Optimal Device-Aware Caching

Samta Shukla and Alhussein A. Abouzeid
Department of Electronic, Computers and Systems Engineering
Rensselaer Polytechnic Institute, USA
{shukls, abouzeid}@rpi.edu

***Abstract*—Caches in Content-Centric Networks (CCN) are increasingly adopting flash memory based storage. The current flash cache technology stores all files with the largest possible "expiry date," i.e. the files are written in the memory so that they are retained for as long as possible. This, however, does not leverage the CCN data characteristics where content is typically short-lived and has a distinct popularity profile. Writing files in a cache using the longest retention time damages the memory device thus reducing its lifetime. However, writing using a small retention time can increase the content retrieval delay, since, at the time a file is requested, the file may already have been expired from the memory. This motivates us to consider a joint optimization wherein we obtain optimal policies for jointly minimizing the content retrieval delay (which is a network-centric objective) and the flash damage (which is a device-centric objective). Caching decisions now not only involve *what* to cache but also for *how long* to cache each file. We design provably optimal policies and numerically compare them against prior policies.**

***Index Terms*—Content-centric network, caching, computing, flash memory, Least Recently Used, First-In-First-out, Farthest-in-Future, Retention time, Markov Decision Process.**

I. Introduction

Flash memory based cache is a principal component of the emerging Content-Centric or Information-Centric Networks (CCN/ ICN) with ubiquitous caching, mobile/cloud computing and device-to-device networking [1]–[4]. One of the main obstacles in flash memory adoption is its high rate of wear out (referred as flash damage) which is directly proportional to the programmed data retention time [4]–[8].

The relationship between data retention and wear out can be briefly described as follows. A flash memory consists of flash cells. Data is stored in a flash memory by *programming* (P) the threshold voltage of each memory cell into two or more non-overlapping voltage windows. A memory cell is *erased* (E) of all the data before it is programmed; erasing data involves removing the charges in the floating gate and setting the threshold voltage to the lowest voltage window. The reliability of a flash (or flash lifetime) is specified in terms of the number of program/erase (P/E) cycles it can endure (e.g., 10^4 to 10^5 P/E cycles) [9]. Depending on the underlying technology, all flash cells are programmed to retain data in cache for a specified duration (from 1 to 10 years), known as the *retention time*. The specified memory retention is achieved by programming data with a high threshold voltage. However, programming at high voltages causes a high wear out to the flash cell thus reducing the memory lifetime [5]–[7].

The current practice in flash technology is not optimal. It writes all files with a fixed maximum/default threshold voltage to get the maximum possible data retention which in turn causes maximum cache damage at each write. Note that the high damage caused is permanent even if the file is evicted from the cache before its retention expires. Clearly, writing every content with maximum retention is wasteful since in CCN some content could be less popular. We aim to obtain optimal retention times by leveraging the content popularity profile which can be locally estimated from the user requests in CCN architectures [10].

We take first steps in reformulating the traditional data caching problem by proposing a *cross-layer optimization* that combines the *cache-level* objective of minimizing the content-retrieval delay and the *device-level* objective of minimizing the device damage[1]. We note that the cache-level and device-level objectives are conflicting: a smaller delay is achieved by writing files with longer retention but that incurs a high damage; a smaller (or zero) damage is achieved by not writing files at all but that causes large delays. Despite the inherent trade-off, earlier work in these areas have progressed largely independently. For example, in the device literature, a recent line of work considers optimizing damage/retention times by dynamically trimming the retention duration of a content based on their refresh cycle durations [12], [13]. Another closely related work considers trading retention time for system performance (such as memory speed and lifetime) [8]. By contrast, the caching literature consists of innumerable attempts to construct policies with high cache hit probability for achieving lower network delays (see [14] and the citations within) while overlooking the device-related aspects.

The key challenge addressed in this paper is to find caching policies for a finite capacity cache, that, in addition to the functions provided by a traditional caching policy, determine optimal file retention times to incur minimum flash damage when subject to a constraint on acceptable network delay. A file written in the cache at time t for a retention duration D is no longer readable from the cache after time $t + D$, thus leading to a cache miss (unless the file is re-written between t and $t + D$ to extend its original retention but that incurs additional damage).

Our first contribution (Section III) is to solve the prob-

[1]A preliminary version of this work appeared in [11]. This paper extends the work in [11] with: (1) A complete description of the online policy in Section V-C. (2) Section VI, wherein we compare the online and offline policies by showing the delay-damage tradeoffs and the competitive ratios. (3) The full proofs for all theorems and lemmas in the Appendices.

lem of *offline caching*, i.e. caching when the content request string is given. We design an optimal offline policy, Damage-Aware REtention (DARE), that returns the optimal retention times for every file without exceeding the optimal cache misses given by Belady's Farthest in Future (FiF) algorithm[2]. We prove analytically and show by simulations that our policy, DARE, by taking retentions into account, achieves a significantly lower cache damage than FiF without increasing the optimal delay (or cache misses).

Our second contribution is to solve the *online caching* problem, i.e. caching when the request string is not given ahead of time. Our optimal online policy, DARE-Δ, approaches the online caching problem in two stages. It first assumes a large cache (a cache with no capacity constraint) and obtains the optimal file retentions by solving an optimization problem (Section IV). A large cache assumption implies that there are no evictions and the cache misses are only because of the files expiring. The policy then extends the results from a large cache case to a cache of finite capacity in Section V. In this case a cache miss can result in a file eviction if the cache is full (Section V). Subsequently, DARE-Δ exploits the optimal retentions obtained for large caches and models the problem of which file to evict at every cache miss as a Markov Decision Process (MDP). In contrast with the usual MDP-based approaches which suffer from the curse of dimensionality, we show that our MDP can be characterized to give a very simple, easy to implement rule for evicting files. Our simulations (Sections IV, V, VI) reinforce the theoretical findings for a range of parameters, caching policies and damage functions for the online case. We note that our work is a significant generalization of [15] where authors found an eviction sequence using MDP but do not consider flash damage constraints in their formulation.

The paper is structured as follows. The modeling prelimiaries are discussed in Section II. We analyze the offline caching problem in Section III. The online caching problem is studied in Section IV (for a large cache) and in Section V (for a finite cache). We simulate our policies in Section VI, provide a survey of related work in Section VII and conclude in Section VIII.

II. Preliminaries

In this section, we explain the model assumptions that are common to all the analytical results in the paper. We also discuss our work in the light of closely related literature.

A. Model Assumptions

1) Cache-level assumptions: Let M denote the set of all files where each file $m \in M$ is of unit size. Files are requested at a cache with finite capacity of size B files. A requested file that is not in the cache results in a cache miss. Upon a cache miss, the requested file is fetched by incurring a retrieval cost (see Section II-B) and is subsequently written in the cache by incurring a retention cost (see Section II-B). Files are served instantaneously in the case of a cache hit. The file arrivals conform to the Independent Reference Model (IRM)[3] [15], [16], where each file is requested with a static probability independent of other requests. We describe our traffic model in more detail in Section IV-A. For tractability, we obtain results for Poisson file arrivals modulated with a suitable popularity distribution – such as ZipF popularity law [15], [16] – and exponentially distributed retention times in our analysis in Sections IV and V[4]. Our work focuses on applications related to ICN/CCN, where the content request probabilities can be estimated/learned due to its receiver-driven architecture [10], [17], [18]. Further, we emulate a ICN/CCN environment by considering a non-uniform, non-constant (random) file retrieval costs which are independent of both arrivals and retention processes.

2) Device-level assumptions: The process of writing files in the flash cache is explained as follows. A memory is divided into various sectors from which a sector is chosen uniformly at random. It is a reasonable assumption since the disk controller in a flash exercises "wear leveling" by spreading writes evenly across the flash chip for causing less damage to the flash lifetime [19]. We neglect the damage caused due to subsequent reads of an already written file and only consider the damage due to writing a file since reading the disk does not require writing or erasing [19]. Subsequently, we model the P/E cycle counts and erasure costs (associated with programming and erasing a file) in a flash memory with the help of a damage function which takes retention times as arguments (see Section III-A). This is justified because the P/E cycle duration is closely related to the retention time. A higher retention is obtained by programming (P) the flash with a very high positive voltage thus requiring a very high negative voltage to erase (E) the data. Finally, while there are only empirical relationships known about flash damage as a function of the depleting cell life [7], we propose and analyze a general mathematical model that captures a wider range of dependence between flash damage and depleting cell life due to file retention (see Section III-A).

B. Cost of fetching and writing a file

Upon a cache miss, the requested file is fetched from the server by incurring a non-uniform random file retrieval cost which is assumed to be exponentially distributed with parameter $\delta(m) \in \mathbb{Z}^+$ for file m. The retrieval cost can be thought of as the cost of obtaining the file from the server based on the time of the day, current server workload, or available channel bandwidth, etc.

After the file is retrieved, it is subsequently written to the cache by incurring a deterministic retention (writing) cost $f(m) \in \mathbb{Z}^+$. Retention cost can be perceived as the storage cost and/or the cost of writing the file to the cache memory

[2]With traditional caching (caching without optimizing memory retentions) Belady's Farthest in Future (FiF) algorithm obtains the optimal cache misses. As FiF is damage/retention agnostic, we account for the flash damage in FiF by assigning the retention time of a file as the duration for which it stays in the cache before it is evicted due to a cache miss.

[3]Although IRM does not take temporal locality into account, it is a widely accepted, standard traffic model in caching literature [15], [16].

[4]Our Markovian formulations in Section V require memoryless arrivals and retention times.

as every write incurs a physical damage that reduces cache lifetime. While there are only empirical relationships known about flash memory damage as a function of memory retention times, we choose the damage function to be an *increasing convex polynomial* in our analysis motivated by the following properties: (1) Memory damage, although a function of several factors, is known to increase with retention time; this is because writing a file at a higher threshold voltage helps in a longer file retention thereby incurring a higher damage [5]–[7]. This justifies the choice of an increasing function. (2) As the flash memory ages, the damage caused increases for the same amount of workload [5]–[7]. This justifies the choice of a convex function. (3) Damage function, $f(\cdot)$, is a complicated, non-linear function with $f(0) = 0$. This justifies the choice of a polynomial damage function as a wide variety of functions can be derived by varying the coefficients of the polynomial. Formally, retention cost is defined as:

Definition 1 (One-shot retention cost)**.** *The one-shot retention cost is the damage caused to the cache due to writing a file for a retention time $Z \in \mathbb{R}^+$, given by $f(Z) \in \mathbb{R}^+$ where $f(\cdot)$ is a convex increasing polynomial of degree n given by $f(Z) = a_n Z^n + a_{n-1} Z^{n-1} + \cdots + a_1 Z + a_0$ with coefficients $a_i \geq 0$, for all $i \geq 1$ and $a_0 = 0$.*

C. DARE caches are different from TTL caches

In this paper, our goal is to design a Damage Aware REtention caching (DARE) policy to minimize flash damage with acceptable delay guarantees. Having a file written for a duration equal to its retention time, as in DARE caches, may appear similar to the TTL caches considered in [14], [20]–[22], where files stay in cache for their TTL (time-to-live) duration. However, our work, even at the conceptual level, is different from TTL caches[5]. DARE caches take retention time distributions as input and output the optimal retention values satisfying the goal. In contrast, TTL caches are devised to simplify the analysis of traditional caching policies (such as LRU, FIFO, RND) which do not optimize any specific quantity. In particular, TTL caches take a damage oblivious existing policy as input to obtain (an asymptotic approximation of) the corresponding TTL distribution as an output (see [14] for a detailed analysis of TTL caches).

D. Summary of prior caching policies

We compare our optimal policies against the performance of the following well-known policies (e.g. see [14]). In these policies, a requested file not already in the cache is inserted. The policies differ in their eviction policies when a cache is full. In Least Recently Used (LRU) policy, the least recently used file is evicted. In First In First Out (FIFO) policy, the file which was written first is evicted. In RaNDom (RND) policy a file is evicted from the cache uniformly at random. Farthest in Future (FiF) policy, also called Belady's Algorithm, evicts the file whose next request is the farthest in time. FiF minimizes the number of cache misses [23] but assumes knowledge of the full time sequence of requests. LRU is widely used since it performs well even for arbitrary request strings. RND and FIFO are very simple to implement in hardware and are seen as a viable alternative of LRU in CCN high-speed routers [16].

A detailed literature survey is presented in Section VII. We now move on to Section III where we design DARE caches when the file request string is known.

[5]Coincidentally, the hit and miss probabilities obtained for DARE with the large cache assumption are the same as the hit and miss probabilities of a TTL cache under the RND caching policy (see Section II-D).

III. Flash-aware Optimal Offline Caching

In this section, we consider the case of offline caching. In this case, the file request string is given ahead of time as a sequence of positive integer-valued indices chosen from a set of M files. Recall the FiF algorithm by Belady [23] which is known to minimize the number of request misses for a cache. Our contribution is in showing that FiF is not optimal with respect to damage. Further, we advance the state-of-the-art by constructing the DARE caching policy which minimizes flash damage by taking no more delay (cache-misses) than Belady's FiF (i.e. the known optimal delay benchmark).

A. System model

In this section, we assume that time of horizon length T is slotted in equal length intervals, and files are requested at the beginning of each slot. A requested file that is not in the cache results in a cache miss. Upon a cache miss, the requested file is fetched and subsequently written in the cache for *at least one slot*. Writing the requested file on *every* miss is called *cache miss allocation* [24] in device literature. We lift this assumption in Section VI where the policy is allowed to skip writing the requested file.

In this section, for exposition, we assume that the file retrieval costs (from the server upon cache miss) are deterministically assigned to $\delta(m) = 1$ unit for all $m \in M$. Thus minimizing delay corresponds to minimizing the number of cache misses. We assume that the one-shot retention cost incurred for writing file $m \in M$ for retention time $R \geq 1$ slots, $R \in \mathbb{Z}^+$ is $f(R) \in \mathbb{Z}^+$ as explained in Section II-B.

Let F, E denote the optimal number of cache misses, the corresponding eviction sequence according to FiF policy. Our goal is to find a policy that determines the *optimal retention times* for each file write without exceeding F.

B. The optimal offline policy, DARE

DARE aims to reduce the cache retention times without changing the cache miss sequence from FiF. It considers every eviction in the optimal eviction sequence given by FiF policy and works backward to find the optimal retention for *each* file write. When a file l is evicted in FiF at time t, DARE finds two different time indices by traversing back from t. First it finds the *latest* (time) slot when l was written in the cache before getting evicted at t; we call it time k. Second, it searches for the time when l was last requested before eviction at t, we call it time j. Our policy stores file l

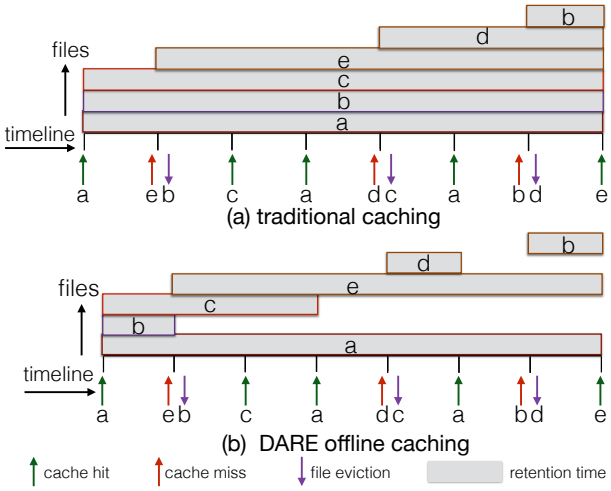

Fig. 1: (a) With traditional caching a file (upon a cache miss) is stored for the maximum retention time. (b) With DARE caching, a file is stored only for the duration for which it is needed.

in the cache at time k for $j-k+1$ slots. Also, the files which are present in the cache (i.e. not evicted) till the last eviction are stored for $T-k+1$ slots, where T represents the total number of slots. Thus, for each evicted file, DARE saves on the number of slots by storing a file for a retention time equal to the difference between the time when it was last requested from the time when it was written latest. Figure 1 illustrates the algorithm.

C. DARE is optimal

We observe that DARE incurs optimal number of cache misses (by definition). Thus, for optimality we only need to prove the non-existence of a policy which incurs less cost than DARE in choosing retention times for files without exceeding the optimal number of cache misses. Theorem 1 proves that DARE is optimal; see proof in Appendix B.

Theorem 1. *DARE is optimal with respect to retention cost over all possible optimal eviction sequences that minimize the number of cache misses.*

D. Numerical study: Cache miss versus damage

The current practice in flash memory technology is to write all files with a very high retention (typically 1-10 years), however, for making a fair comparison among policies we assume that the policies LRU, FIFO, RND and FiF write a file exactly for the time till it is not evicted. Subsequently, we write a file for the calculated optimal retention duration for DARE. Our goal is to demonstrate the optimality of DARE against other caching policies.

We consider a horizon of length $T = 10000$ slots where in each slot file m is requested as per the IRM with probability $\frac{1/m^{\alpha}}{\sum_{j\in M} 1/j^{\alpha}}, m \in M$, where α is the ZipF popularity coefficient. Usually, for web caches and data servers, the ZipF coefficient is found to vary from 0.65 (least skewed) to 1 (most skewed) [14]. Hence, we consider two extremes and set $\alpha = \{0.65, 0.95\}$. Files are requested from a catalogue containing $|M| = 1000$ files and the cache size varies from 50 to 600 files. The damage function for writing files is assumed to be quadratic in retention time. We compute the aggregate damage and cache misses for T slots by evaluating: damage $= \sum_{t=1}^{T} 1_{(m,t)} R_m^2$, and cache miss fraction $= 1/T \sum_{t=1}^{T} 1_{(m,t)}$, where $1_{(m,t)} = 1$ when there is a cache miss for file m at time t and 0 otherwise. We plot the results in Figure 2.

Recall that the cache misses (fraction) for both FiF and DARE are the same (by definition). Thus, it suffices to represent the cache miss variation by plotting a single curve, which is shown by the dotted curve in Figure 2. We observe that as the cache size increases, the fraction of cache misses decreases, as expected, and soon converges to a specific value in steady state. The higher the ZipF-α, the sooner this fraction converges. We also note that a higher α results in a lower value of cache miss (fraction) in steady state. This can be briefly explained as follows. When α increases, the skewness in the file request arrivals increases, i.e. with $\alpha = 0.95$ the popular files are more popular and the unpopular files are less popular, compared to $\alpha = 0.65$. Thus, a highly skewed traffic, by sending fewer requests for unpopular files, begets a lower cache miss count.

The solid lines in Figure 2 show that as the cache size increases, the damage values from both FiF and DARE increase, and gradually both of them converge to a specific value. This implies that the damage savings, calculated as $\frac{\text{damage from FiF}}{\text{damage from DARE}}$ approaches one with increasing cache size. We observe that for smaller caches, a damage savings of upto 2-3 folds can be achieved. We also compare DARE against LRU, FIFO and RND; simulating these policies result in significantly worse damage to the extent that it can not be shown on the figures with the same scale. Similar trends for the ZipF variation follow for the damage curve as observed for the cache miss (fractions) curve.

In this section, we showed that the well-known delay optimal caching policy (FiF) is not damage optimal. Further, we devised a caching policy that achieves optimal damage without exceeding the optimal number of misses given by FiF. The case of offline caching lends insights for online caching where the arrival requests are not known apriori.

IV. Flash-aware Optimal Online Caching for Large Caches

We now consider the case of online caching where the files are requested according to a distribution, however, the exact request string is not known to the policy apriori. We first state the system model for the online caching which applies to Sections IV and V. Our goal is to design a policy that jointly finds the optimal retention times for all files and the optimal eviction sequence in the event of a cache miss. We achieve this goal by designing a policy DARE-Δ which optimizes in two steps. First, in this section (Section IV), it approximates the problem by considering a *large cache* (a cache with no capacity constraint and hence no evictions) and finds the optimal retention times. Subsequently, in Section V, it obtains the optimal file eviction sequence

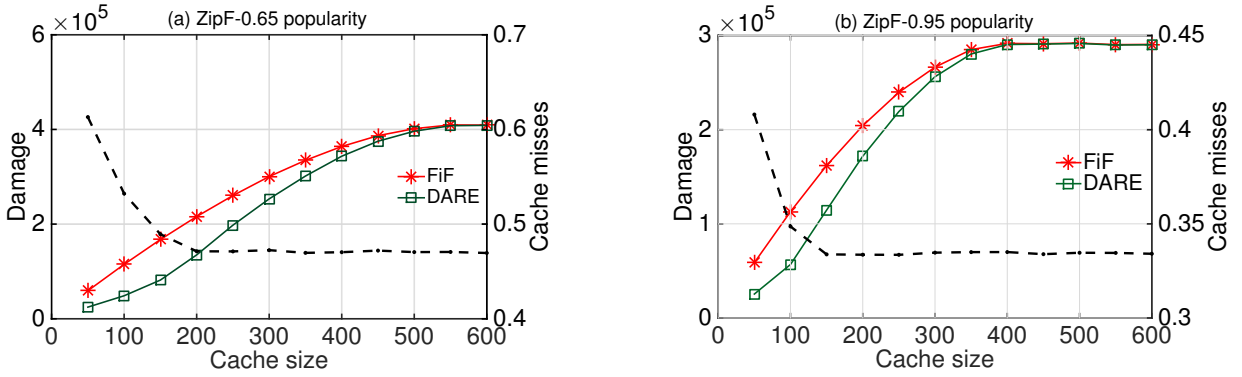

Fig. 2: Cache damage vs. cache miss under IRM for B varying from 50 to 600 files with $|M| = 1000$ files. Damage curve is shown in bold lines as per the legends; the cache misses (fraction) curve is shown as a black dotted line.

given the optimal retention durations. Note that the problem of jointly optimizing over all possible retention times and eviction decisions remains an open problem.

In this section, we formulate an optimization problem called DARE-Δ Retention Formulation (see Section IV-B) to minimize cache damage subject to a constraint on the network delay to find optimal retention times. Our formulation provides an approximate solution due to the large cache assumption, however, our numerical studies in Section VI show that the objective function quickly converges to a steady state with increasing cache size. Having a large cache implies that there are no evictions and there is a cache miss on the requested file only if it has expired from the cache; this assumption[6] is known to decouple files thus facilitating a tractable mathematical analysis [14], [20]. We conclude this section by showing damage-delay trade-offs for different damage functions.

A. System model

1) Traffic model: The file request string is assumed i.i.d. File requests arrive according to the Independent Reference Model (IRM) [15], [16] which assumes the following. (1) All requests are for a fixed collection of M files. (2) The probability of requesting file m is p_m which is static and independent of past or future requests.

We assume that the interarrival times of file $m \in M$, $X(m)$, is exponentially distributed with rate parameter λ_m, and the arrival process across files conform to an independent and homogeneous Poisson process. Under IRM, the probability of requesting file m with interarrival times $X(m)$ and modulated with ZipF-α popularity law is given by $p_m = \frac{\lambda_m}{\sum_{j=1}^{M} \lambda_j}$ where $\lambda_m = 1/m^{\alpha}$, $\forall m \in M$.

2) Cost of retrieving and writing a file: Fetching file m upon a cache miss incurs a retrieval cost that is assumed to be an exponentially distributed random variable with mean $\delta(m) \in \mathbb{R}^+$ for file m and is assumed to be independent of arrivals/retentions. Thus given that an arrival for file m is a miss, by the virtue of independence, the expected (delay) cost for fetching file m by averaging over arrival process, retention times and retrieval costs would equal $\delta(m)$. We assume that the one-shot retention cost incurred for writing file $m \in M$ for retention time $R \geq 1$ slots, $R \in \mathbb{Z}^+$, is $f(R) \in \mathbb{Z}^+$ as in Section II-B. The retention time for file m is assumed to be distributed as an exponential random variable $\mathcal{R}(m)$ with parameter μ_m, $m \in M$. We assume retention time as a random variable to capture the property that writing a file in memory with a retention R leaves a non-zero probability of finding it in cache after time R.

B. Problem formulation for finding optimal retention times

Definition 2 (Optimal online policy)**.** *A policy is online optimal if it finds the values of the retention parameters for each file (i.e. $\{\mu_m\}$) that minimizes the expected cache damage due to successive file writes under the constraint that the expected delay does not exceed $\Delta > 0$.*

To find the optimal online policy (see Definition 2), we first obtain an expression for the miss probability with a single file in the library ($|M| = 1$) and consider the set of all requests to a cache in steady state. Let $\{\mathcal{R}_n\}$ denote[7] the i.i.d. exponential retention time sequences corresponding to arrivals $n = 1, 2, \ldots$ for the single file. Let I_n be the indicator variable defined as follows:

$$I_n = \begin{cases} 1 & \text{if } n^{th} \text{ file request results in a cache miss} \\ 0 & \text{otherwise} \end{cases}$$

Let $X_n(m)$ denote the i.i.d exponential interarrival time between the n^{th} and $n+1^{th}$ request of file m. Note that $I_n = 1$ corresponds to the event $X_n > \mathcal{R}_n$. Thus, $\lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} I_n = \mathbb{P}(X_n > \mathcal{R}_n) = p_{\text{miss}} = \frac{\mu}{\lambda+\mu}$. Similarly, the probability of a cache hit is $p_{\text{hit}} = 1 - p_{\text{miss}} = \frac{\lambda}{\lambda+\mu}$. For the n^{th} file request, we write the file with retention $\mathcal{R}_{n+1}$ if there is a miss (and we do not write otherwise). Thus, the *expected damage*, D, can be expressed as:

$$\begin{aligned} D &= \lim_{N\to\infty} \left[\mathbb{E}_{\mathcal{R}} \left[\frac{1}{N} \sum_{n=1}^{N} I_n \times f(\mathcal{R}_{n+1}) \right] \right] \\ &= \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} I_n \times \mathbb{E}_{\mathcal{R}} \left[f(\mathcal{R}_{n+1}) \right], \qquad (1) \\ &= p_{\text{miss}} \times \mathbb{E}_{\mathcal{R}} \left[f(\mathcal{R}_{n+1}) \right]] \qquad (2) \end{aligned}$$

[6] A large cache assumption was previously considered in [14], [20] in the context of TTL-caches.

[7] We denote the discrete retention time in the offline caching section as R and the continuous retention for the context of online caching as $\mathcal{R}$.

which holds since I_n is independent of the retention time $\mathcal{R}_{n+1}$[8]. Similarly the expected delay constraint can be expressed as $p_{\text{miss}}\delta \leq \Delta$ where δ denotes the mean parameter for file retrieval cost.

We now generalize the analysis for a single file to a set of $|M|$ files. With IRM, the probability of requesting file m is given by p_m, where $p_m = \lambda_m / \sum_{i \in M} \lambda_i, m \in M$. Also, the miss probability of file m upon request is given by, $p_{\text{miss}}(m) = \mathbb{P}(X(m) > \mathcal{R}(m)) = \mu_m/(\mu_m + \lambda_m)$, since the interarrival and retention times are exponentially distributed. Thus the optimization problem becomes:

$$\underset{\mu_m \in \boldsymbol{\mu}}{\text{minimize}} \sum_{m \in M} p_m p_{\text{miss}}(m) \mathbb{E}_{\mathcal{R}}\left[f(\mathcal{R}(m)\right] \tag{3a}$$

$$\text{subject to} \sum_{m \in M} p_m p_{\text{miss}}(m)\delta(m) \leq \Delta \tag{3b}$$

Define $q_m := \lambda_m/(\mu_m + \lambda_m), m \in M$, and substitute the value of the polynomial damage function, $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x$ (as in Definition (1)) in the objective of formulation (3). The objective becomes:

$$\frac{1}{\sum_{m \in M} \lambda_m} \sum_{m \in M} q_m \mu_m \mathbb{E}[a_n \mathcal{R}(m)^n + \cdots + a_1 \mathcal{R}(m)]. \tag{4}$$

Note that $\mathbb{E}[a_k \mathcal{R}(m)^k] = a_k k!/\mu_m^k$ for $\mathcal{R} \sim exp(\mu)$. Also,

$$\frac{1}{\mu_m^k} = \frac{1}{(\mu_m + \lambda_m - \lambda_m)^k} = \frac{1}{(\frac{\lambda_m}{q_m} - \lambda_m)^k} = \left(\frac{q_m/\lambda_m}{1 - q_m}\right)^k. \tag{5}$$

Therefore, substituting (4), (5) in the objective in (3a) gives:

$$\frac{1}{\sum_{m \in M} \lambda_m} \sum_{m \in M} q_m \sum_{k=1}^{n} a_k k! \left(\frac{q_m/\lambda_m}{1 - q_m}\right)^k. \tag{6}$$

Further, the constraint in (3b) can be simplified as:

$$\sum_{m \in M} \lambda_m \delta(m) \left(\frac{\mu_m + \lambda_m - \lambda_m}{\mu_m + \lambda_m}\right) = \sum_{m \in M} \lambda_m \delta(m)(1 - q_m).$$

Now we present the final formulation.

DARE – Δ Retention Formulation:

$$\underset{q_m \in \mathbf{q}}{\text{minimize}} \frac{1}{\sum_{m \in M} \lambda_m} \sum_{m \in M} \sum_{k=1}^{n} a_k k! \frac{q_m^{k+1}}{\lambda_m^k (1 - q_m)^k} \tag{7a}$$

$$\text{subject to:} \frac{1}{\sum_{m \in M} \lambda_m} \sum_{m \in M} \lambda_m \delta(m)(1 - q_m) \leq \Delta \tag{7b}$$

$$0 \leq q_m \leq 1, \forall\, m \in M \tag{7c}$$

The next Lemma proves that the above objective function is convex, the proof of which is in Appendix A.

Lemma 1. *The objective function in the damage formulation (7) is convex.*

The constraint in formulation (7) poses upper and lower bounds on the value of $q_m = \lambda_m/(\mu_m + \lambda_m)$. The boundary cases are: when $q_m = 0$ then $\mu_m = \infty$ which means that files are never written into the cache; alternatively, $q_m = 1$ implies $\mu_m = 0$ meaning that the file is retained forever. Once we obtain optimal q_m's, the optimal $\mu's$ can be obtained by letting $\mu_m = \lambda_m(1 - q_m)/q_m$. The objective function in the optimization problem in (7) is convex (see Lemma 1). We use a MATLAB convex program solver to solve (7) and report the results in Figure 3.

[8] I_n only depends on X_n and $\mathcal{R}_n$ by definition.

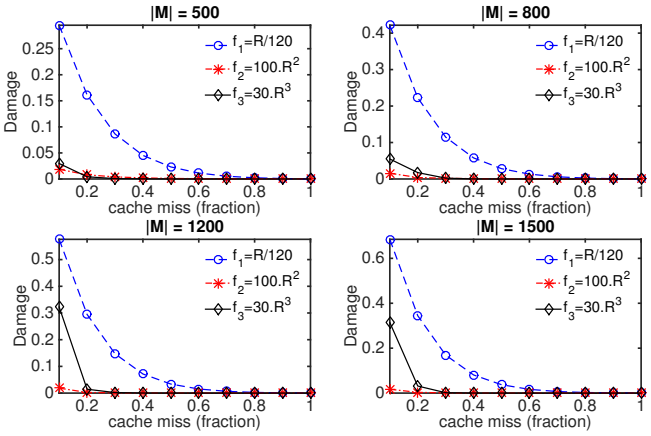

Fig. 3: Impact of increasing number of files and (allowed) fraction of cache misses on objective function for Poisson arrivals modulated with ZipF $\alpha = 0.85$.

C. Damage-delay trade-offs for various damage functions with DARE-Δ

We study the delay-damage trade-offs obtained for three polynomial[9] damage functions (linear, quadratic and cubic) on Poisson arrivals modulated with ZipF popularities ($\lambda_m = 1/m^\alpha, \alpha = 0.85$). For exposition, we assume a deterministic unit retrieval cost for fetching files ($\delta(m) = 1, m \in M$); thus the expected delay becomes equivalent to expected fraction of cache misses ($\Delta = \epsilon \in [0, 1]$). We study the damage function trade-off with increasing ϵ for an increasing number of files $|M|$, as shown in Figure 3.

We observe that damage decreases with increasing ϵ in each case. This is reasonable since a higher ϵ means a relaxed delay constraint which implies that now more files can be written with lower retention values thus incurring less damage. We also observe that the value of damage increases with increasing number of files for the same value of ϵ, which is expected as now more files are written in the cache (causing a higher damage) to achieve the required ϵ.

V. Flash-aware Optimal Online Caching for a Finite Capacity Cache

In this section, we use the same model as defined in Section IV-A with the only difference that now the cache is finite and can contain only B files. Upon a cache miss, a file is written in the cache for a duration given by the optimal retention time obtained in Section IV. A cache miss can result in a file eviction if the cache is full. We aim to obtain the optimal file to evict on every cache miss when the

[9] The problem of finding suitable coefficients for the polynomial damage function could be an independent research problem by itself (left as an open problem for device engineers [12]) and is thus not considered in this work. Our work is concerned with finding optimal caching policies *given* any polynomial damage function.

cache is full using only the knowledge of the past requests and cache contents. We formulate the problem of finding an optimal eviction sequence as a sequential decision problem using the theory of Markov Decision Processes (MDP). We then characterize the optimal solution which results in a very simple, easy to implement rule. We conclude the section by giving an outline of the DARE-Δ policy and comparing its performance with LRU, FIFO, RND policies.

Our work is a significant generalization of [15] where the authors have proposed a stationary, Markovian policy to optimally evict a file when files have non-uniform costs and the cache is finite. In contrast with [15] where the files are evicted *only* upon a cache miss when the cache is full, in our model files leave the cache not only because they are evicted but *also* because their retention time has *expired*. Although subtle, this difference is significant as the minimization is performed over different file sets in both the cases. Hence the optimal solution in [15] is not a solution to our problem and vice versa. Moreover, modeling retention time for every file makes the analysis significantly more involved.

A. Markov Decision Process

1) State Description: We construct an MDP on a continuous time, discrete state space and use uniformization [15] to obtain a discrete time Markov chain (DTMC) from the continuous Markov process. Let $t = 1, 2, \dots T$ denote the time indices corresponding to the state transitions due to file arrivals/departures. Let $\mathbf{S}(t)$ be a state in the Markov Chain denoted by a 3-tuple, $\mathbf{S}(t) = \{S(t), R(t), D(t)\}$, where $S(t)$ is the set of files in the cache at t, $R(t)$ denotes the file requested at time t and $D(t)$ is the first file departing at time t. We assume that a transition is either due to a file arrival or a file departure and *not* both. For a file arrival, $D(t) := 0$ and for a file departure, $R(t) := 0$. Thus, the states of the MDP are of the form $\{S(t), R(t), 0\}$ or $\{S(t), 0, D(t)\}$. Files leave the cache either because they are evicted or because their retention time expires. A file whose retention time expires is said to *depart* from the cache.

The cache state transitions can be summarized as follows. When file $D(t)$ departs from the cache $S(t)$, the cache becomes $S(t) - D(t)$. If there is a file arrival which results in a cache hit (i.e. $R(t) \in S(t)$) then the cache content at time $t+1$ is the same as that at time n (i.e. $S(t+1) = S(t)$). In the case of a cache miss, two cases arise: (1) if the cache is not full then the new file gets added to the cache, i.e. $S(t+1) = S(t) + R(t)$; (2) If the cache is full, then, the state at time $t+1$ is $S(t) + R(t) - U(t)$ where $U(t), U(t) \in S(t) + R(t)$ is the random variable denoting the file evicted on n^{th} arrival on a full cache. Note that we assume *optional* evictions, i.e. the policy may not evict a stored file upon a cache miss (in which case we say that the requested file $R(t)$ itself is instantaneously evicted). Formally,

$$
\begin{aligned}
&S(t+1) = T(\mathbf{S}(t), U(t)) \\
&= \begin{cases} S(t) & \text{if } R(t) \in S(t), |S(t)| \leq B \\ S(t) + R(t) & \text{if } R(t) \notin S(t), |S(t)| < B \\ S(t) + R(t) - U(t) & \text{if } R(t) \notin S(t), |S(t)| = B \\ S(t) - D(t) & \text{if } R(t) = 0, |S(t)| \geq 1 \end{cases}
\end{aligned}
$$

Our goal is to find the optimal eviction sequence $U(t)$, $t = 1, 2, \dots, T$, using MDP by using the optimal values of $D(t)$ (i.e. the retention times $\sim exp(\mu_j)$, $j \in M$) obtained in Section IV.

2) Markovian Policy: It is easy to see that state $\mathbf{S}(t+1)$ only depends on state $\mathbf{S}(t)$ and $U(t)$. Thus, we need to focus only on Markovian policies (deterministic or randomized) that give optimal eviction sequences. Let $\mathcal{P}$ denote the set of all Markovian policies for evicting files. A policy $\pi \in \mathcal{P}$ is of the form $\pi = \{\pi_1, \pi_2, \dots \ \pi_T\}$, where each π_t is a mapping from state $\mathbf{S}(t)$ to the evicted file in $\{0, 1, \dots, M\}$, *i.e.* $U(t) = \pi_t(\mathbf{S}(t))$. We define $U(t) := 0$ when: (1) no eviction decision needs to be made (i.e. $R(t) \in S(t)$); (2) there is a cache miss and $U(t)$ refers to a file not present in cache or request (i.e. $R(t) \notin S(t)$ and $U(t) \notin S(t) + R(t)$). Let $\pi_t(u, \mathbf{S}(t))$ be the probability that policy π evicts file u in state $\mathbf{S}(t)$ on n^{th} arrival, where $u \in \mathcal{M}$, then $\pi_t(u, \mathbf{S}(t))$ satisfies the following properties:

$$
\begin{aligned}
&\sum_{u \in M} \pi_t(u, \mathbf{S}(t)) = 1, \\
&\pi_t(u, \mathbf{S}(t)) = 0 \ \ \forall u > 0 \text{ if } R(t) \in S(t), \\
&\pi_t(u, \mathbf{S}(t)) = 0 \ \ \forall u : u \notin S(t) + R(t), R(t) \notin S(t),
\end{aligned}
$$

3) State transition probabilities: For our DTMC with state transitions due to file request arrivals and file departures, the probability of leaving a state due to an arrival of file r is given by $\hat{p}_r = \lambda_r / \sum_{m \in M} (\lambda_m + \mu_m)$ and due to a departure of file d is $\tilde{p}_d = \mu_d / \sum_{m \in M} (\lambda_m + \mu_m)$ since files have exponential interarrivals and retentions (as defined in Section IV-A). Let $\mathbf{p}$ denote the pmf of these probabilities. Let $\mathbf{P}_\pi, \mathbf{E}_\pi$ denote the probability measure, expectation (respectively) under pmf $\mathbf{p}$ and policy π and let $\mathbf{1}[\cdot]$ be the indicator function then we derive the state transition probabilities as follows:

$$
\mathbf{P}_\pi[U(t) = u | \mathbf{S}(t)] = \pi_t(u, \mathbf{S}(t)), u \in M \tag{8}
$$

$$
\begin{aligned}
&\mathbf{P}_\pi[S(t+1) = \tilde{S}, R(t+1) = r, D(t+1) = 0 | \mathbf{S}(t), U(t)] \\
&= \hat{p}_r \times \mathbf{P}_\pi[S(t+1) = \tilde{S} | \mathbf{S}(t), U(t)] \\
&= \hat{p}_r \times \mathbf{1}[T(\mathbf{S}(t), U(t)) = \tilde{S}]
\end{aligned} \tag{9}
$$

$$
\begin{aligned}
&\mathbf{P}_\pi[S(t+1) = \tilde{S}, R(t+1) = 0, D(t+1) = d | \mathbf{S}(t)] \\
&= \tilde{p}_d \times \mathbf{P}_\pi[S(t+1) = \tilde{S} | \mathbf{S}(t)]
\end{aligned} \tag{10}
$$

Equations (8)-(9) follow since IRM file arrivals are independent of the state of the cache and the time of the request. Equations (9)-(10) apply for every $(\tilde{S}, r, d) \in \mathbf{S}(\mathbf{t} + \mathbf{1})$.

4) Cost function: A one-shot cost $c(m)$ for file m (e.g., sum of mean retrieval and retention cost as in Section IV) is incurred on every cache miss. The expected cost for the horizon of length T under the policy π becomes:

$$
J_c(\pi, T) = \mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}_{[R(t) \notin S(t)]} \times c(R(t)) \right]
$$

The average cost over the horizon of T discrete time steps under policy π is given by, $J_c(\pi) =$

$\limsup_{T\to\infty} \frac{\sum_{m\in M}(\lambda_m+\mu_m)}{T+1} J_c(\pi,T)$[10].

B. The Optimal Eviction Policy

Now we formulate and solve the MDP to find an optimal eviction policy. We define $J_c(\pi,(S,R,D),T)$ as the cost-to-go for the policy π starting in the state $\mathbf{S}=\{S,R,D\}$. Minimizing $J_c(\pi,(S,R,D),T)$ at every possible state will give us the optimal eviction policy.

$$J_c(\pi,(S,R,D),T) := \mathbf{E}_\pi\left[\sum_{t=0}^{T} \mathbf{1}_{[R(t)\notin S(t)]} \times c(R(t)|\mathbf{S}(0)=\{S,R,D\})\right].$$

We will use the *value iteration* approach to solve our problem. The value function minimizes cost-to-go over all policies, i.e. $V_T(S,R,D)=\inf_{\pi\in\mathcal{P}} J_c(\pi,(S,R,D),T)$.

Next, we write the Dynamic Programming Equation (Bellman equation) for this MDP. We form two different recurrence equations for the states of type $(S,r,0)$ and $(S,0,d)$, each accounting for a file request and a departure (recall that no other types of states are possible as we have assumed that file requests and departures are mutually exclusive). We first state the recurrence equations followed by the explanation:

$$\begin{aligned} V_{T+1}(S,r,0) &= \mathbf{1}_{\{r\in S\}}\mathbb{E}_{R^*}[V_T(S,R^*,0)] \\ &+ \mathbf{1}_{\{r\notin S,|S|<B\}}\left(c(r)+\mathbb{E}_{R^*}[V_T(S+r,R^*,0)]\right) \\ &+ \mathbf{1}_{\{r\notin S,|S|=B\}} \\ &\left(c(r)+\min_{u\in S+r}\mathbb{E}_{R^*}[V_T(S+r-u,R^*,0)]\right), \\ &+ \mathbf{1}_{\{r\in S\}}\mathbb{E}_{D^*}[V_T(S,0,D^*)] \\ &+ \mathbf{1}_{\{r\notin S,|S|<B\}}\left(c(r)+\mathbb{E}_{D^*}[V_T(S+r,0,D^*)]\right) \\ &+ \mathbf{1}_{\{r\notin S,|S|=B\}} \\ &\left(c(r)+\min_{u\in S+r}\mathbb{E}_{D^*}[V_T(S+r-u,0,D^*)]\right) \end{aligned} \tag{11}$$

$$\begin{aligned} V_{T+1}(S,0,d) &= \mathbb{E}_{R^*}(V_T(S-d,R^*,0)) \\ &+ \mathbb{E}_{D^*}(V_T(S-d,0,D^*)) \end{aligned} \tag{12}$$

The different terms in (11)-(12) can be explained as follows:

- $V_{T+1}(S,r,0)$ is the value of the objective when optimal action is taken in the state $S,r,0$ at time $t=0$ to minimize the cost over the horizon $[0,T+1]$. The first (or fourth) term in the sum says that when file request r belongs to the cache S then the expected cost for horizon $[0,T]$ due to a file request R^* (or a departure D^*) at time $t=0$ is given by $\mathbb{E}_{R^*}[V_T(S,R^*,0)]$ (or $\mathbb{E}_{D^*}[V_T(S,0,D^*)]$).
- The second and fifth terms differ from the above in that the file r requested at $t=0$ leads to a cache miss but the cache is not full so the requested file is written in the cache (without any eviction) thus increasing the expected cost over the horizon $[0,T]$ by $c(r)$.
- The third and sixth terms capture the case when the cache is full at $t=0$ and there is a cache miss upon request thus leading to a file eviction. The expected cost over the horizon $[0,T]$ is thus obtained by minimizing over all possible evictions, i.e. $u\in S+r$.

Equation 12 represents file d departing from the cache at time $t=0$. Here no cost is incurred over the horizon $[0,T+1]$ since we do not fetch or write a new file. The two terms in the sum infer that the next state could be due to a file request or a file departure at $t=1$.

By inspection, we observe that in Bellman equations (11)-(12), we *only* need to optimize the term: $\min_{u\in S+r}\mathbb{E}_{R^*}[V_T(S+r-u,R^*,0)]$[11]. Theorem 2 characterizes the optimal eviction policy obtained from the minimization.

Theorem 2. *To minimize the expected cost over the horizon $[0,T]$, the optimal eviction policy evicts a file v in the state $(S,r,0)$ that satisfies the following:*

$$v=\arg\min_{u\in S+r}\{p_u c(u)\} \tag{13}$$

whenever the cache is full and there is a cache miss on the request r (i.e. $r\notin S$).

Theorem 2 is proved in Appendix C. It characterizes a *stationary* optimal Markov eviction policy which suggests evicting a file that is requested least often and can be fetched, written with the least cost. Intuitively, the eviction rule seems fairly reasonable. We note that the non-uniform cost function $c(u)$ is general as it can be easily extended to represent an convex combination of various factors such as link congestion, available bandwidth on the channel (from where file u is fetched), etc. [15].

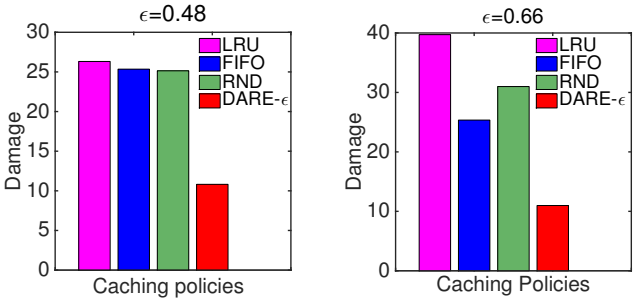

Fig. 4: Damage values for LRU, FIFO, RND and DARE-ϵ with Poisson arrivals: (1) ZipF-0.65 popularity, $\epsilon=0.48$, (2) ZipF-0.95 popularity, $\epsilon=0.66$.

C. DARE-Δ end-to-end policy design

We now outline a complete description of our online policy, DARE-Δ. It consists of two main blocks:

(I) Preprocessing: Given the set of files (M), an acceptable delay (Δ), the file arrival distribution and the cost specifications, obtain the optimal retention time parameters of all files (i.e. $\mu_i, i\in M$) by solving the convex optimization problem in (7). Further, sample retention times $\mathcal{R}_i, i\in M$ where each retention time, $\mathcal{R}_i$, is an exponential random variable sampled from mean μ_i.

[10]It is possible that with an arbitrary policy π the limit may not exist, therefore we use supremum which is a standard practice in the MDP literature.

[11]There are two minimization terms in Bellman equation, however, the second term can be minimized only by minimizing the first term due to the recurrence relation. See details in the Proof of Theorem 2 in Appendix.

(II) Run-time execution: Given a request for file $r \in M$ at time $t \in \{1, 2, \ldots, T\}$, check if the cache S contains the requested file. If so, serve r instantaneously. If not, then find the file k in cache such that $k = \arg\min_{u \in S+r} p_u c(u)$ (see Theorem 2). If file $k = r$, i.e. file k is the request itself then do nothing. If file k is a file from the cache then two cases arise: (a) if the cache is not full then write the requested file r in cache for a retention duration $\mathcal{R}_r$; (b) if the cache is full then write the requested file r with retention $\mathcal{R}_r$ and evict file k.

We claim that the run-time execution takes $O(\log M)$ computations for each time slot. We construct the cache S by using a binary search tree with file indices as keys. Then it takes $O(\log M)$ time to check if S contains the requested file. If it does, the file is served and the algorithm terminates. For the case when file is not present, we maintain a min-heap with $p_u c(u)$ as values (for every $u \in \{S + r\}$), which can find the file k that minimizes $p_u c(u)$ in $O(1)$ time and will return the readjusted heap after (possibly) deleting the file in $O(\log M)$ time. Thus the worst-case execution time is $O(\log M)$ for any slot.

D. Numerical comparison against other online policies

Recall that the caching policies LRU, FIFO and RND *assume* that the files are written in the cache until evicted. For comparison, we embed the notion of retention time in the well-known policies by first assuming that the files are written in the cache for a deterministic time thus incurring a one-shot damage on each write. Second, we even optimize these policies by finding the *best* such time for each policy. We do this by simulating the policies over a wide range of time values and finding a time that yields minimum damage if all files are written in cache for that time.

We consider Poisson arrivals modulated with ZipF-α with $\alpha \in \{0.65, 0.95\}$, respectively. We consider unit delay and fix a value for expected cache misses, $\Delta = \epsilon$, in (7). Further, we simulate DARE-Δ by writing files in the cache for the optimal retention time computed from the solution of (7) with cost $c(m) = f(m)$ for the chosen value of ϵ. The damage function for writing files is assumed to be quadratic in retention time. Upon a cache miss, DARE-Δ evicts the file u with least $p_u c(u)$ (see Theorem 2).

The results in Figure 4 show that *even after optimizing* the existing caching policies over all possible retention times, DARE-Δ outperforms other policies by giving a *2-4 fold* damage savings, thus agreeing with the analytical result (derived in Theorem 2). Moreover, we note that (1) FIFO and RND differ significantly with respect to damage under IRM, but are known to perform similar in terms of cache miss. (2) LRU and RND, which are being actively considered for deploying in CCN caches, perform very poor with respect to damage.

VI. Bridging the gap between offline & online policies

So far we described DARE-Δ and have shown some numerical results on its performance. In this section, we set up a simulation framework to test the large cache approximation. We benchmark the performance of DARE-Δ against LRU and the offline DARE policies. Recall that the offline policies in Section III assume that all cache misses are allocated, i.e. a requested file *must* be written to the cache in the event of a cache miss (see Section III). However, from a device damage perspective, allocating every cache miss is not necessarily optimal. For example, if there is a cache miss on a request for a very unpopular file then it can be served directly by fetching it from the server instead of writing it in the cache at the expense of evicting a more popular file. This practice of selecting when to cache a file and when not to is particularly useful in mitigating expensive write damage in a flash memory [24]. We thus allocate cache misses and obtain the modified variants of DARE and LRU, referred henceforth as DARE* and LRU*. These policies are allowed to not cache the requested file *if* it is going to be requested farthest in the future (for DARE*) or is the least recently used (for LRU*). LRU* operates by assigning recency timers (indicating the time when a file was last requested) with all the files and removing the file with smallest timer value from cache or request. We recall that our online policy already allocates cache misses since it has the option to evict the request itself.

The performance analysis is based on the following parameters. We are interested in time asymptotics so we first generate a long request string corresponding to a horizon of length $T = 10^5$ slots (as in the offline case) or transitions (as in the online case). The generated file requests are sampled from $|M| = 500$ files (of equal size). File requests form a Poisson process modulated with ZipF-ian popularity as before, i.e. the probability of requesting file i is proportional to $1/i^\alpha$, with the sum of request probabilities normalized to one. We consider $\alpha = \{0.65, 0.95\}$ which is common for web caches. We obtain the optimal retention times from Section IV and the optimal file to be evicted on each miss from Section V. The damage function for writing files is assumed to be quadratic in retention time. Cache size B is varied from 5 to 500 files in steps of 55. For each value of cache size, we obtain results and average it over 30 iterations. Damage (or delay) for a particular cache size is calculated by obtaining the average damage (or fraction of cache misses) over all iterations.

A. Damage-delay trade-offs

We evaluate the damage-delay trade-off with increasing cache sizes for two settings. We first show the delay-damage trade-off for DARE* versus LRU* with increasing cache size in Figure 5(a). The solid lines indicate the damage curve and the dotted lines indicate the delay curve. We observe that as the cache size increases, the damage increases and delay decreases, which is consistent with our observations in Section III. Moreover, we note that a higher value of α results in a lower damage. This is expected since popular files get a larger share with increasing α (i.e. the disparity between a popular versus a non-popular file increases) thus sufficing to store a few most popular files. Although we only show the curve for $|M| = 500$ files, we state our observation that the value of damage as well as the cache miss (fraction)

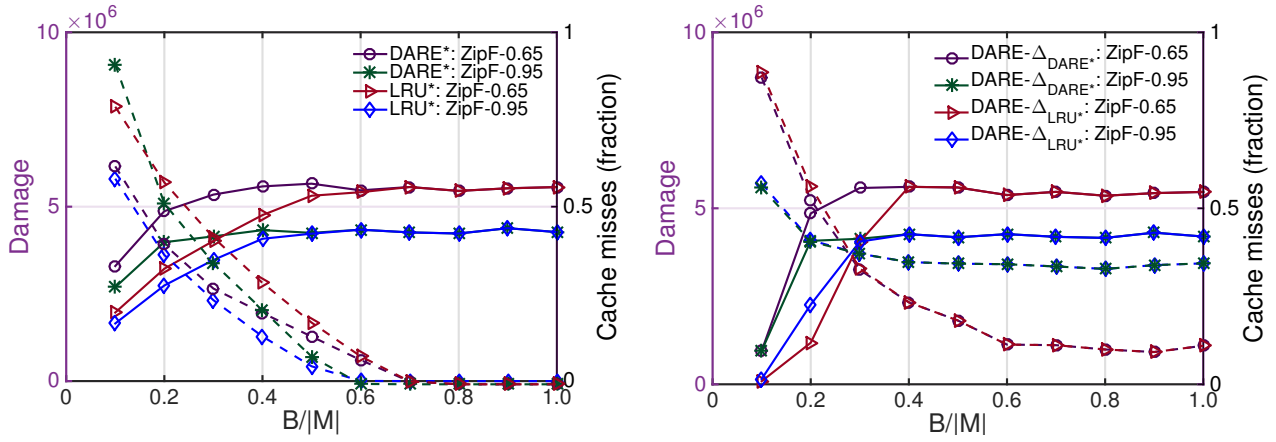

Fig. 5: Delay-damage trade-offs for $|M| = 500$ with increasing cache size B and varying ZipF parameters. The bold lines show damage, the dotted lines show cache miss (fractions).

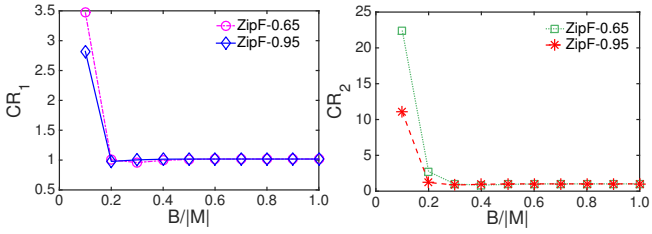

Fig. 6: Competitive ratios for the online policies with varying cache size and ZipF exponents for $|M| = 500$ files.

increases (at each point) as the number of files increase, however, the plot follows the same trend as in Figure 5(a).

We further use the delay (i.e. cache miss fraction, ϵ) obtained from DARE* and LRU* for each cache size and provide it as an input to the online policy DARE-Δ. The results are plotted as DARE-$\Delta_{\text{DARE}*}$ and DARE-Δ_{LRU*}, respectively, in Figure 5 (b). Similar trends as above were observed in damage-delay trade-offs. We also note that both damage and cache misses converge to a steady state with increasing cache size. We also observe that the cache miss (fraction) obtained with ZipF-0.65 is higher than the cache miss fraction with ZipF-0.95 (in steady state) irrespective of the input ϵ. This suggests that cache misses heavily depend on request popularity profile more than the target input ϵ. Also, even though the retention times were obtained by feeding ϵ from DARE* and LRU* (see the dotted lines in Figure 5 (a)), the resultant delay obtained from variants of DARE-Δ was found to be moderately higher than the original ϵ. This is the price of uncertainty paid when shifting from the offline caching, which has a complete knowledge of request arrivals, to the online caching, which only knows the value of the expected delay (cache miss fraction, ϵ).

B. Relative Damage, offline versus online

We now calculate the *cost ratio*, CR, obtained by taking the ratio of the damage incurred from the optimal offline algorithm (i.e. DARE*) with the optimal online algorithm (i.e. DARE-Δ). In our analysis, file requests conform to Poisson distribution thus limiting the possibility of a pathological worst-case input. We obtain CR for a long request string of length $T = 10^5$ (with other parameters same as above) to calculate two quantities. (1) $CR_1 = \frac{\text{damage from DARE*}}{\text{damage from DARE-}\Delta_{\text{DARE*}}}$, (2) $CR_2 = \frac{\text{damage from LRU*}}{\text{damage from DARE-}\Delta_{\text{LRU*}}}$. A value $CR = r$ shows an r-fold superiority of the online algorithm over the offline counterpart. The plot in Figure 6 shows that CR_2 is (approximately) 5-fold higher than CR_1. This is because the damage value from $\text{DARE} - \Delta_{\text{LRU}*}$ (the denominator of CR_1) is approximately 4-5 folds higher than $\text{DARE} - \Delta_{\text{DARE}*}$ (the denominator of CR_2), whereas damage from DARE$*$ (numerator of CR_1) is always within 0.8 to 1 times of damage from LRU$*$ (numerator of CR_2).

We also observe that the online cost is always lower than the offline cost resulting in both CRs' to take a value greater than one. Moreover, we observe that both the CRs' start with a higher value and gradually converge to one. This shows that our optimal online policy converges very fast to the damage performance of the optimal offline policy with increasing cache size. We briefly justify these observations as follows. Recall that the online policy obtains retention times for an infinte capacity cache whereas the offline policy uses a finite capacity cache. The only interaction between offline and online policies is via the delay (or the cache miss fraction, ϵ) which we obtain from the offline policy and pass as a parameter to the online policy. Thus, for smaller caches, due to the cache capacity constraint, the offline policy incurs a higher cost compared to the online policy which assumes an uncapacitated cache while calculating file retentions. Nevertheless, as the cache size grows, the discrepancy between the two policies vanishes and the cost incurred by both offline and online policies match up.

VII. Related Work

Content caching is employed by a wide range of technologies such as mobile/content-centric networks [25]–[28], heterogeneous networks [29], content distribution networks and data centers [30], [31]. Caching algorithms can be classified in two categories. The first category consists of *replacement policies* where a finite amount of cache memory is monitored to maintain the most recent/requested contents

in the cache. Various algorithms of this kind (such as LRU, FIFO, RND and hybrids thereof) are surveyed in [32], [33]. Replacement algorithms have been analyzed mostly under the IRM model [14], [34]. Despite the simplicity of replacement based algorithms, their analysis becomes very complicated even for an isolated cache. Several approximations have been proposed to analyze replacement caches of reasonable sizes. A key example is is the well-known Che's approximation for the widely used LRU caches proposed in [35] which was theoretically justified in [36]. This result was recently generalized to consider renewal traffic [16], and consequently extended to analyze different caching policies (see [37] and the citations therein). For example, in [38] a fast iterative scheme is proposed for FIFO which is similar to the algorithm proposed in [16] for RND caching policy.

The second category of caching algorithms use a *timer-driven eviction* model (or TTL caches) where content stored for a certain time is evicted from the cache upon timer expiration [14], [21], [34], [39]. Originally, TTL caches were designed as a tool to model the otherwise hard to analyze replacement based policies, while having no objective of their own. Since then several modifications have been suggested: for example, authors in [39], [40] analyze TTL caches to maximize hit rate in an online setting. Our work is similar to timer-driven eviction policy like TTL caches as in [39], [40] with the difference that we aim at minimizing cache damage subject to reasonable delay guarantees.

There exists cache replacement algorithms for content-centric networks with proposals to limit the admission of files in cache [26]–[28]. In [27] caching files in highly connected nodes in the network is proposed. [28] proposed storing files in randomly chosen nodes along the content delivery path; [26] advocated caching popular files in every cache, the aim of both being increasing the hit-rate. Although it is outside the scope of this work, we believe that extending our work to consider joint storage and content delivery/routing will prove superior to these policies as these policies do not take into account storage damage but only focus on maximizing the hit rate.

Although extensive efforts have been devoted to maximizing flash memory lifetime, prior work focused mainly on using flash memory as a general-purpose SSD in replacement of traditional hard disks. SSDs are not aware of any data semantics, and strive to guarantee a very long retention time (a few years) for all data. This apparently does not match with the characteristics of content-centric networks where contents have lifetime [10], [17], [18]. A recent work in [31] proposed a framework for optimizing the performance of flash caches by reducing random write overheads. While their framework is shown to outperform FIFO with respect to cache hit rates, it is not very clear if it is optimal with respect to our metric/other caching policies.

Prior works on designing caching policies for non-uniform cost using MDP can be found in [15], [41]. In [15], authors have considered uniform file sizes but non-uniform file retrieval costs and have shown that the optimal policy corresponds to a stationary Markovian policy which evicts the file with minimum $p_u c(u)$ for a file u in cache. Further, [41] generalized the analysis in [15] to non-uniform file sizes and showed that the optimal policy becomes history-dependent as well as NP-Hard to compute. We note that our work is a significant generalization of [15] where authors did not consider flash damage constraints in their formulation.

VIII. Conclusions

This paper advances the state-of-the-art of traditional data caching literature when applied to CCN caches by proposing a cross-layer optimization for the network layer objective of minimizing the content retrieval delay and the device layer objective of minimizing the flash damage. We analyze the delay-damage trade-offs for both offline and online caching to obtain optimal damage-aware caching policies. Our results demonstrate that our policies achieve significant damage reductions when compared to the traditional caching policies with the same delay bounds. This advocates using damage-aware caching policies in data intensive applications where flash memory cost and wear-out are of critical importance.

This work can be extended to several possible directions. For example, by considering temporal correlations in file request arrivals or by extending the problem to a network of caches. It is an open problem to devise a framework that jointly optimizes over both retention times (i.e. all possible distributions) and eviction sequences. Moreover, it would be interesting to generalize the damage/retention time modeling by taking more device-related aspects into account (such as the region on the flash chip and the age of the cache memory). Finally, it would be of theoretical interest to solve the problem for non-uniform file sizes.

Acknowledgements

This material is based upon work supported in part by the U.S. National Science Foundation under grant numbers 1422153, and a Tekes FiDiPro Fellow award with University of Oulu, Oulu, Finland.

References

[1] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *ACM SIGCOMM, 2009*, ser. SIGCOMM '09, 2009.

[2] B. J. Ko, V. Pappas, R. Raghavendra, Y. Song, R. B. Dilmaghani, K.-w. Lee, and D. Verma, "An Information-centric Architecture for Data Center Networks," in *ACM ICN Workshop, 2012*.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *ACM Communication, 2010*.

[4] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer, "Mercury: Host-side Flash Caching for the Data Center," in *IEEE MSST*, 2012.

[5] Y. Lu, J. Shu, and W. Wang, "ReconFS: A Reconstructable File System on Flash Storage," in *USENIX, FAST*, 2014.

[6] X. Jimenez, D. Novo, and P. Ienne, "Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance," in *USENIX FAST*, 2014.

[7] J. Jeong, S. S. Hahn, S. Lee, and J. Kim, "Lifetime Improvement of NAND Flash-based Storage Systems Using Dynamic Program and Erase Scaling," in *USENIX, FAST*, 2014.

[8] K. Vaid, "Designing SSDs for Large Scale Cloud Workloads," http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2014/20140807_Keynote10_Microsoft_Vaid.pdf/, 2014.

[9] P. Desnoyers, "What Systems Researchers Need to Know about NAND Flash," in *5th USENIX Workshop on Hot Topics in Storage and File Systems*, 2013.

[10] V. A. Siris, X. Vasilakos, and G. C. Polyzos, "Efficient Proactive Caching for Supporting Seamless Mobility," *arXiv preprint arXiv:1404.4754*, 2014.

[11] S. Samta and A. A. Abouzeid, "On Designing Optimal Memory Damag Aware Caching Policies for Content-Centric Networks," in *IEEE WiOpt, 2016*.

[12] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing NAND Flash-based SSDs via Retention Relaxation," *USENIX FAST, 2012*.

[13] L. Shi, K. Wu, M. Zhao, D. Liu, J. Xue, and E. Sha, "Retention Trimming for Lifetime Improvement of Flash Memory Storage Systems," *IEEE TCAD*, 2015.

[14] N. É. C. Fofack, "On Models for Performance Analysis of a Core Cache Network and Power Save of a Wireless Access Network," Ph.D. dissertation, Université Nice Sophia Antipolis, 2014.

[15] O. Bahat and A. Makowski, "Optimal replacement policies for nonuniform cache objects with optional eviction," in *IEEE INFOCOM*, 2003.

[16] V. Martina, M. Garetto, and E. Leonardi, "A Unified Approach to the Performance Analysis of Caching Systems," in *IEEE INFOCOM*, 2014.

[17] S. Dernbach, N. Taft, J. Kurose, U. Weinsberg, C. Diot, and A. Ashkan, "Cache Content-Selection Policies for Streaming Video Services," *IEEE INFOCOM, 2016*.

[18] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing Dynamic Content in Caches with Small Population," *CoRR*, 2016.

[19] I. Koltsidas and S. D. Viglas, "Data Management over Flash Memory," in *ACM SIGMOD, 2011*.

[20] J. Jung, A. Berger, and H. Balakrishnan, "Modeling TTL-based Internet Caches," in *IEEE INFOCOM*, 2003.

[21] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact Analysis of TTL Cache Networks: The Case of Caching Policies Driven by Stopping Times," in *ACM SIGMETRICS, 2014*.

[22] N. Choungmo Fofack, D. Towsley, M. Badov, M. Dehghan, and D. L. Goeckel, "An Approximate Analysis of Heterogeneous and General Cache Networks," Tech. Rep.

[23] L. A. Belady, "A Study of Replacement Algorithms for a Virtual-storage Computer," *IBM Systems journal, 1966*.

[24] T. Pritchett and M. Thottethodi, "SieveStore: A Highly-selective, Ensemble-level Disk Cache for Cost-performance," in *ACM SIGARCH ISCA, 2010*.

[25] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, 2014.

[26] C. Bernardini, T. Silverston, and O. Festor, "Mpc: Popularity-based caching strategy for content centric networks," in *2013 IEEE International Conference on Communications (ICC)*, 2013.

[27] "Cache less for more in information-centric networks (extended version)," *Computer Communications, 2013*.

[28] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic In-network Caching for Information-centric Networks," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ser. ICN '12, 2012.

[29] J. G. Andrews, "Seven ways that HetNets are a Cellular Paradigm Shift," *IEEE Communications Magazine*, 2013.

[30] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An Analysis of Facebook Photo Caching," in *ACM SOSP, 2013*, 2013.

[31] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li, "RIPQ: Advanced Photo Caching on Flash for Facebook," in *USENIX FAST, 2015*, 2015.

[32] A. Balamash and M. Krunz, "An Overview of Web Caching Replacement Algorithms," *IEEE Communications Surveys Tutorials*, 2004.

[33] J. Wang, "A Survey of Web Caching Schemes for the Internet," *SIGCOMM Computer Communication Review, 1999*.

[34] N. C. Fofack, M. Dehghan, D. Towsley, M. Badov, and D. L. Goeckel, "On the Performance of General Cache Networks," in *Proceedings of the ACM VALUETOOLS, 2014*.

[35] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: Modeling, Design and Experimental Results," *IEEE Journal on Selected Areas in Communications*, 2002.

[36] C. Fricker, P. Robert, and J. Roberts, "A Versatile and Accurate Approximation for LRU Cache Performance," *CoRR, 2012*.

[37] N. Gast and B. Van Houdt, "Transient and Steady-state Regime of a Family of List-based Cache Replacement Algorithms," in *ACM SIGMETRICS, 2015*.

[38] A. Dan and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes," *SIGMETRICS Performance Evaluation Review, 1990*.

[39] A. Ferragut, I. Rodriguez, and F. Paganini, "Optimizing TTL Caches Under Heavy-Tailed Demands," *SIGMETRICS Performance Evaluation Review, 2016*.

[40] D. S. Berger, S. Henningsen, F. Ciucu, and J. B. Schmitt, "Maximizing Cache Hit Ratios by Variance Reduction," *SIGMETRICS Performance Evaluation Review, 2015*.

[41] Y. Su and L. V. Lakshmanan, "On efficient replacement policies for cache objects with non-uniform sizes and costs," ftp://ftp.cs.ubc.ca/local/techreports/2009/TR-2009-20.pdf, 2009.

APPENDICES

APPENDIX A: PROOF OF LEMMA 1

Proof. We want to show that the function, $h(\mathbf{q}, \boldsymbol{\lambda}, M) := \frac{1}{\sum_{m=1}^{M}\lambda_m}\sum_{m=1}^{M}\sum_{k=1}^{n} a_k k! \frac{q_m^{k+1}}{\lambda_m^k (1-q_m)^k}$ is convex. Let $\sigma(k,m,M) := \frac{a_k k!}{\lambda_m^k \sum_{m=1}^{M}\lambda_m}$. Note that $\sigma(k,m,M)$ does not depend on q_m. We prove the claim by first showing it for $M = 1$, where we have $h(q_1,\lambda_1,1) = \sum_{k=1}^{n}\sigma(k,1,1)\frac{q_1^{k+1}}{(1-q_1)^k}$. Let $H(q_1,\lambda_1,1)$ be the double derivative of $h(q_1,\lambda_1,1)$. For convexity, we require, $H(q_1,\lambda_1,1) = \frac{\partial^2 h}{\partial q_1^2} \geq 0$. Now, the first derivative with respect to q_1 is:

$$\frac{\partial h}{\partial q_1} = \sum_{k=1}^{n}\sigma(k,1,1)\frac{q_1^k(k+1-q_1)}{(1-q_1)^{k+1}}$$

and, the second derivative, after simplifying, is:

$$\frac{\partial^2 h}{\partial q_1^2} = \sum_{k=1}^{n}\sigma(k,1,1)\frac{q_1^{k-1}(1-q_1)^k k(k+1)}{(1-q_1)^{2(k+1)}}$$

which is well defined and non-negative as every term in the expression is non-negative provided $q_1 \in [0,1)$.
Thus the objective in (7) is convex because it is the sum of M different convex functions, thus proving the claim. ■

APPENDIX B: PROOF OF THEOREM 1

Proof. The optimality theorem follows from Lemmas 2, 3.

Lemma 2. *For the eviction sequence given by FiF policy, DARE policy gives optimal offline cost.*

Proof. DARE incurs the same delay cost as FiF (since both incur optimal cache misses). Thus to prove the lemma it suffices to prove the non-existence of a policy which assigns retention times to files as per the eviction sequence from FiF by incurring the same number of cache misses (delay cost) but a less retention cost than DARE. Let P^* be that policy. This implies that there exists a file l which is retained in cache for more slots with DARE compared to that with P^*. For l, there must exist a time triplet (t_0, t_1, t_2) – where $t_0 < t_1 < t_2$ – such that for the interval (t_0, t_1), l is present in cache with both the policies; however, for the interval (t_1, t_2), file l is only present in cache with DARE but not with P^*, thus causing a less retention cost with P^*. DARE is designed to store l in cache only for the time it is useful, indicating that l is stored for the duration (t_1, t_2) to account

for a request for file l at time t_2. However, at t_2, P^* will result in a cache miss since it did not have l cached, thus increasing the optimal number of cache misses by one. A contradiaction. This proves the claim. ■

Let $\mathcal{F}$ denote the family of eviction sequences with optimal (F) misses. While DARE is built on eviction sequence from FiF, E, we prove that DARE is optimal over all $J \in \mathcal{F}$.

Lemma 3. *The retention cost incurred with $J \in \mathcal{F}, J \neq E$ is greater or equal to the retention cost incurred by E.*

Proof. We provide a brief proof sketch due to space constraints. Suppose there exists an optimal sequence $J^* \in \mathcal{F}$, $J^* \neq E$, such that the total retention cost with J^* is less than that with E. We transform every eviction in J^* to the evictions in E using the *exchange argument* and show that the one-shot retention cost of a file in E is a permutation of the one-shot retention cost of a file in J^*, thus making the cumulative retention costs equal. A contradiction. ■

This completes the proof of Theorem 1. ■

APPENDIX C: PROOF OF THEOREM 2

Proof. There are two minimization terms in Bellman equations (11)-(12). We use induction for minimizing the first term in (11), i.e., $\min_{u \in S+r} \mathbb{E}_{R^*}[V_T(S+r-u, R^*, 0)]$. We then show that for the second term, i.e. $\min_{u \in S+r} \mathbb{E}_{D^*}[V_T(S+r-u, 0, D^*)]$, the proof proceeds by induction similar to minimization of the first term and it reduces to minimizing the first term itself due to (11), (12) after simplification.
We define $v = \arg\min\{u \in S+r : p_u c(u)\}$, then, $\mathbb{E}[V_T(S+r-v, R^*, 0)] = \min_{u \in S+r} \mathbb{E}[V_T(S+r-u, R^*, 0)]$. The proof proceeds by induction on $T = 0, 1, \ldots$. At each step, we want to show that
$\mathbb{E}[V_T(S+r-u, R^*, 0)] - \mathbb{E}[V_T(S+r-v, R^*, 0)] \geq 0$
The basis step: Fix the initial state, request as (S, r). Thus, $V_0(S, r, 0) = 1_{\{r \notin S, |S|<B\}} 2c(r) + 1_{\{r \notin S, |S|=B\}} 2c(r)$, and,

$$\begin{aligned}
&\mathbb{E}[V_0(S+r-u, R^*, 0)] \\
=&\mathbb{E}[(1_{R^* \notin \{S+r-u\}, |S+r-u|<B} + 1_{R^* \notin \{S+r-u\}, |S+r-u|=B}) 2c(R^*)] \\
=&\mathbb{E}[1_{R^* \notin \{S+r-u\}} 2c(R^*)] \\
=&\mathbb{E}[1_{R^* \notin \{S+r\}} 2c(R^*)] + \mathbb{E}[1_{R^*=u} 2c(R^*)]
\end{aligned}$$

Now we write an expression for $\mathbb{E}[V_0(S+r-v, R^*, 0)]$, and observe that,

$$\begin{aligned}
&\mathbb{E}[V_0(S+r-u, R^*, 0)] - \mathbb{E}[V_0(S+r-v, R^*, 0)] \\
&= \mathbb{E}[1_{R^* \notin \{S+r\}} 2c(R^*)] + \mathbb{E}[1_{R^*=u} 2c(R^*)] \\
&- \mathbb{E}[1_{R^* \notin \{S+r\}} 2c(R^*)] - \mathbb{E}[1_{R^*=v} 2c(R^*)] \\
&= 2(p_u c(u) - p_v c(v))
\end{aligned}$$

That is the claim is true with $T = 0$.
The Induction step: Assume that the claim is true for some fixed $T > 0$. Fix $(S, r, 0)$ and $(S, 0, d)$ with $r \notin S$ and $d \in S$. We need to show that, for $u \in S + r$, we have, $\mathbb{E}[V_{T+1}(S+r-u, R^*, 0)] - \mathbb{E}[V_{T+1}(S+r-v, R^*, 0)] \geq 0$. To show this, we take the expectation of $V_{T+1}(S+r-u, R^*, 0)$ and use (11) to get,

$$\begin{aligned}
&\mathbb{E}[V_{T+1}(S+r-u, R^*, 0)] \\
&= P[R^* \in S+r-u] E[V_T(S+r-u, R^{**}, 0))] && (14)\\
&+ P[R^* \in S+r-u] E[V_T(S+r-u, 0, D^{**}))] && (15)\\
&+ \mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|<B} * 2c(R^*)] && (16)\\
&+ \mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|=B} * 2c(R^*)] && (17)\\
&+ \mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|<B} \mathbb{E}(V_T(S+r-u+R^*, R^{**}, 0))] && (18)\\
&+ \mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|<B} \mathbb{E}(V_T(S+r-u+R^*, 0, D^{**}))] && (19)\\
&+ \mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|=B} \widehat{V}_T(S+r-u, R^*, 0)] && (20)\\
&+ \mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|=B} \widehat{V}_T(S+r-u, 0, R^*)] && (21)
\end{aligned}$$

where $\widehat{V}_T(S+r-u, R^*, 0) := \min_{u' \in S+r-u+R^*} \mathbb{E}[V_T(S+r-u+R^*-u', R^{**}, 0)]$ and $\widehat{V}_T(S+r-u, 0, R^*) := \min_{u' \in S+r-u+R^*} \mathbb{E}[V_T(S+r-u+R^*-u', 0, D^{**})]$. Now, we state some *reductions* (22) to (25) which will be instrumental in getting to the proof.

$$\begin{aligned}
&P[R^* \in S+r-u] \mathbb{E}[V_T(S+r-u, R^{**}, 0)] \\
&= P[R^* \in S+r-(u,v)] \mathbb{E}[V_T(S+r-u, R^{**}, 0)] \\
&+ p_v \mathbb{E}[V_T(S+r-u, R^{**}, 0)] && (22)\\
&\mathbb{E}[1_{R^* \notin \{S+r-u\}} c(R^*)] = \mathbb{E}[1_{R^* \notin \{S+r\}} c(R^*)] + p_u c(u) && (23)\\
&\mathbb{E}[1_{\{R^* \notin S+r-u\}} \widehat{V}_T(S+r-u, R^*, 0)] \\
&= \mathbb{E}[1_{R^* \notin \{S+r\}} \widehat{V}_T(S+r-u, R^*, 0)] \\
&+ p_u \widehat{V}_T(S+r-u, u, 0) && (24)\\
&\widehat{V}_T(S+r-u, u, 0) = \min_{u' \in S+r} \mathbb{E}[V_T(S+r-u', R^{**}, 0)] \\
&= \mathbb{E}[V_T(S+r-v, R^{**}, 0)] && (25)
\end{aligned}$$

With the machinery to simplify the expressions, we write the difference in terms with u and v for (14) through (21). Recall reduction (22) to express $14(u) - 14(v)$ as:

$$\begin{aligned}
&P[R^* \in S+r-(u,v)] \mathbb{E}[V_T(S+r-u, R^{**}, 0)] \\
&+ p_v \mathbb{E}[V_T(S+r-u, R^{**}, 0)] \\
&- P[R^* \in S+r-(u,v)] \mathbb{E}[V_T(S+r-v, R^{**}, 0)] \\
&- p_u \mathbb{E}[V_T(S+r-v, R^{**}, 0)] \\
&= P[R^* \in S+r-(u,v)] \\
&\times (\mathbb{E}[V_T(S+r-u, R^{**}, 0)] - \mathbb{E}[V_T(S+r-v, R^{**}, 0)]) \\
&+ p_v \mathbb{E}[V_T(S+r-u, R^{**}, 0)] \\
&- p_u \mathbb{E}[V_T(S+r-v, R^{**}, 0)]
\end{aligned}$$

We know by induction on T that, $\mathbb{E}[V_T(S+r-u, R^{**}, 0)] - \mathbb{E}[V_T(S+r-v, R^{**}, 0)] \geq 0$ holds. Thus, to prove $14(u) - 14(v) \geq 0$, we need, $p_v \mathbb{E}[V_T(S+r-u, R^{**}, 0)] - p_u \mathbb{E}[V_T(S+r-v, R^{**}, 0)] \geq 0$

Next, we invoke reduction (23), define $p_S(B) = P(|S| < B)$ and simplify $16(u) - 16(v)$ as follows:

$$\begin{aligned}
&\mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|<B} 2c(R^*)] \\
&- \mathbb{E}[1_{R^* \notin \{S+r-v\}, |S+r-v|<B} 2c(R^*)]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{R^* \notin \{S+r-u\}} P[R^*||S|<B]P[|S|<B]2c(R^*) \\
&\quad - \sum_{R^* \notin \{S+r-v\}} P[R^*||S|<B]P[|S|<B]2c(R^*) \\
&= \sum_{R^* \notin \{S+r-u\}} 2p_S(B) \times p_{R^*} c(R^*) \\
&\quad - \sum_{R^* \notin \{S+r-v\}} p_{R^*} p_S(B) 2c(R^*) \\
&= \sum_{R^* \notin \{S+r\}} 2p_S(B) \times p_{R^*} c(R^*) + 2p_S(B) \times p_u c(u) \\
&\quad - \sum_{R^* \notin \{S+r\}} p_{R^*} p_S(B) 2c(R^*) - 2p_S(B) \times p_v c(v) \\
&= 2p_S(B)(p_u c(u) - p_v c(v))
\end{aligned}
$$

which is ≥ 0 because $p_u c(u) \geq p_v c(v)$ by definition of v. Similarly, we show $17(u) - 17(v) \geq 0$.
Now, we invoke reduction (24) and consider $18(u) - 18(v)$.

$$
\begin{aligned}
&\mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|<B} \mathbb{E}(V_T(S+r-u+R^*, R^{**}, 0))] \\
&\quad - \mathbb{E}[1_{R^* \notin \{S+r-v\}, |S+r-v|<B} \mathbb{E}[V_T(S+r-v+R^*, R^{**}, 0))] \\
&= \sum_{R^* \notin \{S+r-u\}} P[R^*||S|<B]P[|S|<B] \\
&\quad \mathbb{E}[V_T(S+r-u+R^*, R^{**}, 0)] \\
&\quad - \sum_{R^* \notin \{S+r-v\}} P[R^*||S|<B]P[|S|<B] \\
&\quad \mathbb{E}[V_T(S+r-v+R^*, R^{**}, 0)] \\
&= \sum_{R^* \notin \{S+r-u\}} p_{R^*} p_S(B) \mathbb{E}[V_T(S+r-u+R^*, R^{**}, 0)] \\
&\quad - \sum_{R^* \notin \{S+r-v\}} p_{R^*} p_S(B) \mathbb{E}[V_T(S+r-v+R^*, R^{**}, 0)] \\
&= \sum_{R^* \notin \{S+r\}} p_{R^*} p_S(B) (\mathbb{E}[V_T(S+r-u+R^*, R^{**}, 0)] \\
&\quad - \mathbb{E}[V_T(S+r-v+R^*, R^{**}, 0)] \\
&\quad + (p_u - p_v) p_S(B) \mathbb{E}[V_T(S+r, R^{**}, 0)]
\end{aligned}
$$

which is ≥ 0 by induction, since $\mathbb{E}(V_T(S+r-u+R^*, R^{**}, 0))] \geq \mathbb{E}(V_T(S+r-v+R^*, R^{**}, 0))]$ and $p_u \geq p_v$ by definition of v. Similarly, we show that $19(u) - 19(v) \geq 0$. Finally, $20(u) - 21(v)$ becomes:

$$
\begin{aligned}
&\mathbb{E}[1_{R^* \notin \{S+r-u\}, |S+r-u|=B} \widehat{V}_T(S+r-u, R^*, 0)] \\
&\quad - \mathbb{E}[1_{R^* \notin \{S+r-v\}, |S+r-v|=B} \widehat{V}_T(S+r-v, R^*, 0)] \\
&= \sum_{R^* \notin \{S+r-u\}} P[R^*||S|=B] \times P[|S|=B] \widehat{V}_T(S+r-u, R^*, 0)] \\
&\quad - \sum_{R^* \notin \{S+r-v\}} P[R^*||S|=B] P[|S|=B] \widehat{V}_T(S+r-v, R^*, 0)] \\
&= \sum_{R^* \notin \{S+r\}} (1-p_S(B)) p_{R^*} \\
&\quad (\widehat{V}_T(S+r-u, R^*, 0) - \widehat{V}_T(S+r-v, R^*, 0)) \\
&\quad + (1-p_S(B))(p_u \widehat{V}_T(S+r-u, u, 0) - p_v \widehat{V}_T(S+r-v, v, 0))
\end{aligned}
$$

The first term in the above expression is always ≥ 0 by induction. Now, we only need to show that the following sum is non-negative by using the definition of $\widehat{V}_T(.)$:

$$
\begin{aligned}
&p_v \mathbb{E}[V_T(S+r-u, R^{**}, 0)] - p_u \mathbb{E}[V_T(S+r-v, R^{**}, 0)] \\
&+(1-p_S(B))[p_u \mathbb{E}(V_T(S+r-v, R^{**}, 0)) \\
&\quad - p_v \mathbb{E}[V_T(S+r-v, R^{**}, 0)]] \\
&= p_v \mathbb{E}[V_T(S+r-u, R^{**}, 0)] \\
&\quad - [(1-p_S(B))(p_v - p_u) - p_u] \mathbb{E}[V_T(S+r-v, R^{**}, 0)]
\end{aligned}
$$

Now $\mathbb{E}[\mathrm{V_T(S+r-v, R^{**}, 0)}] \leq \mathbb{E}[\mathrm{V_T(S+r-u, R^{**}, 0)}]$, by definition of v therefore the above expression is non-negative if: $p_v \geq (1-p_S(B))(p_v - p_u) - p_u$ which is equivalent to showing $2p_u \geq p_S(B)(p_u - p_v)$. This holds since $2p_u = p_u + p_u \geq p_u - p_v \geq p_S(B)(p_u - p_v)$. This completes minimizing the first term.

Proving the result for the second minimization term: We summarize the steps to prove the result for the expression $\min_{u \in S+r} \mathbb{E}_{D^*}[V_T(S+r-u, 0, D^*)]$. Assuming v is the same as above, we have $\mathbb{E}[V_T(S+r-v, 0, D^*)] = \min_{u \in S+r} \mathbb{E}[V_T(S+r-u, 0, D^*)]$. Our goal is to show $\mathbb{E}[V_T(S+r-u, 0, D^*)] - \mathbb{E}[V_T(S+r-v, 0, D^*)] \geq 0$ by induction on $T = 1, 2 \ldots$. Let d be the departing file. Note that $\mathbb{E}[V_0(S-d, 0, D^*)] = 0$ by (12), thus $V_1(S, 0, d) = \mathbb{E}[V_0(S-d, R^*, 0)] + \mathbb{E}[V_0(S-d, 0, D^*)] = 2c(R^*)(1_{\{R^* \notin S-d, |S-d| \leq B\}})$. Therefore, after a few algebraic manipulations we get $\mathbb{E}[V_1(S+r-u, 0, D^*)] - \mathbb{E}[V_1(S+r-v, 0, D^*)] = 2(p_u c(u) - p_v c(v))$. showing that the claim is true for the base case with $T = 1$.
We next assume that claim holds for some $T > 1$ and prove that it holds for $T+1$. Fix $(S, r, 0)$, $(S, 0, d)$ with $r \notin S$, $d \in S$. We need to show, $\forall u \in S+r$, $\mathbb{E}[V_{T+1}(S+r-u, 0, D^*)] - \mathbb{E}[V_{T+1}(S+r-v, 0, D^*)] \geq 0$. By taking expectation of $V_{T+1}(S+r-u, 0, D^*)$ and invoking (12), we have, $\mathbb{E}_{D^*}[V_{T+1}(S+r-u, 0, D^*)] = \mathbb{E}_{D^*}[[\mathbb{E}_{R^*}(V_T(S+r-u-D^*, R^*, 0))] + [\mathbb{E}_{D^{**}}(V_T(S+r-u-D^*, 0, D^{**}))]]$. The first term in the argument of $\mathbb{E}_{D^*}[.]$ is the same as solving the induction step for minimizing the first term. Also, due to recurrence (11), (12), the second term is equivalent to solving $\min_{u \in S+r} \mathbb{E}_{D^*}[V_T(S+r-u, 0, D^*)]$ which is true by induction.
This completes the proof of Theorem 2. ■

Samta Shukla received a B.E. in Electronics and Telecommunication Engineering from B.I.T, Durg, India, in 2010, and an M.S. in Computer Networks from Indian Institute of Science, Bangalore in 2013. Since 2013, she is a PhD graduate at Rensselaer Polytechnic Institute, Troy, NY, majoring in Computer Systems. Her dissertation is on stochastic optimization with a focus on problems in data caching and storage. She held a PhD internship at IBM Research in 2014.

A. A. Abouzeid is a Professor in the Department of Electrical, Computer and Systems Engineering at Rensselaer Polytechnic Institute, Troy NY. He is a visiting Professor and Finnish Distinguished Professor Fellow with the Department of Communications Engineering, University of Oulu, Oulu, Finland. From 2008 to 2010 he served as a Program Director at the National Science Foundation (NSF), Arlington, VA. His research is in the field of computer networks, focusing primarily on wireless systems and mobile computing. He founded WiFiUS, an international NSF funded virtual institute composed of 58 principal investigators from 29 institutions in the US and Finland. He received a CAREER award from NSF in 2006. He serves/served as an Associate Editor for several IEEE journals and magazines, and as a technical program committee member/chair for several ACM/IEEE conferences.