DR-Cache: Distributed Resilient Caching with Latency Guarantees

Jian Li[1,*], Truong Khoa Phan[2,*], Wei Koong Chai[3], Daphne Tuncer[2], George Pavlou[2], David Griffin[2], Miguel Rio[2]
[1]University of Massachusetts Amherst, [2]University College London, [3]Bournemouth University
[1]jianli@cs.umass.edu, [2]{t.phan,d.tuncer,g.pavlou,dgriffin,miguel.rio}@ucl.ac.uk, [3]wchai@bournemouth.ac.uk
*Co-primary authors

***Abstract*—The dominant application in today's Internet is content streaming, which is increasingly relying on caches to meet the stringent conditions on the latency between content servers and end-users. These systems routinely face the challenges of limited bandwidth capacities and network server failures, which degrade caching performance. In this paper, we study the problem of optimally allocating content over a resilient caching network, in which each cache may fail under some situations. Given content request rates and multiple routing paths, we formulate an optimization problem to maximize the expected caching gain, i.e., the reduction of latency due to intermediate caching. The offline version of this problem is NP-hard. We first propose a centralized, offline algorithm and show that a solution with (1-1/e) approximation ratio to the optimal can be constructed. We then propose a distributed ascent algorithm based on the concave relaxation of the expected gain. Informed by the results of our analysis, we finally propose a distributed resilient caching algorithm (DR-Cache) that is simple and adaptive to network failures. We show numerically that DR-Cache significantly outperforms other candidate algorithms under synthetic requests, as well as real world traces over a class of network topologies.**

I. Introduction

Today a common method of delivering content to end users is through the use of caching infrastructures [24] [16]. By keeping copies of content in a set of universally deployed or distributed cache nodes, these infrastructures enable content requests to be served as close as possible to the end users, reducing as such latency costs and contributing to improve quality of experience.

As the Internet is evolving, however, its resilience to failures is becoming more critical. By analyzing the IS-IS routing updates of the Sprint network, an operational IP backbone network, the authors in [18] showed that failures are not uncommon events in networks. In addition, it was demonstrated in [12] that network failures can even occur in highly reliable data centers. In recent years, several initiatives have investigated edge cloud computing architectures to enable caching services in the network edge [14], [25]. However, small servers deployed in edge clouds are more likely to fail due to power supply shutdowns, hardware failures and so on [11]. In fact, it is reported that IT downtime due to equipment failures costs enterprises billions of dollars a year [1]. In the area of information-centric networking (ICN), the authors in [6] investigated the provision of information resilience through a caching approach based on the concept of modularity.

Failures and caching performance are inherently tied together in cache networks since failures result in further searching and routing which, in turn, can significantly increase latency. Substantial effort has been invested over time in the development of novel approaches to drive caching decisions in cache networks, e.g., content/service placement strategies [2], [3], [5], [17], [21], [22] and replication schemes [7], [9], [20], [26]. However, one of the main limitations of these approaches is that they do not explicitly take network failures into account when making caching decisions.

In this paper, we propose an optimal and distributed content allocation algorithm (DR-Cache) that maximizes the expected caching gain in the presence of failures across a cache network. DR-Cache is a lightweight algorithm that does not need a priori knowledge of content request patterns and requires only a minimal quantity of information exchanged between cache nodes. In addition, network failure probability is explicitly taken into account in caching decision which shows significant improvement on system performance compared with traditional non-resilient caching algorithms.

The main contributions of this paper are as follows:
(1) We formulate an optimization problem to determine the optimal content allocation across a resilient cache network, which captures both routing latency and network failures. The deterministic and combinatorial version of this problem is NP-hard. We propose a centralized, offline algorithm to achieve a solution with an approximation ratio of $1 - 1/e$.
(2) Since the request information and network topology is usually not known in advance, we then propose a distributed gradient ascent algorithm based on the concave relaxation of the expected caching gain, and show that it can converge to a solution with a $(1 - 1/e)$ approximation ratio to the optimal.
(3) Informed by the results of our analysis, we propose a distributed resilient caching algorithm (DR-Cache) that is simple and adaptive to network failures.
(4) We conduct extensive numerical studies under both synthetic requests and real world traces over a range of network topologies. We show that DR-Cache significantly outperforms other candidate algorithms. Content retrieval latency is reduced

This research was sponsored in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, in part by CHIST-ERA CONCERT under Grant I1402, and in part by H2020 5G-MEDIA project under Grant 761699.

by up to 57%, while the number of failed requests is reduced by 8% and routing overhead is reduced by 16.7%.

Related Work and Organization

The key focus of this work is to propose a content allocation scheme to incorporate network failures, specifically in caching systems, to achieve low latency. Our ideas are built and borrowed from a large body of existing work in this area. However, to the best of our knowledge, DR-Cache is the first caching system that employs replication and captures network failures to reduce latencies.

Due to network failures, maintaining a single copy of each piece of content in a cache network can result in low performance due to content unavailability. Replication [9], [23] has been used to increase content redundancy, which further improves the performance in terms of cache hits. However, [15] showed that simple universal replication strategies are not optimal when there is a significant variation in the popularity of content: cache space can be wasted on less popular contents.

Many studies provide insights on the offline caching optimization problem [2], [5], [10], [13], [19], [20]. The one most close to our problem is [15], we rely and expand upon this work. [15] considered a caching network and assumed that only a fixed and unique path is used for routing the request and that all caches (servers) are stable. However, in our framework, due to network failures, i.e., instability of each node, we assume that multiple paths are available for each request. These are the two major differences with [15]. Since we capture network failures using multiple paths, showing that (i) there exists a centralized, offline algorithm that can achieve a constant approximation ratio to the optimal; and (ii) there exists a distributed resilient caching algorithm that is simple and adaptive to network failures without prior knowledge of network topology and request information, are quite intricate. These are main contributions in this paper.

The paper is organized as follows. We introduce the model and formally formulate our optimization problem in Section II. We present the centralized algorithm in Section III. Distributed gradient ascent algorithm and distributed resilient caching algorithm (DR-Cache) are presented in Section IV. Finally, we provide extensive numerical results in Section V, and we present our conclusions in Section VI.

II. Analytical Model

We consider a general cache network, in which each node has a finite cache capacity to store content and failures can occur on any node. We use "node" as a general term to indicate a router in Information/Content Centric Networks (ICN/CCN [16]) or a server in Content Distribution Networks (CDN) [24] and edge cloud clusters [14], [25]. We assume that there exists an origin server, which is the persistent store of the content requested by end users. Each end user generates requests for content, which are forwarded through the network towards the origin server. Content requests are satisfied on the occurrence of the first "cache hit". Here, we consider the case that there exists multiple paths for each request, but only one path is used for routing each time. Whenever a cache hit occurs, the requested content is sent back along the reverse direction of the path. Intermediate nodes can then cache the requested content to serve future requests. Our objective is to determine the optimal content allocation that maximizes the expected caching gain.

Cache Network: We represent the cache network as a general graph $\mathcal{G} = (V, E)$. Each node has a finite cache capacity, denote it as $C_j \in \mathbb{N}_+$ for $j \in V$. Let $\mathcal{N} = \{1, 2, \cdots, n\}$ be the set of equal-size content in the library considered in our system. For each node $j \in V$ and content $i \in \mathcal{N}$, we denote

$$x_{j,i} = \begin{cases} 1, & \text{if node } j \text{ caches content } i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Denote the global allocation as $X = \{x_{j,i}\}_{j \in V, i \in \mathcal{N}} \in \{0,1\}^{|V| \times |\mathcal{N}|}$. Thus based on the capacity constraint on each node, we have

$$\sum_{i \in \mathcal{N}} x_{j,i} \leq C_j, \quad \forall j \in V. \tag{2}$$

Network Resilience: We capture the resilience of each node with a stability score $s_j \in [0, 1]$ for $j \in V$. A higher value of s_j means node j is more stable, while a smaller s_j value means that node j is more likely to fail. In other words, node j fails with probability $(1 - s_j)$ over the timeframe under consideration. As shown in [11], [12], the failure probability of a node is affected by a combination of several factors. However, deriving an exact model to predict the failure probability is out of the scope of this work.

A request routed along a path fails if one of the intermediate nodes on the path is broken. For example, given a path with four nodes (p_1, p_2, p_3, p_4) and suppose node p_3 caches the requested content. If node p_2 breaks, the request fails since it cannot reach node p_3. Therefore, it is prudent for each node to take the stability information (including that of itself and other nodes along the path) into account when it decides to cache the content. In this work, we propose an algorithm under which node j makes a decision to cache the requested content i with probability $f_{j,i}$, which relies on the stability of each of the nodes on the path. More details are presented in Section IV.

Content Requests and Routing Latency: We assume that requests to content are routed along paths in $\mathcal{G}$. We denote a request as $(i, \mathcal{P}^i)$, where i is the requested content, and $\mathcal{P}^i$ is a set of corresponding paths. We assume that each content i can be routed along a set of possible paths $\mathcal{P}^i = (\mathcal{P}^i_1, \cdots, \mathcal{P}^i_I)$, for constant $I \in \mathbb{N}_+$. The probability of routing along path $\mathcal{P}^i_l$ is given by q^i_l, we denote the probability distribution over $\mathcal{P}^i$ as $\mathcal{Q}^i = (q^i_1, \cdots, q^i_I)$, satisfying $\sum_{l=1}^{I} q^i_l = 1$. Once a request is generated, one path is used for routing, and a new path is selected only if that path fails. Formally, a path $\mathcal{P}^i_l$ is a sequence $\{p^i_{l,1}, \cdots, p^i_{l,|\mathcal{P}^i_l|}\}$ of nodes $p^i_{l,j}$ such that the edge $(p^i_{l,j}, p^i_{l,j+1}) \in E$, for $j = 1, \cdots, |\mathcal{P}^i_l| - 1$. We say that a request $(i, \mathcal{P}^i)$ is well routed in $\mathcal{G}$, if there exist a set of paths $\mathcal{P}^i$ and an origin server, which is the end node $p^i_{l,|\mathcal{P}^i_l|}$ on each possible path $\mathcal{P}^i_l$, that contains the requested content i.

We denote all the requests as $\mathcal{R}$. Also each request arrives according to an independent Poisson process, we denote the request rate by $\lambda_{(i,\mathcal{P}^i)}$ for request $(i,\mathcal{P}^i) \in \mathcal{R}$.

A request $(i,\mathcal{P}^i) \in \mathcal{R}$ is routed over the cache network $\mathcal{G}$ following a path $\mathcal{P}_l^i \in \mathcal{P}^i$. Once it reaches a cache storing that content, a response message is generated and carries the requested content and the stability information of each node on the path. The response message is passed along the reverse direction of $\mathcal{P}_l^i$ to the end-user who sent the request. We assume that each edge along the path has a delay (latency) $d_{(u,v)} > 0, \forall (u,v) \in E$. Since we consider a symmetric routing, i.e., $d_{(u,v)} = d_{(v,u)}$, the delay from searching and responding are identical, our objective of maximizing caching gain is equivalent to maximizing caching gain from response messages. Therefore, in the following, we only consider the delay from response messages. For brevity, in the remainder of this section, we remove the superscript $^{\cdot i}$ of each node $p_{l,j}^i$ on path $\mathcal{P}_l^i \in \mathcal{P}^i$.

Therefore, if a request $(i,\mathcal{P}^i) \in \mathcal{R}$ is well routed in the network $\mathcal{G}$, the corresponding delay can be expressed as

$$\begin{aligned} &L_{(i,\mathcal{P}^i)} = L_{(i,\mathcal{P}^i)}(X) \\ &= \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}} \prod_{j}^{k} (1 - x_{p_{l,j},i}), \end{aligned} \tag{3}$$

where $u(\cdot)$ is a continuous and decreasing convex function on $[0,1]$ with $u(0) < M$ for $M < \infty$ and $u(1) = 1$. It captures the impact of node stability along the path on the latency. Intuitively, (3) means that if node $p_{l,k+1}$ caches content i, then the corresponding delay should be the sum of delay associated with all edges on the reverse path from node $p_{l,k+1}$ to the end-user, since all nodes between them do not store content i. Equation (3) represents the expected delay a request $(i,\mathcal{P}^i)$ can experience when it is routed along the path $\mathcal{P}^i$. Note that if any of the intermediate nodes between $p_{l,1}$ and $p_{l,k+1}$ breaks, the request on this path fails, which results in a large latency. Thus, we use $u(\cdot)$ in (3) to capture this feature. For example, if node $p_{l,k}$ is unstable, the value of $u(s_{p_{l,k}})$ is high, then another path with more stable nodes might be selected to minimize the corresponding delay for the request $(i,\mathcal{P}^i)$.

Therefore, the total expected delay for all requests $(i,\mathcal{P}^i) \in \mathcal{R}$ is defined as

$$\begin{aligned} L(X) = &\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} L_{(i,\mathcal{P}^i)} = \\ &\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}} \prod_{j}^{k} (1 - x_{p_{l,j},i}). \end{aligned} \tag{4}$$

Caching Gain Optimization: Now, if we consider the case that there is no cache available in the network, i.e., all requests from end-users are served by the origin server, then the expected delay is given as

$$L_0 = \sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}}. \tag{5}$$

Our objective is to find a feasible allocation X to minimize the total expected delay, which is equivalent to maximizing the expected caching gain, defined as

$$\begin{aligned} G(X) = L_0 - L(X) = &\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) \\ &\cdot d_{p_{l,k+1}p_{l,k}} \left(1 - \prod_{j}^{k} (1 - x_{p_{l,j},i})\right). \end{aligned} \tag{6}$$

Hence, we consider the following optimization problem

$$\begin{aligned} \max \quad & G(X) \\ \text{s.t.} \quad & x_{j,i} \in \{0,1\}, \quad \forall j \in V, i \in \mathcal{N}, \\ & \sum_{i\in\mathcal{N}} x_{j,i} \le C_j, \quad \forall j \in V. \end{aligned} \tag{7}$$

Theorem 1. *The optimization problem in (7) is NP-hard.*

The proof is straightforward from [8]. In this paper, we aim at designing a distributed algorithm that captures the resilience of the network and produces a feasible allocation within a constant approximation ratio to the optimal, without knowing the request information and network topology.

III. Centralized Algorithm

Before presenting distributed algorithms, we first consider a centralized, offline algorithm that achieves a constant approximated solution to (7) in polynomial time. It is easy to check that the optimization problem defined in (7) is a submodular optimization problem under matroid constraints [8] and there exists a $1/2$-approximation algorithm that can be constructed. In the following, we consider the *randomized swap rounding* algorithm [8] that can improve the approximation ratio to $1 - 1/e$.

The centralized approximation algorithm consists of two steps: (1) We turn the integer programming problem to a convex optimization problem whose solution is equal to or greater than the optimal one by relaxing the binary variable $x_{j,i}$ from $\{0,1\}$ to $[0,1]$, $\forall j \in V$ and $\forall i \in \mathcal{N}$. We show that the obtained solution is within a constant approximation from the optimal solution to the original problem. (2) We round the possibly fractional solutions from the convex optimization to possible solutions to the original integer problem. In the following, we present the key steps of the application of *randomized swap rounding scheme* [8] in our problem, and refer the interested reader to [8] for further details.

A. Convex Relaxation

The objective of convex relaxation is to relax the original integer programming to a convex optimization problem. Suppose that the binary variables $x_{j,i}$ for all $j \in V$ and $i \in \mathcal{N}$ are independent Bernoulli random variables. Let $\mathcal{F}$ be the corresponding joint probability distribution in $[0,1]^{|V|\times|\mathcal{N}|}$. Denote $f_{j,i}$ as the marginal probability that node $j \in V$ caches content $i \in \mathcal{N}$. Then, we have

$$f_{j,i} = \mathbb{P}_{\mathcal{F}}[x_{j,i} = 1] = \mathbb{E}_{\mathcal{F}}[x_{j,i}], \tag{8}$$

where $\mathbb{P}_{\mathcal{F}}[\cdot]$ and $\mathbb{E}_{\mathcal{F}}[\cdot]$ are the probability and expectation with respect to (w.r.t.) $\mathcal{F}$. Denote the corresponding relaxed global allocation as $F = \{f_{j,i}\}^{|V|\times|\mathcal{N}|}$. Then we have

$$\begin{aligned}
&\mathbb{E}_{\mathcal{F}}[G(X)] \\
=&\mathbb{E}_{\mathcal{F}}\Bigg[\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}} \\
&\qquad \cdot \left(1-\prod_j (1-x_{p_{l,j},i})\right)\Bigg] \\
=&\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}} \\
&\qquad \cdot \mathbb{E}_{\mathcal{F}}\left[1-\prod_j (1-x_{p_{l,j},i})\right] \\
\overset{(a)}{=}&\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}} \\
&\qquad \cdot \left(1-\prod_j (1-\mathbb{E}_{\mathcal{F}}\left[x_{p_{l,j},i}\right])\right) \\
\triangleq& G(F),
\end{aligned} \tag{9}$$

where (a) holds true since all nodes are independent and unique on a feasible path. This is known as multi-linear relaxation of $G(X)$ [8]. Now consider the following relaxed convex optimization problem

$$\begin{aligned}
\max \quad & G(F) \\
\text{s.t.} \quad & f_{j,i} \in [0,1], \quad \forall j \in V, i \in \mathcal{N}, \\
& \sum_{i\in\mathcal{N}} f_{j,i} \le C_j, \quad \forall j \in V.
\end{aligned} \tag{10}$$

If we denote the optimal solutions to (7) and (10) by X^* and F^*, respectively. Then we immediately have

$$G(X^*) \le G(F^*), \tag{11}$$

since (10) achieves the maximization of the same objective function over a larger domain.

However, (9) is not concave, in order to achieve a convex optimization problem, we consider the following approximated function

$$\begin{aligned}
\tilde{G}(\tilde{F}) = &\sum_{(i,\mathcal{P}^i)\in\mathcal{R}} \lambda_{(i,\mathcal{P}^i)} \sum_{l=1}^{I} q_l^i \sum_{k=1}^{|\mathcal{P}_l^i|-1} u(s_{p_{l,k+1}}) d_{p_{l,k+1}p_{l,k}} \\
&\cdot \min\left\{1, \sum_{j=1}^{k} f_{p_{l,j},i}\right\},
\end{aligned} \tag{12}$$

which is concave.

Now we consider the following relaxed approximated convex optimization problem

$$\begin{aligned}
\max \quad & \tilde{G}(\tilde{F}) \\
\text{s.t.} \quad & f_{j,i} \in [0,1], \quad \forall j \in V, i \in \mathcal{N}, \\
& \sum_{i\in\mathcal{N}} f_{j,i} \le C_j, \quad \forall j \in V.
\end{aligned} \tag{13}$$

Theorem 2. *[8] Given the optimal solutions F^* to (10) and $\tilde{F}^*$ to (13), we have*

$$\left(1-\frac{1}{e}\right) G(F^*) \le \tilde{G}(\tilde{F}^*) \le G(F^*). \tag{14}$$

Hence, from (11) and (14), we know the optimal solution to (10) is within a constant factor from that of (7). Also note that the convex optimization problem (13) can be solved in strongly polynomial time [8]. In the following, we show that the optimal solution $\tilde{F}^*$ can be rounded to a feasible solution to (7) in finite steps.

B. Randomized Rounding

Convex relaxation enables us to solve a convex optimization problem and obtain an optimal solution $\tilde{F}^*$, however, this solution may not satisfy the constraint of the original problem, i.e., $\tilde{F}^*$ may contain fractional solutions. Hence, to produce a constant approximation solution to (7), $\tilde{F}^*$ has to be rounded. Here, we use the randomized swap rounding scheme [8].

We use the following rounding steps: (i) Denote $f_{j',i}$ as a convex combination of $f_{p_{l,j},i}$, i.e., $f_{j',i} = \sum_{l=1}^{I} q_l^i f_{p_{l,j},i}$, where $\sum_{l=1}^{I} q_l^i = 1$; (ii) Merge any two components in $f_{j',i}$ into a new component with the coefficient equals to the sum of the corresponding coefficients of the merged two components; (iii) Continue the merge operation of $I-1$ steps to obtain the new variable $f_{j',i}$, and repeat the process for all possible content $i \in \mathcal{N}$, after which we obtain $|\mathcal{N}| \times |V|$ variables for the caching probability of these content in the network; (iv) If all are integer, we are done; otherwise, pick two fractional variables, since the capacity is integer, there must exist at least two fractional solutions; (v) transform the mass between these two variables, after which at least one of them becomes 0 or 1 and the objective function is at least as good as before after the transform [8]; (vi) repeat step (iv) and (v) until there are no fractional solutions. Denote the obtained integral solution as $\tilde{X}$. The caching gain G after each rounding step is at least as good as that at previous step [8].

Theorem 3. *Following the above randomized rounding policy, the rounding solutions $\tilde{X}$ satisfies*

$$\left(1-\frac{1}{e}\right) G(X^*) \le \left(1-\frac{1}{e}\right) G(F^*) \le \tilde{G}(\tilde{F}^*) \le G(\tilde{X}), \tag{15}$$

hence there exists a $\left(1-\frac{1}{e}\right)$ approximation solution to (7).

Since the number of variables in the networks is finite, the above rounding steps conclude in finite steps.

IV. Distributed Resilient Caching Algorithm

In many applications, the requests and network topology are usually dynamic and not a priori known. Hence, a distributed algorithm that can provide an optimal content allocation without knowing the requests and adapt to changes are preferred. In this section, we design distributed algorithms for *caching decision-making* to improve content reachability in resilient caching networks.

A. Gradient Ascent Algorithm

A natural decentralized approach to maximizing $\tilde{G}(\tilde{F})$ is to gradually move the random variables towards the optimal point using the gradient ascent algorithm. Since $\tilde{G}(\cdot)$ is concave, the algorithm converges to a solution within a constant approximation ratio $(1-1/e)$ to the optimal solution to the original optimization problem (7) by Theorem 3.

However, we cannot directly compute the gradients from a centralized viewpoint since $\tilde{G}(\cdot)$ is not differentiable over the whole feasible region, we need to consider the subgradient. Similar to the solution in [15], each node needs to estimate the subgradient relying only on the local request and response information. Once we achieve a solution to (13), which may contain a fractional solution, we need to determine the allocation of content in each cache. Since each node needs to determine the cache content in a distributed way, we cannot directly use the randomized swap rounding.

We address both these issues by showing that the subgradients can be estimated in a distributed way only with information arrival at each node, and we can easily obtain a feasible randomized rounding solution. The proposed approach is based on the algorithm presented in [15] that we significantly extend by taking into account multi-path routing and network failures. We briefly introduce the gradient ascent algorithm as applied to our problem, referring the interested readers to [15] for further details.

Algorithm Overview: We measure the information and update the caching allocation upon each request. Each node has its own marginal $f_j \in [0,1]^{|\mathcal{N}|}$ for $j \in V$, i.e., j keeps track of its own marginal probability $f_{j,i}$ of caching content i. We call f_j as the state of node j. Upon each request, each node adapts its state vector f_j and then reshuffles the content in the cache.

To be more specific, upon the k-th request, node j first estimates its own subgradient g_j of $\tilde{G}(\tilde{F})$ w.r.t.s to f_j with the information available from the request, i.e., $g_j(\tilde{F}^{(k)}) \in \partial_{f_j}\tilde{G}(\tilde{F}^{(k)})$, where $\partial_{f_j}\tilde{G}(\tilde{F}^{(k)})$ is the set of subgradients of $\tilde{G}$ w.r.t. f_j. Given this subgradient, node j computes its own state as

$$f_j(\tilde{F}^{(k+1)}) \leftarrow \mathcal{P}_{\mathcal{S}_j}\left(f_j(\tilde{F}^{(k)}) + \eta_k g_j(\tilde{F}^{(k)})\right), \qquad (16)$$

where $\eta_k > 0$ is the step size and $\mathcal{P}_{\mathcal{S}_j}$ is the projection to node j's set of $\mathcal{S}_j = \{f_j \in [0,1]^{|\mathcal{N}|} : \sum_{i\in\mathcal{N}} f_{j,i} = C_j\}$.

Now we describe how to compute the estimate g_j of the subgradient $\partial_{f_j}\tilde{G}(\tilde{F}^{(k)})$ upon each request, which only requires the information available at each local node from the response message traversing the node. We drop the superscript $^{(k)}$ for brevity. Consider a request $(i,\mathcal{P}^i)$ and suppose that path $\mathcal{P}^i_l \in \mathcal{P}^i$ is selected, then:

(i) a request message is sent out along the path until it hits a node that caches content i. Every time it traverses an edge $(j',j) \in E$, node j adds the latency $d_{j',j}$ to latency counter H and its stability score s_j to counter χ, then we have $H_{j,i} = \sum_{k=1}^{j-1} d_{k,k+1}$, and $\chi_{j,i} = \sum_{k=1}^{j-1} s_k d_{k,k+1}$;

(ii) a response message is sent down in the reverse direction carrying the latency counter information as well as request stability counter χ and cache stability counter ψ. If a node caches the content, it adds its cache stability to the counter ψ and subject the score to counter χ. Since the response message traverses every node on the reverse direction of the path, every node on the path learns the values of these counters: $\psi_{k,i} = \sum_{k}^{j-1} s_k d_{k+1,k} 1_{x_{k,i}=1}$ and $\chi_{k,i} = \sum_{k'=1}^{j-1} s_{k'} d_{k'+1,k'} - \sum_{k}^{j-1} s_k d_{k+1,k}$;

(iii) given the information learned from the request, each node k computes its estimate:

$$f_{k,i} = \left(\chi_{k,i} + \psi_{k,i}\right)/H_{k,i}. \qquad (17)$$

Note that, only nodes on the reverse path that the request $(i,\mathcal{P}^i)$ traverses need to compute it.

Finally, given $f_j(\tilde{F}^{(k+1)})$, each node $j \in V$ reshuffles its cached content independently of other nodes in the network, i.e., node j selects a random variable $x_{j,i} = 1$ with probability $f_{j,i}(\tilde{F}^{(k+1)})$ independently of any other nodes in V with a joint distribution satisfying $\mathcal{F}$.

Remark 1. *It is easy to show that the estimate of the subgradient is unbiased. We skip the technical proofs demonstrating the unbias of the estimator and the convergence of the algorithm due to space constraints.*

B. Distributed Resilient Caching Algorithm

The gradient ascent algorithm has a drawback that the estimation of the gradient and the decision of caching are made at the end of a request, which may incur additional traffic to retrieve the requested content. In the following, we propose a new distributed resilient caching algorithm (DR-Cache) that is simple and does not have the above drawback.

Algorithm Overview: Consider a request $(i,\mathcal{P}^i) \in \mathcal{R}$. Suppose a path $\mathcal{P}^i_l$ is selected from $\mathcal{P}^i$ with probability q^i_l. Note that under our model, a request is routed on only one path each time. For simplicity, we remove the subscribe $_{,l}$, and denote the path as $\mathcal{P}^i$ directly.

Step 1*:* Once the request $(i,\mathcal{P}^i)$ is generated, it is then sent along the path $\mathcal{P}^i$ until it reaches one node that caches the content i, i.e., $x_{j,i} = 1$ for $j = 1,\cdots,|\mathcal{P}^i|$. The request message for a request $(i,\mathcal{P}^i)$ contains a request stability counter χ and the current latency H. Whenever the request traverses a node along the path, the stability score of the node is added to the counter χ and the latency counter H is increased by that along the edge:

$$\chi_{j,i} = \sum_{k=1}^{j-1} s_k d_{k,k+1}, \quad \forall j = 1,\cdots,|\mathcal{P}^i|,$$
$$H_{j,i} = \sum_{k=1}^{j-1} d_{k,k+1}. \qquad (18)$$

Step 2*:* Upon finding the cached content i (suppose it is node j on the path $\mathcal{P}^i$), a response message carrying content i is sent back to the end user along the reverse path. The response message for a request $(i,\mathcal{P}^i)$ contains a request stability counter χ, a cache stability counter ψ and the latency counter H. Note that H does not change along the reverse path.

Therefore, node k on the reverse path learns the following information:

$$H_{k,i} \equiv H_{j,i}, \quad \text{where } x_{j,i} = 1 \text{ and } k = 1, \cdots, j,$$
$$\chi_{k,i} = \chi_{j,i} - \sum_{k}^{j-1} s_k d_{k+1,k}, \quad k = 1, \cdots, j,$$
$$\psi_{k,i} = \sum_{k}^{j-1} s_k d_{k+1,k} 1_{x_{k,i}=1}. \tag{19}$$

Step 3*:* Based on the received information, node k computes the probability of caching content i. This computation captures the stability of nodes on the path, as well as the latency to find a node (say node j as above) on the path that caches content i, which is defined as follows

$$f_{k,i} = 1 - \frac{1}{H_{k,i}} \left(\theta_{\chi,k}\chi_{k,i} + \theta_{\psi,k}\psi_{k,i}\right) \cdot g_k(p_{k,i}), \tag{20}$$

where $p_{k,i}$ is the popularity of content i at node k and $g_k(\cdot)$ is a decreasing smooth function. $\theta_{\chi,k}$ and $\theta_{\psi,k}$ are two weighted parameters, representing the greediness of caching.

Step 4*:* Finally, after computing the probability, node k decides to cache content i with probability $f_{k,i}$. If a content needed to be evicted, then an eviction policy (e.g. random replacement (RR), FIFO or LRU [17]) is applied.

The above steps are summarized in Algorithm 1. Note that node k decides to cache the content directly without computing $f_{k,i}$ if there is available cache capacity (line $13-15$).

Algorithm 1 Distributed Resilient Caching Algorithm

1: *When node k receives a message*
2: **if** request message for content i from node k' **then**
3: $H_{k,i} = H_{k',i} + d_{k',k}$
4: **if** k has the requested content **then**
5: $\psi_{k,i} = s_k d_{k',k}$
6: $\chi_{k,i} = \chi_{k',i} - s_k d_{k',k}$
7: send the response message back
8: **else**
9: $\chi_{k,i} = \chi_{k',i} + s_k d_{k',k}$
10: forward the request to next hop
11: **else if** response message with content i from node k' **then**
12: update content popularity $p_{k,i}$
13: $\chi_{k,i} = \chi_{k',i} - s_k d_{k',k}$
14: **if** $C_k > 0$ **then** *//C_k is available capacity of k*
15: cache the content, update $C_k = C_k - 1$
16: $\psi_{k,i} = \psi_{k',i} + s_k d_{k',k}$
17: **else**
18: Compute caching probability $f_{k,i}$
19: **if** the decision is to cache the content **then**
20: $\psi_{k,i} = \psi_{k',i} + s_k d_{k',k}$
21: Replace the content using an eviction policy
22: forward the response message to the next hop

Remark 2. *Some key ideas behind Algorithm 1:*
(a) χ captures the stability of all nodes on the path towards the end user. ψ indicates the stability of the nodes that have cached the content on the response path.
(b) From Equation (20), we can see that (i) If $(\chi+\psi)/H$ is small, the content should be cached only in unstable nodes, thus the current node should cache the content with higher probability such that the content could serve future requests even if those unstable nodes fail; (ii) If $(\chi+\psi)/H$ is large, the content should be cached in stable nodes. Thus the current node sets lower caching probability for this content to save space for other content; (iii) If the content i is more popular at node k, then it should be cached with a higher probability, this is captured by the popularity function $g_k(\cdot)$.
(c) Caching probability increases when response message is close to end-user (see the example in section IV-C).

C. Illustrative Example

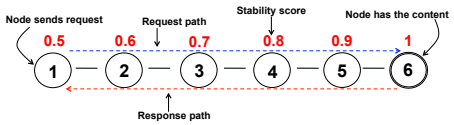


Fig. 1: Illustrative Example of a Simple Path.

We begin our analysis of the aforementioned insights with a simple but representative example as shown in Figure 1, where node 6 stores the requested content. We consider a simple case when $g_k(p_{k,i}) = 1$ and $\theta_{\chi,k} = \theta_{\psi,k} = 1$ for $\forall i \in \mathcal{N}, k \in V$, then the decision probability in (20) reduces to

$$f_{k,i} = 1 - \left(\chi_{k,i} + \psi_{k,i}\right)/H_{k,i}. \tag{21}$$

W.l.o.g., we assume the latency of each link is 1, and stability scores are given as 0.5, 0.6, 0.7, 0.8, 0.9 and 1, respectively.

TABLE I: $\chi_{k,i}$, $\psi_{k,i}$ and $f_{k,i}$ updated on the response path

Node	$\chi_{k,i}$	$\psi_{k,i}$	$H_{k,i}$	$f_{k,i}$	$\chi^*_{k,i}$	$\psi^*_{k,i}$
6	4.5	0	6		3.5	1
5	3.5	1	6	0.25	2.6	1
4	2.6	1	6	0.4	1.8	1
3	1.8	1	6	0.53^*	1.1	$1+0.7=1.7$
2	1.1	1.7	6	0.53	0.5	1.7
1	0.5	1.7	6	0.63^*	0	$1.7+0.5=2.2$

For the request message, (χ, H) is carried along the message, and based on Algorithm 1, the information updated from node 1 to node 6 are as follows: $(0.5, 1)$, $(1.1, 2)$, $(1.8, 3)$, $(2.6, 4)$, $(3.5, 5)$ and $(4.5, 6)$. Since node 6 caches the requested content, a "cache hit" occurs. Upon the occurrence of a cache hit, the content is sent back along the reverse direction of the path. A response message containing (χ, ψ, H) are sent back at the same time. Each node learns this information, updates its own (χ, ψ, H) and then computes its own probability of caching this content through Equation (21) (assume that all nodes are full, i.e. $C_k = 0$). Denote (χ_j, ψ_j, H_j) as the message node j receives from parent node and $(\chi^*_j, \psi^*_j, H_j)$ as the message node j sends to its child node. Hence, we have $(\chi_j, \psi_j, H_j) = (\chi^*_{j+1}, \psi^*_{j+1}, H_{j+1})$ for $j = 1, \cdots, 5$. From the request message, we have $(\chi_6, \psi_6, H_6) = (4.5, 0, 6)$. Since node 6 caches the content, then $(\chi^*_6, \psi^*_6, H_6) = (3.5, 1, 6) = (\chi_5, \psi_5, H_5)$. Given this information, nodes compute the probability f_k based on (21) as shown in Table I. Suppose nodes 3 and 1 decide to cache the content, then they add their stabilities to ψ^*.

V. Numerical Studies

We evaluate the performance of Algorithm 1 (DR-Cache) through a series of experiments using both synthetic and real traffic traces over a class of network topologies. We compare DR-Cache with traditional non-resilient path replication (PR) algorithms using different eviction policies: random replacement (RR), FIFO and LRU [17]. We find that DR-Cache can significantly outperform traditional PR algorithms. The highlights of the evaluation results are:

- DR-Cache can reduce the number of failed requests up to 8%, and improve the caching gain as much as 57%, compared to PR.
- DR-Cache reduces the routing overhead as much as 16.7% compared to PR.

TABLE II: Network Topologies and Experiment Parameters

Network	$\|V\|$	$\|E\|$	$\|\mathcal{N}\|$	$\|\mathcal{R}\|$	$\|\mathcal{N}^*\|$	$\|\mathcal{R}^*\|$
Abilene	12	15	70	20352	73	2857
Nobel EU	28	41	162	27119	169	4144
Zib54	54	81	313	33169	325	9382
Erdos-Renyi	100	915	578	37159	600	13432

A. Methodology

Experiment setup: The networks we consider are summarized in Table II, including one synthetic network (Erdos-Renyi (ER)) and three real backbones. The size of these networks ranges from 12 to 100 nodes. $|\mathcal{N}|$ and $|\mathcal{R}|$ represent the number of unique contents and requests under synthetic traffic, respectively, whereas $|\mathcal{N}^*|$ and $|\mathcal{R}^*|$ are corresponding numbers under real traffic traces [4]. The failure probability of devices is usually less than 50% in real systems [12], [18]. Thus, we associate each node (cache) with a stability score, randomly chosen as $s_j \in [0.5, 1]$ in numerical experiments. As nodes' stabilities are stochastic variables, we run each simulation 100 times to get the mean result. We evaluate the performance of candidate algorithms by varying the cache size of each node from $1 - 10\%$ of the total number of content.

We generate the synthetic traffic as follows. We first randomly select an origin server and assume that it has all the requested contents. To ensure path overlap, we randomly select $|\mathcal{Q}|$ nodes from V that are the only nodes to generate requests. We generate a set of requests starting from a random node in $\mathcal{Q}$, with the requested content selected from $\mathcal{N}$ following a Zipf distribution with parameter 1.2. In order to confirm that our results also hold in real systems, we run similar experiments using traces from IRCache project [4], with an attention on data gathered from the SD Network Proxy in Feb. 2013. A detailed study shows that these traces capture the regional traffic and exhibit significant non-stationaries due to daily traffic fluctuations.

Evaluation metrics: Our primary metrics for comparisons are *number of failed requests*, *average time caching gain per request* and *routing overhead* across the resilient networks.

Number of failed request: A request is routed over the shortest path between user and the origin server. If failures occur on the path, then a 3-hop scoped-flooding [27] is chosen. A request is said to be failed if it cannot be satisfied using flooding.

Average time caching gain per request: we measure the caching gain $\bar{G}$ using the latency defied as follows

$$\bar{G} = L_0 - L(X), \tag{22}$$

where L_0 is the latency without caching, and $L(X)$ is the latency when requests are served from intermediate nodes with content allocation X. For simplicity, we set the latency d of all edges equals to 1 in our experiments. In such a case, the average time caching gain is equivalent to the average number of hops saved.

Routing Overhead: The number of requests routed through 3-hop scoped-flooding, which introduces substantial volume of traffic to the networks.

In DR-Cache, when a request for content i reaches node k, it decides to cache it with probability $f_{k,i}$, given in Equation (20). In our numerical experiments, we need to consider the specific values of the parameters $\theta_{\chi,k}, \theta_{\psi,k}$ and $g_k(p_{k,i})$. Following the discussion on the impact of these parameters on $f_{k,i}$ in Section IV-B, here we consider three classes of the form for $f_{k,i}$, denoted as F_1, F_2 and F_3,

$$F_1 = 1 - (\chi + \psi)/H, \tag{23}$$

$$F_2 = 1 - (\theta_\chi \chi + \theta_\psi \psi)/H, \tag{24}$$

$$F_3 = 1 - log_{10}((1 - p_{k,i})/p_{k,i}) \cdot (\chi + \psi)/H, \tag{25}$$

where $p_{k,i} \in (0, 1]$ is the popularity of content i seen by node k, which is updated whenever node k receives a content. In the following, we characterize the impact of $\theta_{\chi,k}, \theta_{\psi,k}$ and $g_k(p_{k,i})$ on the performance of candidate algorithms through F_1, F_2 and F_3.

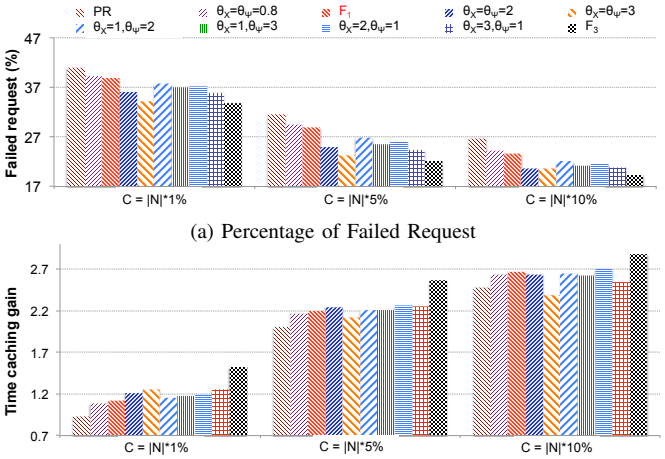
B. DR-Cache vs. PR using LRU

We first compare the performance of DR-Cache using F_1, F_2 and F_3 with the traditional non-resilient PR under synthetic traffics of Nobel EU network. LRU is applied to both if an eviction occurs. We consider the cache size of each node to be $C = 1\%$, 5% and 10% of the total number of content.

The performance of DR-Cache using F_1, F_2 with different values of $(\theta_\chi, \theta_\psi)$, F_3 and PR are shown in Figure 2. It is obvious that increasing the cache size improves the performance for all algorithms. More importantly, we observe, even with the simplest form F_1, DR-Cache always outperforms PR in terms of lower number of failed requests and higher time caching gain. Since F_1 does not capture all necessary information for cache decision-making, the performance gain is limited. For example, F_1 reduces around 3% number of failed requests compared to PR.

As discussed in Section IV-B, θ_χ and θ_ψ capture the greediness of caching decision-making. From (24), if we choose smaller values of θ_χ and θ_ψ (e.g. $\theta_\chi = \theta_\psi = 0.8$ in Figure 2), we manually increase the caching probability at each node since $F_2 > F_1$ with other parameters fixed. Similarly, choosing larger values of θ_χ and θ_ψ (e.g. $\theta_\chi > 1$ and $\theta_\psi > 1$) forces the caching probability of each node to be decreased as $F_2 < F_1$. In particular, if the values of θ_χ and θ_ψ make $F_2 < 0$, the node never caches this content when the cache is full.

We characterize the impact of θ_χ and θ_ψ on the performance through F_2, shown in Figure 2. We can see: (i) F_2 outperforms PR in all cases; (ii) smaller values degrade the performance, the performance of F_2 with $\theta_\chi = \theta_\psi = 0.8$ is worse than F_1 in terms of larger number of failed requests and smaller value of caching gain. Intuitively, smaller value of $\theta_\chi = \theta_\psi = 0.8$ increases the caching probability compared to F_1, however, in real applications or even with the synthetic traffic we consider, most content $(70-80\%)$ have only been requested once, there is no need to increase caching probabilities. We also simulate other smaller values of θ_χ, θ_ψ, similar trends exist and are omitted here due to space constraints. We mainly focus on the case $\theta_\chi, \theta_\psi > 1$, i.e., decreasing the greediness of caching. (iii) We analyze multiple combinations of $\theta_\chi, \theta_\psi > 1$, and all cases we consider significantly improve the performance, shown in Figure 2. For example, when $C = |\mathcal{N}| \times 1\%$, $\theta_\chi = \theta_\psi = 3$ results in best performance w.r.t. lower failed requests and larger caching gain. However, this is not the best choice for $C = |\mathcal{N}| \times 5\%$ and $C = |\mathcal{N}| \times 10\%$. Note that further increasing θ_χ, θ_ψ results in a decrease in performance as $F_2 \leq 0$, nodes never cache new content.

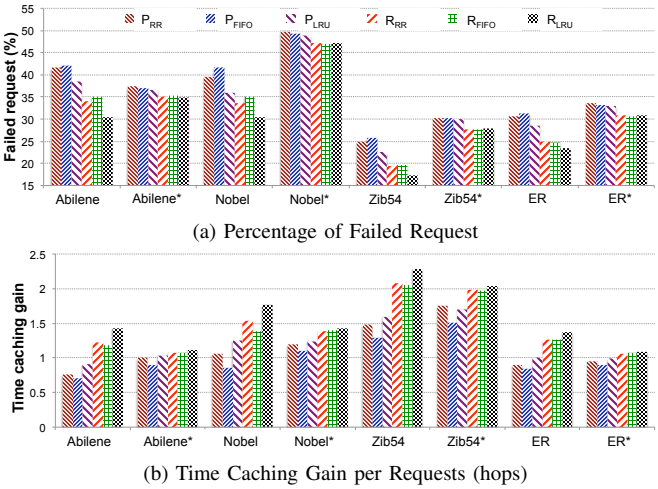


(a) Percentage of Failed Request

(b) Time Caching Gain per Request (hops)

Fig. 2: PR vs. DR-Cache using F_1, F_2 and F_3.

Intuitively, more popular content should be cached with a higher probability, however, neither F_1 nor F_2 captures the content popularity. Finally, we introduce content popularity into decision-making, given in Equation (20). With the analysis above of the impact θ_χ and θ_ψ on the performance, for simplicity, we consider the case that $\theta_\chi = \theta_\psi = 1$ and $g_k(p_{k,i}) = \log_{10}((1-p_{k,i})/p_{k,i})$. Note that $g_k(p_{k,i})$ is decreasing in $p_{k,i}$, hence F_3 is increasing in $p_{k,i}$. It is consistent with the idea that more popular content should be cached with higher probability. Moreover, as we have observed, most content have $p_{k,i} < 0.01$, i.e., $g_k(p_{k,i}) > 1$. As a result, F_3 is consistent with the observation that F_2 improves the performance with $\theta_\chi, \theta_\psi > 1$. We can see in Figure 2 that F_3 significantly outperforms all other algorithms. Similar trends hold under other networks using both synthetic and real traces, and hence are omitted here due to space constraints.



(a) Percentage of Failed Request

(b) Time Caching Gain per Requests (hops)

Fig. 3: Performance vs. different eviction policies with cache size $C = |\mathcal{N}| \times 1\%$. Superscript * represent real traffic trace.

C. DR-Cache vs. PR using Different Eviction Policies

The results discussed in Section V-B are with LRU eviction policy. Now we compare the performance of DR-Cache and PR with other candidate eviction policies: RR, FIFO and LRU. Due to space limitations, we only present the results for $C = |\mathcal{N}| \times 1\%$, similar trends observed for $C = |\mathcal{N}| \times 5\%$ and $C = |\mathcal{N}| \times 10\%$. Furthermore, through the analysis in Section V-B, here we only consider DR-Cache using F_3. The results for both synthetic and real traffic traces are given in Figure 3.

Again, DR-Cache outperforms PR under any eviction policy used here. For instance, considering Abilene network with LRU, DR-Cache (R_{LRU}) reduces 8% of the number of failed requests while improves the caching gain as much as 57% compared to PR (P_{LRU}). Also we note that LRU always outperforms RR and FIFO in both DR-Cache and PR algorithms.

(a) Percentage of Failed Request

(b) Time Caching Gain per Requests (hops)

Fig. 4: Performance vs. Different Cache Sizes.

D. Impact of Cache Size on Performance

In this subsection, we characterize the impact of cache size on the performance. We increase the cache size by 1% of the total number of contents each time. Due to space limitations, we show the results for $C_1 = |\mathcal{N}| \times 1\%$, $C_2 = |\mathcal{N}| \times 5\%$ and

$C_3 = |\mathcal{N}| \times 10\%$. Also we consider LRU in DR-Cache (R_{LRU}) and PR (P_{LRU}). From Figure 4, we notice that increasing the cache size enhances the performance under all cases, as expected. More interestingly, DR-Cache can achieve better performance than PR even with smaller cache size, i.e., DR-Cache improves the *cache efficiency*. For example, in Nobel network, DR-Cache with cache size C_2 has less number of failed requests than that for PR with cache size C_3.

E. Performance in Stable Network

Now we investigate how DR-Cache performs in stable networks (node stability scores equal to 1), i.e., no failed requests. From Figure 5, we can see that DR-Cache still achieves better performance for most of these networks. This is because under DR-Cache, the caching probability is increased when the response message is close to end-users (see Remark 2(c), section IV-B), which improves the caching gain.

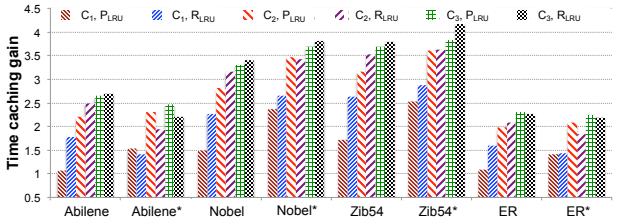


Fig. 5: Time Caching Gain in Stable Networks.

F. Overhead in Routing

In our numerical experiments, given a set of feasible paths, we choose the shortest path first under both DR-Cache and PR algorithms. If this path fails, then a 3-hop scoped-flooding is used. In practice, network-wide flooding is rarely used due to its significant traffic overhead. In this subsection, we study the overhead induced by DR-Cache and PR, shown in Figure 6. We can see that DR-Cache significantly outperforms PR. For example, in Abilene network with capacity C_1, the overhead, in terms of flooding requests can be significantly reduced by 16.7% if using DR-Cache rather than PR.

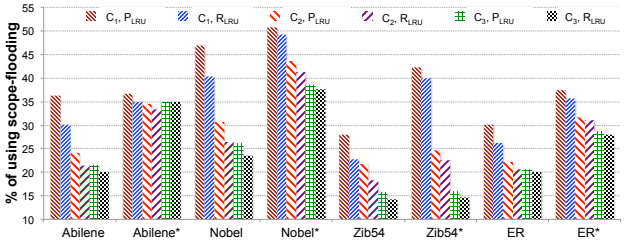


Fig. 6: Percentage of Using Scope-flooding.

VI. Conclusion

This paper aims to design a distributed algorithm for making caching decisions in unreliable networks. We proposed both a centralized algorithm and DR-Cache - a simple and adaptive distributed algorithm for disruptive networks. We evaluated DR-Cache against traditional path replication (PR) algorithms using several network topologies with both synthetic and real traffic traces. The numerical results showed that DR-Cache significantly outperforms PR. In future work, we will incorporate both node and link failures into the formulation.

References

[1] http://www.networkcomputing.com/networking/high-price-it-downtime/856595126.

[2] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan. Optimal Content Placement for a Large-Scale VoD System. *IEEE/ACM Transactions on Networking*, 24:2114 – 2127, 2016.

[3] I. Baev, R. Rajaraman, and C. Swamy. Approximation Algorithms for Data Placement Problems. *SIAM Jour. Comp.*, 38(4):1411–1429, 2008.

[4] G. Bianchi, A. Detti, A. Caponi, and N. Blefari Melazzi. Check before Storing: What is the Performance Price of Content Integrity Verification in LRU Caching? *ACM SIGCOMM CCR*, 2013.

[5] S. Borst, V. Gupta, and A. Walid. Distributed Caching Algorithms for Content Distribution Networks. In *IEEE INFOCOM*, 2010.

[6] W. K. Chai, V. Sourlas, and G. Pavlou. Providing Information Resilience Through Modularity-based Caching in Perturbed Information-centric Networks. In *IEEE/ACM International Teletraffic Congress*, 2017.

[7] H. Che, Y. Tung, and Z. Wang. Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. *Selected Areas in Communications, IEEE Journal on*, 20(7):1305–1314, 2002.

[8] C. Chekuri, J. Vondrak, and R. Zenklusen. Dependent Randomized Rounding via Exchange Properties of Combinatorial Structures. In *IEEE FOCS*, 2010.

[9] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *ACM SIGCOMM*, 2002.

[10] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman. On the Complexity of Optimal Routing and Content Caching in Heterogeneous Networks. In *IEEE INFOCOM*, 2015.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *ACM SIGOPS Operating Systems Review*, 2003.

[12] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *ACM SIGCOMM*, 2011.

[13] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire. Femtocaching: Wireless Video Content Delivery Through Distributed Caching Helpers. In *IEEE INFOCOM*, 2012.

[14] I. Hou, T. Zhao, S. Wang, and K. Chan. Asymptotically Optimal Algorithm for Online Reconfiguration of Edge-Clouds. In *ACM MobiHoc*, 2016.

[15] S. Ioannidis and E. Yeh. Adaptive Caching Networks with Optimality Guarantees. In *ACM SIGMETRICS*, 2016.

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

[17] J. Li, S. Shakkottai, J. C. S. Lui, and V. Subramanian. Accurate Learning or Fast Mixing? Dynamic Adaptability of Caching Algorithms. *Arxiv preprint arXiv:1701.02214*, 2017.

[18] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, Y. Ganjali, and C. Diot. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Transactions on Networking*, 16:749 – 762, 2008.

[19] N. K. Panigrahy, J. Li, and D. Towsley. Hit rate vs. hit probability based cache utility maximization. In *ACM MAMA*, 2017.

[20] N. K. Panigrahy, J. Li, F. Zafari, D. Towsley, and P. Yu. What, When and Where to Cache: A Unified Optimization Approach. *Arxiv preprint arXiv:1711.03941*, 2017.

[21] T. K. Phan, D. Griffin, E. Maini, and M. Rio. Utility-maximizing Server Selection. In *IFIP Networking*, 2016.

[22] T. K. Phan, D. Griffin, E. Maini, and M. Rio. Utility-centric Networking: Balancing Transit Costs with Quality of Experience. *IEEE/ACM Transactions on Networking*, 2018.

[23] E. J. Rosensweig, D. S. Menasche, and J. Kurose. On the Steady-State of Cache Networks. In *IEEE INFOCOM*, 2013.

[24] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An Analysis of Internet Content Delivery Systems. *ACM SIGOPS*, 2002.

[25] L. Tong, Y. Li, and W. Gao. A Hierarchical Edge Cloud Architecture for Mobile Computing. In *IEEE INFOCOM*, 2016.

[26] R. Van Renesse and F. B. Schneider. Chain Replication for Supporting High Throughput and Availability. In *OSDI*, 2004.

[27] L. Wang, S. Bayhan, J. Ott, J. Kangasharju, S. A, and J. Crowcroft. Pro-Diluvian: Understanding Scoped-Flooding for Content Discovery in Information-Centric Networking. In *ACM ICN*, 2015.