomial over $\mathcal{D}_{\boldsymbol{\lambda}}$, given by:

$$(f_1 \cdot f_2)(\mathbf{x}) = \sum_{(r,r') \in \mathcal{R}_1 \times \mathcal{R}_2} \beta_r \beta'_r \prod_{t \in \mathcal{I}_1(r) \cup \mathcal{I}_2(r')} (1 - x_t)$$

Proof: To see this, observe that

$$f_1(\mathbf{x}) f_1(\mathbf{x}) = \sum_{(r,r') \in \mathcal{R}_1 \times \mathcal{R}_2} \beta_r \beta'_r \prod_{t \in \mathcal{I}_1(r) \cap \mathcal{I}_2(r')} (1 - x_t)^2 \times \prod_{t \in \mathcal{I}_1(r) \triangle \mathcal{I}_2(r')} (1 - x_t)$$

where $\triangle$ is the symmetric set difference. On the other hand, as $(1 - x_t) \in \{0, 1\}$, we have that $(1 - x_t)^2 = (1 - x_t)$, and the lemma follows. □

Hence, if $\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})$ is a W-DNF polynomial, by (31) and Lemma 10, so is $\rho_e^{k+1}(\mathbf{x}, \boldsymbol{\lambda})$.□

B. Proof of Theorem 2

We begin by bounding the bias of estimator (29). Indeed, given a set of continuous functions $\{C_{(u,v)}\}_{(u,v) \in E}$ where their first $L+1$ derivatives within their operating regime, $[0, 1)$, are upperbounded by a finite constant, W, the bias of estimator $\mathbf{z} \equiv [z_{vi}]_{v \in V, i \in \mathcal{C}}$, where z_{vi} is defined by (27), is given by

$$B \equiv ||\mathbf{z} - \nabla G(\mathbf{y})||_2 = ||\sum_{e \in E} \frac{1}{(L+1)!} C_e^{(L+1)}(\rho'_e)(\rho_e - \rho_e^*)^{L+1}||_2, \tag{32}$$

where $\rho'_e \in [\rho_e^*, \rho_e]$. To compute the bias, we note that $\rho_e, \rho_e^* \in [0, 1]$. Specifically, we assume $\rho_e, \rho_e^* \in [0, 1)$. Hence, $|\rho_e - \rho_e^*| \leq 1$, and $C_e^{(L+1)}(\rho'_e) \leq \max\{C_e^{(L+1)}(\rho_e), C_e^{(L+1)}(\rho_e^*)\} < \infty$. In particular, let $W = \max_{e \in E} C_e^{(L+1)}(\rho'_e)$. Then, it is easy to compute the following upper bound on the bias of $\mathbf{z}$:

$$B \leq \frac{W|E|}{(L+1)!}. \tag{33}$$

In addition, note that G is linear in y_{vi}, and hence [1]:

$$\frac{\partial G}{\partial y_{vi}} = \mathbb{E}[F(\mathbf{x})|x_{vi}=1] - \mathbb{E}[F(\mathbf{x})|x_{vi}=0] = \mathbb{E}[C(\mathbf{x})|x_{vi}=0] - \mathbb{E}[C(\mathbf{x})|x_{vi}=1] \geq 0, \quad (34)$$

which is ≥ 0 due to monotonicity of $F(\mathbf{x})$. It is easy to verify that $\frac{\partial^2 G}{\partial y_{vi}^2} = 0$. For $(v_1,i_1) \neq (v_2,i_2)$, we can compute the second derivative of G [1] as given by

$$\begin{aligned}\frac{\partial^2 G}{\partial y_{v_1 i_1} \partial y_{v_2 i_2}} &= \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1}=1, x_{v_2 i_2}=0] \\ &\quad + \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1}=0, x_{v_2 i_2}=1] \\ &\quad - \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1}=1, x_{v_2 i_2}=1] \\ &\quad - \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1}=0, x_{v_2 i_2}=0] \leq 0,\end{aligned}$$

which is ≤ 0 due to the supermodularity of $C(\mathbf{x})$. Hence, $G(\mathbf{y})$ is component-wise concave [1].

In addition, it is easy to see that for $\mathbf{y} \in \tilde{\mathcal{D}}$, $|G(\mathbf{y})|$, $||\nabla G(\mathbf{y})||$, and $||\nabla^2 G(\mathbf{y})||$ are bounded by $C(\mathbf{x}_0)$, $C(\mathbf{x}_0)|\mathcal{C}||V|$, and $2C(\mathbf{x}_0)(|\mathcal{C}||V|)^2$, respectively. Consequently, ∇G is P-Lipschitz continuous, with $P = 2C(\mathbf{x}_0)(|\mathcal{C}||V|)^2$.

In the kth iteration of the Continuous Greedy algorithm, let $\mathbf{m}^* = \mathbf{m}^*(\mathbf{y}_k) := (\mathbf{y}^* \vee (\mathbf{y}_k + \mathbf{y}_0)) - \mathbf{y}_k = (\mathbf{y}^* - \mathbf{y}_k) \vee \mathbf{y}_0 \geq \mathbf{y}_0$, where $x \vee y := (\max\{x_i, y_i\})_i$. Since $\mathbf{m}^* \leq \mathbf{y}^*$ and $\mathcal{D}$ is closed-down, $\mathbf{m}^* \in \mathcal{D}$. Due to monotonicity of G, it follows

$$G(\mathbf{y}_k + \mathbf{m}^*) \geq G(\mathbf{y}^*). \quad (35)$$

We introduce univariate auxiliary function $g_{\mathbf{y},\mathbf{m}}(\xi) := G(\mathbf{y} + \xi\mathbf{m}), \xi \in [0,1], \mathbf{m} \in \tilde{\mathcal{D}}$. Since $G(\mathbf{y})$ is component-wise concave, then, $g_{\mathbf{y},\mathbf{m}}(\xi)$ is concave in $[0,1]$. In addition, since $g_{\mathbf{y}_k,\mathbf{m}^*}(\xi) = G(\mathbf{y}_k + \xi\mathbf{m}^*)$ is concave for $\xi \in [0,1]$, it follows

$$\begin{aligned} &g_{\mathbf{y}_k,\mathbf{m}^*}(1) - g_{\mathbf{y}_k,\mathbf{m}^*}(0) \\ &\quad = G(\mathbf{y}_k + \mathbf{m}^*) - G(\mathbf{y}_k) \\ &\quad \leq \frac{dg_{\mathbf{y}_k,\mathbf{m}}(0)}{d\xi} \times 1 = \langle \mathbf{m}^*, \nabla G(\mathbf{y}_k)\rangle. \end{aligned} \quad (36)$$

Now let $\mathbf{m}_k$ be the vector chosen by Algorithm 2 in the kth iteration. We have

$$\langle \mathbf{m}_k, \mathbf{z}(\mathbf{y}_k)\rangle \geq \langle \mathbf{m}^*, \mathbf{z}(\mathbf{y}_k)\rangle. \quad (37)$$

For the LHS, we have

$$\begin{aligned}\langle \mathbf{m}_k, \mathbf{z}\rangle &= \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k)\rangle + \langle \mathbf{m}_k, \mathbf{z} - \nabla G(\mathbf{y}_k)\rangle \\ &\overset{(i)}{\leq} \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k)\rangle + ||m_k||_2 \cdot |\mathbf{z} - \nabla G(\mathbf{y}_k)||2 \\ &\leq \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k)\rangle + DB. \end{aligned} \quad (38)$$

where $D = \max_{\mathbf{m}\in\tilde{\mathcal{D}}} \|\mathbf{m}\|_2 \leq |V| \cdot \max_{v\in V} c_v$, is the upperbound on the diameter of $\tilde{\mathcal{D}}$, B is as defined in (33), and (i) follows from Cauchy-Schwarz inequality. Similarly, we have for the RHS of that (37)

$$\langle \mathbf{m}^*, \mathbf{z}(\mathbf{y}_k)\rangle \geq \langle \mathbf{m}^*, \nabla G(\mathbf{y}_k)\rangle - DB. \quad (39)$$

It follows

$$\begin{aligned}\langle \mathbf{m}_k, \nabla G(\mathbf{y}_k)\rangle + 2DB &\geq \langle \mathbf{m}^*, \nabla G(\mathbf{y}_k)\rangle \\ &\overset{(a)}{\geq} G(\mathbf{y}_k + \mathbf{m}^*) - G(\mathbf{y}_k) \overset{(b)}{\geq} G(\mathbf{y}^*) - G(\mathbf{y}_k), \end{aligned} \quad (40)$$

where (a) follows from (36), and (b) follows from (35).

Using the P-Lipschitz continuity property of $\frac{dg_{\mathbf{y}_k,\mathbf{m}_k}(\xi)}{d\xi}$ (due to P-Lipschitz continuity of ∇G), it is straightforward to see that

$$\begin{aligned}-\frac{P\gamma_k^2}{2} &\leq g_{\mathbf{y}_k,\mathbf{m}_k}(\gamma_k) - g_{\mathbf{y}_k,\mathbf{m}_k}(0) - \gamma_k \cdot \frac{dg_{\mathbf{y}_k,\mathbf{m}_k}(0)}{d\xi} \\ &= G(\mathbf{y}_k + \gamma_k\mathbf{m}_k) - G(\mathbf{y}_k) - \gamma_k < \mathbf{m}_k, \nabla G(\mathbf{y}_k) >, \end{aligned} \quad (41)$$

hence,

$$\begin{aligned} &G(\mathbf{y}_{k+1}) - G(\mathbf{y}_k) \\ &\quad \geq \gamma_k \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k)\rangle - \frac{P\gamma_k^2}{2} \\ &\quad \geq \gamma_k \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k)\rangle - \frac{P\gamma_k^2}{2} \\ &\quad \overset{(c)}{\geq} \gamma_k (G(\mathbf{y}^*) - G(\mathbf{y}_k)) - 2\gamma_k DB - \frac{P\gamma_k^2}{2}, \end{aligned} \quad (42)$$

where (c) follows from (40), respectively. By rearranging the terms and letting $k = K-1$, we have

$$\begin{aligned} &G(\mathbf{y}_K) - G(\mathbf{y}^*) \\ &\geq \prod_{j=0}^{K-1}(1-\gamma_j)(G(\mathbf{y}_0) - G(\mathbf{y}^*)) \\ &\quad - 2DB\sum_{j=0}^{K-1}\gamma_j - \frac{P}{2}\sum_{j=0}^{K-1}\gamma_j^2 \\ &\overset{(e)}{\geq} (G(\mathbf{y}_0) - G(\mathbf{y}^*))\exp\{-\sum_{j=0}^{K-1}\gamma_j\} - 2DB\sum_{j=0}^{K-1}\gamma_j - \frac{P}{2}\sum_{j=0}^{K-1}\gamma_j^2, \end{aligned}$$

where (e) is true since $1 - x \leq e^{-x}, \forall x \geq 0$, and $G(\mathbf{y}_0) \leq G(\mathbf{y}^*)$ holds due to the greedy nature of Algorithm 2 and monotonicity of G. In addition, Algorithm 2 ensures $\sum_{j=0}^{K-1}\gamma_j = 1$. It follows

$$G(\mathbf{y}_K) - (1 - \frac{1}{e})G(\mathbf{y}^*) \geq e^{-1}G(\mathbf{y}_0) - 2DB - \frac{P}{2}\sum_{j=0}^{K-1}\gamma_j^2. \quad (43)$$

This result holds for general stepsizes $0 < \gamma_j \leq 1$. The RHS of (43) is indeed maximized when $\gamma_j = \frac{1}{K}$, which is the assumed case in Algorithm 2. In addition, we have $\mathbf{y}_0 = \mathbf{0}$, and hence, $G(\mathbf{y}_0) = 0$. Therefore, we have

$$G(\mathbf{y}_K) - (1 - \frac{1}{e})G(\mathbf{y}^*) \geq -2DB - \frac{P}{2K}. \quad (44)$$

APPENDIX C
BEYOND $M/M/1$ QUEUES

C. Proof of Lemma 7

For an arbitrary network of M/M/k queues, the traffic load on queue $(u,v) \in E$ is given as

$$a_{(u,v)}(\mathbf{x}) = \frac{\sum_{r\in\mathcal{R}:(v,u)\in p^r} \lambda^r \prod_{k'=1}^{k_{p^r}(v)} (1 - x_{p^r_{k'} i^r})}{k\mu_{(u,v)}}, \quad (45)$$

which is similar to that of M/M/1 queues, but normalized by the number of servers, k. Hence, $a_{(u,v)}(\mathbf{x})$ is submodular in $\mathbf{x}$.

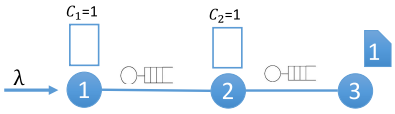


Fig. 9. A simple network with finite-capacity queues.

TABLE III
RESULTS OF $\rho_{u,v}(\mathbf{x})$'S FOR DIFFERENT CACHING CONFIGURATIONS

$[x_{11}, x_{21}]$	$\rho_{3,2}$	$\rho_{2,1}$
$[0,0]$	$\frac{\lambda}{\mu_{3,2}}$	$\frac{\lambda(1-p^L_{3,2})}{\mu_{2,1}}$
$[1,0]$	0	0
$[0,1]$	0	$\frac{\lambda}{\mu_{2,1}}$
$[1,1]$	0	0

For an M/M/k queue, the probability that an arriving packet finds all servers busy and will be forced to wait in queue is given by Erlang C formula [30], which follows

$$P^Q_{(u,v)}(\mathbf{x}) = \frac{b_{(u,v)}(\mathbf{x})(k a_{(u,v)}(\mathbf{x}))^k}{k!(1-a_{(u,v)}(\mathbf{x}))}, \tag{46}$$

where

$$b_{(u,v)}(\mathbf{x}) = \left[\sum_{n=0}^{k-1} \frac{(k a_{(u,v)}(\mathbf{x}))^n}{n!} + \frac{(k a_{(u,v)}(\mathbf{x}))^k}{k!(1-a_{(u,v)}(\mathbf{x}))}\right]^{-1}, \tag{47}$$

is the normalizing factor. In addition, the expected number of packets waiting for or under transmission is given by

$$\mathbb{E}[n_{(u,v)}(\mathbf{x})] = k a_{(u,v)}(\mathbf{x}) + \frac{a_{(u,v)}(\mathbf{x}) P^Q_{(u,v)}(\mathbf{x})}{1-a_{(u,v)}(\mathbf{x})}. \tag{48}$$

Lee and Cohen [33] show that $P^Q_{(u,v)}(\mathbf{x})$ and $\mathbb{E}[n_{(u,v)}(\mathbf{x})]$ are strictly increasing and convex in $a_{(u,v)}(\mathbf{x})$, for $a_{(u,v)}(\mathbf{x}) \in [0,1)$. In addition, a more direct proof of convexity of $\mathbb{E}[n_{(u,v)}(\mathbf{x})]$ was shown by Grassmann in [34]. Hence, Both $P(\mathbf{x}) := \sum_{(u,v)\in E} P^Q_{(u,v)}(\mathbf{x})$ and $N(\mathbf{x}) := \sum_{(u,v)\in E} \mathbb{E}[n_{(u,v)}(\mathbf{x})]$ are increasing and convex. Due to Theorem 1, we note that both functions are non-increasing and supermodular in $\mathbf{x}$, and the proof is complete.

D. Proof of Lemma 8

Let $\rho_{(u,v)}(\mathbf{x})$ be the traffic load on queue $(u,v) \in E$, as defined by (4). It can be shown that the average number of packets in queue $(u,v) \in E$ is of form [30]

$$\mathbb{E}[n_{(u,v)}(\mathbf{x})] = \rho_{(u,v)}(\mathbf{x}) + \frac{\rho^2_{(u,v)}(\mathbf{x})}{2(1-\rho_{(u,v)}(\mathbf{x}))}. \tag{49}$$

It is easy to see that this function is strictly increasing and convex in $\rho_{(u,v)}(\mathbf{x})$ for $\rho_{(u,v)}(\mathbf{x}) \in [0,1)$. Due to Theorem 1, $N(\mathbf{x}) := \sum_{(u,v)\in E} \mathbb{E}[n_{(u,v)}(\mathbf{x})]$ is non-increasing and supermodular in $\mathbf{x}$, and the proof is complete.

E. Proof of Lemma 9

Consider the network of $M/M/1/k$ queues in Fig. 9, where node 1 is requesting content 1 from node 3, according to a Poisson process with rate λ. For simplicity, we only consider the traffic for content 1. For queues $(2,1)$ and $(3,2)$, it is easy to verify that the probability of packet drop at queues $(u,v) \in \{(2,1),(3,2)\}$ is given by

$$p^L_{(u,v)}(\rho_{(u,v)}) = \frac{\rho_{u,v}(\mathbf{x})^k(1-\rho_{(u,v)}(\mathbf{x}))}{1-\rho_{(u,v)}(\mathbf{x})^{k+1}}, \tag{50}$$

where $\rho_{(u,v)}(\mathbf{x})$ is the traffic load on queue (u,v), and it can be computed for$(2,1)$ and $(3,2)$ as follows:

$$\rho_{(2,1)}(x_{11},x_{21}) = \frac{\lambda(1-x_{11})(1-p^L_{(3,2)})}{\mu_{(2,1)}}, \tag{51}$$

$$\rho_{(3,2)}(x_{11},x_{21}) = \frac{\lambda(1-x_{11})(1-x_{21})}{\mu_{(3,2)}}. \tag{52}$$

Using the results reported in Table III, it is easy to verify that ρ's are not monotone in $\mathbf{x}$. Hence, no strictly monotone function of ρ's are monotone in $\mathbf{x}$. In addition, it can be verified that ρ's are neither submodular, nor supermodular in $\mathbf{x}$. To show this, let sets $A = \emptyset$, and $B = \{(1,1)\}$, correspond to caching configurations $[0,0]$ and $[1,0]$, respectively. Note that $A \subset B$, and $(2,1) \notin B$. Since $\rho_{(3,2)}(A \cup \{(2,1)\}) - \rho_{(3,2)}(A) = -\frac{\lambda}{\mu_{(3,2)}} \not\geq 0 = \rho_{(3,2)}(B \cup \{(2,1)\}) - \rho_{(3,2)}(B)$, then $\rho_{(3,2)}$ is not submodular. Consequently, no strictly monotone function of $\rho_{(3,2)}$ is submodular. Similarly, as $\rho_{(2,1)}(A \cup \{(2,1)\}) - \rho_{(2,1)}(A) = \frac{\lambda p^L_{(3,2)}}{\mu_{(2,1)}} \not\leq 0 = \rho_{(2,1)}(B \cup \{(2,1)\}) - \rho_{(2,1)}(B)$, $\rho_{(2,1)}$ is not supermodular. Thus, no strictly monotone function of $\rho_{(2,1)}$ is supermodular.

REFERENCES

[1] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a monotone submodular function subject to a matroid constraint," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1740–1766, Jan. 2011.

[2] M. Mahdian, A. Moharrer, S. Ioannidis, and E. Yeh, "Kelly cache networks," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Apr. 2019, pp. 217–225.

[3] F. P. Kelly, *Reversibility and Stochastic Networks*. Cambridge, U.K.: Cambridge Univ. Press, 2011.

[4] R. G. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[5] R. Nelson, *Probability, Stochastic Processes, and Queueing Theory: The Mathematics of Computer Performance Modeling*, 1st ed. Berlin, Germany: Springer, 2010.

[6] H. Chen and D. D. Yao, *Fundamentals of Queueing Networks: Performance, Asymptotics, and Optimization*, vol. 46. New York, NY, USA: Springer, 2013.

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2009, pp. 1–12.

[8] 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 *Proc. 1st Int. Conf. Inf.-Centric Netw. (INC)*, 2014, pp. 117–126.

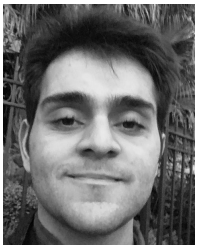
[9] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 113–124, Jun. 2016.

[10] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[11] M. Dehghan *et al.*, "On the complexity of optimal routing and content caching in heterogeneous networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 936–944.

[12] N. Laoutaiis, S. Syntila, and L. Stavrakakis, "Meta algorithms for hierarchical Web caches," in *Proc. IEEE Int. Conf. Perform., Comput., Commun.*, Apr. 2004, pp. 445–452.
[13] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: Modeling, design and experimental results," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 7, pp. 1305–1314, Sep. 2002.
[14] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 6, pp. 505–519, Jun. 2004.
[15] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.
[16] K. P. Naveen, L. Massoulie, E. Baccelli, A. C. Viana, and D. Towsley, "On the interaction between content caching and request assignment in cellular cache networks," in *Proc. 5th Workshop All Things Cellular, Oper., Appl. Challenges (AllThingsCellular)*, 2015, pp. 37–42.
[17] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation caching and routing algorithms for massive mobile data delivery," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2013, pp. 3534–3539.
[18] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. ICS*, 2002, pp. 84–95.
[19] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, p. 177, Oct. 2002.
[20] A. A. Ageev and M. I. Sviridenko, "Pipage rounding: A new method of constructing algorithms with proven performance guarantee," *J. Combinat. Optim.*, vol. 8, no. 3, pp. 307–328, Sep. 2004.
[21] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1258–1275, Jun. 2018.
[22] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM J. Comput.*, vol. 38, no. 4, pp. 1411–1429, Jan. 2008.
[23] Y. Bartal, A. Fiat, and Y. Rabani, "Competitive algorithms for distributed data management," *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 341–358, Dec. 1995.
[24] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *Proc. 17th Annu. ACM-SIAM Symp. Discrete Algorithm (SODA)*, 2006, pp. 611–620.
[25] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale VoD system," in *Proc. CoNEXT*, 2010, pp. 1–12.
[26] A. Krause and D. Golovin, "Submodular function maximization," in *Tractability: Practical Approaches to Hard Problems*. Cambridge, U.K.: Cambridge Univ. Press, Feb. 2014.
[27] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, Dec. 1978.
[28] J. Vondrak, "Optimal approximation for the submodular welfare problem in the value oracle model," in *Proc. 14th Annu. ACM Symp. Theory Comput. (STOC)*, 2008, pp. 67–74.
[29] G. L. Nemhauser and L. A. Wolsey, "Best algorithms for approximating the maximum of a submodular set function," *Math. Oper. Res.*, vol. 3, no. 3, pp. 177–188, Aug. 1978.
[30] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data Networks*, vol. 2. Upper Saddle River, NJ, USA: Prentice-Hall, 1992,
[31] G. Calinescu, R. Chekuri, M. Pál, and J. Vondrák, "Maximizing a submodular set function subject to a matroid constraint," in *Proc. IPCO*, 2007, pp. 182–196.
[32] C. Chekuri, J. Vondrak, and R. Zenklusen, "Dependent randomized rounding via exchange properties of combinatorial structures," in *Proc. IEEE 51st Annu. Symp. Found. Comput. Sci.*, Oct. 2010, pp. 575–584.
[33] H. L. Lee and M. A. Cohen, "A note on the convexity of performance measures of M/M/C queueing systems," *J. Appl. Probab.*, vol. 20, no. 4, pp. 920–923, Dec. 1983.
[34] W. Grassmann, "The convexity of the mean queue size of the M/M/C queue with respect to the traffic intensity," *J. Appl. Probab.*, vol. 20, no. 4, pp. 916–919, Dec. 1983.

Milad Mahdian received the B.S. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2012, and the M.S. and Ph.D. degrees in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2014 and 2017, respectively. He is currently the Vice President in Technology at Goldman Sachs Group, Inc., New York, NY, USA, developing the new low-latency electronic trading platform for Goldman Sachs. His main research interests are on non-linear and stochastic optimization techniques, computer networking, and adaptive algorithm development of low-latency caching systems.



Armin Moharrer received the B.Sc. degree in electrical engineering from the Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2015, and the M.Sc. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2018. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Northeastern University, under the supervision of Prof. Stratis Ioannidis. His research focuses on distributed algorithms and data mining.

Stratis Ioannidis received the B.Sc. degree in electrical and computer engineering from the National Technical University of Athens, Greece, in 2002, and the M.Sc. and Ph.D. degrees in computer science from the University of Toronto, Canada, in 2004 and 2009, respectively. Prior to joining Northeastern University, he was a Research Scientist at the Technicolor research centers, Paris, France, and Palo Alto, CA, USA, as well as at Yahoo Labs, Sunnyvale, CA. He is currently an Assistant Professor with the Electrical and Computer Engineering Department, Northeastern University, Boston, MA, USA, where he also holds a courtesy appointment with the Khoury College of Computer Sciences. He is a recipient of the NSF CAREER Award, a Google Faculty Research Award, and the Best Paper Award at the 2017 ACM Conference on Information-centric Networking (ICN) and the 2019 IEEE DySPAN Conference.

Edmund Yeh received the B.S. degree (Hons.) in electrical engineering and Phi Beta Kappa from Stanford University, in 1994, the M.Phil. degree in engineering from Cambridge University on the Winston Churchill Scholarship in 1995, and the Ph.D. degree in electrical engineering and computer science from MIT, under Prof. R. Gallager, in 2001. He was previously an Assistant and Associate Professor of electrical engineering, computer science, and statistics at Yale University. He is currently a Professor of electrical and computer engineering at Northeastern University, Boston, MA, USA. He is a 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 the 2015 IEEE International Conference on Communications (ICC) Communication Theory Symposium.