Fundamental Limits on the Regret of Online Network-Caching

RAJARSHI BHATTACHARJEE,
Dept. of Electrical Engineering, Indian Institute of Technology Madras, India
SUBHANKAR BANERJEE,
Dept. of Electrical Engineering, Indian Institute of Technology Madras, India
ABHISHEK SINHA,
Dept. of Electrical Engineering, Indian Institute of Technology Madras, India

Optimal caching of files in a content distribution network (CDN) is a problem of fundamental and growing commercial interest. Although many different caching algorithms are in use today, the fundamental performance limits of network caching algorithms from an online learning point-of-view remain poorly understood to date. In this paper, we resolve this question in the following two settings: (1) a single user connected to a single cache, and (2) a set of users and a set of caches interconnected through a bipartite network. Recently, an online gradient-based *coded* caching policy was shown to enjoy sub-linear regret. However, due to the lack of known regret lower bounds, the question of the optimality of the proposed policy was left open. In this paper, we settle this question by deriving tight non-asymptotic regret lower bounds in both of the above settings. In addition to that, we propose a new Follow-the-Perturbed-Leader-based *uncoded* caching policy with near-optimal regret. Technically, the lower-bounds are obtained by relating the online caching problem to the classic probabilistic paradigm of *balls-into-bins*. Our proofs make extensive use of a new result on the expected load in the most populated half of the bins, which might also be of independent interest. We evaluate the performance of the caching policies by experimenting with the popular MovieLens dataset and conclude the paper with design recommendations and a list of open problems.

CCS Concepts: • **Networks → Network performance analysis**; • **Mathematics of computing → Probabilistic algorithms**.

Additional Key Words and Phrases: network-caching; online algorithms; regret bounds; fundamental limits

ACM Reference Format:
Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. 2020. Fundamental Limits on the Regret of Online Network-Caching. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 2, Article 25 (June 2020), 31 pages. https://doi.org/10.1145/3392143

1 INTRODUCTION AND RELATED WORK

The classical caching problem, which seeks to make popular content quickly accessible by prefetching them in low-latency storage, has been extensively studied in the literature. The core idea of caching has been used in many diverse domains, including improving the CPU paging

Authors' addresses: Rajarshi Bhattacharjee,
Dept. of Electrical Engineering, Indian Institute of Technology Madras, India, brajarshi91@gmail.com; Subhankar Banerjee, Dept. of Electrical Engineering, Indian Institute of Technology Madras, India, ee16s048@ee.iitm.ac.in; Abhishek Sinha, Dept. of Electrical Engineering, Indian Institute of Technology Madras, India, abhishek.sinha@ee.iitm.ac.in.

2476-1249/2020/6-ART25 $15.00
https://doi.org/10.1145/3392143

performance via L_1/L_2 caches [63], web-caching by Content Distribution Networks [4, 8, 49, 54, 56], and low-latency wireless video delivery through Femtocaching [60]. With the exponential growth of internet video traffic and the advent of new services consuming high bandwidth, such as augmented and virtual reality (AR/VR), the importance of caching for ensuring the quality of service (QoS) is on the rise [15]. Top CDN providers, such as Amazon AWS and Microsoft Azure, now offer caching as a service [16, 67].

Several caching algorithms have been proposed in the literature. The MIN algorithm [65] is an optimal *offline* caching policy that assumes that the entire file request sequence is known non-causally in advance. MIN is often used as a benchmark for comparing the performance of the online caching policies. Among the online policies, the Least Recently Used policy (LRU), the Least Frequently Used policy (LFU) [39], the FIFO policy [20], and the online coded caching policy [52] have been studied extensively. The paper [42] shows how to augment online algorithms with a machine-learned oracle to design caching policies with low competitive ratios. However, the performance guarantees available for most of the online caching policies are highly contingent upon some a priori assumptions on the generative model of the file request sequence [13, 58]. The paper [34] analyzed the performance of the LRU caching policy with an i.i.d. file request sequence, also known as the Independent Reference Model [66]. Under a Markovian assumption, the paper [52] shows that a coded caching policy outperforms the LRU policy. The paper [24] develops a unified framework for analyzing several popular caching policies, again with a stationary file request model. On a different line of work, the papers [35, 36, 43, 44] derive information-theoretic lower bounds and efficient caching, computing, and coding schemes to facilitate bandwidth-efficient delivery of the cached files to the users.

With frequent addition of new content to the library, mobility of the users, Femtocaching with small caches, and change in the popularity distribution with time, the assumption of stationary file popularity barely holds in practice [64]. This prompts us to consider the problem of caching from an online learning point-of-view with no a priori statistical assumption on the file request sequence. The article [50] gives an excellent overview of CDN caching along this line. Our work is inspired by the recent paper [51], which describes an online gradient-based coded caching policy (OGA), and proves a sub-linear regret upper-bound for the same. Interestingly, they also show that popular uncoded caching policies, such as LRU, LFU, and FIFO, suffer from linear regrets in the worst case. In fact, no uncoded caching policy with a sub-linear regret is known previously in the literature. More seriously, no regret lower bound is known for the network-caching problem.

In contrast to the multi-armed bandits setting [5, 19, 26, 38, 46], relatively few results are known for the regret lower bounds for online convex optimization problems. Technically, the network-caching problem is an instance of an online convex optimization problem with a piecewise linear reward function and polytope constraints. The paper [3] establishes a minimax regret lower bound for linear cost functions with hyper ball constraints. A regret lower bound for the *unconstrained* linear cost functions has been obtained in [31]. The papers [30, 32] prove logarithmic regret bounds for online stochastic strongly convex problems. However, to the best of our knowledge, except for [51], the regret for a linear cost function with a simplex and several box constraints, which arise in the context of the single cache problem, has not been studied before. Moreover, the problem of lower bounding the regret for a piecewise linear cost function with polytope constraints, which arise in the context network-caching, is entirely open.

The above considerations inspire us to ask the following two questions in this paper:
Question 1. What is the fundamental performance limit of all online caching policies *regardless* of their operational constraints or computational complexity?
Question 2. Can a simple, distributed network-caching strategy be designed which meets the above fundamental limit?

In answering Question 1, we derive universal regret lower bounds that also apply to computationally intensive caching policies, which can completely change the profile of the cached contents at *every* time slot. Surprisingly enough, we answer Question 2 in the affirmative. In particular, our matching upper and lower bounds reveal that a simple gradient-based incremental coded caching policy is regret-optimal. Moreover, we propose a new Follow-the-Perturbed-Leader-based uncoded caching policy that has near-optimal regret. Hence, one of the key take-away points from this paper is that there exist computationally cheap caching policies that perform excellently in an online setting even with adversarial request sequence.

Our contributions: In the process of answering the above questions, we make the following key technical contributions:

(1) Lower bound for a single cache: In Theorem (2), we prove a tight non-asymptotic regret lower bound for the single-cache problem. This result improves upon the previously known *asymptotic* regret lower bound in [51], which can be arbitrarily loose for sufficiently large library size.

(2) Lower bounds for caching networks: In Theorems (6) and (7), we derive non-asymptotic sub-linear regret lower bounds for bipartite caching networks. We also show that the lower bounds are tight within constant factors. To the best of our knowledge, this is the first known regret lower bound for piecewise linear functions with polytope constraints. Hence, our results also contribute to the growing literature on online convex optimization.

(3) Near-optimal uncoded caching policy: Although the above lower bounds certify the optimality of the gradient-based coded caching policy of [51], it was an open problem whether one could achieve the optimal regret without coding. In Algorithm 1, we propose a simple uncoded caching policy based on the Follow-the-Perturbed-Leader (FTPL) paradigm, which uses cumulative frequency counts with added Gaussian noise for caching decisions. Theorem 3 shows that the FTPL policy has near-optimal regret. Our theoretical results nicely complement some recent promising empirical observations on large-scale caching systems employing similar frequency-based caching policies [17, 22, 45].

(4) New proof techniques: In this paper, the regret lower bounds are established by relating the online caching problem to the classic probabilistic setup of *balls-into-bins* via Lemma 1. In this Lemma, we derive a tight non-asymptotic lower bound to the expected total load in the most populated n bins when m balls are randomly thrown into $2n$ bins. For our lower bound proofs, we are particularly interested in the regime $m \gg n$. It is to be noted that classical results on randomized load balancing, such as [25, 47, 48], are not relevant to this setting because they apply only to the regime where $m = O(n)$. Furthermore, we note that previous works, such as [37, 53], derive asymptotic high-probability bounds for the maximum load for various asymptotic regimes of m and n. However, since we are primarily interested in the *non-asymptotic expected* value of Max-Load, these asymptotic results do not suffice for our purpose. Consequently, we tackle this problem from the first principles, culminating in Lemma 1. To the best of our knowledge, this is the first paper where a connection between online learning and the framework of balls-into-bins has been explicitly brought out and exploited in proving regret lower bounds.

(5) Numerical experiments. In Section 4, we compare the performance of different caching policies on the popular MovieLens 1 M dataset [1]. Our experiments reveal that the theoretically sound FTPL policy outperforms other competitive caching policies in terms of long-term average regret on this trace.

2 SINGLE CACHE

In this section, we begin our investigation with the single cache problem. We establish a key technical lemma on the *balls-into-bins* problem, which is used in all of our lower bound proofs. Our analysis in this section improves upon the best-known regret lower bound for the single caches [51] and paves the way for analyzing the bipartite caching problem in Section 3.

2.1 System Model

In the classical caching problem with a single cache, there is a library of N distinct files. A cache with a limited storage capacity can store at most C files at any time slot. In practice, the cache capacity is significantly smaller compared to the entire library size (*i.e.,* $C \ll N$). As an example, we may think of caching the movie files in the Netflix edge-servers, where the library size N increases every day as new movies are released, but the physical cache-size in the edge-server remains constant on the relevant time-scale. Time is slotted, and a user may request at most one file at a time. The file request at time slot t is represented by an N-dimensional binary vector $\boldsymbol{x}_t \in \{0, 1\}^N$, where $x_{tf} = 1$ if the f^{th} file is requested by the user at time t, and is zero otherwise (one-hot encoding). Following an online caching policy π, files are cached at every time slot *before* the request for that slot arrives. We do not make any statistical assumptions on the file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$. Thus, we may as well assume that the requests are made by an omniscient adversary who has complete knowledge of the cached contents and the caching policy in use. See Figure 1 for a schematic.

Coded Caching: In this paper, we consider both coded and uncoded caching. In the classical uncoded caching, complete files are cached. On the other hand, in coded caching, the original files are first encoded using fountain codes (a class of rateless erasure codes), e.g., Raptor code [41, 61], and then some of the resulting coded symbols are cached. These codes have the property that an original file consisting of k source symbols can be recovered (with high probability) by combining any subset of k' coded symbols, where k' needs to be only slightly larger than k. Hence, for decoding, it does not matter which encoding symbols are combined, as long as the decoder has access to sufficiently many encoded symbols. We will see that coding offers distinct advantages for network caching. These codes also admit highly efficient linear time encoding and decoding operations. The rateless codes are routinely used in P2P data streaming, large scale data centers, and CDNs [68].

Caching configuration: The cache configuration at time t is represented by an N-dimensional vector $\boldsymbol{y}_t^\pi \in [0, 1]^N$, where $y_{t,f}^\pi$ denotes the fraction of the file f cached at time t under the policy π [1]. Naturally, in the uncoded case $y_{t,f}^\pi \in \{0, 1\}, \forall f, t$. The set of all admissible caching configuration is denoted by $\mathcal{Y}$ where

$$\mathcal{Y} = \left\{ \boldsymbol{y} \in [0, 1]^N : \sum_{f=1}^{N} y_f \leq C \right\}, \tag{1}$$

where C is the capacity of the cache. The caching decision $\boldsymbol{y}_t$ may be randomized and may depend on the file request sequence and caching decisions up to time $t-1$. Any requested file, not present in the cache, is routed to a central server (which is assumed to host every file in the library) and accrues zero rewards.

[1] Whenever the caching policy π is clear from the context, we will drop the superscript π to simplify the notation.

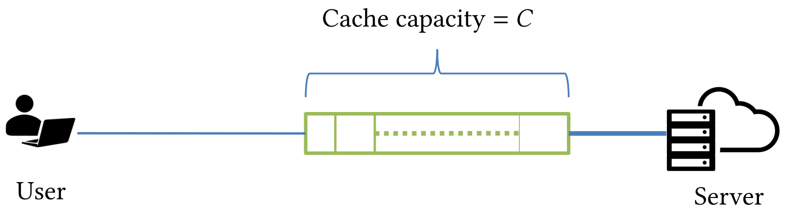

Fig. 1. Caching with a single cache

2.2 Performance metrics

2.2.1 ***Reward.*** A popular performance metric for any online caching policy is its average *hit rate*, *i.e.*, the average number of requested files already present in the cache so that the files can be quickly retrieved. In this connection, let the user-generated file request sequence be denoted by $\{\boldsymbol{x}_t\}_{t\geq 1}$. The total reward up to time T accrued by a caching policy, which responds to the file requests by setting the cache configuration to $\boldsymbol{y}_t$, at time t, is denoted by $Q\big(\{\boldsymbol{x}_t\}_1^T, \{\boldsymbol{y}_t\}_1^T\big)$. We assume that the cumulative reward over a time horizon T has an additive structure, which may be obtained by summing up the rewards obtained at every slot up to time T, *i.e.,*

$$Q\big(\{\boldsymbol{x}_t\}_1^T, \{\boldsymbol{y}_t\}_1^T\big) = \sum_{t=1}^{T} q(\boldsymbol{x}_t, \boldsymbol{y}_t). \tag{2}$$

The one-slot reward function $q(\cdot,\cdot)$ captures the reward obtained per slot. Intuitively, $q(\boldsymbol{x}_t, \boldsymbol{y}_t)$ denotes the extent of cache-hits for the request vector $\boldsymbol{x}_t$ against the cache configuration vector $\boldsymbol{y}_t$. The function $q(\cdot,\cdot)$ takes different functional forms depending on whether we consider (a) single cache, or (b) a network of caches. For the case of a single cache, we define the one-slot reward to be the amount of the requests successfully served by the cache, *i.e.,*

$$q(\boldsymbol{x}_t, \boldsymbol{y}_t) \equiv \boldsymbol{x}_t \cdot \boldsymbol{y}_t. \tag{3}$$

A generalized definition of the above one-slot reward function applicable to caching networks will be given in Section 3.

2.2.2 ***Regret.*** Since we do not make any assumption on the user-generated file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$, it is futile to attempt to optimize the total reward (*i.e.,* hit rate). This is because, at any slot, an omniscient adversary can always request a file that is not present in the cache, thus yielding a total of zero hits. To obtain a non-trivial performance measure, we cast the caching problem into the framework of online learning. This prompts us to compare the performance of any policy with the best offline *stationary* optimal policy [59]. Let the vector $\boldsymbol{y}^*$ denote any fixed stationary cache configuration vector. The vector $\boldsymbol{y}^*$ may be selected offline after seeing the entire request sequence $\{\boldsymbol{x}_t\}_{t=1}^T$. Following the usual convention in the online learning literature, we define the regret $R_T^\pi(\{\boldsymbol{x}_t\}_1^T)$ for a request sequence $\{\boldsymbol{x}_t\}_1^T$ to be the difference in the reward obtained by the best stationary caching configuration $\boldsymbol{y}^*$ and that of the online policy π. Mathematically[2],

$$R_T^\pi(\{\boldsymbol{x}_t\}_1^T) := \sup_{\boldsymbol{y}^*\in\mathcal{Y}} \bigg(Q\big(\{\boldsymbol{x}_t\}_1^T, \{\boldsymbol{y}^*\}_1^T\big) - Q\big(\{\boldsymbol{x}_t\}_1^T, \{\boldsymbol{y}_t\}_1^T\big)\bigg). \tag{4}$$

[2]Recall that the cache configuration sequence $\{\boldsymbol{y}_t\}_{t\geq 1}$ is determined by the policy π.

The regret R_T^π of any caching policy π up to time T is defined to be the maximum regret over all admissible request sequences, *i.e.,*

$$R_T^\pi := \sup_{\{\boldsymbol{x}_t\}_1^T} R_T^\pi(\{\boldsymbol{x}_t\}_1^T). \tag{5}$$

Discussion: Note that, minimizing the regret as defined in Eqn. (5) optimizes against the worst-case or "adversarial" file request sequence. The adversarial assumption of the request-sequence is quite standard in the literature on caching. In fact, the classical competitive analysis of the caching algorithms makes use of this assumption [6]. In this sequel, we will see that strong performance guarantees against worst-case request patterns lead to robust caching algorithms both in theory and in practice.

2.3 Standing Assumptions

In order to put our results in the proper context, we now list the main assumptions behind our analysis. Note that these assumptions also apply to the results in [51].

(1) **Generalized caching policies:** We relax a requirement of the standard caching policies (*e.g.,* LRU, LFU), which require only items appearing in the miss-stream to be downloaded to the cache from the server at every slot. Instead, we allow the caching policies to load any number of arbitrary items to the cache at a slot (subject to the cache capacity constraint), irrespective of whether those items were requested or not in the previous slot.
(2) **Zero Download Cost:** Since we are interested in the fundamental limits of online caching performance, we assume that downloading content from the remote server to the cache does not cost anything. That being said, the caching algorithms that we propose in this paper are flexible enough to offer a smooth trade-off between the cost of caching and hit ratio. We explore this direction empirically in Section 4.1.
(3) **Zero Download Delay:** At every time slot, files are assumed to be cached before the request for that slot arrives [56].
(4) **Uniform file sizes:** We assume that all files have a unit size. The effect of variable file-sizes on caching has been studied recently in [10].
(5) **Negligible Coding overhead:** With coded caching, we assume that files are encoded with an efficient erasure code so that the coding overhead is negligible [61].
(6) **One request per slot:** For simplicity, we assume that each user requests at most one file per slot.

Obviously, all of our lower bounds hold when some or all of the above assumptions are relaxed.

2.4 Regret lower bounds - preliminaries

Using the simple observation that the maximum of a set of real numbers is at least equal to their average, for any joint probability distribution $P(\{\boldsymbol{X_t}\}_{t=1}^T)$ on the file request sequence $\{\boldsymbol{X_t}\}_{t=1}^T$, the regret in Eqn. (5) may be lower bounded as follows:

$$R_T^\pi \geq \mathbb{E}_{\{\boldsymbol{X}_t\}_1^T} R_T^\pi(\{\boldsymbol{X}_t\}_1^T). \tag{6}$$

The joint distribution $P(\cdot)$ needs to be chosen carefully to ensure the tractability of evaluating the expectation in (6), as well as the tightness of the resulting bound. The above technique is known as the *probabilistic method* popularized by Erdős [7]. In all of our proofs, we lower bound the quantity in Eqn. (4) by a suitable *binary* cache configuration vector $\boldsymbol{y}^*$ (thus $\boldsymbol{y}^*$ corresponds to uncoded caching). As a consequence, all of our lower bounds remain valid in both uncoded and coded caching.

2.5 Regret lower bound for unit-sized cache

We first consider a single cache of unit capacity and derive a universal non-asymptotic regret bound. As we will see in the sequel, understanding this special case lays the foundation for our subsequent analysis of caching networks serving multiple users.

Theorem 1 (Lower bound for a single-cache of unit capacity). *The regret R_T^π of any online caching policy, for a library-size $N = 2$ and cache-capacity $C = 1$, is lower bounded as:*

$$R_T^\pi \geq \sqrt{\frac{T}{2\pi}} - \frac{1}{2\sqrt{2\pi T}}, \ \forall T \geq 1.$$

Proof. Denote the cache configuration selected by the caching policy π at slot t by the vector $\boldsymbol{y}_t = \begin{pmatrix}\gamma_t & 1-\gamma_t\end{pmatrix}'$, where γ_t denotes the fraction of File_1 cached by the policy π ($0 \leq \gamma_t \leq 1$). Recall the definition of regret R_T in this context:

$$R_T^\pi = \sup_{\{\boldsymbol{x}_t\}_1^T} \sup_{\boldsymbol{y}^* \in \mathcal{Y}} \left(\boldsymbol{y}^* \cdot \sum_{t=1}^T \boldsymbol{x}_t - \sum_{t=1}^T \boldsymbol{y}_t \cdot \boldsymbol{x}_t\right), \tag{7}$$

where the set of all admissible configurations $\mathcal{Y}$ is given by Eqn. (1). Denote the file request vector at slot t by $\boldsymbol{x}_t = \begin{pmatrix}w_t & 1-w_t\end{pmatrix}'$, where $w_t \in \{0,1\}$. It can be easily seen that for a given file request sequence $\{\boldsymbol{x}_t\}_1^T$, an optimal choice of the fixed cache configuration vector $\boldsymbol{y}^*$ in Eqn. (7) is given as follows:

$$\boldsymbol{y}^* = \begin{cases} \begin{pmatrix}1 & 0\end{pmatrix}', & \text{if } \sum_{t=1}^T w_t \geq T/2, \\ \begin{pmatrix}0 & 1\end{pmatrix}', & \text{if } \sum_t w_{t=1}^T < T/2. \end{cases}$$

To prove a universal regret lower bound, we need to show the existence of an adversarial file request sequence $\{\boldsymbol{x}_t\}_1^T$ under which the caching policy π performs poorly. Towards this, let $\{W_t\}_{t\geq 1}$ be a sequence of i.i.d. uniform Bernoulli random variables such that $\mathbb{P}(W_t = 0) = \mathbb{P}(W_t = 1) = \frac{1}{2}$. Construct a random file request sequence $\boldsymbol{X}_t = \begin{pmatrix}W_t & 1-W_t\end{pmatrix}'$. The regret incurred for the sequence $\{\boldsymbol{X}_t\}_1^T$ may be obtained from Eqn. (7) as:

$$\begin{aligned} & R_T^\pi(\{\boldsymbol{X}_t\}_t) \\ = \ & \max\left\{\sum_{t=1}^T W_t, T - \sum_{t=1}^T W_t\right\} - \sum_{t=1}^T \big(\gamma_t W_t + (1-\gamma_t)(1-W_t)\big) \\ = \ & \max\big\{Z, T-Z\big\} - \sum_{t=1}^T \big(2\gamma_t W_t + 1 - \gamma_t - W_t\big), \end{aligned}$$

where the r.v. $Z \equiv \sum_{t=1}^T W_t$, being the summation of T i.i.d. uniform Bernoulli random variables, is Binomially distributed with the parameters $(T, 1/2)$. Using linearity of expectation, we can write

$$\mathbb{E}_{\{\boldsymbol{X}_t\}_1^T}\big(R_T^\pi(\{\boldsymbol{X}_t)\}_1^T\big) = \mathbb{E}\big(\max\big\{Z, T-Z\big\}\big) - T/2, \tag{8}$$

where we have used the fact that $\mathbb{E}(W_t) = 1/2$ and the caching decision γ_t is independent of the incoming request $\boldsymbol{X}_t$. Observe that, we can write

$$\max\big\{Z, T-Z\big\} = \frac{T}{2} + |Z - T/2|. \tag{9}$$

Thus, combining Eqns. (8) and (9), we have

$$\mathbb{E}_{\{\boldsymbol{X}_t\}_1^T}\big(R_T^\pi(\{\boldsymbol{X}_t\})_1^T\big) = \mathbb{E}\big|Z - T/2\big|. \tag{10}$$

The mean absolute deviation for a symmetric binomial random variable may be computed in closed form by using De Moivre's formula ([9], Eqn. (1)) as follows:

$$\mathbb{E}\Big|Z - \frac{T}{2}\Big| = \frac{1}{2^T}\Big(\lfloor\frac{T}{2}\rfloor + 1\Big)\binom{T}{\lfloor\frac{T}{2}\rfloor + 1}. \tag{11}$$

Eqn. (11), in combination with a non-asymptotic form of Stirling's formula [55], yields the following *non-asymptotic* lower bound

$$\mathbb{E}\Big|Z - \frac{T}{2}\Big| \geq \sqrt{\frac{T}{2\pi}} - \frac{1}{2\sqrt{2\pi T}}, \ \forall T \geq 1. \tag{12}$$

For details of the above calculations, please refer to Appendix 8.1. Equation (12), coupled with Eqn. (10), shows the existence of a file request sequence $\{\boldsymbol{x}_t\}_1^T$ such that

$$R_T^\pi(\{\boldsymbol{x}_t\}_1^T) \geq \sqrt{\frac{T}{2\pi}} - \frac{1}{2\sqrt{2\pi T}}, \ \forall T \geq 1.$$

□

Remarks: It can be seen that the above proof and the lower bound in Theorem 1 continue to hold even if we let the library size to be $N \geq 2$. This observation follows by constructing a randomized file request sequence where the first two files are requested with probability $\frac{1}{2}$ each and other files are requested with zero probabilities.

2.6 Lower bound for caches of an arbitrary size

We now extend the previous result to caches with arbitrary size $C \geq 1$. This extension is non-trivial. We will see in the sequel that our analysis naturally leads us to investigate a random variable arising in connection with the classic probabilistic framework of *balls-into-bins*, where a number of balls are thrown uniformly and independently at random to some bins [53]. The following lemma, which might be of independent interest, gives a non-asymptotic lower bound to the total number of balls in the most populated half of the bins.

Lemma 1 (Total occupancy in the most popular half). *Suppose that T balls are thrown independently and uniformly at random into $2C$ bins. Let the random variable $M_C(T)$ denote the number of balls in the most populated C bins. Then*

$$\mathbb{E}(M_C(T)) \geq \frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \frac{(\sqrt{2}+1)C^{3/2}}{2\sqrt{2\pi T}} - \sqrt{\frac{2}{\pi}}\frac{C^2}{T}.$$

Proof outline: The proof proceeds by pairing up the bins to form *super bins* (see Fig. 8), and then selects the most-occupied bin in each super bin to obtain a lower bound on $M_C(T)$. Finally, we conclude the proof by appealing to the mean-deviation bound in Eqn. (12). Please refer to Section 7.1 for the complete proof of Lemma 1.

To improve readability, in the rest of the paper we will rephrase the above bound as

$$\mathbb{E}\big(M_C(T)\big) \geq \frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}),$$

with the understanding that an explicit form of the lower order terms may be obtained by using Lemma 1, if required. The following Theorem is the main result of this section.

Theorem 2 (Lower bound for a single cache of arbitrary capacity). *The regret R_T^π of any online caching policy π, for a library size N and cache capacity C with $N \geq 2C$, is lower bounded as*

$$R_T^\pi \geq \sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}), \ \forall\, T \geq 1. \tag{13}$$

Proof outline: The proof proceeds along the lines of Theorem 1, where the t^{th} file requested $\boldsymbol{X}_t$ is chosen independently and uniformly at random from the *first* $2C$ files from the library. Then, we show that the reward accrued by the best static cache configuration $\boldsymbol{y}^*$ corresponds (in distribution) to the total number of balls in the most populated half of the bins. We then conclude the proof by appealing to Lemma 1. Please refer to Section 7.2 for the complete proof of Theorem 2.

Comparison with Theorem 1 of [51]: In the single cache setting with the same set of assumptions, the paper [51] establishes a rather loose *asymptotic* regret lower bound of $\sqrt{\frac{C}{N\pi}}\sqrt{CT}$, which could be arbitrarily smaller than the lower bound given in Eqn. (13) when the library size N is sufficiently large. Theorem 2 improves the result in [51] in two ways. First, it proves a regret lower bound that is *independent* of N. As a consequence, we will soon see that it implies that the gradient-based coded policy of [51] is regret-optimal up to a constant factor. Theorem 2 also implies near-optimality of the uncoded FTPL policy described in the next section. Second, unlike the regret lower bound in [51], the bound in Eqn. (2) is *non-asymptotic*, thus giving a valid lower bound for any $T \geq 1$.

2.7 Achievability

We note that many popular classical uncoded caching policies, such as LRU, LFU, and FIFO, have linear regrets (Proposition 1 of [51]). This can be simply understood from the following example: consider the single cache setting with $N = 2, C = 1$ and an alternating file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1} = \{1, 2, 1, 2, 1, 2, \ldots\}$. Since any missed content is always immediately loaded to the cache, each of the above policies gets zero cumulative hits (*cf.* Assumption 1). On the other hand, caching either one of the files forever achieves a total of $\frac{T}{2}$ cumulative hits for a horizon of length T. Thus, all of the above policies have $\Omega(T)$ regret. This is surprising as the LRU and FIFO policies are known to have a finite competitive ratio [6]. To the best of our knowledge, no uncoded caching policy with sub-linear regret is known in the literature.

Achievability with uncoded caching: Making use of the theory of Online Structured Learning [18], we now propose a simple Follow the Perturbed Leader (FTPL)-based *uncoded* caching policy, which achieves $O(\sqrt{T})$ expected regret against an oblivious adversary. The FTPL policy maintains a cumulative running count of the number of times a file was requested so far (by Assumption 3, this feedback is immediate). This count is then perturbed by adding i.i.d. Gaussian noise of zero mean and an appropriate variance to each of the count values. Finally, at each time slot, the top C files with the highest perturbed count are loaded to the cache. Note that, here we implicitly exploit the freedom granted by Assumptions 1 and 2. The FTPL policy is formally described below in Algorithm 1.

Algorithm 1 FTPL policy for Uncoded Caching

```
1: count ← 0, $\eta \leftarrow \frac{1}{(4\pi \log N)^{1/4}}\sqrt{\frac{T}{C}}$
2: for $t = 1$ to $T$ do
3:     count ← count + $\boldsymbol{x}_t$
4:     Sample $\boldsymbol{\gamma}_t \sim \mathcal{N}(0, \boldsymbol{I}_{N\times N})$
5:     perturbed count ← count + $\eta\boldsymbol{\gamma}_t$
6:     Sort perturbed count in decreasing order and load the top $C$ files in the cache.
7: end for
```

Theorem 3 (Achievability with uncoded caching). *In the single cache setting, the FTPL uncoded caching policy achieves the following upper bound for expected regret (the expectation is taken over the randomness of the algorithm)*

$$\mathbb{E}_{\{\boldsymbol{\gamma}_t\}_{t\geq 1}}(R_T^{FTPL}) \leq 1.51(\log N)^{1/4}\sqrt{CT}.$$

The proof of Theorem 3 follows a similar line of arguments as the proof of Theorem 1 of [18]. However, in this paper, we tighten the regret upper bound of [18] further by a factor of $O(\sqrt{C})$. This improvement follows by taking into account the constraint that only one file is requested per slot and any feasible cache configuration respects a natural box constraint. This tightening is essential in order to match the FTPL upper bound with the lower bound given in Theorem 2. The details of the proof are given in Section 7.3.
The lower bound of Theorem 2 shows that FTPL is regret-optimal up to poly-logarithmic factors in the library size N. In the following, we show that this extra poly-log factor may be removed if we allow coded caching.

Achievability with coded caching: Making use of Assumption 5, in [51], Corollary 2, the authors showed that an Online Gradient Ascent (OGA)-based single-server caching policy, serving one file-request per slot, achieves a regret of value at most $\sqrt{2CT}$. To avoid repetition, a description of the general version of the OGA policy in the context of network caching will be given in Section 4, which subsumes the single cache case.

3 CACHING IN A CONTENT DISTRIBUTION NETWORK

We now begin investigating the problem of optimal caching in a Content Distribution Network (CDN). In this problem, there is a set of geographically distributed users who periodically request files to a content provider (*e.g.,* Netflix). The content provider maintains a global network of data centers, each caching some files up to its capacity. A user's file-request may be served by its neighboring data centers if the requested file exists in any of the neighboring caches. This client-server architecture gives rise to a bipartite content distribution network with the set of users and the set of caches constituting its two parts [60]. For a detailed case-study on the above caching architecture, including the global distribution of the data centers and performance measurement in the context of Netflix CDN, please refer to [12]. In the following Section, we show how the tools and techniques developed in the previous section for a single cache may be generalized to address this more challenging problem.

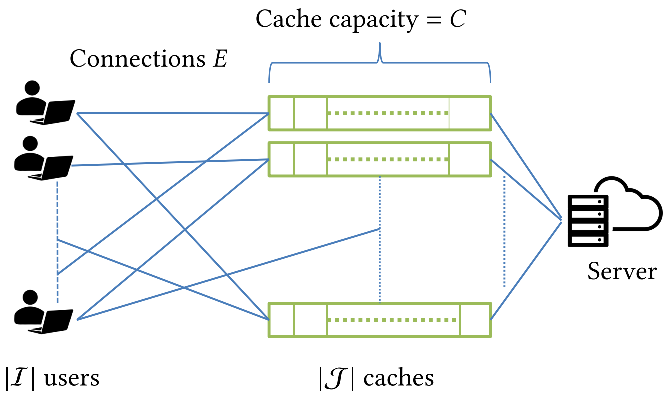

Fig. 2. Schematic of a Bipartite Caching Network

3.1 System Model

We now formalize the above system model for caching in a CDN. A set of users $\mathcal{I} = \{1, 2, \ldots, I\}$ is connected to a set of caches $\mathcal{J} = \{1, 2, \ldots, J\}$ in the form of a bipartite network. For simplicity, we assume that the caches are homogeneous in the sense that each cache has the same storage capacity C. As before, the library size (*i.e.,* the number of all possible files) N is assumed to be sufficiently large. The connection between the users and the caches is represented by the bipartite graph $(\mathcal{I}, \mathcal{J}, E)$. The set of caches connected to a user $i \in \mathcal{I}$ is denoted by $\partial^+(i) \equiv \{j \in \mathcal{J} : (i, j) \in E\}$. Similarly, the set of users connected to a cache $j \in \mathcal{J}$ is denoted by $\partial^-(j) \equiv \{i \in \mathcal{I} : (i, j) \in E\}$. The *in-degree* of the cache j is defined as $d_j \equiv |\partial^-(j)|, j \in \mathcal{J}$. For the sake of simplicity, we assume the network to be right d-regular, *i.e.,* $d_j = d, \forall j \in \mathcal{J}$. See Figure 2 for a schematic.
Each user requests one file per time slot. Each file-request may be served by any (one or more) neighboring caches. As before, the file-request generated by a user i at time t is one-hot encoded by an N-dimensional binary vector $\boldsymbol{x}_t^i$, with the interpretation that $x_{tf}^i = 1$ if and only if the f^{th} file is requested by the user i at time t. The cache configuration of the j^{th} cache at time t is represented by the N-dimensional vector $\boldsymbol{y}_t^j$, with each component denoting the fraction of the corresponding coded file cached. The cache configuration $\boldsymbol{y}_t$ must always satisfy the cache-capacity constraints. Thus, the set of all feasible cache configuration $Y_{\mathcal{J}}$ is given by:

$$Y_{\mathcal{J}} = \left\{ (\boldsymbol{y}^j, j \in \mathcal{J}) : \sum_{f=1}^{N} y_f^j \leq C, \forall j \in \mathcal{J}, \boldsymbol{0} \leq \boldsymbol{y} \leq \boldsymbol{1} \right\}. \tag{14}$$

For uncoded caching, $y_{ft}^j \in \{0, 1\}, \ \forall j, f, t$. Assumptions from Section 2.3 continue to apply.

3.2 Reward and Regret

For content distribution networks, it will be useful to distinguish between *elastic* and *inelastic* contents, as defined below. Making this distinction is essential due to the possibility that, in a caching network, the same content may be cached and retrieved from more than one cache at a time. Recall that, for rateless codes, it is only the *total number* of received encoded symbols that determine the decoding quality.

Elastic contents: We call a content to be *elastic* if receiving multiple layers (*i.e.,* resolutions) of the same content improves its overall utility for the users. Examples of elastic contents include multi-bitrate video files for adaptive streaming [27], [23], multi-resolution HD videos [33], and erasure-coded files in fault-tolerant distributed file systems, such as Hadoop [21], [62]. In this setting, an incoming file request $\boldsymbol{x}_t^i$ from the i^{th} user can be satisfied by fetching and combining parts of the cached layers of contents from different neighboring caches $j \in \partial^+(i)$. Accordingly, for elastic contents, we define the one-slot reward to be the aggregate of the cache-hits at that slot, *i.e.,*

$$q_{\text{elastic}}(\boldsymbol{x}_t, \boldsymbol{y}_t) \equiv \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \bigg(\sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j \bigg). \tag{15}$$

Hence, a user's utility increases linearly as she receives more layers of the requested content from different neighboring caches.

Inelastic contents: We call a content to be *inelastic* if the content has only a single layer (resolution), and a user is fully satisfied if she is able to retrieve the original content. Popular examples of inelastic contents include traditional webpages, databases, documents, and single resolution images. Similar to the elastic case, if the files are encoded by rate less MDS codes, a request from a user may be satisfied by combining different fractional parts of the content (from different neighboring caches), with the fractions adding up to unity. We define the one-slot reward for inelastic content to be

$$q_{\text{inelastic}}(\boldsymbol{x}_t, \boldsymbol{y}_t) \equiv \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \min \bigg\{ \mathbf{1}, \big(\sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j \big) \bigg\}, \tag{16}$$

where $\mathbf{1}$ is the all-one vector, and the $\min(\cdot,\cdot)$ operator outputs a vector whose components are the pointwise minimum of the corresponding components of the arguments. In comparison with the definition in Eqn. (15), the $\min(\mathbf{1},\cdot)$ operator in Eqn. (16) takes into account the fact that receiving multiple fractions of an inelastic content, summing up to more than one, does not add additional utility. Hence, unlike the elastic contents, inelastic contents have bounded rewards. We note that the reward in Eqn. (16) coincides with the utility definition in Eqn. (13) of [51]. The regret of any caching policy is defined exactly in the same way as in the single cache case via Eqns. (4) and (5).

3.3 Achievability for Caching Networks

In the following, we describe a simple and distributed gradient-based *coded* caching policy that achieves $O(\sqrt{T})$ regret in both elastic and inelastic settings. We then propose an extension of the Follow-the-Perturbed-Leader-based *uncoded* caching policy given in Algorithm 1, which also achieves near-optimal regret in the elastic setting.

Achievability with coded caching [51]: Let $q(\boldsymbol{x}, \boldsymbol{y})$ be a generic one-slot reward function, which is concave in the cache-configuration vector $\boldsymbol{y}$ (*e.g.,* $q(\cdot,\cdot)$ could be chosen to be either $q_{\text{elastic}}(\cdot,\cdot)$ or $q_{\text{inelastic}}(\cdot,\cdot)$). Let $\boldsymbol{g}_t$ be a supergradient of $q(\boldsymbol{x}_t, \boldsymbol{y})$ at $\boldsymbol{y} = \boldsymbol{y}_t$. The paper [51] describes the following Online Gradient Ascent (OGA)-based caching policy: starting from any initial feasible configuration $\boldsymbol{y}_0 \in Y_{\mathcal{J}}$, iterate as follows:

$$\boldsymbol{y}_{t+1} = \Pi_{Y_{\mathcal{J}}}(\boldsymbol{y}_t + \eta \boldsymbol{g}_t), \tag{17}$$

where $\boldsymbol{Y}_{\mathcal{J}}$ is the set of all feasible cache configurations given in Eqn. (14), $\boldsymbol{\Pi}_{Y_{\mathcal{J}}}(\cdot)$ is the Euclidean projection operator on the set $\boldsymbol{Y}_{\mathcal{J}}$, and $\eta > 0$ is an appropriate step-size parameter. For the single user- single cache setting of Section 2, we simply set $|\mathcal{I}| = |\mathcal{J}| = 1$.

Distributed implementation: The OGA-based caching policy can be implemented at each cache in a distributed fashion with locally available information only. This is true because

(1) A separable supergradient $\boldsymbol{g}_t$ for both the objective functions $q_{\text{elastic}}(\cdot,\cdot)$ and $q_{\text{inelastic}}(\cdot,\cdot)$ may be obtained, such that the gradient ascent steps in Eqn. (17) can be carried out locally. For an expression of such a supergradient, see Eqn. (30).

(2) Since the cache capacity constraints are separable for different caches, the projection operation $\mathbf{\Pi}_{Y_{\mathcal{J}}}(\cdot)$ may be carried out separately for each cache as $\mathbf{\Pi}_{Y_{\mathcal{J}}}(\boldsymbol{y}) = \{\Pi_{\mathcal{Y}}(\boldsymbol{y}_j),\ j \in \mathcal{J}\}$, where $\mathcal{Y}$ is the single cache constraint set given by Eqn. (1).

We have the following achievability result for OGA:

Theorem 4 (Achievability with coded caching [51]). *For both elastic and inelastic contents, the OGA-based caching policy (17) with step size $\eta = \frac{\sqrt{2C}}{d\sqrt{T}}$, achieves the following upper-bound on regret for a right d-regular bipartite network:*

$$R_T^{OGA} \leq d|\mathcal{J}|\sqrt{2CT}.$$

Proof outline: In this proof, we appeal to Theorem 2 of [51], which gives a generic regret upper bound for the OGA-based caching policy. We conclude the proof of Theorem 4 by computing the diameter of the feasible set $Y_{\mathcal{J}}$ subject to the cache-capacity constraints. Note that we can not directly use the regret upper-bound given in Theorem 3 of [51] because there the authors make an assumption that only one user out of $|\mathcal{I}|$ users may request for contents at a slot. In our model, there is no such restriction so that all $|\mathcal{I}|$ users may simultaneously request for contents at a slot. Please refer to Section 7.4 for a proof of Theorem 4.

Achievability with uncoded caching: For uncoded caching in a bipartite network, we propose a simple extension of the FTPL policy given in Algorithm 1 for a single cache. In this extension, each of the $|\mathcal{J}|$ caches *independently* implements the FTPL policy irrespective of whether the content is elastic or inelastic. We have the following achievability result for elastic contents:

Theorem 5 (Achievability with uncoded caching for elastic contents). *For elastic contents, the FTPL caching policy with the noise parameter $\eta = \frac{d}{(4\pi \log N)^{1/4}}\sqrt{\frac{T}{C}}$ yields the following upper-bound on expected regret for a right d-regular bipartite network:*

$$\mathbb{E}_{\{\boldsymbol{\gamma}_t\}_t}(R_T^{FTPL}) \leq 1.51(\log N)^{1/4} d|\mathcal{J}|\sqrt{CT}.$$

Please refer to Section 7.5 for a proof of Theorem 5.

3.4 Converse for Caching Networks

The question of regret-optimality of the OGA policy (17) for caching networks was left open in [51] due to lack of known lower bounds. In the following, we prove tight universal lower bounds for the regret, which applies to both coded and uncoded caching.

Theorem 6 (Lower bound for elastic contents). *For caching elastic contents in a bipartite network in the above set up with $N \geq 2C$, the regret of any online caching policy π is lower bounded as:*

$$R_T^{\pi} \geq d|\mathcal{J}|\sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}),\ \forall T \geq 1.$$

Proof outline: In this proof, we construct a *common* randomized file request sequence $\{\boldsymbol{X}_t\}_{t\geq 1}$, which is identical for each user. In other words, all users request the same random file at each slot. Thus, unlike most other applications of the probabilistic method, which usually proceeds with i.i.d. random variables, we consider a set of mutually *dependent* file request sequence. The expected reward accrued by any caching policy is then obtained by using the statistical symmetry of the file requests and the linearity of expectation. Finally, the static optimal caching configuration is identified, and the reward accrued by the optimal stationary policy is lower bounded by appealing to Lemma 1. Combining the above two results yields the regret lower bound. Please refer to Section 7.6 for the complete proof of Theorem 6.

The following theorem gives regret lower bound for caching inelastic content in a bipartite network.

Theorem 7 (Lower bound for inelastic contents). *For caching inelastic contents in a bipartite network in the above set up with $N \geq 2C|\mathcal{J}|$, the regret of any online caching policy π is lower bounded as:*

$$R_T^\pi \geq d\sqrt{\frac{|\mathcal{J}|CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}),\ \forall T \geq 1.$$

Proof outline: The principal difficulty in extending the argument from the proof of Theorem 6 to the inelastic case is the presence of non-linearity in the reward function (16) in the form of $\min(\cdot,\cdot)$ operator. As a result, it becomes difficult to analyze the expected reward accrued by the optimal stationary caching configuration. To get around this obstacle, we lower bound the reward of the optimal caching configuration with the help of a carefully constructed sub-optimal caching configuration $Y_\perp$. Interestingly, under the caching configuration $Y_\perp$, the non-linearity of the reward function vanishes, which leads to tractable analysis. Similar to the proof of Theorem 6, this proof also uses an identical (*i.e.,* dependent) file request sequence across all users. Please refer to Section 7.7 for the complete proof of Theorem 7.

Tightness: Comparing the regret upper bounds in Theorems 4 and 5 with the lower bounds in Theorems 6 and 7, we see that, for elastic contents, the OGA and FTPL network caching policies are regret-optimal up to a constant and poly-log factor respectively. For inelastic contents, the OGA policy is regret-optimal up to a factor of $O(\sqrt{|\mathcal{J}|})$.

4 EXPERIMENTS

Dataset description: In this section, we compare the performance of the existing caching policies with the proposed FTPL policy using a popular and stable benchmark - MovieLens 1 M dataset [1, 29]. This dataset contains $\sim$ 1 M ratings for $N \sim 3700$ movies, along with the timestamps of the ratings. The ratings were given by $\sim$ 6000 unique users. For our experiments, we assume that the users request movies from a CDN (such as Netflix) in the same chronological order as the recorded timestamps of the ratings. Hence, it is only the sequential order of the time-stamps of the requested files that matters in our experiment. We ignore the actual time-stamps and file sizes. A histogram of the movie request frequencies by taking all users together is shown in Fig. 3.

Experimental Setup: Following the standard industry practice, the cache capacity C of each cache is set to be a fixed α fraction of the total library size N, where we take $\alpha = 0.01$. For the bipartite caching scenario, we assume that a total of $|\mathcal{I}| = 10$ users are connected to $|\mathcal{J}| = 4$ caches. Each cache has in-degree $d = 3$. The first cache is connected to user 1, 2, and 3, the second cache is

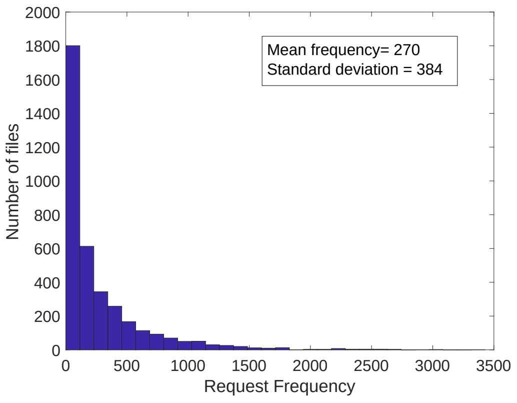

Fig. 3. Histogram of the frequency of requests of the MovieLens 1M dataset

connected to user 4, 5, and 6, the third cache is connected to user 7, 8, and 9, and the fourth cache is connected to user 1, 3 and 10. The entire dataset with $\sim 1M$ entries is uniformly divided into 10 disjoint blocks. Each user is allocated one block of the dataset. It is assumed that each user makes requests serially from its allocated block in the chronological order.

Results: The time-averaged regrets for different caching policies in the single cache setting and the bipartite network setting for both elastic and inelastic contents are plotted in Figure 4 (see the following page). From the plots in Figures 4 (a), 3 (b), and 3 (c), we conclude that the FTPL and LFU policies have the best performance in terms of average regret uniformly in all cases, and there is hardly any noticeable difference between their performance for large enough T. These two policies also perform very close to the theoretical lower bounds (corresponding to the worst-case request sequence). On the other hand, we find that the LRU policy has the worst performance, followed by OGA, which performs only marginally better. It is quite surprising to find that the uncoded caching policy FTPL outperforms the coded caching policy OGA in all scenarios. Figure 5 compares the time-averaged hit-rates of different policies (up to time $T = 10^5$ slots) for different cache-sizes in the single cache setting. It can be seen from the plot that the FTPL policy achieves a hit-rate of 80% when the cache-size is only 30% of the library size. At this cache-size, the LRU policy achieves a hit-rate of $\sim$ 70%. Figure 6 shows the variation of the average regret as the cache capacity is increased from 1 percent to 10 percent of the library size N for a fixed time $T = 5 \times 10^4$. These plots confirm that the regret increases with the cache size. However, we find that the increase in the regret is smallest for the LFU and FTPL policies followed by the OGA and the LRU policy.

4.1 Experiments with synthetic data - Cost-of-Caching vs Hit-Rate

The principal focus of this paper is on the fundamental limits of caching from an online learning point-of-view. Consequently, by following the assumption (2) of Section 2.3, we did not associate any cost with the downloading of new content to the cache at every slot so far. However, from an operational point-of-view, this is quite a strong assumption as most flash-based CDN caches (*e.g.,* used by Akamai, Netflix) wear out very quickly with repeated read-writes. Intuitively, there should be a trade-off between the Cost of Caching [3] and Hit-rates. In this Section, we argue that our algorithms are flexible enough to admit a smooth compromise between these two.

[3]Formally, we define the *Cost-of-Caching* at a slot to be the number of new files downloaded to the cache from the central server.

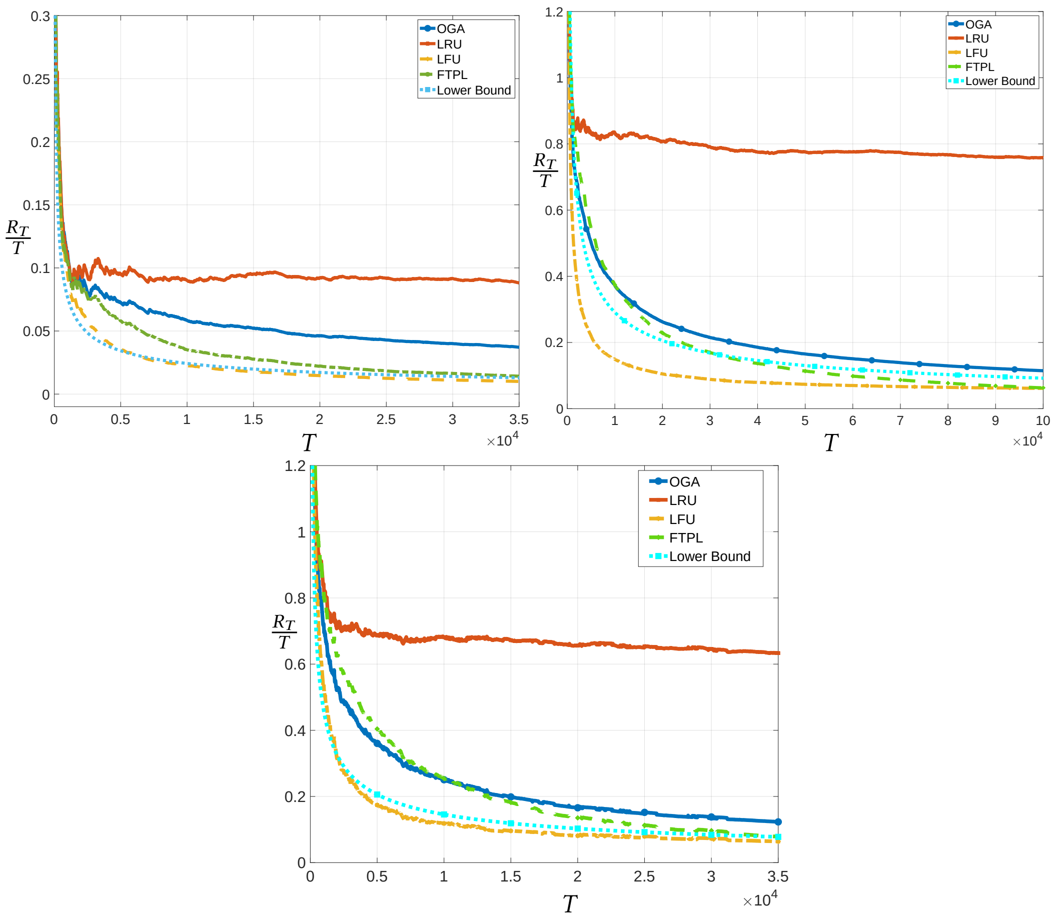

Fig. 4. Comparison among the caching policies in terms of time-averaged regret $\frac{R_T(\{\boldsymbol{x}_t\})}{T}$ (viz. Definition (4)) for (a) single cache (b) bipartite caching network with elastic contents, and (c) bipartite caching network with inelastic contents, where the file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ is obtained from the MovieLens dataset. The bipartite network has 10 users and 4 caches, each connected to 3 users. The capacity of each cache is chosen to be 1 percent of the library size in all cases.

To motivate this trade-off, let us first consider the greedy version of the proposed FTPL policy, which does not add any noise to the frequency counts (*i.e.*, $\eta = 0$). In the literature, this is also known as the Follow the Leader (FTL) policy. Since a user requests only one file at a time, the Hamming distance between two consecutive frequency count vectors is at most one. Hence, the FTL policy needs to cache at most one new file per slot. Thus, the cost of caching under the FTL policy is minimal. Unfortunately, it is well-known in the literature that the FTL policy suffers from linear regret [14]. This observation motivates us to use the FTPL policy with some non-zero noise. By varying the variance of the added Gaussian noise, we can strike a balance between the cost of caching and the hit rates of the caching policy.

To experimentally demonstrate this trade-off, we consider a single cache setting with library size $N = 5$ and cache capacity $C = 2$. The file request sequence is periodic and is given by

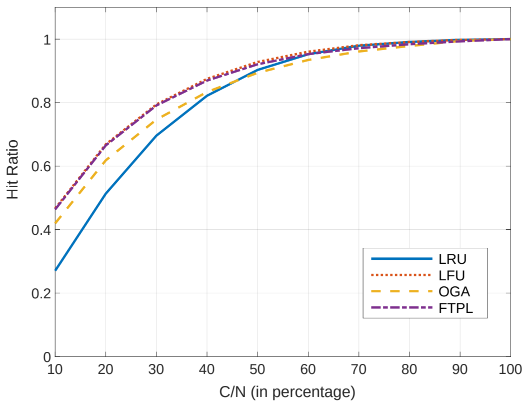

Fig. 5. Variation of the cache hit-ratios for different policies up to the time $T = 10^5$ for the MovieLens dataset in the single cache setting. The FTPL policy achieves a hit rate of 80% when the cache-size C is only 30% of the library size N.

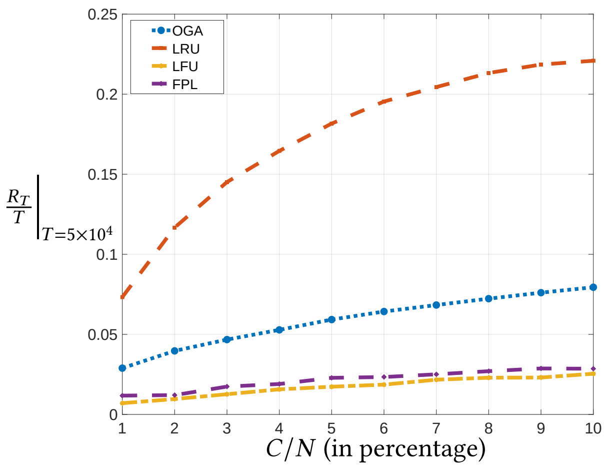

Fig. 6. Variation of time-averaged regret as a function of the cache-capacity (C) in the single cache setting for different caching policies

$\{\underbrace{5,4,3,2,1}, \underbrace{5,4,3,2,1}, 5,4,\ldots\}$. Clearly, classical policies, *e.g.,* LRU, LFU, have zero hit-rate on this request sequence. Figure 7 shows the variation of time-averaged cost of caching and hit-rate as a function of the noise level (standard-deviation) η. It is evident from the plots that (1) the FTL policy (which corresponds to $\eta = 0$) has hit-rate precisely zero, (2) hit-rate increases with increasing noise-level (η). However, this increased hit-rate comes at the expense of a higher cost of caching.

5 CONCLUSION AND FUTURE WORK

In this paper, we obtain tight sub-linear regret lower bounds for the online caching problem for single caches and bipartite caching networks. In the process, we derive a key technical result on the *balls-into-bins* problem and utilize the result in deriving all our lower bounds. We also propose a new randomized caching policy, called FTPL, which is shown to be both sound in theory and superior in practice. We envision the following future research directions stemming from this work:

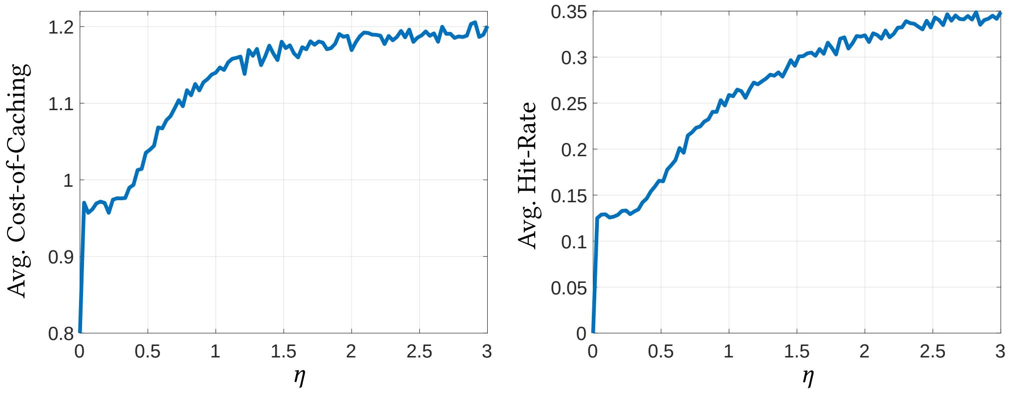

Fig. 7. Variation of time-averaged Cost-of-Caching and Hit-rate with the noise level η.

(1) As an immediate follow-up, it will be interesting to narrow down the $O(\sqrt{|\mathcal{J}|})$ gap between the lower and upper bounds on regret for inelastic contents in a bipartite caching network. Moreover, obtaining a regret guarantee for the FTPL policy for bipartite networks with inelastic contents would be nice. Relaxing Assumption (6) and allowing a bounded number $r \geq 1$ requests per user per slot will enhance the scope of the results. (2) We defined the reward functions primarily with the online performance of the caching policies (*i.e.*, hit rates) in mind. In particular, our reward definitions do not take into account the system cost associated with cache replacements at every slot (viz. Experiments in Section 4.1). It would be interesting to find the optimal trade-off between the hit-rates and the cost of cache replacements. (3) A variation of the caching problem arises in the context of inventory management where, instead of digital files, physical commodities are stored in the caches (*e.g.*, retail stores). Requests for the commodities arrive sequentially. The requested commodities, which are currently present in the cache, are immediately removed from the cache (*e.g.*, sold). Hence, unlike in our setting, there is no scope of "coding", and it would make sense to cache multiple copies of the same commodity at the same slot, subject to the cache capacity constraints (c.f., Eqn. (1)) . The performance of FTPL-like randomized uncoded caching policies will be interesting to investigate in this setup. (4) Finally, it will also be interesting to go beyond the single-hop setting of bipartite caching networks and design a regret-optimal joint routing and caching policy for multi-hop CDNs [40, 50].

6 ACKNOWLEDGEMENT

This work is supported by the Qualcomm grant IND-417880. The authors would like to thank the anonymous reviewers for many constructive comments, which helped to improve the overall presentation of the paper.

7 PROOF OF THE RESULTS

7.1 Proof of Lemma 1

We index the bins sequentially as $1, 2, \ldots, 2C$. Next, we logically combine every two consecutive bins $\{(2i-1, 2i)\}, 1 \leq i \leq C$, to obtain C *Super bins* (See Figure 8). Let us denote the (random) number of balls in the i^{th} super bin by $X_i, j = 1, 2, \ldots, C$. Conditioned on the r.v. X_i, the number of balls in the corresponding bins: $2i-1$ and $2i$ are jointly distributed as $(Z, X_i - Z)$, where Z is a binomial random variable with parameter $(X_i, \frac{1}{2})$. Let H_i denote the maximum number of balls

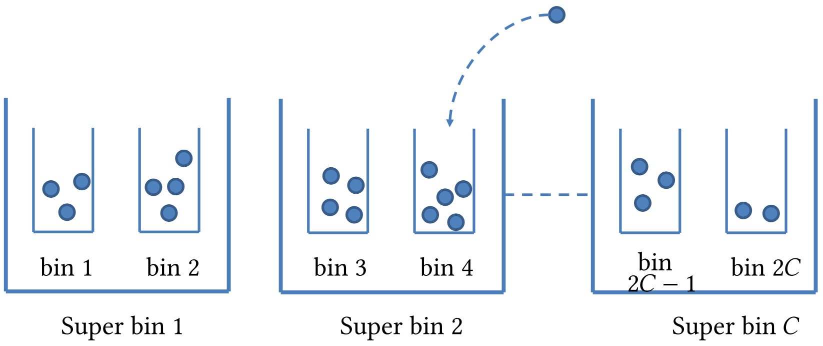

Fig. 8. Illustrating the construction of Super bins

between the corresponding bins $2i-1$ and $2i$. Then, as shown in the proof Theorem 1, when $X_i > 0$:

$$\mathbb{E}(H_i|X_i) \geq \frac{X_i}{2} + \sqrt{\frac{X_i}{2\pi}} - \frac{1}{2\sqrt{2\pi X_i}}, \ \forall 1 \leq i \leq C. \tag{18}$$

Since $M_C \geq \sum_{i=1}^{C} H_i$, we have

$$\begin{aligned} & \mathbb{E}(M_C) \\ \geq \ & \mathbb{E}\Big(\sum_{i=1}^{C} H_i\Big) = \sum_{i=1}^{C} \mathbb{E}(H_i) \overset{(a)}{=} C\mathbb{E}(H_1) \overset{(b)}{=} C\mathbb{E}\big(H_1 \mathbb{1}(X_1 > 0)\big) \\ \overset{(c)}{=} \ & C\mathbb{E}\mathbb{E}(H_1 \mathbb{1}(X_1 > 0)|X_1) \\ \overset{(d)}{\geq} \ & C\mathbb{E}\Big(\frac{X_1 \mathbb{1}(X_1 > 0)}{2} + \sqrt{\frac{X_1}{2\pi}}\mathbb{1}(X_1 > 0) - \frac{\mathbb{1}(X_1 > 0)}{2\sqrt{2\pi X_1}}\Big) \\ \overset{(e)}{=} \ & \frac{C}{2}\mathbb{E}(X_1) + \frac{C}{\sqrt{2\pi}}\mathbb{E}(\sqrt{X_1}) - \frac{C}{2\sqrt{2\pi}}\mathbb{E}\Big(\frac{\mathbb{1}(X_1 > 0)}{\sqrt{X_1}}\Big), \end{aligned} \tag{19}$$

where the equality (a) follows from the fact that the random variables $\{H_i\}_{i=1}^{C}$ have identical distribution. For equation (b), we write

$$H_1 = H_1 \mathbb{1}(X_1 = 0) + H_1 \mathbb{1}(X_1 > 0).$$

Now, observe that if $X_1 = 0$ then $H_1 = 0$ a.s. Hence, almost surely, we have $H_1 = H_1 \mathbb{1}(X_1 > 0)$. The equation (c) follows from the tower property of conditional expectation, the inequality (d) follows from the bound (18), and the equality (e) follows from the facts that $X_1 = X_1 \mathbb{1}(X_1 > 0)$, $\sqrt{X_1} = \sqrt{X_1}\mathbb{1}(X_1 > 0)$. The Lemma now follows by using the bounds on moments of the binomial distribution as computed next.

Bounding the Expectations in Eqn. (19)*:* For bounding the middle term in Eqn. (19), consider the factorization [28]:

$$\sqrt{x} - \Big(1 + \frac{x-1}{2} - \frac{(x-1)^2}{2}\Big) = \frac{\sqrt{x}}{2}(\sqrt{x}-1)^2(\sqrt{x}+2).$$

The RHS is non-negative for any $x \geq 0$. Thus, we have the following algebraic inequality:

$$\sqrt{x} \geq 1 + \frac{x-1}{2} - \frac{(x-1)^2}{2}, \ \forall \, x \geq 0.$$

Replacing the variable x with the random variable $\frac{X_1}{\mathbb{E}(X_1)}$ point wise, we have almost surely,

$$\sqrt{\frac{X_1}{\mathbb{E}(X_1)}} \geq 1 + \frac{\frac{X_1}{\mathbb{E}(X_1)} - 1}{2} - \frac{(\frac{X_1}{\mathbb{E}(X_1)} - 1)^2}{2}.$$

Taking expectation of both sides, the above yields

$$\mathbb{E}(\sqrt{X_1}) \geq \sqrt{\mathbb{E}(X_1)}\Big(1 - \frac{\mathsf{Var}(X_1)}{2(\mathbb{E}(X_1))^2}\Big). \tag{20}$$

Finally, recall that $X_1 \sim \mathsf{Binom}(T, \frac{1}{C})$. Hence, $\mathbb{E}(X_1) = \frac{T}{C}$ and $\mathsf{Var}(X_1) = T\frac{1}{C}(1 - \frac{1}{C}) \leq \frac{T}{C}$. Using this, Eqn. (20) yields the following lower bound

$$\mathbb{E}(\sqrt{X_1}) \geq \sqrt{\frac{T}{C}} - \frac{1}{2}\sqrt{\frac{C}{T}}. \tag{21}$$

For bounding the last term in Eqn. (19), we write

$$\frac{\mathbb{1}(X_1 > 0)}{\sqrt{X_1}} \leq 1\mathbb{1}\big(X_1 \leq \frac{T}{2C}\big) + \sqrt{\frac{2C}{T}}.$$

Recall that $X_1 \sim \mathsf{Binom}(T, \frac{1}{C})$. Taking expectation of both sides of the above inequality, we have

$$\mathbb{E}\Big(\frac{\mathbb{1}(X_1 > 0)}{\sqrt{X_1}}\Big) \leq \mathbb{P}(X_1 \leq \frac{1}{2}\mathbb{E}(X_1)) + \sqrt{\frac{2C}{T}}.$$

The probability term in the above expression may be bounded using Chebyshev's inequality as follows:

$$\begin{aligned}
\mathbb{P}\big(X_1 \leq \frac{1}{2}\mathbb{E}(X_1)\big) &= \mathbb{P}\big(X_1 - \mathbb{E}(X_1) \leq -\frac{1}{2}\mathbb{E}(X_1)\big) \\
&\leq \mathbb{P}\big(|X_1 - \mathbb{E}(X_1)| \geq \frac{1}{2}\mathbb{E}(X_1)\big) \\
&\leq \frac{4\mathsf{Var}(X_1)}{(\mathbb{E}(X_1))^2} = 4\frac{C-1}{T}
\end{aligned}$$

Taking the above bounds together, we obtain

$$\mathbb{E}\Big(\frac{\mathbb{1}(X_1 > 0)}{\sqrt{X_1}}\Big) \leq \sqrt{\frac{2C}{T}} + \frac{4C}{T}. \tag{22}$$

Finally, combining Eqns. (19), (21), and (22) together, we obtain

$$\mathbb{E}(M_C) \geq \frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \frac{(\sqrt{2}+1)C^{3/2}}{2\sqrt{2\pi T}} - \sqrt{\frac{2}{\pi}}\frac{C^2}{T}. \blacksquare$$

7.2 Proof of Theorem 2

To lower bound the regret R_T^π in equation (7), as in Section 2.4, we consider a random file request sequence $\{X_t\}_{t\geq 1}$, each sampled independently and uniformly at random from the set of first $2C$ unit vectors $\{\boldsymbol{e}_i, 1 \leq i \leq 2C\}$ of dimension N [4]. In other words, at every time slot, the user independently requests a random file from the set of first $2C$ files uniformly at random.

[4]Recall that, the unit vector $\boldsymbol{e}_i$ has one at its i^{th} coordinate and zeros everywhere else.

The expected reward obtained by any caching policy, given by the second term in Eqn. (4), is now easy to evaluate:

$$\mathbb{E}\Big(\sum_{t=1}^{T} \boldsymbol{Y}_t \cdot \boldsymbol{X}_t\Big) \overset{(a)}{=} \sum_{t=1}^{T} \boldsymbol{Y}_t \cdot \big(\mathbb{E}\boldsymbol{X}_t\big) = \frac{1}{2C}\sum_{t=1}^{T}\sum_{k=1}^{2C} \boldsymbol{Y}_{tk} \overset{(b)}{\leq} \frac{T}{2}, \tag{23}$$

where, in (a), we have used the fact that the caching decision $\boldsymbol{Y}_t$ made at time t is independent of the incoming file request $\boldsymbol{X}_t$, and in (b), we have made use of the cache-capacity constraint (1):

$$\sum_{k=1}^{2C} \boldsymbol{Y}_{tk} \leq \sum_{k=1}^{N} \boldsymbol{Y}_{tk} \leq C,$$

in addition to the fact that

$$\mathbb{E}(X_{tk}) = \begin{cases} \frac{1}{2C}, \ \forall 1 \leq k \leq 2C \\ 0 \text{ o.w.} \end{cases}$$

Note that, for any given file request sequence $\{\boldsymbol{x}_t\}_{t=1}^{T}$, the optimal *offline* stationary cache configuration vector is obtained by caching the most popular C files. Hence, the optimal vector $\boldsymbol{y}^* \in [0,1]^N$, corresponding to the first term in Eqn. (4) is obtained by simply setting the coordinates corresponding to the *maximum C coordinates* of the N-dimensional vector $\sum_{t=1}^{T} \boldsymbol{x}_t$ to unity. Given the distribution of the file request sequence, it immediately follows that the reward accrued by the optimal stationary policy $\boldsymbol{Y}^* \cdot \sum_{t=1}^{T} \boldsymbol{X}_t$ is identically distributed to the total number of balls in the most heavily loaded C bins when a total of T balls are randomly thrown into $2C$ bins. Finally, invoking Lemma 1, we have

$$\mathbb{E}\Big(\boldsymbol{Y}^* \cdot \sum_{t=1}^{T} \boldsymbol{X}_t\Big) \geq \frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}). \tag{24}$$

Combining equations (23) and (24), we obtain the following lower bound for regret in the single cache setting:

$$\begin{aligned} R_T &\geq \mathbb{E}_{\{\boldsymbol{X}_t\}_{t=1}^{T}}\Big(\boldsymbol{Y}^* \cdot \sum_{t=1}^{T} \boldsymbol{X}_t - \sum_{t=1}^{T} \boldsymbol{Y}_t \cdot \boldsymbol{X}_t\Big) \\ &\geq \sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}). \ \blacksquare \end{aligned}$$

7.3 Proof of Theorem 3

Our proof follows a similar line of arguments as the proof of Theorem 1 of [18]. However, we improve the regret upper bound by a factor of $O(\sqrt{C})$. This additional improvement results from making use of the constraint that only one file is requested at every slot. The notations of [18] are slightly altered in order to remain consistent throughout the paper.

Let the set $\mathcal{Y}$ denote the set of all possible uncoded caching configuration in the single cache setting. Clearly, $|\mathcal{Y}| = \binom{N}{C}$. Define the potential function

$$\Phi_\eta(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\gamma}\sim\mathcal{N}(0,I)}\Big[\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{x} + \eta\boldsymbol{\gamma}\rangle\Big].$$

Also, denote the cumulative file request arrivals to the cache up to time $t-1$ by $\boldsymbol{X}_t = \sum_{\tau=1}^{t-1} \boldsymbol{x}_\tau$ with $\boldsymbol{X}_1 = \boldsymbol{0}$. Then, as shown in Eqn. (3) of [18], the expected regret of the FTPL policy in Algorithm 1 with noise variance η^2 is upper bounded as [5]

$$\mathbb{E}(R_T) \leq \Phi_\eta(\boldsymbol{X}_1) + \frac{1}{2}\sum_{t=1}^{T}\langle \boldsymbol{x}_t, \nabla^2\Phi_\eta(\tilde{\boldsymbol{x}}_t)\boldsymbol{x}_t\rangle, \tag{25}$$

for some $\tilde{\boldsymbol{x}}_t$ connecting the line segment $\boldsymbol{X}_t$ and $\boldsymbol{X}_{t+1}$. Following standard notation, the operator $\langle \cdot, \cdot \rangle$ denotes the inner-product of the two arguments, and the operator $\nabla^2(\cdot)$ denotes the Hessian of the argument.
Next, we bound each of the above two terms separately. The first term may be bounded in the same way as in [18]:

$$\Phi_\eta(\boldsymbol{X}_1) \leq \eta\sqrt{2C\log\binom{N}{C}} \leq C\eta\sqrt{2\log N},$$

where the last inequality follows from the fact that $\binom{N}{C} \leq N^C$. Since only one file is requested at every slot, the quadratic form above may be upper bounded as

$$\langle \boldsymbol{x}_t, \nabla^2\Phi_\eta(\tilde{\boldsymbol{x}_t})\boldsymbol{x}_t\rangle \leq \max_{i,\boldsymbol{x}}\left(|\nabla^2\Phi_\eta(\boldsymbol{x})|\right)_{ii}. \tag{26}$$

Moreover, following Lemma 7 of [2], we have that

$$\left(\nabla^2\Phi_\eta(\boldsymbol{x})\right)_{ij} = \frac{1}{\eta}\mathbb{E}\left[\hat{y}(\tilde{\boldsymbol{x}}_t + \eta\boldsymbol{\gamma})_i\gamma_j\right],$$

where $\hat{y}(z) \in \arg\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{y}, z\rangle$. Hence, using Jensen's inequality we have that

$$\left(|\nabla^2\Phi_\eta(\boldsymbol{x})|\right)_{ii} \leq \frac{1}{\eta}\mathbb{E}\left[|\hat{y}(\tilde{\boldsymbol{x}}_t + \eta\boldsymbol{\gamma})_i||\gamma_i|\right] \overset{(a)}{\leq} \frac{1}{\eta}\mathbb{E}\left[|\gamma_i|\right] \overset{(b)}{=} \frac{1}{\eta}\sqrt{\frac{2}{\pi}}, \tag{27}$$

where the inequality (a) follows from the fact that for all $\boldsymbol{y} \in \mathcal{Y}$, we have $y_i \in \{0, 1\}$, and the equality (b) follows from the fact that $\gamma_i \sim \mathcal{N}(0, 1)$. Hence, substituting the above bounds in Eqn. (25), we have the following upper bound on expected regret under the FTPL caching policy:

$$\mathbb{E}(R_T) \leq C\eta\sqrt{2\log N} + \frac{T}{\eta\sqrt{2\pi}}.$$

Finally, choosing $\eta = \frac{1}{(4\pi\log N)^{1/4}}\sqrt{\frac{T}{C}}$, yields the following regret upper bound for the FTPL policy:

$$\mathbb{E}_{\{\boldsymbol{\gamma}_t\}_t}(R_T^{\text{FTPL}}) \leq 1.51(\log N)^{1/4}\sqrt{CT}. \blacksquare$$

7.4 Proof of Theorem 4

From Eqn. (15), we know that $q_{\text{elastic}}(\boldsymbol{x}, \boldsymbol{y})$ is linear (and hence, concave) in the cache-configuration vector $\boldsymbol{y}$. Moreover, since the pointwise minimum of linear functions is concave [11], it follows from Eqn. (16) that the reward function $q_{\text{inelastic}}(\boldsymbol{x}, \boldsymbol{y})$ is also concave in the cache-configuration vector $\boldsymbol{y}$. To obtain a regret upper bound for the the OGA algorithm (17), we appeal to Theorem 2 of [51], which states that with an appropriate choice of the step-size parameter η,

$$R_T^{\text{OGA}} \leq \text{diam}(Y_{\mathcal{J}})L\sqrt{T}, \tag{28}$$

where $\text{diam}(Y_{\mathcal{J}})$ denotes the Euclidean diameter [57] of the set $Y_{\mathcal{J}}$ defined in (14), and L is an upper-bound for the 2-norm of the (super) gradient of the reward function.

[5]The signs are flipped as we are in the rewards maximization setting, as opposed to the loss minimization setting of [18].

For bounding the diameter, consider any two vectors $\boldsymbol{y}$ and $\boldsymbol{z}$ from the set $Y_{\mathcal{J}}$. We have

$$\begin{aligned} ||\boldsymbol{y}-\boldsymbol{z}||_2^2 &= \sum_{j\in\mathcal{J}}\sum_{f=1}^{N}(y_f^j - z_f^j)^2 \\ &\overset{(a)}{\leq} \sum_{j\in\mathcal{J}}\sum_{f=1}^{N}|y_f^j - z_f^j| \\ &\overset{(b)}{\leq} \sum_{j\in\mathcal{J}}\sum_{f=1}^{N}y_f^j + \sum_{j\in\mathcal{J}}\sum_{f=1}^{N}z_f^j \overset{(c)}{\leq} 2CJ, \end{aligned}$$

where the inequality (a) follows from the fact that $|y_f^j - z_f^j| \leq 1$, the inequality (b) follows from triangle inequality, and finally, inequality (c) follows from the cache-capacity constraints. Since, the above bound is valid for any two vectors in the set $Y_{\mathcal{J}}$, it follows that

$$\text{diam}(Y_{\mathcal{J}}) \equiv \sup_{\boldsymbol{y},\boldsymbol{z}\in Y_{\mathcal{J}}} ||\boldsymbol{y}-\boldsymbol{z}|| \leq \sqrt{2CJ}. \tag{29}$$

Next, we bound the norm of the supergradients of the reward functions. Recall,

$$q_{\text{elastic}}(\boldsymbol{x},\boldsymbol{y}) = \sum_{i\in\mathcal{I}} \boldsymbol{x}^i \cdot \Big(\sum_{j\in\partial^+(i)} \boldsymbol{y}^j\Big).$$

Hence, we have the following expression for the supergradient of the objective function:

$$\Big(\nabla_{\boldsymbol{y}} q_{\text{elastic}}(\boldsymbol{x},\boldsymbol{y})\Big)_f^j = \sum_{i\in\partial^-(j)} x_f^i. \tag{30}$$

Thus,

$$||\nabla_{\boldsymbol{y}} q_{\text{elastic}}(\boldsymbol{x},\boldsymbol{y})||_2^2 = \sum_{j\in\mathcal{J}}\sum_{f}\Big(\sum_{i\in\partial^-(j)} x_f^i\Big)^2. \tag{31}$$

On the other hand, since $\boldsymbol{x} \geq \boldsymbol{0}$, it follows that the vector $\nabla_{\boldsymbol{y}} q_{\text{elastic}}(\boldsymbol{x},\boldsymbol{y})$ can be taken to be a supergradient of the concave function $q_{\text{inelastic}}(\boldsymbol{x},\boldsymbol{y})$ w.r.t. the argument $\boldsymbol{y}$. Thus, to obtain the regret upper bound, it only remains to upper bound the RHS of Eqn. (31). We have:

$$\Big(\sum_{i\in\partial^-(j)} x_f^i\Big)^2 \overset{(a)}{\leq} d \sum_{i\in\partial^-(j)} (x_f^i)^2 \overset{(b)}{=} d \sum_{i\in\partial^-(j)} x_f^i,$$

where the inequality (a) follows from Cauchy-Schwartz inequality and (b) follows from the fact that each x_f^i's are either zero or one. Hence, the RHS of Eqn. (31) is upper bounded as follows:

$$\begin{aligned} \sum_{j\in\mathcal{J}}\sum_{f}\Big(\sum_{i\in\partial^-(j)} x_f^i\Big)^2 &\leq d\sum_{j\in\mathcal{J}}\sum_{f}\sum_{i\in\partial^-(j)} x_f^i \\ &= d\sum_{j\in\mathcal{J}}\sum_{i\in\partial^-(j)}\sum_{f} x_f^i \\ &\overset{(a)}{\leq} d\sum_{j\in\mathcal{J}}\sum_{i\in\partial^-(j)} 1 = d^2|\mathcal{J}|, \end{aligned}$$

where (a) follows from the facts that each user can request at most one file per slot, and the network is right d-regular. Hence, for both elastic and inelastic reward functions, the 2-norm of the supergradients is upper bounded as

$$L \leq d\sqrt{|\mathcal{J}|}. \tag{32}$$

Hence, combining Eqns. (28), (29), and (32), we have that

$$R_T^{\text{OGA}} \leq d|\mathcal{J}|\sqrt{2CT}. \blacksquare$$

7.5 Proof of Theorem 5

Let R_T^{single} denote the regret of a single cache under the FTPL caching policy. Due to linearity of the reward functions for elastic contents, the regret for bipartite network caching may be simply upper bounded by summing the regret for each of the $|\mathcal{J}|$ constituent single caches, *i.e.,*

$$R_T \leq |\mathcal{J}|R_T^{\text{single}}. \tag{33}$$

However, note that we can not use the regret upper bound from Theorem 3 to upper bound the regret of the individual caches, as, unlike the setting of Theorem 3, each individual caches may receive up to d requests from its neighboring users per slot. Hence, we need to modify the proof of Theorem 3 to take up to d requests into account. Following the same line of argument as in the proof of Theorem 7.3, the inequality (26) may be replaced by the following

$$\langle \boldsymbol{x}_t, \nabla^2\Phi_\eta(\tilde{\boldsymbol{x}_t})\boldsymbol{x}_t)\rangle \leq d^2 \max_{i,j,\boldsymbol{x}} \big(|\nabla^2\Phi_\eta(\boldsymbol{x})|\big)_{ij}. \tag{34}$$

Eqn. (34) follows from the fact that since the number of non-zero terms in $\boldsymbol{x}_t$ is at most d (*i.e.,* $||\boldsymbol{x}_t||_0 \leq d, ||\boldsymbol{x}_t||_1 \leq d$). Finally, proceeding similarly as in Eqn. (27), we conclude that

$$\big(|\nabla^2\Phi_\eta(\boldsymbol{x})|\big)_{ij} \leq \frac{1}{\eta}\sqrt{\frac{2}{\pi}}, \ \forall \boldsymbol{x}, i, j.$$

Plugging in the above in Eqn. (25) results in the following regret bound for the FTPL caching policy:

$$\mathbb{E}(R_T^{\text{single}}) \leq C\eta\sqrt{2\log N} + d^2\frac{T}{\eta\sqrt{2\pi}}. \tag{35}$$

Choosing $\eta = \frac{d}{(4\pi \log N)^{1/4}}\sqrt{\frac{T}{C}}$, and combining Eqns (33) and (35), we have the following regret upper bound for the FTPL policy for caching elastic contents in a d-right regular bipartite caching network:

$$\mathbb{E}_{\{\boldsymbol{\gamma}_t\}_t}(R_T) \leq 1.51(\log N)^{1/4}d|\mathcal{J}|\sqrt{CT}. \blacksquare$$

7.6 Proof of Theorem 6

We prove a slightly general result without assuming the bipartite caching network to be right d-regular. Let $\boldsymbol{x}_t^i$ be the file request vector from the i^{th} user, and $\boldsymbol{y}_t^j$ be the cache configuration vector of the j^{th} cache selected by the policy π at time t, where $i \in \mathcal{I}, j \in \mathcal{J}$. Hence, due to the elastic nature of the content, the total reward G_T^π accrued by the caching policy π is given by Eqn. (15):

$$G_T^\pi = \sum_{t=1}^T \sum_{i\in\mathcal{I}} \boldsymbol{x}_t^i \cdot \sum_{j\in\partial^+(i)} \boldsymbol{y}_t^j.$$

Recall that, while computing the regret of a policy π via Eqn. (4), we compare the reward accrued by the policy π to that of a stationary caching policy π^* equipped with the hindsight knowledge (*i.e.,* π^* knows the entire file request sequence $\{\boldsymbol{x}_t\}_1^T$ in advance, *viz.,* Eqns. (4)- (5)). Let $\boldsymbol{y}_*^j$ be the optimal stationary cache configuration vector at the server $j \in \mathcal{J}$ set by the policy π^*. Then the reward accrued by the stationary caching policy π^* is given by:

$$G_T^{\pi^*} = \sum_{t=1}^{T} \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \sum_{j \in \partial^+(i)} \boldsymbol{y}_*^j.$$

Rearranging the terms, we have

$$G_T^{\pi^*} = \sum_{j \in \mathcal{J}} \boldsymbol{y}_*^j \cdot \Big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{x}_t^i \Big). \tag{36}$$

Since the j^{th} cache has a total capacity of C, from the above equation, it is clear that the optimal stationary configuration $\boldsymbol{y}_*^j$ corresponds to caching the most popular C files requested by all of j's in-neighbours taken together. Following the probabilistic method described in Section 2.4, we now construct a randomized file request sequence which are *identical* for each user, *i.e.*, all users request the *same* random file at every time slot. At time slot t, the file request from the i^{th} user is given by the vector $\boldsymbol{X}_t^i = \boldsymbol{X}_t$, where the vector $\boldsymbol{X}_t$ is sampled independently and uniformly at random from the set of first $2C$ unit vectors $\{\boldsymbol{e}_i \in \mathbb{R}^N, 1 \leq i \leq 2C\}$. With this set up, the expected reward accrued by the policy π (*i.e.,* the second term in Eqn. (4)) may be evaluated as:

$$\begin{aligned}
\mathbb{E}\Big(\sum_{t=1}^{T} \sum_{i \in \mathcal{I}} X_t^i \cdot \sum_{j \in \partial^+(i)} Y_t^j \Big) &= \sum_{t=1}^{T} \sum_{i \in \mathcal{I}} \sum_{j \in \partial^+(i)} \sum_{f=1}^{N} \mathbb{E}(X_{tf}) Y_{tf}^j \\
&\overset{(a)}{=} \frac{1}{2C} \sum_{t=1}^{T} \sum_{i \in \mathcal{I}} \sum_{j \in \partial^+(i)} \sum_{f=1}^{2C} Y_{tf}^j \\
&\overset{(b)}{\leq} \frac{1}{2C} \sum_{t=1}^{T} \sum_{i \in \mathcal{I}} \sum_{j \in \partial^+(i)} C \overset{(c)}{=} \frac{T}{2} \sum_j d_j,
\end{aligned}$$

where the equation (a) uses the fact that the random variables $\boldsymbol{X}_t$ and $\boldsymbol{Y}_t$ are independent and $\mathbb{E}(X_{tf}) = \frac{1}{2C}, \forall t, f$. The inequality (b) follows from the fact that each cache has capacity C, and the equality (c) follows from interchanging the order of the summations.
From Eqn. (36), we can express the expected total reward $G_T^{\pi^*}$ accrued by the stationary policy π^* as:

$$\mathbb{E}(G_T^{\pi^*}) = \sum_{j \in \mathcal{J}} \mathbb{E}(M_C^j),$$

where M_C^j denotes the sum of the largest C coordinates of the following N-dimensional random vector:

$$\sum_{t=1}^{T} \sum_{i \in \delta^-(j)} \boldsymbol{X}_t^i = d_j \sum_{t=1}^{T} \boldsymbol{X}_t.$$

Thus, it follows that the random variable M_C^j is statistically identical to d_j times the total number of balls in the most occupied C bins when T number of balls are thrown independently and uniformly at random into $2C$ bins. Finally, appealing to Lemma 1, we have

$$\mathbb{E}(M_C^j) \quad \geq \quad d_j \Bigg(\frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}) \Bigg). \tag{37}$$

Finally, combining the above, we have the following regret lower bound for caching elastic contents in a bipartite caching network:

$$R_T^\pi \geq \sqrt{\frac{CT}{2\pi}}\left(\sum_{j\in\mathcal{J}} d_j\right) - \Theta(\frac{1}{\sqrt{T}}) = d|\mathcal{J}|\sqrt{\frac{CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}). \blacksquare$$

7.7 Proof of Theorem 7

We begin our analysis with the following two observations:

(1) From the definition of the reward functions (15) and (16), in general, we have,

$$q_{\text{inelastic}}(\boldsymbol{x}, \boldsymbol{y}) \leq q_{\text{elastic}}(\boldsymbol{x}, \boldsymbol{y}), \ \forall \boldsymbol{x}, \boldsymbol{y}. \tag{38}$$

(2) In the special case, when different servers $j \in \mathcal{J}$ cache *different* items, *i.e.,* $\boldsymbol{y}^i \cdot \boldsymbol{y}^j = 0, \forall i \neq j$, we have

$$q_{\text{inelastic}}(\boldsymbol{x}, \boldsymbol{y}) = q_{\text{elastic}}(\boldsymbol{x}, \boldsymbol{y}), \ \forall \boldsymbol{x}, \boldsymbol{y}. \tag{39}$$

Eqn. (39) follows from the fact that in the product $\boldsymbol{x}_t^i \cdot \left(\sum_{j\in\partial^+(i)} \boldsymbol{y}_t^j\right) = \sum_{j\in\partial^+(i)} \boldsymbol{x}_t^i \cdot \boldsymbol{y}_t^j$, only one of the dot-product terms could be strictly positive, as different servers cache different items and the vector $\boldsymbol{x}_t^i$ has only one positive component. Moreover, the value of each dot product is at most one, as each server stores at most one copy of each file.
With these two observations at hand, we now relate our argument below to the proof of Theorem 6 to obtain a regret lower bound for inelastic contents. Similar to the proof of Theorem 6, we construct a randomized and *identical* file request sequence $\{\boldsymbol{X}_t^i \equiv \boldsymbol{X}_t\}_{t\geq 1}$ for each user, where the random variable $\boldsymbol{X}_t$ is sampled independently and uniformly at random from the set of *first* $2C|\mathcal{J}|$ unit vectors $\{\boldsymbol{e}_i \in \mathbb{R}^N, 1 \leq i \leq 2C|\mathcal{J}|\}$. Hence, using observation (1) and working similarly as in Eqn. (37), the expected total reward G_T^π accrued by any online caching policy may be upper bounded as

$$G_T^\pi \leq \mathbb{E}\left(\sum_{t=1}^T \sum_{i\in\mathcal{I}} \boldsymbol{X}_t^i \cdot \sum_{j\in\partial^+(i)} Y_t^j\right) \leq \frac{T}{2|\mathcal{J}|}\sum_{j\in\mathcal{J}} d_j = \frac{Td}{2}. \tag{40}$$

Note that, unlike the elastic setting, due to the presence of the min operator in the reward function (16), obtaining an optimal cache configuration vector $\boldsymbol{Y}^*$ is non-trivial in the inelastic setting. However, since we only require a lower bound to the total expected reward accrued by the optimal static policy π^*, any suitably constructed sub-optimal caching configuration will serve the purpose, provided we can evaluate its reward. Towards this end, in the following, we construct an $N|\mathcal{J}|$ dimensional stationary cache-configuration vector $\boldsymbol{Y}_\perp$ (obtained by stacking the N-dimensional cache-configuration vectors $\boldsymbol{Y}_\perp^j$ together $\forall j \in \mathcal{J}$) with the property that

$$Y_\perp^i \cdot Y_\perp^j = 0, \ \forall i \neq j, \ i, j \in \mathcal{J}. \tag{41}$$

The cache-configuration vector $\boldsymbol{Y}_\perp$ is constructed by first sorting the vector $\boldsymbol{v} \equiv \sum_{t=1}^T \boldsymbol{X}_t$ in non-increasing order. Then, we let the cache configuration $Y_\perp^j$ correspond to the set of C files from rank $(j-1)C+1$ to jC. Clearly, the above construction ensures property (41). Let the operator $S_{\boldsymbol{v}}(m, n)$ denote the sum of the coordinates running from m to n of the vector $\boldsymbol{v}$, sorted in non-increasing order. With the above construction, since all users make the same file request at each time (*i.e.,*

$\boldsymbol{X}_t^i = \boldsymbol{X}_t, \forall i$), we have $\forall j \in \mathcal{J}$:

$$\begin{aligned} Y_{\perp}^j \cdot \Big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{X}_t^i \Big) &= dY_{\perp}^j \cdot \boldsymbol{v} \\ &= dS_{\boldsymbol{v}}\Big((j-1)C+1, jC\Big). \end{aligned} \tag{42}$$

Finally, the reward accrued by the optimal stationary policy π^* may be lower-bounded as:

$$\begin{aligned} G_T^{\pi^*} &\overset{(a)}{\geq} \mathbb{E}\Big(\sum_{j \in \mathcal{J}} Y_{\perp}^j \cdot \big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{X}_t^i \big) \Big) \\ &\overset{(b)}{=} \sum_{j \in \mathcal{J}} \mathbb{E}\Big(Y_{\perp}^j \cdot \big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{X}_t^i \big) \Big) \\ &\overset{(c)}{=} d \sum_{j \in \mathcal{J}} \mathbb{E}\Big(S_{\boldsymbol{v}}\big((j-1)C+1, jC\big) \Big) \\ &= d\mathbb{E}\Big(\sum_{j \in \mathcal{J}} S_{\boldsymbol{v}}\big((j-1)C+1, jC\big) \Big) \\ &= d\mathbb{E}\big(S_{\boldsymbol{v}}(1, |\mathcal{J}|C)\big) \\ &\overset{(d)}{\geq} \frac{Td}{2} + d\sqrt{\frac{|\mathcal{J}|CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}), \end{aligned} \tag{43}$$

where the inequality (a) follows from Eqn. (39) of observation 2, the equation (b) follows from the linearity of expectation, and the inequality (c) follows from Eqn. (42). Finally, we realize that the random variable $S_{\boldsymbol{v}}(1, |\mathcal{J}|C)$ is distributed as the total load of the most loaded $|\mathcal{J}|C$ bins when a total of T balls are thrown uniformly at random to $2|\mathcal{J}|C$ bins. Hence, the inequality (d) follows from an application of Lemma 1. Finally, combining Eqns. (40) and (43), we have

$$R_T^{\pi} \geq G_T^{\pi^*} - G_T^{\pi} \geq d\sqrt{\frac{|\mathcal{J}|CT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}). \blacksquare$$

REFERENCES

[1] [n.d.]. MovieLens 25M Dataset. https://grouplens.org/datasets/movielens/. Accessed: 01-19-2019.

[2] Jacob Abernethy, Chansoo Lee, Abhinav Sinha, and Ambuj Tewari. 2014. Online linear optimization via smoothing. In *Conference on Learning Theory*. 807–823.

[3] Jacob Abernethy and Alexander Rakhlin. 2008. Optimal strategies and minimax lower bounds for online convex games. In *In Proceedings of the Nineteenth Annual Conference on Computational Learning Theory*.

[4] Charu Aggarwal, Joel L Wolf, and Philip S. Yu. 1999. Caching on the world wide web. *IEEE Transactions on Knowledge and data Engineering* 11, 1 (1999), 94–107.

[5] Shipra Agrawal and Navin Goyal. 2013. Further optimal regret bounds for thompson sampling. In *Artificial intelligence and statistics*. 99–107.

[6] Susanne Albers. 1996. *Competitive online algorithms.* Citeseer.

[7] Noga Alon and Joel H Spencer. 2004. *The probabilistic method.* John Wiley & Sons.

[8] EC Amazon. 2015. Amazon web services. *Available in: http://aws. amazon. com/es/ec2/(November 2012)* (2015).

[9] Daniel Berend and Aryeh Kontorovich. 2013. A sharp estimate of the binomial mean absolute deviation with applications. *Statistics & Probability Letters* 83, 4 (2013), 1254–1259.

[10] Daniel S Berger, Nathan Beckmann, and Mor Harchol-Balter. 2018. Practical bounds on optimal caching with variable object sizes. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 2 (2018), 1–38.

[11] Dimitri P Bertsekas and Athena Scientific. 2015. *Convex optimization algorithms*. Athena Scientific Belmont.

[12] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2018. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn. *ACM SIGCOMM Computer Communication Review* 48, 1 (2018), 28–34.

[13] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker, et al. 1999. Web caching and Zipf-like distributions: Evidence and implications. In *Ieee Infocom*, Vol. 1. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 126–134.

[14] Nicolo Cesa-Bianchi and Gábor Lugosi. 2006. *Prediction, learning, and games*. Cambridge university press.

[15] Jacob Chakareski. 2017. VR/AR immersive communication: Caching, edge computing, and transmission trade-offs. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 36–41.

[16] David Chappell et al. 2010. Introducing the windows azure platform. *David Chappell & Associates White Paper* (2010).

[17] Ludmila Cherkasova and Gianfranco Ciardo. 2001. Role of aging, frequency, and size in web cache replacement policies. In *International Conference on High-Performance Computing and Networking*. Springer, 114–123.

[18] Alon Cohen and Tamir Hazan. 2015. Following the perturbed leader for online structured learning. In *International Conference on Machine Learning*. 1034–1042.

[19] Richard Combes, Stefan Magureanu, Alexandre Proutiere, and Cyrille Laroche. 2015. Learning to rank: Regret lower bounds and efficient algorithms. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 231–244.

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

[21] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. 2010. Network coding for distributed storage systems. *IEEE transactions on information theory* 56, 9 (2010), 4539–4551.

[22] Gil Einziger, Roy Friedman, and Ben Manes. 2017. Tinylfu: A highly efficient cache admission policy. *ACM Transactions on Storage (ToS)* 13, 4 (2017), 1–31.

[23] D. H. Finstad, H. K. Stensland, H. Espeland, and P. Halvorsen. 2011. Improved Multi-Rate Video Encoding. In *2011 IEEE International Symposium on Multimedia*. 293–300. https://doi.org/10.1109/ISM.2011.53

[24] 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.

[25] Gaston H Gonnet. 1981. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM (JACM)* 28, 2 (1981), 289–304.

[26] Steffen Grünewälder, Jean-Yves Audibert, Manfred Opper, and John Shawe-Taylor. 2010. Regret bounds for Gaussian process bandit problems. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 273–280.

[27] Chuang Gu, Chun-Wei Chan, William Chen, Stacey Spears, Thomas W Holcomb, Chih-Lung Lin, and Sanjeev Mehrotra. 2013. Multiple bit rate video encoding using variable bit rate and dynamic resolution for adaptive video streaming. US Patent 8,396,114.

[28] Ori Gurel-Gurevich. [n.d.]. Expectation of square root of binomial r.v. MathOverflow. arXiv:https://mathoverflow.net/q/121424 https://mathoverflow.net/q/121424 URL:https://mathoverflow.net/q/121424 (version: 2013-02-10).

[29] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[30] Elad Hazan, Amit Agarwal, and Satyen Kale. 2007. Logarithmic regret algorithms for online convex optimization. *Machine Learning* 69, 2-3 (2007), 169–192.

[31] Elad Hazan and Sanjeev Arora. 2006. *Efficient algorithms for online convex optimization and their applications*. Princeton University Princeton.

[32] Elad Hazan and Satyen Kale. 2014. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *The Journal of Machine Learning Research* 15, 1 (2014), 2489–2512.

[33] Thomas W Holcomb, Shankar Regunathan, Chih-lung Bruce Lin, and Sridhar Srinivasan. 2008. Multi-resolution video coding and decoding. US Patent 7,379,496.

[34] Predrag R Jelenković and Xiaozhu Kang. 2008. Characterizing the miss sequence of the LRU cache. *ACM SIGMETRICS Performance Evaluation Review* 36, 2 (2008), 119–121.

[35] M. Ji, G. Caire, and A. F. Molisch. 2016. Fundamental Limits of Caching in Wireless D2D Networks. *IEEE Transactions on Information Theory* 62, 2 (Feb 2016), 849–869. https://doi.org/10.1109/TIT.2015.2504556

[36] M. Ji, M. F. Wong, A. M. Tulino, J. Llorca, G. Caire, M. Effros, and M. Langberg. 2015. On the fundamental limits of caching in combination networks. In *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. 695–699. https://doi.org/10.1109/SPAWC.2015.7227127

[37] Valentin Fedorovich Kolchin, Boris Aleksandrovich Sevastyanov, and Vladimir Pavlovich Chistyakov. 1978. Random allocations. (1978).

[38] Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvari. 2015. Tight regret bounds for stochastic combinatorial semi-bandits. In *Artificial Intelligence and Statistics*. 535–543.

[39] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies.. In *SIGMETRICS*, Vol. 99. Citeseer, 1–4.

[40] Boxi Liu, Konstantinos Poularakis, Leandros Tassiulas, and Tao Jiang. 2019. Joint Caching and Routing in Congestible Networks of Arbitrary Topology. *IEEE Internet of Things Journal* (2019).

[41] M Luby, A Shokrollahi, M Watson, T Stockhammer, and L Minder. 2011. RaptorQ forward error correction scheme for object delivery–rfc 6330. *IETF Request For Comments* (2011).

[42] Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive caching with machine learned advice. *arXiv preprint arXiv:1802.05399* (2018).

[43] Mohammad Ali Maddah-Ali and Urs Niesen. 2014. Fundamental limits of caching. *IEEE Transactions on Information Theory* 60, 5 (2014), 2856–2867.

[44] M. A. Maddah-Ali and U. Niesen. 2016. Coding for caching: fundamental limits and practical challenges. *IEEE Communications Magazine* 54, 8 (August 2016), 23–29. https://doi.org/10.1109/MCOM.2016.7537173

[45] Bruce M Maggs and Ramesh K Sitaraman. 2015. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review* 45, 3 (2015), 52–66.

[46] Stefan Magureanu, Richard Combes, and Alexandre Proutière. 2014. Lipschitz Bandits: Regret Lower Bound and Optimal Algorithms. In *Proceedings of The 27th Conference on Learning Theory, COLT 2014, Barcelona, Spain, June 13-15, 2014*. 975–999. http://proceedings.mlr.press/v35/magureanu14.html

[47] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

[48] Michael David Mitzenmacher. 1996. *The Power of Two Choices in Randomized Load Balancing*. Ph.D. Dissertation. UNIVERSITY of CALIFORNIA at BERKELEY.

[49] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19.

[50] Georgios Paschos, George Iosifidis, and Giuseppe Caire. 2019. Cache Optimization Models and Algorithms. *arXiv preprint arXiv:1912.12339* (2019).

[51] Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. 2019. Learning to Cache With No Regrets. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 235–243.

[52] Ramtin Pedarsani, Mohammad Ali Maddah-Ali, and Urs Niesen. 2016. Online coded caching. *IEEE/ACM Transactions on Networking (TON)* 24, 2 (2016), 836–845.

[53] Martin Raab and Angelika Steger. 1998. "Balls into bins"— A simple and tight analysis. In *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 159–170.

[54] Amr Rizk, Michael Zink, and Ramesh Sitaraman. 2017. Model-based design and analysis of cache hierarchies. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 1–9.

[55] Herbert Robbins. 1955. A remark on Stirling's formula. *The American mathematical monthly* 62, 1 (1955), 26–29.

[56] Elisha J Rosensweig, Daniel S Menasche, and Jim Kurose. 2013. On the steady-state of cache networks. In *2013 Proceedings IEEE INFOCOM*. IEEE, 863–871.

[57] Walter Rudin et al. 1964. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York.

[58] D. N. Serpanos, G. Karakostas, and W. H. Wolf. 2000. Effective caching of Web objects using Zipf's law. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532)*, Vol. 2. 727–730 vol.2. https://doi.org/10.1109/ICME.2000.871464

[59] Shai Shalev-Shwartz et al. 2012. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning* 4, 2 (2012), 107–194.

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

[61] Amin Shokrollahi. 2006. Raptor codes. *IEEE/ACM Transactions on Networking (TON)* 14, SI (2006), 2551–2567.

[62] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. 2010. The hadoop distributed file system.. In *MSST*, Vol. 10. 1–10.

[63] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. 2006. *Operating system principles*. John Wiley & Sons.

[64] Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, and Saverio Niccolini. 2015. Unravelling the impact of temporal and geographical locality in content caching systems. *IEEE Transactions on Multimedia* 17, 10 (2015), 1839–1854.

[65] Benjamin Van Roy. 2007. A short proof of optimality for the MIN cache replacement algorithm. *Information processing letters* 102, 2-3 (2007), 72–73.

[66] Sarut Vanichpun and Armand M Makowski. 2004. The output of a cache under the independent reference model: where did the locality of reference go?. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*. 295–306.
[67] Jinesh Varia, Sajee Mathew, et al. 2014. Overview of amazon web services. *Amazon Web Services* (2014), 1–22.
[68] Chuan Wu and Baochun Li. 2007. rStream: resilient and optimal peer-to-peer streaming with rateless codes. *IEEE Transactions on Parallel and Distributed Systems* 19, 1 (2007), 77–92.

8 APPENDIX

8.1 Calculations for the Mean Deviation bound of Theorem 1, Eqn. (12)

We recall Robbin's form of Stirling's formula [55], which will be useful in obtaining a non-asymptotic bound for regret.

$$\sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n}e^{\frac{1}{12n+1}} \leq n! \leq \sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n}e^{\frac{1}{12n}}. \tag{44}$$

Consider the following two possible cases.

Case I: T is even

In this case, we lower bound the Mean absolute deviation in Eqn. (11) as follows:

$$\begin{aligned}
\mathbb{E}|Z-\frac{T}{2}| &= \frac{2}{2^{T+1}}(\lfloor\frac{T}{2}\rfloor+1)\binom{T}{\lfloor\frac{T}{2}\rfloor+1}\\
&= \frac{1}{2^T}\frac{T!(\frac{T}{2})}{(\frac{T}{2})!(\frac{T}{2})!}\\
&\overset{(a)}{\geq} \frac{1}{2^T}\frac{1}{\sqrt{2\pi}}(\frac{T}{2})\frac{T^{T+\frac{1}{2}}}{(\frac{T}{2})^{T+1}}e^{\frac{1}{12T+1}-\frac{1}{3T}}\\
&= \frac{\sqrt{T}}{\sqrt{2\pi}}e^{-\frac{9T+1}{3T(12T+1)}}\\
&\overset{(b)}{\geq} \sqrt{\frac{T}{2\pi}}-\frac{9T+1}{3\sqrt{2\pi T}(12T+1)}
\end{aligned}$$

where the inequality (a) follows from the bound (44) and the inequality (b) follows from the fact that $e^{-x} \geq (1-x)$ for all $x \in \mathbb{R}$. Using the fact that $\frac{9T+1}{12T+1} < 1$ for all $T > 0$, the mean deviation can be lower bounded as:

$$\mathbb{E}|Z-\frac{T}{2}| \geq \sqrt{\frac{T}{2\pi}}-\frac{1}{3\sqrt{2\pi T}} \tag{45}$$

Case II: T is odd

In this case, the binomial coefficient $\binom{T}{\lfloor\frac{T}{2}\rfloor+1}$ may be lower bounded as follows:

$$\begin{aligned}
&\binom{T}{\lfloor\frac{T}{2}\rfloor+1}\\
=\ &\frac{T!}{(\frac{T+1}{2})!(\frac{T-1}{2})!}\\
\geq\ &\frac{1}{\sqrt{2\pi}}\frac{T^{T+\frac{1}{2}}}{(\frac{T^2-1}{4})^{\frac{T}{2}}}\frac{2}{T+1}e^{-\frac{1}{T}(\frac{1}{6+\frac{6}{T}}+\frac{1}{6-\frac{6}{T}}-\frac{1}{12+\frac{1}{T}})}.
\end{aligned}$$

Note that

$$\frac{T^{T+\frac{1}{2}}}{(\frac{T^2-1}{4})^{\frac{T}{2}}} \geq \frac{T^{T+\frac{1}{2}}}{(\frac{T^2}{4})^{\frac{T}{2}}} = 2^T\sqrt{T}.$$

Also, for $T \geq 3$

$$\frac{1}{6+\frac{6}{T}} + \frac{1}{6-\frac{6}{T}} - \frac{1}{12+\frac{1}{T}} \leq \frac{1}{2}.$$

This gives us the following lower bound

$$\binom{T}{\lfloor \frac{T}{2} \rfloor + 1} \geq \frac{1}{\sqrt{2\pi}} \frac{2^{T+1}\sqrt{T}}{T+1}(1-\frac{1}{2T}).$$

For $T = 1$, the inequality holds good by direct inspection. Hence, the mean deviation from Eqn. (11) is lower bounded as

$$\begin{aligned} \mathbb{E}|Z - \frac{T}{2}| &\geq \frac{1}{2^T}(\frac{T+1}{2})\frac{1}{\sqrt{2\pi}}\frac{2^{T+1}\sqrt{T}}{T+1}(1-\frac{1}{2T}) \\ &\geq \sqrt{\frac{T}{2\pi}} - \frac{1}{2\sqrt{2\pi T}}. \end{aligned} \tag{46}$$

Finally, combining equations (45) and (46) together, we have the following lower bound for all $T \geq 1$

$$\mathbb{E}|Z - \frac{T}{2}| \geq \sqrt{\frac{T}{2\pi}} - \frac{1}{2\sqrt{2\pi T}},$$

which verifies Eqn. (12).

Received January 2020; revised February 2020; accepted March 2020