Jointly Optimal Routing and Caching for Arbitrary Network Topologies

Stratis Ioannidis
Northeastern University
Electrical and Computer Engineering
360 Huntington Avenue, 409DA
Boston, MA, USA
ioannidis@ece.neu.edu

Edmund Yeh
Northeastern University
Electrical and Computer Engineering
360 Huntington Avenue, 409DA
Boston, MA, USA
eyeh@ece.neu.edu

ABSTRACT

We study a problem of fundamental importance to ICNs, namely, minimizing routing costs by jointly optimizing caching and routing decisions over an arbitrary network topology. We consider both source routing and hop-by-hop routing settings. The respective offline problems are NP-hard. Nevertheless, we show that there exist polynomial time approximation algorithms producing solutions within a constant approximation from the optimal. We also produce distributed, adaptive algorithms with the same approximation guarantees. We simulate our adaptive algorithms over a broad array of different topologies. Our algorithms reduce routing costs by several orders of magnitude compared to prior art, including algorithms optimizing caching under fixed routing.

CCS CONCEPTS

• **Networks → Network performance analysis**;

KEYWORDS

Caching, forwarding, routing, distributed optimization

ACM Reference format:
Stratis Ioannidis and Edmund Yeh. 2017. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. In *Proceedings of ICN '17, Berlin, Germany, September 26–28, 2017,* 11 pages.
DOI: 10.1145/3125719.3125730

1 INTRODUCTION

Optimally placing resources in a network and routing requests toward them is a problem as old as the Internet itself. It is of paramount importance in information centric networks (ICNs) [28, 50], but also naturally arises in a variety of networking applications such as web-cache design [12, 33, 51], wireless/femtocell networks [37, 39, 45], and peer-to-peer networks [14, 35], to name a few. Motivated by this problem, we study a *caching network*, i.e., a network of nodes augmented with additional storage capabilities. In such a network, some nodes act as designated content servers, permanently storing content and serving as "caches of last resort". Other nodes generate requests for content that are forwarded towards these designated servers. If, however, an intermediate node in the path towards a server stores the requested content, the request is satisfied early: i.e., the request ceases to be forwarded, and a content copy is sent over the reverse path towards the request's source.

This abstract setting naturally captures ICNs. Designated servers correspond to traditional web servers permanently storing content, while nodes generating requests correspond to customer-facing gateways. Intermediate, cache-enabled nodes correspond to storage-augmented routers in the Internet's backbone: such routers forward requests but, departing from traditional network-layer protocols, immediately serve requests for content they store. An extensive body of research, both theoretical [8, 12, 22, 23, 26, 36, 41, 42] and experimental [12, 28, 33, 35, 43, 51], has focused on modeling and analyzing networks of caches in which *routing is fixed*, and requests follow predetermined paths. For example, shortest paths to the nearest designated server are often used. Given routes to be followed, and the demand for items, the above works aim to model and analyze (theoretically or empirically) the behavior of different caching algorithms deployed over intermediate nodes.

It is not a priori clear whether fixed routing and, more specifically, routing towards the nearest server is the appropriate design choice for such networks. This is of special interest in the context of ICNs, where delegating routing decisions to another protocol amounts to an "incremental" deployment. For example, in such a deployment, requests can be forwarded towards the closest designated web servers over paths determined according to, e.g., existing routing protocols such as OSPF or BGP [31]. Subsequent caching decisions by intermediate routers affect only where–within a given path–requests are satisfied. An alternative is to *jointly* optimize both routing *and* caching decisions simultaneously. Doing so however poses a significant challenge, precisely because this joint optimization is inherently combinatorial. Indeed, jointly optimizing routing and caching decisions with the objective of, e.g., minimizing routing costs, is an NP-hard problem, and constructing a distributed approximation algorithm is far from trivial [9, 21, 26, 45].

This state of affairs gives rise to the following questions. First, is it possible to design *distributed, adaptive, and tractable algorithms jointly optimizing both routing and caching decisions over arbitrary cache network topologies, with provable performance guarantees*? Identifying such algorithms is important precisely due to the combinatorial nature of the problem at hand. Second, presuming such algorithms exist, *do they yield significant performance improvements*

ICN '17, Berlin, Germany

978-1-4503-5122-5/17/09…$15.00
DOI: 10.1145/3125719.3125730

over fixed routing protocols? Answering this question in the affirmative may justify the potential increase in protocol complexity due joint optimization. It can also inform future ICN design, indicating whether full optimization is preferable, or whether an incremental approach in which routing and caching are separate suffices.

Our goal is to provide rigorous, comprehensive answers to these two questions. We make the following contributions:

- We show, by constructing a counterexample, that fixed routing (and, in particular, routing towards the nearest server) can be *arbitrarily suboptimal* compared to jointly optimizing caching and routing decisions. Intuitively, joint optimization affects routing costs drastically because *exploiting path diversity increases caching opportunities*.
- We propose a formal mathematical framework for joint routing and caching optimization. We consider both *source routing* and *hop-by-hop* routing strategies, the two predominant classes of routing protocols over the Internet [31].
- We study the offline version of the joint routing and caching optimization problem, which is NP-hard, and construct a polynomial-time $1 - 1/e$ approximation algorithm.
- We provide a *distributed*, *adaptive* algorithm that converges to joint routing and caching strategies that are, globally, within a $1 - 1/e$ approximation ratio from the optimal.
- We evaluate our distributed algorithm over 9 synthetic and 3 real-life network topologies, and show that it significantly outperforms the state of the art: it reduces routing costs by a factor between 10 and 1000, for a broad array of competitors, including both fixed and dynamic routing protocols.

The remainder of this paper is organized as follows. We review related work in Section 2, and present our mathematical model of a caching network in Section 3. Our main results are presented in Section 4. A numerical evaluation of our algorithms over several topologies is presented in Section 5, and we conclude in Section 6.

2 RELATED WORK

There are several adaptive, distributed approaches determining how to populate caches under fixed routing. A simple, elegant, and ubiquitous algorithm is *path replication* [14], sometimes also referred to as "leave-copy-everywhere" (LCE) [33]: once a request for an item reaches a cache, every downstream node receiving the response caches the item. Several variants of this principle exist. In "leave-copy-down" (LCD), a copy is placed *only* in the node immediately preceding the cache storing the requested item [32, 33], while "move-copy-down" (MCD) also removes the present upstream copy. Probabilistic variants have also been proposed [40]. To evict items, traditional eviction policies like Least Recently Used (LRU), Least Frequently Used (LFU), First In First Out (FIFO), and Random Replacement (RR) are typically used. Several works [33, 40, 43, 44, 48] have experimentally studied the performance of these protocols and variants over a broad array of topologies. Despite the advantages of simplicity and elegance inherent in path replication, when targeting an optimization objective such as, e.g., minimizing total routing costs, path replication combined with all of the above eviction and replication policies is known to be arbitrarily suboptimal [26].

There is a vast literature on the performance of eviction policies like LRU, FIFO, LFU, etc., on a single cache, and the topic is classic [2, 16, 20, 24, 30]. Nevertheless, the study of *networks of caches* still poses significant challenges. A significant breakthrough in this area has been the so-called *Che approximation* [12, 23], which postulates that the hit rate of an LRU cache can be well approximated under the assumption that items stay in the cache for a constant time. This approximation is quite accurate in practice [23], and its success motivated extensive research in so-called *time-to-live* (TTL) caches. A series of recent works have focused on identifying how to set TTLs to (a) approximate the behavior of known eviction policies, (b) describe hit-rates in closed-form formulas [8, 12, 17, 22, 36]. Despite these advances, none of the above works address issues of routing cost minimization over multiple hops, which is our goal.

In their seminal paper [14] introducing path replication, Cohen and Shenker also introduced the abstract problem of finding a content placement that minimizes routing costs. The authors show that path replication combined with a constant rate of evictions leads to an allocation that is optimal, in equilibrium, when nodes are visited through uniform sampling. Unfortunately, this optimality breaks down when uniform sampling is replaced by routing over arbitrary topologies [26]. Several papers have studied complexity and optimization issues of cost minimization as an offline caching problem under restricted topologies [4–6, 9, 21, 45]. With the exception of [45], these works model the network as a bipartite graph: nodes generating requests connect directly to caches, and demands are satisfied a single hop, and do not readily generalize to arbitrary topologies. In general, the *pipage rounding* technique of Ageev and Sviridenko [3] (see also [10, 47]) yields again a constant approximation algorithm in the bipartite setting, while approximation algorithms are also known for several variants of this problem [5, 6, 9, 21]. Excluding [9], all these works focus only on centralized solutions of the offline caching problem; none considers jointly optimizing caching *and* routing decisions.

In earlier work [26], we consider a setting in which routes are fixed, and only caching decisions are optimized in an adaptive, distributed fashion. We extend [26] to incorporate routing decisions, both through source and hop-by-hop routing. We show that a variant of pipage rounding [3] can be used to construct a poly-time approximation algorithm, that also lends itself to a distributed, adaptive implementation. Crucially, our evaluations in Section 5 show that jointly optimizing caching *and* routing significantly improves performance compared to fixed routing, reducing the routing costs by as much as three orders of magnitude compared to [26].

Several recent works study caching and routing *jointly*, in more restrictive settings than the ones we consider here. The benefit of routing towards nearest replicas, rather than towards nearest designated servers, has been observed empirically [11, 13, 19]. Deghan et al. [18], Abedini and Shakkotai [1], and Xie et al. [49] all study joint routing and content placement schemes in a bipartite, single-hop setting. In all three cases, minimizing the single-hop routing cost reduces to solving a linear program; Naveen et al. [37] extend this to other, non-linear (but still convex) objectives of the hit rate, still under single-hop, bipartite routing constraints. None of these approaches generalize to a multi-hop setting, which leads to non-convex formulations (see Section 3.6); addressing this lack of

convexity is one of our technical contributions. Closer to our work, a multi-hop, multi-path setting is formally analyzed by Carofiglio et al. [11] under the assumption that requests by different users follow non-overlapping paths. The authors show that under appropriate conditions on request arrival rates, this assumption leads to a convex optimization problem. Our approach addresses the lack of convexity in its full generality, for arbitrary topologies, request arrival rates, and overlapping paths.

The problem we study is also related to more general placement problems, including the allocation of virtual machines (VMs) to hosts in cloud computing [7, 25, 34, 46]–see also [29], that jointly optimizes placement and routing in this context. This is a harder problem: heterogeneity of host resources and VM requirements leads to multiple knapsack-like constraints (one for each resource) per host. Our storage constraints are simpler; as a result, in contrast to [7, 25, 29, 34, 46], we can provide poly-time, distributed algorithms with provable approximation guarantees.

3 MODEL

We begin by presenting our formal model, extending [26] to account for both caching *and* routing decisions. Our analysis applies to two routing variants: (a) *source routing* and (b) *hop-by-hop routing*. In both cases, we study two types of strategies: *deterministic* and *randomized*. For example, in source routing, requests for an item originating from the same source may be forwarded over several possible paths, given as input. In deterministic source routing, *only one* is selected and used *for all subsequent requests* with this origin. In contrast, a randomized strategy samples a new path to follow independently with each new request. We also use similar deterministic and randomized analogues both for caching strategies as well as for hop-by-hop routing strategies.

Randomized strategies subsume deterministic ones, and are arguably more flexible and general. This begs the question: why study both? There are three reasons. First, optimizing deterministic strategies naturally relates to submodular maximization subject to matroid constraints, allowing us to leverage related combinatorial optimization techniques. Second, the online, distributed algorithms we propose to construct randomized strategies rely on the solution to the offline, deterministic problem. Finally, and most importantly: deterministic strategies turn out to be equivalent to randomized strategies! As we show in Thm. 4.4, the smallest routing cost attained by randomized strategies is exactly the same as the one attained by deterministic strategies.

3.1 Network Model and Content Requests

Consider a network represented as a directed, symmetric[1] graph $G(V, E)$. Content items (e.g., files, or file chunks) of equal size are to be distributed across network nodes. Each node is associated with a cache that can store a finite number of items. We denote by C the set of possible content items, i.e., the *catalog*, and by $c_v \in \mathbb{N}$ the cache capacity at node $v \in V$: exactly c_v content items can be stored in v. The network serves content requests routed over the graph G. A request (i, s) is determined by (a) the item $i \in C$ requested, and (b) the source $s \in V$ of the request. We denote by $\mathcal{R} \subseteq C \times V$ the set of all requests. Requests of different types $(i, s) \in \mathcal{R}$ arrive according to independent Poisson processes with arrival rates $\lambda_{(i,s)} > 0$, $(i, s) \in \mathcal{R}$.

[1] A directed graph is symmetric when $(i, j) \in E$ implies that $(j, i) \in E$.

Common Notation	
$G(V, E)$	Network graph, with nodes V and edges E
C	Item catalog
c_v	Cache capacity at node $v \in V$
w_{uv}	Weight of edge $(u, v) \in E$
$\mathcal{R}$	Set of requests (i, s), with $i \in C$ and source $s \in V$
$\lambda_{(i,s)}$	Arrival rate of requests $(i, s) \in \mathcal{R}$
$\mathcal{S}_i$	Set of designated servers of $i \in C$
x_{vi}	Variable indicating whether $v \in V$ stores $i \in C$
ξ_{vi}	Marginal probability that v stores i
X	Global caching strategy of x_{vi}s, in $\{0, 1\}^{\lvert V\rvert \times \lvert C\rvert}$
Ξ	Expectation of caching strategy matrix X
T	Duration of a timeslot in online setting
w_{uv}	weight/cost of edge (u, v)
$\mathsf{supp}(\cdot)$	Support of a probability distribution
$\mathsf{conv}(\cdot)$	Convex hull of a set
Source Routing	
$\mathcal{P}_{(i,s)}$	Set of paths request $(i, s) \in \mathcal{R}$ can follow
P_{SR}	Total number of paths
p	A simple path of G
$k_p(v)$	The position of node $v \in p$ in path p.
$r_{(i,s),p}$	Variable indicating whether $(i, s) \in \mathcal{R}$ is forwarded over $p \in \mathcal{P}_{(i,s)}$
$\rho_{(i,s),p}$	Marginal probability that s routes request for i over p
r	Routing strategy of $r_{(i,s),p}$s, in $\{0, 1\}^{\sum_{(i,s)\in\mathcal{R}} \lvert\mathcal{P}_{(i,s)}\rvert}$.
ρ	Expectation of routing strategy vector r
$\mathcal{D}_{\mathsf{SR}}$	Feasible strategies (r, X) of MaxCG-S
RNS	Route to nearest server
RNR	Route to nearest replica
Hop-by-Hop Routing	
$G^{(i)}$	DAG with sinks in $\mathcal{S}_i$
$E^{(i)}$	Edges in DAG $G^{(i)}$
$G^{(i,s)}$	Subgraph of $G^{(i)}$ including only nodes reachable from s
$\mathcal{P}^u_{(i,s)}$	Set of paths in $G^{(i,s)}$ from s to u.
P_{HH}	Total number of paths
$r^{(i)}_{uv}$	Variable indicating whether u forwards a request for i to v
$\rho^{(i)}_{uv}$	Marginal probability that u forwards a request for i to v
r	Routing strategy of $r^i_{u,v}$s, in $\{0, 1\}^{\sum_{i\in C} \lvert E^{(i)}\rvert}$.
ρ	Expectation of routing strategy vector r
$\mathcal{D}_{\mathsf{HH}}$	Feasible strategies (r, X) of MaxCG-HH

Table 1: Notation Summary

For each item $i \in C$ there is a fixed set of *designated server* nodes $\mathcal{S}_i \subseteq V$, that always store i. A node $v \in \mathcal{S}_i$ permanently stores i in *excess memory outside its cache*. Thus, the placement of items to designated servers is fixed and outside the network's design.

A request (i, s) is routed over a path in G towards a designated server. However, forwarding terminates upon reaching *any intermediate cache* that stores i. At that point, a response carrying i is sent over the reverse path, i.e., from the node where the cache hit occurred, back to source node s. Both caching *and* routing decisions are network design parameters, which we define formally below.

3.2 Caching Strategies

We study two types or caches: *deterministic* and *randomized*.

Deterministic caches. For each node $v \in V$, we define v*'s caching strategy* as a vector $x_v \in \{0, 1\}^{|C|}$, where $x_{vi} \in \{0, 1\}$, for $i \in C$, is the binary variable indicating whether v stores content item i. As v can store no more than c_v items, we have that:

$$\sum_{i \in C} x_{vi} \leq c_v, \text{ for all } v \in V. \tag{1}$$

We define the *global caching strategy* as the matrix $X = [x_{vi}]_{v\in V, i\in C} \in \{0, 1\}^{|V|\times|C|}$, whose rows comprise the caching strategies of each node.

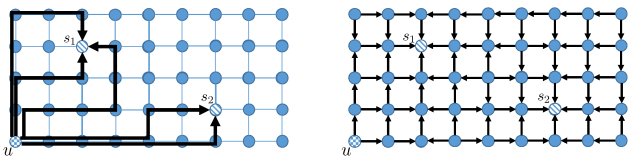

Figure 1: Source Routing vs. Hop-by-Hop routing. In source routing, shown left, source node u on the bottom left can choose among 5 possible paths to route a request to one of the designated servers storing i (s_1, s_2). In hop-by-hop routing, each intermediate node in the network selects the next hop among one of its neighbors in a DAG, whose sinks are the designated servers.

Randomized caches. In the case of randomized caches, the caching strategies x_v, $v \in V$, are random variables. We denote by:

$$\xi_{vi} \equiv \mathbf{P}[x_{vi} = 1] = \mathbb{E}[x_{v,i}] \in [0,1], \text{ for } i \in C, \tag{2}$$

the marginal probability that node v caches item i, and by $\Xi = [\xi_{vi}]_{v\in V, i\in C} = \mathbb{E}[X] \in [0,1]^{|V|\times|C|}$, the corresponding expectation of the global caching strategy.

3.3 Source Routing Strategies

Recall that requests are routed towards designated server nodes. In source routing, for every request $(i,s) \in C \times V$, there exists a $\mathcal{P}_{(i,s)}$ of *paths* that the request can follow towards a designated server in $\mathcal{S}_i$. A source node s can forward a request among any of these paths, but we assume each response follows the same path as its corresponding request. Formally, a path p of length $|p| = K$ is a sequence $\{p_1, p_2, \ldots, p_K\}$ of nodes $p_k \in V$ such that $(p_k, p_{k+1}) \in E$, for every $k \in \{1, \ldots, |p|-1\}$. We make the following natural assumptions on the set of paths $\mathcal{P}_{(i,s)}$. For every $p \in \mathcal{P}_{(i,s)}$: (a) p starts at s, i.e., $p_1 = s$; (b) p is simple, i.e., it contains no loops; (c) the last node in p is a designated server for item i, i.e., if $|p| = K$, $p_K \in \mathcal{S}_i$; and (d) no other node in p is a designated server for i, i.e., if $|p| = K$, $p_k \notin \mathcal{S}_i$, for $k = 1, \ldots, K-1$. Given a path p and a $v \in p$, denote by $k_p(v)$ is the position of v in p; i.e., $k_p(v)$ equals to $k \in \{1, \ldots, |p|\}$ such that $p_k = v$. As in the case of caches, we consider both deterministic and randomized routing strategies.

Deterministic Routing. Given sets $\mathcal{P}_{(i,s)}$, $(i,s) \in \mathcal{R}$, the *routing strategy* of a source $s \in V$ w.r.t. request $(i,s) \in \mathcal{R}$ is a vector $r_{(i,s)} \in \{0,1\}^{|\mathcal{P}_{(i,s)}|}$, where $r_{(i,s),p} \in \{0,1\}$ is a binary variable indicating whether s selected path $p \in \mathcal{P}_{(i,s)}$. These satisfy:

$$\textstyle\sum_{p\in\mathcal{P}_{(i,s)}} r_{(i,s),p} = 1, \text{ for all } (i,s) \in \mathcal{R}. \tag{3}$$

indicating that exactly one path is selected. Let $P_{\mathsf{SR}} = \sum_{(i,s)\in\mathcal{R}} |\mathcal{P}_{(i,s)}|$ be the total number of paths. We refer to the vector $r = [r_{(i,s),p}]_{(i,s)\in\mathcal{R}, p\in\mathcal{P}_{(i,s)}} \in \{0,1\}^P$, as the *global routing strategy*.

Randomized Routing. In the case of randomized routing, variables $r_{(i,s)}$, $(i,s) \in \mathcal{R}$ are random. We randomize routing by allowing requests to be routed over a random path in $\mathcal{P}_{(i,s)}$, selected *independently of all past requests (at s or elsewhere)*. We denote by

$$\rho_{(i,s),p} \equiv \mathbf{P}[r_{(i,s),p} = 1] = \mathbb{E}[r_{(i,s),p}], \text{ for } p \in \mathcal{P}_{(i,s)}, \tag{4}$$

the probability that path p is selected by s, and by $\rho = [\rho_{(i,s),p}]_{(i,s)\in\mathcal{R}, p\in\mathcal{P}_{(i,s)}} = \mathbb{E}[r] \in [0,1]^P$ the expectation of the global routing strategy r.

Remark. We make no a priori assumptions on P_{SR}, the total number of paths used during source routing; moreover, we allow paths to overlap. The complexity of our offline algorithm, and the rate of convergence of our distributed, adaptive algorithm depend on P_{SR}. In practice, if the number of possible paths is, e.g., exponential in $|V|$, it makes sense to restrict each $\mathcal{P}_{(i,s)}$ to a small subset of possible paths, or to use hop-by-hop routing instead, which, as discussed below, restricts the maximum number of paths considered.

3.4 Hop-by-Hop Routing Strategies

Under hop-by-hop routing, each node along the path makes an individual decision on where to route a request message. When a request for item i arrives at an intermediate node $v \in V$, node v determines how to forward the request to one of its neighbors. The decision depends on i but *not* on the request's source. This limits the paths a request may follow, making hop-by-hop routing less expressive than source routing. On the other hand, reducing the space of routing strategies reduces complexity. In adaptive algorithms, it also speeds up convergence, as routing decisions w.r.t. i are "learned" across requests by different sources.

To ensure loop-freedom, we must assume that forwarding decisions are restricted to a subset of possible neighbors in G. For each $i \in C$, we denote by $G^{(i)}(V, E^{(i)})$ a graph that has the following properties: (a) $G^{(i)}$ is a subgraph of G, i.e., $E^{(i)} \subseteq E$; (b) $G^{(i)}$ is a directed acyclic graph (DAG); and (c) a node v in $G^{(i)}$ is a sink if and only if it is a designated server for i, i.e., $v \in \mathcal{S}_i$. Note that, given G and $\mathcal{S}_i$, $G^{(i)}$ can be constructed in polynomial time using, e.g., the Bellman-Ford algorithm [15]. Indeed, requiring that v forwards requests for $i \in C$ only towards neighbors with a smaller distance to a designated server in $\mathcal{S}_i$ results in such a DAG. A distance-vector protocol [31] can form this DAG in a distributed fashion. We assume that every node $v \in V$ can forward a request for item i *only to a neighbor in* $G^{(i)}$. Then, the above properties of $G^{(i)}$ ensure both loop freedom and successful termination.

Deterministic Routing. For any node $s \in V$, let $G^{(i,s)}$ be the induced subgraph of $G^{(i)}$ which results from removing any nodes in $G^{(i)}$ not reachable from s. For any u in $G^{(i,s)}$, let $\mathcal{P}^u_{(i,s)}$ be the set of all paths in $G^{(i,s)}$ from s to u, and denote by $P_{\mathsf{HH}} = \sum_{(i,s)\in C} \sum_{u\in V} |\mathcal{P}^u_{(i,s)}|$. We denote by $r^{(i)}_{uv} \in \{0,1\}$, for $(u,v) \in E^{(i)}$, $i \in C$, the decision variable indicating whether u forwards a request for i to v. The *global routing strategy* is $r = [r^{(i)}_{uv}]_{i\in C,(u,v)\in E^{(i)}} \in \{0,1\}^{\sum_{i\in C}|E^{(i)}|}$, and satisfies

$$\textstyle\sum_{v:(u,v)\in E^{(i)}} r^{(i)}_{uv} = 1, \quad \text{for all } v \in V, i \in C. \tag{5}$$

Note that, in contrast to source routing strategies, that have length P_{SR}, hop-by-hop routing strategies have length at most $|C||E|$.

Randomized Routing. As in the case of source routing, we also consider randomized hop-by-hop routing strategies, whereby each request is forwarded independently from previous routing decisions to one of the possible neighbors. We again denote by

$$\begin{aligned}\rho &= [\rho^{(i)}_{uv}]_{i\in C,(u,v)\in E^{(i)}} = [\mathbb{E}[r^{(i)}_{uv}]]_{i\in C,(u,v)\in E^{(i)}} \\ &= \left[\mathbf{P}[r^{(i)}_{uv} = 1]\right]_{i\in C,(u,v)\in E^{(i)}} \in [0,1]^{\sum_{i\in C}|E^{(i)}|},\end{aligned} \tag{6}$$

the vector of corresponding (marginal) probabilities of routing decisions at each node v.

3.5 Offline vs. Online Setting

To reason about the caching networks we have proposed, we consider two settings: the *offline* and *online* setting. In the offline setting, all problem inputs (demands, network topology, cache capacities, etc.) are known apriori to, e.g., a system designer. At time $t = 0$, the system designer selects (a) a caching strategy X, and (b) a routing strategy r. Both can be either deterministic or randomized, but both are also static: they do not change as time progresses. In the case of caching, cache contents (selected deterministically or at random at $t = 0$) remain static for all $t \geq 0$. In the case of routing decisions, the distribution over paths (in source routing) or neighbors (in hop-by-hop routing) remains static, but each request is routed independently of previous requests.

In the online setting, no a priori knowledge of the demand, i.e., the rates of requests $\lambda_{(i,s)}$, $(i,s) \in \mathcal{R}$ is assumed. Both caching *and* routing strategies change through time via a *distributed, adaptive* algorithm. Time is slotted, and each slot has duration $T > 0$. During a timeslot, both caching and and routing strategies remain fixed. Nodes have access only to local information: they are aware of their graph neighborhood and state information they maintain locally. They exchange messages, including both normal request and response traffic, as well as (possibly) control messages, and may adapt their state. At the conclusion of a time slot, each node changes its caching and routing strategies. Changes made by v depend only on its neighborhood, its current local state, as well as on messages that node v received in the previous timeslot. Both caching and routing strategies during a timeslot may be deterministic or randomized. Implementing a caching strategy at the conclusion of a timeslot involves changing cache contents, which incurs additional overhead; if T is large, however, this cost is negligible compared to the cost of transferring items during a timeslot.

3.6 Optimal Routing and Caching

We are now ready to formally pose the problem of jointly optimizing caching and routing. We pose here the *offline* problem, in which problem inputs are given, and static caching and routing strategies are determined (jointly) at time $t = 0$. Nonetheless, we will devise distributed, adaptive algorithms that do not a priori know the demand, but still converge to (probabilistic) strategies that are within a constant approximation of the (offline) optimal.

To capture costs (e.g., latency, money, etc.), we associate a *weight* $w_{uv} \geq 0$ with each edge $(u,v) \in E$, representing the cost of transferring an item across this edge. We assume that costs are solely due to response messages that carry an item, while request forwarding costs are negligible. We do not assume that $w_{uv} = w_{vu}$. We describe the cost minimization objectives under source and hop-by-hop routing below.

Source Routing. The cost for serving a request $(i,s) \in \mathcal{R}$ under source routing is:

$$C_{\mathsf{SR}}^{(i,s)}(r,X) = \sum_{p \in \mathcal{P}_{(i,s)}} r_{(i,s),p} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \prod_{k'=1}^{k} (1 - x_{p_{k'}i}). \tag{7}$$

Intuitively, (7) states that $C_{\mathsf{SR}}^{(i,s)}$ includes the cost of an edge (p_{k+1}, p_k) in the path p if (a) p is selected by the routing strategy, and (b) *no* cache preceding this edge in p stores i.

In the deterministic setting, we seek a global caching and routing strategy (r,X) minimizing the aggregate expected cost, defined as:

$$C_{\mathsf{SR}}(r,X) = \textstyle\sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} C_{\mathsf{SR}}^{(i,s)}(r,X), \tag{8}$$

with $C_{\mathsf{SR}}^{(i,s)}$ given by (7). That is, we wish to solve:

MinCost-SR

$$\text{Minimize:} \quad C_{\mathsf{SR}}(r,X) \tag{9a}$$

$$\text{subj. to:} \quad (r,X) \in \mathcal{D}_{\mathsf{SR}} \tag{9b}$$

where $\mathcal{D}_{\mathsf{SR}} \subset \mathbb{R}^{P_{\mathsf{SR}}} \times \mathbb{R}^{|V|\times|C|}$ is the set of (r,X) satisfying the routing, capacity, and integrality constraints, i.e.:

$$\textstyle\sum_{i\in C} x_{vi} = c_v, \qquad \text{for all } v \in V, \tag{10a}$$

$$\textstyle\sum_{p\in\mathcal{P}_{(i,s)}} r_{(i,s),p} = 1, \qquad \text{for all } (i,s) \in \mathcal{R}, \tag{10b}$$

$$x_{vi} \in \{0,1\}, \qquad \text{for all } v \in V, i \in C, \text{ and} \tag{10c}$$

$$r_{(i,s),p} \in \{0,1\}, \qquad \text{for all } p \in \mathcal{P}_{(i,s)}, (i,s) \in \mathcal{R}. \tag{10d}$$

This problem is NP-hard, even in the case where routing is fixed: see Shanmugam et al. [45] for a reduction from the 2-Disjoint Set Cover Problem.

Hop-By-Hop Routing. Similarly to (7), under hop-by-hop routing, the cost of serving (i,s) can be written as:

$$C_{\mathsf{HH}}^{(i,s)}(r,X) = \textstyle\sum_{(u,v)\in G^{(i,s)}} w_{vu} \cdot r_{uv}^{(i)} (1 - x_{ui}) \cdot \sum_{p\in\mathcal{P}_{(i,s)}^{u}} \prod_{k'=1}^{|p|-1} r_{p_{k'}p_{k'+1}}^{(i)} (1 - x_{p_{k'}i}). \tag{11}$$

We wish to solve:

MinCost-HH

$$\text{Minimize:} \quad C_{\mathsf{HH}}(r,X) \tag{12a}$$

$$\text{subj. to:} \quad (r,X) \in \mathcal{D}_{\mathsf{HH}} \tag{12b}$$

where $C_{\mathsf{HH}}(r,X) = \sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} C_{\mathsf{HH}}^{(i,s)}(r,X)$ is the expected routing cost, and $\mathcal{D}_{\mathsf{HH}}$ is the set of $(r,X) \in \mathbb{R}^{\sum_{i\in C}|E^{(i)}|} \times \mathbb{R}^{|V|\times|C|}$ satisfying the constraints:

$$\textstyle\sum_{i\in C} x_{vi} = c_v, \qquad \text{for all } v \in V, \tag{13a}$$

$$\textstyle\sum_{v:(u,v)\in E^{(i)}} r_{uv}^{(i)} = 1 \qquad \text{for all } v \in V, i \in C, \tag{13b}$$

$$x_{vi} \in \{0,1\}, \qquad \text{for all } v \in V, i \in C, \text{ and} \tag{13c}$$

$$r_{uv}^{(i)} \in \{0,1\}, \qquad \text{for all } (u,v) \in E^{(i)}, i \in C. \tag{13d}$$

Randomization. The above routing cost minimization problems can also be stated in the context of randomized caching and routing strategies. For example, in the case of source routing, assuming (a) independent caching strategies across nodes selected at time $t = 0$, with marginal probabilities given by Ξ, and (b) independent routing strategies at each source, with marginals given by ρ (also independent from caching strategies), all terms in C_{SR} contain products of independent random variables; this implies that:

$$\mathbb{E}[C_{\mathsf{SR}}(r,X)] = C_{\mathsf{SR}}[\mathbb{E}[r], \mathbb{E}[X]] = C_{\mathsf{SR}}(\rho, \Xi), \tag{14}$$

where the expectation is taken over the randomness of both caching and routing strategies. The expected routing cost thus depends on the routing and caching strategies only through the expectations ρ

and Ξ. As a result, under randomized routing and caching strategies, MinCost-SR becomes (see [27] for the derivation):

$$\text{Minimize:} \quad C_{\mathsf{SR}}(\rho, \Xi) \tag{15a}$$
$$\text{subj. to:} \quad (\rho, \Xi) \in \mathsf{conv}(\mathcal{D}_{\mathsf{SR}}) \tag{15b}$$

where $\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$ is the convex hull of $\mathcal{D}_{\mathsf{SR}}$; this is precisely the set defined by (10) with integrality constraints (10c), (10d) relaxed. The objective function C_{SR} is *not convex* and the relaxed problem (15) is therefore *not* a convex optimization problem. This is in stark contrast to single-hop settings, that often can naturally be expressed as linear programs [1, 18, 37].

A similar derivation can be done for hop-by-hop routing. Assuming again independent caches and independent routing strategies, it can be shown that optimizing over randomized hop-by-hop strategies is equivalent to

$$\text{Minimize:} \quad C_{\mathsf{HH}}(\rho, \Xi) \tag{16a}$$
$$\text{subj. to:} \quad (\rho, \Xi) \in \mathsf{conv}(\mathcal{D}_{\mathsf{HH}}), \tag{16b}$$

where $\mathsf{conv}(\mathcal{D}_{\mathsf{HH}})$ the convex hull of $\mathcal{D}_{\mathsf{HH}}$. This, again, is a non-convex optimization problem.

3.7 Fixed Routing

When the global routing strategy r is fixed, (9) reduces to

$$\text{Minimize:} \quad C_{\mathsf{SR}}(r, X) \tag{17a}$$
$$\text{subj. to:} \quad X \text{ satisfies (10a) and (10c)} \tag{17b}$$

MinCost-HH can be similarly restricted to caching only. We studied this restricted optimization in earlier work [26]. In particular, under given global routing strategy r, we cast (17) as a maximization problem as follows. Let

$$C_0^r = C_{\mathsf{SR}}(r, 0) = \sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} \sum_{p\in\mathcal{P}_{(i,s)}} r_{(i,s),p} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \tag{18}$$

be the cost when all caches are empty (i.e., X is the zero matrix 0). Note that this is a constant that does not depend on X. Consider the following maximization problem:

$$\text{Maximize:} \quad F_{\mathsf{SR}}^r(X) = C_0^r - C_{\mathsf{SR}}(r, X) \tag{19a}$$
$$\text{subj. to:} \quad X \text{ satisfies (10a) and (10c)} \tag{19b}$$

This problem is equivalent to (17), in that a feasible solution to (19) is optimal if and only if it also optimal for (17). The objective $F_{\mathsf{SR}}^r(X)$, referred to as the *caching gain* in [26], is monotone, non-negative, and submodular, while the set of constraints on X is a set of matroid constraints. As a result, for any r, there exist standard approaches for constructing a polynomial time approximation algorithm solving the corresponding maximization problem (19) within a $1 - 1/e$ factor from its optimal solution [26, 45]. In addition, we show [26] that an approximation algorithm based on a technique known as *pipage rounding* [3] can be converted into a distributed, adaptive version with the same approximation ratio.

3.8 Greedy Routing Strategies

In the case of source routing, we identify two "greedy" deterministic routing strategies, that are often used in practice, and play a role in our analysis. We say that a global routing strategy r is a *route-to-nearest-server* (RNS) strategy if all paths it selects are least-cost paths to designated servers, irrespectively of cache contents. Formally, for all $(i,s) \in \mathcal{R}$, $r_{(i,s),p^*} = 1$ for some $p^* \in \arg\min_{p\in\mathcal{P}_{(i,s)}} \sum_{k=1}^{|p|-1} w_{p_{k+1},p_k}$, while $r_{(i,s),p} = 0$ for all other $p \in \mathcal{P}_{(i,s)}$ s.t. $p \neq p^*$. Similarly, given a caching strategy X, we say that a global routing strategy r is *route-to-nearest-replica* (RNR) strategy if, for all $(i,s) \in \mathcal{R}$, $r_{(i,s),p^*} = 1$ for some $p^* \in \arg\min_{p\in\mathcal{P}_{(i,s)}} \sum_{k=1}^{|p|-1} w_{p_{k+1},p_k} \prod_{k'=1}^{k} (1 - x_{p_{k'}i})$, while $r_{(i,s),p} = 0$ for all other $p \in \mathcal{P}_{(i,s)}$ s.t. $p \neq p^*$. In contrast to RNS strategies, RNR strategies depend on the caching strategy X. Note that RNS and RNR strategies can be defined similarly in the context of hop-by-hop routing.

4 MAIN RESULTS

We present our main results in this section, extending the analysis in [26] to the joint optimization of both caching and routing decisions. We provide an analysis of both source and hop-by-hop routing; proofs of theorems are omitted, and are provided in our technical report [27].

4.1 Routing to Nearest Server Is Suboptimal

A simple approach, followed by most works that optimize caching separately from routing, is to always route requests to the nearest designated server storing an item (i.e., use an RNS strategy). It is therefore interesting to ask how this simple heuristic performs compared to a solution that attempts to solve (9) by *jointly* optimizing caching and routing. It is easy to see that RNS and, more generally, routing that ignores caching strategies, can lead to arbitrarily suboptimal solutions:

Theorem 4.1. *For any $M > 0$, there exists a caching network for which the route-to-nearest-server strategy r' satisfies*

$$\min_{X:(r',X)\in\mathcal{D}_{\mathsf{SR}}} C_{\mathsf{SR}}(r', X) / \min_{(r,X)\in\mathcal{D}_{\mathsf{SR}}} C_{\mathsf{SR}}(r, X) = \Theta(M). \tag{20}$$

In other words, routing to the nearest server can be arbitrarily suboptimal, incurring a cost arbitrarily larger than the cost of the optimal jointly optimized routing and caching policy. The network that exhibits this behavior is shown in Fig. 2, and a proof of the theorem can be found in [27]. In short, a source node s generates requests for items 1 and 2 that are permanently stored on designated server t. There are two alternative paths towards t, each passing through an intermediate node with cache capacity 1 (i.e., able to store only one item). Under shortest path routing, requests for both items are forwarded over the path of length $M + 1$ towards t; fixing routes this way leads to a cost of $M + 1$ for at least one of the items, irrespectively of which item is cached in the intermediate node. On the other hand, if routing and caching decisions are jointly optimized, requests for the two items can be forwarded to different paths, allowing both items to be cached, and reducing the cost for both requests to at most 2.

This example illustrates that joint optimization of caching *and* routing decisions benefits the system by *increasing path diversity*. In turn, increasing path diversity can increase caching opportunities, thereby leading to reductions in caching costs. This is consistent with our experimental results in Section 5.

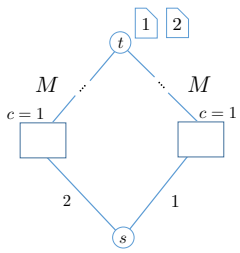

Figure 2: A simple example illustrating the benefits of path diversity. A source node s generates requests for items 1 and 2, permanently stored on designated server t. Intermediate nodes on the are two alternative paths towards t have capacity 1. Numbers above edges indicate costs.

4.2 Offline Source Routing.

Expected Caching Gain. Before presenting a distributed, adaptive joint routing and caching algorithm, we first turn our attention to the offline problem MinCost. As in the solution by [26] described in Section 3.7, we cast this first as a maximization problem. Let C_0 be the constant:

$$C^0_{\mathsf{SR}} = \sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} \sum_{p\in\mathcal{P}_{(i,s)}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k}. \tag{21}$$

Then, given a pair of strategies (r, X), we define the *expected caching gain* $F_{\mathsf{SR}}(r, X)$ as follows:

$$F_{\mathsf{SR}}(r, X) = C^0_{\mathsf{SR}} - C_{\mathsf{SR}}(r, X), \tag{22}$$

where C_{SR} is the aggregate routing cost given by (8). Note that C^0_{SR} upper bounds the expected routing cost, so that $F_{\mathsf{SR}}(r, X) \geq 0$. We seek to solve the following problem, equivalent to MinCost:

$$\begin{aligned} &\text{MaxCG-S} \\ \text{Maximize:} \quad & F_{\mathsf{SR}}(r, X) && \text{(23a)} \\ \text{subj. to:} \quad & (r, X) \in \mathcal{D}_{\mathsf{SR}} && \text{(23b)} \end{aligned}$$

The selection of the constant C^0_{SR} is not arbitrary: this is precisely the value that allows us to approximate F_{SR} via the concave relaxation L_{SR} below–c.f. Eq. (26).

Approximation Algorithm. Its equivalence to MinCost implies that MaxCG-S is also NP-hard. Nevertheless, we show that there exists a polynomial time approximation algorithm for MaxCG-S:

Theorem 4.2. *There exists an algorithm that terminates within a number of steps that is polynomial in $|V|$, $|C|$, and P_{SR}, and produces a strategy $(r', X') \in \mathcal{D}_{\mathsf{SR}}$ such that*

$$F_{\mathsf{SR}}(r', X') \geq (1 - 1/e) \max_{(r,X)\in\mathcal{D}_{\mathsf{SR}}} F_{\mathsf{SR}}(r, X). \tag{24}$$

In Sec. 5 we show that, in spite of attaining approximation guarantees w.r.t. F_{SR} rather than C_{SR}, the resulting approximation algorithm has excellent performance in practice in terms of minimizing routing costs. In particular, we can reduce routing costs by a factor as high as 10^3 compared to fixed routing policies, including [26].

We briefly describe the algorithm below, leaving details to [27]. Consider the concave function $L_{\mathsf{SR}} : \mathsf{conv}(\mathcal{D}_{\mathsf{SR}}) \to \mathbb{R}_+$, defined as:

$$\begin{aligned} L_{\mathsf{SR}}(\rho, \Xi) = \sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} \sum_{p\in\mathcal{P}_{(i,s)}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \cdot \\ \min\left\{1, 1 - \rho_{(i,s),p} + \sum_{k'=1}^{k} \xi_{p_{k'}i}\right\}. \end{aligned} \tag{25}$$

Then, L_{SR} closely approximates F_{SR} [27]:

$$(1 - 1/e)\, L_{\mathsf{SR}}(\rho, \Xi) \leq F_{\mathsf{SR}}(\rho, \Xi) \leq L_{\mathsf{SR}}(\rho, \Xi), \tag{26}$$

for all $(\rho, \Xi) \in \mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$. Our constant-approximation algorithm for MaxCG-S comprises two steps. *First,* obtain

$$(\rho^*, \Xi^*) \in \arg\max_{(\rho,\Xi)\in\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})} L_{\mathsf{SR}}(\rho, \Xi). \tag{27}$$

As L_{SR} is a concave function and $\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$ is convex, the above maximization is a convex optimization problem. In fact, it can be reduced to a linear program, so it can be solved in polynomial time [38]. *Second,* round the (possibly fractional) solution $(\rho^*, \Xi^*) \in \mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$ to an integral solution $(r, X) \in \mathcal{D}_{\mathsf{SR}}$ such that $F_{\mathsf{SR}}(r, X) \geq F_{\mathsf{SR}}(\rho^*, \Xi^*)$. This rounding is deterministic and also takes place in polynomial time.

The rounding technique used in our proof of Thm. 4.2 has the following immediate implication:

Corollary 4.3. *There exists an optimal solution (r^*, X^*) to MaxCG-S (and hence, to MinCost-SR) in which r^* is an route-to-nearest-replica (RNR) strategy w.r.t. X^*.*

Although, in light of Theorem 4.1, Corollary 4.3 suggests an advantage of RNR over RNS strategies, we note it does not give any intuition on how to construct an optimal RNR solution.

Equivalence of Deterministic and Randomized Strategies. We can also show the following result regarding *randomized* strategies. For μ a probability distribution over $\mathcal{D}_{\mathsf{SR}}$, let $\mathbb{E}_\mu[C_{\mathsf{SR}}(r, X)]$ be the expected routing cost under μ. Then, the following equivalence theorem holds:

Theorem 4.4. *The deterministic and randomized versions of MinCost-SR attain the same optimal routing cost, i.e.:*

$$\begin{aligned} \min_{(r,X)\in\mathcal{D}_{\mathsf{SR}}} C_{\mathsf{SR}}(r, X) &= \min_{(\rho,\Xi)\in\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})} C_{\mathsf{SR}}(\rho, \Xi) \\ &= \min_{\mu:\mathsf{supp}(\mu)=\mathcal{D}_{\mathsf{SR}}} \mathbb{E}_\mu[C_{\mathsf{SR}}(r, X)] \end{aligned} \tag{28}$$

The first equality of the theorem implies that, surprisingly, there is *no inherent advantage in randomization*: although randomized strategies constitute a superset of deterministic strategies, the optimal attainable routing cost (or, equivalently, caching gain) is the same for both classes. The second equality implies that assuming independent caching and routing strategies is as powerful as sampling routing and caching strategies from an arbitrary joint distribution. Thm. 4.4 generalizes Thm. 5 of [26], which pertains to optimizing caching alone.

4.3 Online Source Routing

The algorithm in Thm. 4.2 is offline and centralized: it assumes full knowledge of the input, including demands and arrival rates, which are rarely a priori available in practice. To that end, we turn our attention to solving MaxCG-S in the online setting, in the absence of any a priori knowledge of the demand, and seek an algorithm that is both adaptive and distributed.

As described in 3.5, in the online setting, time is partitioned into slots of equal length $T > 0$. Caching and routing strategies are randomized as described in Sec. 3: at the beginning of a timeslot, nodes place a random set of contents in their cache, independently of each other; upon arrival, a new request is routed over a random path, selected independently of (a) all past routes followed, and

Algorithm 1 Projected Gradient Ascent

Execute the following for each $v \in V$ and each $(i, s) \in \mathcal{R}$:
Pick arbitrary state $(\rho^{(0)}, \Xi^{(0)}) \in \mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$.
for each timeslot $k \geq 1$ **do**
  for each $v \in V$ **do**
    Compute the sliding average $\bar{\xi}_v^{(k)}$.
    Sample a feasible $x_v^{(k)}$ from a distribution with marginals $\bar{\xi}_v^{(k)}$.
    Place items $x_v^{(k)}$ in cache.
    Collect measurements and, at the end of the timeslot, compute estimate o f $\partial_{\xi_v} L_{\mathsf{SR}}(\rho^k, \Xi^{(k)})$.
    Adapt to new state $\xi_v^{(k+1)}$ in the direction of the gradient with step-size γ_k, projecting back to $\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$.
  end for
  for each $(i, s) \in \mathcal{R}$ **do**
    Compute the sliding average $\bar{\rho}_{(i,s)}^{(k)}$.
    Whenever a new request arrives, sample $p \in \mathcal{P}_{(i,s)}$ from distribution $\bar{\rho}_{(i,s)}^{(k)}$.
    Collect measurements and, at the end of the timeslot, compute estimate of $\partial_{\rho_{(i,s)}} L_{\mathsf{SR}}(\rho^k, \Xi^{(k)})$.
    Adapt to new state $\rho_{(i,s)}^{(k+1)}$ in the direction of the gradient with step-size γ_k, projecting back to $\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$.
  end for
end for

(b) of caching decisions. Our next theorem shows that, in steady state, the expected caching gain of the jointly constructed routing and caching strategies is within a constant approximation of the optimal solution to the offline problem MaxCG-S:

Theorem 4.5. *There exists a distributed, adaptive algorithm under which the randomized strategies sampled during the k-th slot* $(r^{(k)}, X^{(k)}) \in \mathcal{D}_{\mathsf{SR}}$ *satisfy*

$$\lim_{k\to\infty} \mathbb{E}[F_{\mathsf{SR}}(r^{(k)}, X^{(k)})] \geq (1 - 1/e) \max_{(r,X)\in\mathcal{D}_{\mathsf{SR}}} F_{\mathsf{SR}}(r, X). \tag{29}$$

Note that, despite the fact that the algorithm has no prior knowledge of the demands, the guarantee provided is w.r.t. an optimal solution of the *offline* problem (23). Moreover, in light of Thm. 4.4, our adaptive algorithm is $1 - \frac{1}{e}$-competitive w.r.t. optimal offline *randomized* strategies as well. Our algorithm naturally generalizes [26]: when the path sets $\mathcal{P}_{(i,s)}$ are singletons, and routing is fixed, our algorithm coincides with the cache-only optimization algorithm in [26]. Interestingly, the algorithm casts routing and caching *in the same control plane*: the same quantities are communicated through control messages to adapt both the caching and routing strategies.

Algorithm Overview. We give a brief overview of the distributed, adaptive algorithm that attains the guarantees of Theorem 4.5 below. The algorithm is summarized in Algorithm 1. Recall that time is partitioned into slots of equal length $T > 0$. Caching and routing strategies are randomized as described in Sec. 3: at the beginning of a timeslot, nodes place a random set of contents in their cache, independently of each other; upon arrival, a new request is routed over a random path, selected independently of (a) all past routes followed, and (b) of caching decisions.

More specifically, nodes in the network maintain the following state information. Each node $v \in G$ maintains locally a vector $\xi_v \in [0, 1]^{|C|}$, determining its randomized caching strategy. Moreover, for each request $(i, s) \in \mathcal{R}$, source node s maintains a vector $\rho_{(i,s)} \in [0, 1]^{|\mathcal{P}_{(i,s)}|}$, determining its randomized routing strategy. Together, these variables represent the global state of the network, denoted by $(\rho, \Xi) \in \mathsf{conv}(\mathcal{D}_{\mathsf{SR}})$. When the timeslot ends, each node performs the following four tasks:

(1) **Subgradient Estimation**. Each node uses measurements collected during the duration of a timeslot to construct estimates of the gradient of L_{SR} w.r.t. its own local state variables. As L_{SR} is not everywhere differentiable, an estimate of a *subgradient* of L_{SR} is computed instead.
(2) **State Adaptation.** Nodes adapt their local caching and routing state variables ξ_v, $v \in V$, and $\rho_{(i,s)}$, $(i, s) \in \mathcal{R}$, pushing them towards a direction that increases L_{SR}, as determined by the estimated subgradients.
(3) **State Smoothening.** Nodes compute "smoothened" versions $\bar{\xi}_v$, $v \in V$, and $\bar{\rho}_{(i,s)}$, $(i, s) \in \mathcal{R}$, interpolated between present and past states. This is needed on account of the non-differentiability of L_{SR}.
(4) **Randomized Caching and Routing.** After smoothening, each node v reshuffles the contents of its cache using the smoothened caching marginals $\bar{\xi}_v$, producing a random placement (i.e., caching strategy x_v) to be used throughout the next slot. Moreover, each node $s \in V$ routes requests $(i, s) \in \mathcal{R}$ received during next timeslot over random paths (i.e., routing strategies $r_{(i,s)}$) sampled in an i.i.d. fashion from the smoothened marginals $\bar{\rho}_{(i,s)}$.

Together, these steps ensure that, in steady state, the expected caching gain of the jointly constructed routing and caching strategies is within a constant approximation of the optimal solution to the offline problem MaxCG-S. We formally describe the constituent subgradient estimation, state adaptation, smoothening, and random sampling steps in detail in [27]. We also characterize the overhead of the protocol, specifying control messages, and proposing a modification that reduces this overhead.

Convergence Guarantees. The proof of the convergence of the algorithm relies on the following key lemma:

Lemma 4.6. *Let* $(\bar{\rho}^{(k)}, \bar{\Xi}^{(k)}) \in \mathcal{D}_2$ *be the smoothened state variables at the k-th slot of Algorithm 1, and* $(\rho^*, \Xi^*) \in \arg\max_{(\rho,\Xi)\in\mathsf{conv}(\mathcal{D}_{\mathsf{SR}})} L_{\mathsf{SR}}(\rho, \Xi)$. *Then, for* γ_k *the step-size used in projected gradient ascent,*

$$\varepsilon_k \equiv \mathbb{E}[L_{\mathsf{SR}}(\rho^*, \Xi^*) - L_{\mathsf{SR}}(\bar{\rho}^{(k)}, \bar{\Xi}^{(k)})] \leq \frac{D^2 + M^2 \sum_{\ell=\lfloor k/2 \rfloor}^{k} \gamma_\ell^2}{2 \sum_{\ell=\lfloor k/2 \rfloor}^{k} \gamma_\ell},$$

where $D = \sqrt{2|V| \max_{v\in V} c_v + 2|\mathcal{R}|}$, *and*

$$M = W|V|\Lambda\sqrt{(|V||C|P^2 + |\mathcal{R}|P)(1 + \frac{1}{\Lambda T})}.$$

In particular, $\varepsilon_k = O(1/\sqrt{k})$ *for* $\gamma = 1/\sqrt{k}$.

Lemma 4.6 establishes that Algorithm 1 converges arbitrarily close to an optimizer of L_{SR}. As, by (26), this is a close approximation of F_{SR}, the limit points of the algorithm are with the $1-1/e$ from the optimal. Crucially, Lemma 4.6 can be used to determine the rate of convergence of the algorithm, by determining the number of steps required for ε_k to reach a desired threshold δ. Moreover, through quantity M, Lemma 4.6 establishes a tradeoff w.r.t. T: increasing T decreases the error in the estimated subgradient, thereby reducing the total number of steps till convergence, but also increases the time taken by each step.

Graph	$\|V\|$	$\|E\|$	$\|C\|$	$\|\mathcal{R}\|$	$\|Q\|$	c_v	$\|\mathcal{P}_{(i,s)}\|$	$\bar{C}_{\mathsf{SR}}^{\mathsf{PGA}}$
cycle	30	60	10	100	10	2	2	20.17
grid-2d	100	360	300	1K	20	3	30	0.228
hypercube	128	896	300	1K	20	3	30	0.028
expander	100	716	300	1K	20	3	30	0.112
erdos-renyi	100	1042	300	1K	20	3	30	0.047
regular	100	300	300	1K	20	3	30	0.762
watts-strogatz	100	400	300	1K	20	3	2	35.08
small-world	100	491	300	1K	20	3	30	0.029
barabasi-albert	100	768	300	1K	20	3	30	0.187
geant	22	66	10	100	10	2	10	1.28
abilene	9	26	10	90	9	2	10	0.911
dtelekom	68	546	300	1K	20	3	30	0.025

Table 2: Graph Topologies and Experiment Parameters.

4.4 Hop-by-Hop Routing

A similar analysis to the one we outlined above applies to hop-by-hop routing, both in the offline and online setting. We state again the main theorems here; proofs can again be found in [27].

Offline Setting. As in the case of source routing, we define the constant: $C^0_{\mathsf{HH}} = \sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} \sum_{(u,v)\in G^{(i,s)}} w_{vu} |\mathcal{P}^u_{(i,s)}|$. Using this constant, we define the caching gain maximization problem to be:

$$\text{MaxCG-HH}$$
$$\text{Maximize:} \quad F_{\mathsf{HH}}(r, X) \tag{30a}$$
$$\text{subj. to:} \quad (r, X) \in \mathcal{D}_{\mathsf{HH}} \tag{30b}$$

where $F_{\mathsf{HH}}(r, X) = C^0_{\mathsf{HH}} - \sum_{(i,s)\in\mathcal{R}} \lambda_{(i,s)} C^{(i,s)}_{\mathsf{HH}}(r, X)$ is the expected caching gain. This is again an NP-hard problem, equivalent to (12). We can again construct a constant approximation algorithm for MaxCG-HH:

Theorem 4.7. *There exists an algorithm that terminates within a number of steps that is polynomial in $|V|$, $|C|$, and P_{HH}, and produces a strategy $(r', X') \in \mathcal{D}_{\mathsf{HH}}$ such that*

$$F_{\mathsf{HH}}(r', X') \geq (1 - 1/e) \max_{(r,X)\in\mathcal{D}_{\mathsf{HH}}} F_{\mathsf{HH}}(r, X). \tag{31}$$

Online Setting. Finally, as in the case of source routing, we can provide a distributed, adaptive algorithm for hop-by-hop routing as well.

Theorem 4.8. *There exists a distributed, adaptive algorithm under which the randomized strategies sampled during the k-th slot $(r^{(k)}, X^{(k)}) \in \mathcal{D}_{\mathsf{HH}}$ satisfy*

$$\lim_{k\to\infty} \mathbb{E}[F_{\mathsf{HH}}(r^{(k)}, X^{(k)})] \geq \left(1 - 1/e\right) \max_{(r,X)\in\mathcal{D}_{\mathsf{SR}}} F_{\mathsf{HH}}(r, X). \tag{32}$$

We note again that the distributed, adaptive algorithm attains an expected caching gain within a constant approximation from the *offline* optimal.

5 EVALUATION

We simulate our distributed, adaptive algorithm for MaxCG-S over a broad variety of both synthetic and real networks. We compare its performance to traditional caching policies, combined with both static and dynamic multi-path routing.

Experiment Setup. We consider the topologies in Table 2. For each graph $G(V, E)$ in Table 2, we generate a catalog of size $|C|$, and assign to each node $v \in V$ a cache of capacity c_v. For every item $i \in C$, we designate a node selected u.a.r. from V as a designated server for this item; the item is stored outside the designate server's cache. We assign a weight to each edge in E selected u.a.r. from the interval $[1, 100]$. We also select a random set of Q nodes as the possible request sources, and generate a set of requests $\mathcal{R} \subseteq C \times V$ by sampling exactly $|\mathcal{R}|$ from the set $C \times Q$, uniformly at random. For each such request $(i, s) \in \mathcal{R}$, we select the request rate $\lambda_{(i,s)}$ according to a Zipf distribution with parameter 1.2; these are normalized so that average request rate over all $|Q|$ sources is 1 request per time unit. For each request $(i, s) \in \mathcal{R}$, we generate $|\mathcal{P}_{(i,s)}|$ paths from the source $s \in V$ to the designated server of item $i \in C$. In all cases, this path set includes the shortest path to the designated server. We consider only paths with stretch at most 4.0; that is, the maximum cost of a path in $P_{(i,s)}$ is at most 4 times the cost of the shortest path to the designated source. The values of $|C|$, $|\mathcal{R}|$ $|Q|$, c_v, and $\mathcal{P}_{(i,s)}$ for each experiment are given in Table 2.

Online Caching and Routing Algorithms. We compare the performance of our joint caching and routing projected gradient ascent algorithm (PGA) to several competitors. In terms of caching, we consider four traditional eviction policies for comparison: Least-Recently-Used (LRU), Least-Frequently-Used (LFU), First-In-First-Out (FIFO), and Random Replacement (RR). We combine these policies with path-replication [14, 28]: once a request for an item reaches a cache that stores the item, every cache in the reverse path on the way to the query source stores the item, evicting stale items using one of the above eviction policies. We combine the above caching policies with three different routing policies. In route-to-nearest-server (-S), only the shortest path to the nearest designated server is used to route the message. In uniform routing (-U), the source s routes each request (i, s) on a path selected uniformly at random among all paths in $\mathcal{P}_{(i,s)}$. We combine each of these (static) routing strategies with each of the above caching strategies use. For instance, LRU-U indicates LRU evictions combined with uniform routing. Note that PGA-S, i.e., our algorithm restricted to RNS routing, is exactly the single-path routing algorithm proposed in [26]. To move beyond static routing policies for LRU, LFU, FIFO, and RR, we also combine the above traditional caching strategies with an adaptive routing strategy, akin to our algorithm, with estimates of the expected routing cost at each path used to adapt routing strategies. During a slot, each source node s maintains an average of the routing cost incurred when routing a request over each path. At the end of the slot, the source decreases the probability $\rho_{(i,s),p}$ that it will follow the path p by an amount proportional to the average, and projects the new strategy to the simplex. For fixed caching strategies, this dynamic routing scheme converges to a route-to-nearest-replica (RNS) routing, which we expect by Cor. 4.3 to have good performance. We denote this routing scheme with the extension -D. Note that all algorithms we simulate are online.

Experiments and Measurements. Each experiment consists of a simulation of the caching and routing policy, over a specific topology, for a total of 5000 time units. To leverage PASTA, we collect measurements during the duration of the execution at exponentially distributed intervals with mean 1.0 time unit. At each measurement epoch, we extract the current cache contents in the network and construct $X \in \{0, 1\}^{|V|\times|C|}$. Similarly, we extract the current routing strategies $\rho_{(i,s)}$ for all requests $(i, s) \in \mathcal{R}$, and construct the global routing strategy $\rho \in [0, 1]^{P_{\mathsf{SR}}}$. Then, we evaluate the *expected routing cost* $C_{\mathsf{SR}}(\rho, X)$. We report the average $\bar{C}_{\mathsf{SR}}$ of these values across measurements collected after a warmup phase, during 1000

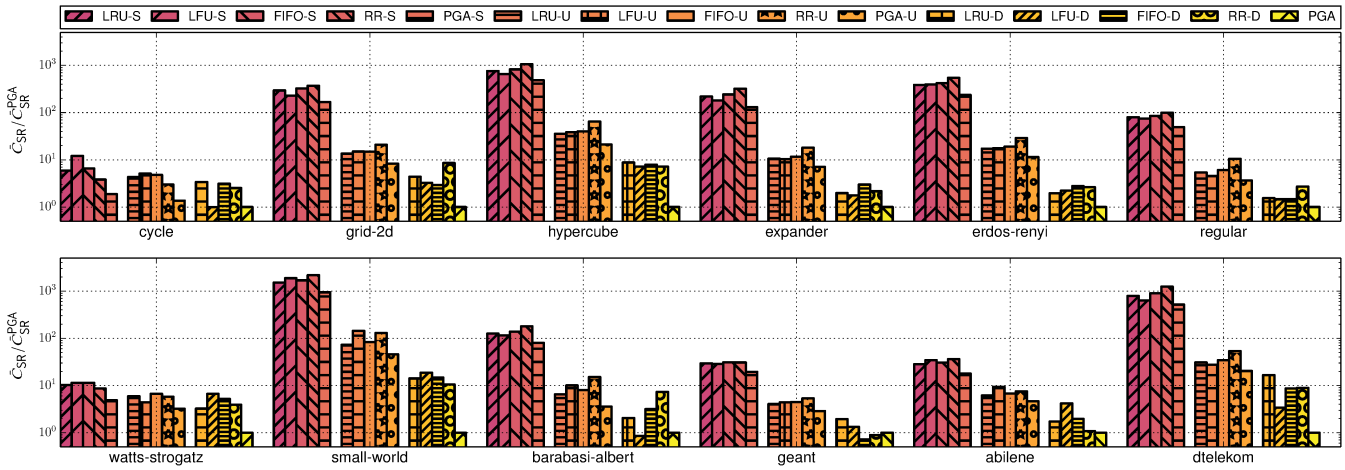

Figure 3: Ratio of expected routing cost $\bar{C}_{\mathsf{SR}}$ to routing cost $\bar{C}_{\mathsf{SR}}^{\mathsf{PGA}}$ under our PGA policy, for different topologies and strategies. For each topology, each of the three groups of bars corresponds to a routing strategy, namely, RNS/shortest path routing (-S), uniform routing (-U), and dynamic routing (-D). The algorithm presented in [26] is PGA-S, while our algorithm (PGA), with ratio 1.0, is shown last for reference purposes; values of of $\bar{C}_{\mathsf{SR}}^{\mathsf{PGA}}$ are given in Table 2.

Graph	LRU-S	PGA-S	LRU-U	PGA-U	LRU	PGA
cycle	0.47	865.29	0.47	436.14	6.62	148.20
grid-2d	0.08	657.84	0.08	0.08	0.08	0.08
hypercube	0.21	924.75	0.21	0.21	0.21	0.21
expander	0.38	794.27	0.38	0.38	0.38	0.38
erdos-renyi	3.08	870.84	0.25	0.25	0.25	0.25
regular	1.50	1183.97	0.05	8.52	0.05	11.49
watts-strogatz	11.88	158.39	7.80	54.90	19.22	37.05
small-world	0.30	955.48	0.30	0.30	0.30	0.30
barabasi-albert	1.28	1126.24	1.28	6.86	1.28	7.58
geant	0.09	1312.96	1.85	12.71	0.09	14.41
abilene	3.44	802.66	3.44	23.08	5.75	14.36
dtelekom	0.30	927.24	0.30	0.30	0.30	0.30

Table 3: Convergence times, in simulation time units, for LRU and PGA caching strategies with different routing variants. Total simulation time is 5K time units. In almost all cases, convergence to steady state occurs much faster than our warm-up period (1K time units).

and 5000 time units of the simulation; that is, if t_i are the measurement times, then $\bar{C}_{\mathsf{SR}} = \frac{1}{t_{\mathsf{tot}} - t_w} \sum_{t_i : \in [t_w, t_{\mathsf{tot}}]} C_{\mathsf{SR}}(\rho(t_i), X(t_i))$.

Performance w.r.t Routing Costs. The relative performance of the different strategies to our algorithm is shown in Figure 3. With the exception of `cycle` and `watts-strogatz`, where paths are scarce, we see several common trends across topologies. First, simply moving from RNS routing to uniform, multi-path routing, reduces the routing cost by a factor of 10. Even without optimizing routing or caching, simply increasing path options increases the available caching capacity. For all caching policies, optimizing routing through the dynamic routing policy (denoted by `-D`), reduces routing costs by another factor of 10. Finally, jointly optimizing routing and caching leads to a reduction by an additional factor between 2 and 10 times. In several cases, PGA outperforms RNS routing (including [26]) by 3 orders of magnitude.

Convergence. In Table 3, we show the convergence time for the different variants of LRU and PGA-convergence times for other algorithms can be found in our techreport [27]. We define the convergence time to be the time at which the time-average caching gain reaches 95% of the expected caching gain attained at steady state. LRU converges faster than PGA, though it converges to a sub-optimal stationary distribution. Interestingly, both `-U` and adaptive routing reduce convergence times for PGA, in some cases (like `grid-2d` and `dtelekom`) to the order of magnitude of LRU: this is because path diversification reduces contention: it assigns contents to non-overlapping caches, which are populated quickly with distinct contents.

6 CONCLUSIONS

We have constructed joint caching and routing schemes with optimality guarantees for arbitrary network topologies. Identifying schemes that lead to improved approximation guarantees, especially on the routing cost directly rather than on the caching gain, is an important open question. Equally important is to incorporate queuing and congestion. In particular, accounting for queueing delays and identifying delay-minimizing strategies is open even under fixed routing. Such an analysis can also potentially be used to understand how different caching and routing schemes affect both delay optimality and throughput optimality.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Navid Abedini and Srinivas Shakkottai. 2014. Content caching and scheduling in wireless networks with elastic and inelastic traffic. *IEEE/ACM Transactions on Networking* 22, 3 (2014), 864–874.

[2] Dimitris Achlioptas, Marek Chrobak, and John Noga. 2000. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science* 234, 1 (2000), 203–218.

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

[4] David Applegate, Aaron Archer, Vijay Gopalakrishnan, Seungjoon Lee, and Kadangode K Ramakrishnan. 2010. Optimal content placement for a large-scale VoD system. In *CoNext*.

[5] Ivan Baev, Rajmohan Rajaraman, and Chaitanya Swamy. 2008. Approximation algorithms for data placement problems. *SIAM J. Comput.* 38, 4 (2008), 1411–1429.

[6] Yair Bartal, Amos Fiat, and Yuval Rabani. 1995. Competitive algorithms for distributed data management. *J. Comput. System Sci.* 51, 3 (1995), 341–358.

[7] Daniel M Batista, Nelson LS Da Fonseca, and Flavio K Miyazawa. 2007. A set of schedulers for grid networks. In *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 209–213.

[8] Daniel S Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. 2014. Exact analysis of TTL cache networks. *IFIP Performance* (2014).

[9] Sem Borst, Varun Gupta, and Anwar Walid. 2010. Distributed caching algorithms for content distribution networks. In *INFOCOM*.

[10] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. 2007. Maximizing a submodular set function subject to a matroid constraint. In *Integer programming and combinatorial optimization*. Springer, 182–196.

[11] Giovanna Carofiglio, Léonce Mekinda, and Luca Muscariello. 2016. Joint forwarding and caching with latency awareness in information-centric networking. *Computer Networks* 110 (2016), 133–153.

[12] Hao Che, Ye Tung, and Zhijun Wang. 2002. Hierarchical web caching systems: Modeling, design and experimental results. *Selected Areas in Communications* 20, 7 (2002), 1305–1314.

[13] Raffaele Chiocchetti, Dario Rossi, Giuseppe Rossini, Giovanna Carofiglio, and Diego Perino. 2012. Exploit the known or explore the unknown?: Hamlet-like doubts in icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*. ACM, 7–12.

[14] Edith Cohen and Scott Shenker. 2002. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM*.

[15] T Cormen, C Leiserson, R Rivest, and C Stein. 2009. *Introduction to Algorithms*. MIT Press.

[16] Asit Dan and Don Towsley. 1990. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *SIGMETRICS*, Vol. 18. ACM.

[17] Mostafa Dehghan, Laurent Massoulie, Don Towsley, Daniel Menasche, and YC Tay. 2015. A utility optimization approach to network cache design. In *INFOCOM*.

[18] Mostafa Dehghan, Anand Seetharam, Bo Jiang, Ting He, Theodoros Salonidis, Jim Kurose, Don Towsley, and Ramesh Sitaraman. 2014. On the complexity of optimal routing and content caching in heterogeneous networks. In *INFOCOM*.

[19] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, KC Ng, Vyas Sekar, and Scott Shenker. 2013. Less pain, most of the gain: Incrementally deployable icn. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 147–158.

[20] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. 1992. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics* 39, 3 (1992), 207–229.

[21] Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. 2006. Tight approximation algorithms for maximum general assignment problems. In *SODA*.

[22] N Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. 2012. Analysis of TTL-based cache networks. In *VALUETOOLS*.

[23] Christine Fricker, Philippe Robert, and James Roberts. 2012. A versatile and accurate approximation for LRU cache performance. In *ITC*.

[24] Erol Gelenbe. 1973. A unified approach to the evaluation of a class of replacement algorithms. *IEEE Trans. Comput.* 100, 6 (1973), 611–618.

[25] Brian Guenter, Navendu Jain, and Charles Williams. 2011. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *INFOCOM, 2011 Proceedings IEEE*. IEEE, 1332–1340.

[26] Stratis Ioannidis and Edmund Yeh. 2016. Adaptive Caching Networks with Optimality Guarantees. In *ACM SIGMETRICS*.

[27] Stratis Ioannidis and Edmund Yeh. 2017. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. (2017). http://arxiv.org/abs/1708.05999.

[28] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. 2009. Networking named content. In *CoNEXT*.

[29] Joe Wenjie Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. 2012. Joint VM placement and routing for data center traffic engineering. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2876–2880.

[30] WF King. 1971. *Analysis of paging algorithms*. Technical Report. Thomas J. Watson IBM Research Center.

[31] James F Kurose and Keith W Ross. 2007. *Computer networking: a top-down approach*. Addison Wesley.

[32] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. 2006. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation* 63, 7 (2006), 609–634.

[33] Nikolaos Laoutaris, Sofia Syntila, and Ioannis Stavrakakis. 2004. Meta algorithms for hierarchical web caches. In *ICPCC*.

[34] Wubin Li, Johan Tordsson, and Erik Elmroth. 2011. Virtual machine placement for predictable and time-constrained peak loads. In *International Workshop on Grid Economics and Business Models*. Springer, 120–134.

[35] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. 2002. Search and replication in unstructured peer-to-peer networks. In *ICS*.

[36] Valentina Martina, Michele Garetto, and Emilio Leonardi. 2014. A unified approach to the performance analysis of caching systems. In *INFOCOM*.

[37] KP Naveen, Laurent Massoulié, Emmanuel Baccelli, Aline Carneiro Viana, and Don Towsley. 2015. On the Interaction between Content Caching and Request Assignment in Cellular Cache Networks. In *ATC*.

[38] Christos H Papadimitriou and Kenneth Steiglitz. 1982. *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

[39] Konstantinos Poularakis, George Iosifidis, and Leandros Tassiulas. 2013. Approximation caching and routing algorithms for massive mobile data delivery. In *GLOBECOM*.

[40] Ioannis Psaras, Wei Koong Chai, and George Pavlou. 2012. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*. ACM, 55–60.

[41] Elisha J Rosensweig, Jim Kurose, and Don Towsley. 2010. Approximate models for general cache networks. In *INFOCOM, 2010 Proceedings IEEE*. IEEE, 1–9.

[42] Elisha J Rosensweig, Daniel S Menasche, and Jim Kurose. 2013. On the steady-state of cache networks. In *INFOCOM*.

[43] Dario Rossi and Giuseppe Rossini. 2011. *Caching performance of content centric networks under multi-path routing (and more)*. Technical Report. Telecom ParisTech.

[44] Giuseppe Rossini and Dario Rossi. 2014. Coupling caching and forwarding: Benefits, analysis, and implementation. In *Proceedings of the 1st international conference on Information-centric networking*. ACM, 127–136.

[45] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. 2013. Femtocaching: Wireless content delivery through distributed caching helpers. *Transactions on Information Theory* 59, 12 (2013), 8402–8413.

[46] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. 2010. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 228–235.

[47] Jan Vondrák. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*.

[48] Yonggong Wang, Zhenyu Li, Gareth Tyson, Steve Uhlig, and Gaogang Xie. 2013. Optimal cache allocation for content-centric networking. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*. IEEE, 1–10.

[49] Haiyong Xie, Guangyu Shi, and Pengwei Wang. 2012. TECC: Towards collaborative in-network caching guided by traffic engineering. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2546–2550.

[50] Edmund Yeh, Tracey Ho, Ying Cui, Michael Burd, Ran Liu, and Derek Leong. 2014. VIP: A framework for joint dynamic forwarding and caching in named data networks. In *ICN*.

[51] Yuanyuan Zhou, Zhifeng Chen, and Kai Li. 2004. Second-level buffer cache management. *Parallel and Distributed Systems* 15, 6 (2004), 505–519.