Published in: 2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)

On Designing Optimal Memory Damage Aware Caching Policies for Content-Centric Networks

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

***Abstract*— Caches in Content-Centric Networks (CCN) are increasingly adopting flash memory based storage owing to its efficiency. 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 suit the CCN data characteristics where contents are typically short-lived and have a distinct popularity profile.**

Writing files in cache using the longest retention time damages the flash memory thus reducing the flash 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 propose 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 and *where* to cache but also for *how long* is each file cached. The optimality of our policies are shown analytically, and are compared against prior policies using simulations.

I. Introduction

Flash memory based cache is a principal component of the emerging Content-Centric Networks (CCN) with ubiquitous caching, 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 which is directly proportional to the programmed data retention time.

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* (PE) cycles it can endure (e.g., 10^4 to 10^5 PE cycles) [5]. 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 [6]–[8].

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. Clearly, writing every content with maximum retention is wasteful since in CCN some content could be less popular or some more popular; an information which can be leveraged to obtain optimal retention times.

We take a first step in reformulating the traditional data caching problem by proposing a *cross-layer optimization* approach that combines the network-level objective of minimizing the content-retrieval delay and the device-level objective of minimizing the device damage. Earlier work in these areas have progressed largely independently. For example, recent works in device literature optimize damage/retention times by dynamically trimming the retention duration of a content based on the given information on data lifetime (such as the refresh cycle duration) [9], [10]; authors in [11] consider trading retention time for system performance (e.g., memory speed, lifetime). On the other hand, the caching literature consists of innumerable attempts to construct policies with high cache hit probability for achieving lower network delays (see [12] and the citations within) while overlooking the device-related aspects.

The key challenge addressed in this paper is to find caching policies, 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 at time t for a retention duration D is no longer readable from the cache if it is not requested at any time between t to $t+D$ thus leading to a cache miss. We assume that the retention times are *renewed* at each cache hit; this is consistent with the CCN paradigm that emphasis on moving popular content as close as possible to the users.

Our first contribution (Section II) is that we solve the problem of *offline caching*, i.e., caching when the content request string is given. We design an optimal policy that returns the optimal retention times for each *file write* by taking least number of cache misses. We note that, with the traditional caching (caching without optimizing on memory retentions), the caching policy by Belady [13] is known to give least number of cache misses. We compare our policy with popular alternatives like Least Recently Used (LRU),

Random (RND), First-In-First-Out (FIFO), and Belady's Farthest in Future (FiF) by accounting for the flash damage caused by writing files in cache for a duration until they are evicted due to a cache miss. Our simulations demonstrate that our policy, by taking retentions into account, achieves lower cache damage without increasing the cache-misses.

Our second contribution is that we solve the *online caching* problem, i.e., caching when the request string is not given. Towards this, we first assume a large cache (a cache with no capacity constraint) and obtain the optimal file retentions by solving an optimization formulation (Section III). Second, we extend the results from a large cache to a cache of finite capacity where a cache miss can result in a file eviction (if the cache is full) (Section IV). We exploit the optimal retentions obtained for large caches and model the problem of which file to evict at every cache miss as a Markov Decision Process (MDP). Further, we take a step ahead to characterize the solution of the MDP which leads to a very simple, easy to implement rule. Our simulations 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 [14] where the authors do not consider flash damage constraints in their formulation.

Our model for online caching is based on the following key assumptions. The file arrivals conform to the Independent Reference Model (IRM) [1] [14], [15] (where file i is requested with a static probability p_i independent of other requests). For tractability, we obtain results for poisson file arrivals modulated with a suitable popularity distribution (such as, ZipF) and exponentially distributed retention times in our analysis. While there are only empirical relationships known about flash damage as a function of the depleting cell life [8], we take the first step to mathematically model the flash damage as a function of retention times.

Having a file written for its retention time may appear similar to the TTL-caches considered in [16]–[19], however, we are the first to design memory damage minimizing policies for CCN caches. Our work, even at the conceptual level is different from TTL caches because we focus on designing policies for finite capacity caches to minimize damage, unlike the infinite capacity TTL caches which take a damage oblivious policy as input. Analyzing a finite capacity cache is paticularly applicable for CCN routers which are known to have small caches [12].

We compare our optimal policies against the performance from these well-known policies from the literature.

- LRU: In Least Recently Used policy, upon arrival of a request, a file not already in cache is inserted. If the cache is full, then the least recently used file is evicted.
- FIFO: First In First Out policy differs from LRU in file eviction in that if the cache is full, then the file which was written first (i.e., longest time ago) is evicted.
- RND: RaNDom policy differs from LRU in file eviction in that RND evicts a file from the cache uniformly at random.
- FiF (Belady's Algorithm): Farthest in Future policy [13] is optimal to minimize the number of cache misses. File insertion is the same as LRU; upon miss, FiF evicts the file whose next request is latest.

We note that LRU is a simple, widely used and efficient caching policy which performs well even for arbitrary request strings. RND and FIFO on the other hand, are very simple to implement in hardware; they are seen as a viable replacement of LRU in CCN high-speed routers [15]. FiF, although optimal, requires the knowledge of the entire request string.

[1] Although IRM does not take temporal locality into account, it is a widely accepted, standard traffic model in caching literature [14], [15].

II. Flash-aware Optimal Offline Caching

Consider the case of offline caching – where the request string $\mathcal{R}$ is given as a sequence of the file indices chosen from a set of M files. The main purpose of studying the offline case is to act as a reference model for other online algorithms. Briefly recall the Farthest-in-Future algorithm by Belady [13] which is known to minimize delay for a cache.

Our contribution is in showing that the well-known delay optimal policy, FiF, is not optimal with respect to damage. Further, we advance the state-of-the-art by constructing a policy OPT_{off} that minimizes damage by taking no more delay (cache-misses) than Belady's FiF (i.e. the known optimal delay benchmark).

A. System Model

For this section we assume that the time is slotted in equal length intervals; file requests arrive at the beginning of each slot. The files have unit length each; the cache memory has a finite capacity of size B. A requested file not in cache results in a cache miss and is written in cache for at least a slot. Files are served instantaneously in case of a cache hit.

We develop a suitable cost-metric to compare the damage performance of policies. Let the one-shot damage caused due to writing a file with retention time R be directly proportional to R, i.e., cost of writing is αR, where $\alpha > 0$ is a constant. We choose a linear cost for the ease of illustration and to lend insights to the case of online caching. We define the flash damage cost as the aggregate one-shot damage caused by writing all files in the cache upon a miss.

Let F_{OPT} be the optimal number of cache misses and E be the ordered sequence of evictions according to the FiF policy. Our goal is to find a policy, OPT_{off}, that determines, for each cache miss, the *optimal retention times* for each file write without exceeding F_{OPT} cache misses.

B. Finding optimal offline policy

Our policy is based on the following principle: OPT_{off} considers every eviction in the optimal eviction sequence given by the FiF caching policy and works backward to find the optimal retention for *each* file write. When a file f is evicted in FiF at time t, OPT_{off} finds two different time indices by traversing back from t. First, it finds the *latest* (time) slot when f was written in the cache before getting evicted at t; we call it time k. Second, it searches for the

time when f was last requested before eviction at t, we call it time j. Our policy stores file f 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 taken care of similarly. Thus, for each evicted file, OPT_{off} 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. Example 1 illustrates the algorithm.

Example 1. Assume that the cache size is $B = 3$ and at time $t = 0$ it contains files $\{a, b, c\}$. Consider the request string shown in Table I.

TABLE I
SEQUENCE OF EVICTIONS AND CACHE EVOLUTION WITH EACH REQUEST UNDER OPT_{off}.

Request (slot)	Request string	File evicted	Files in cache
1	a	-	{a,b,c}
2	e	b	{a,c,e}
3	c	-	-do-
4	a	-	-do-
5	d	c	{a,d,e}
6	a	-	-do-
7	b	d	{a,b,e}
8	e	-	-do-
9	a	-	-do-

For each eviction, OPT_{off} calculates the retention time backwards. Consider slot 5 when a request for file d results in a miss, and file c is evicted as per the solution of FiF. We find the last time when c was requested, i.e., $j-1=3 \implies j=4$. Note that, file c was in cache starting from time $t=0$. Hence, file c will be written for time $j-k=4-0=4$ slots. Similarly, it is easy to see that the output from OPT_{off} is to write both files a and e for 9 slots (since, files a and e are never evicted) and files b, d for 1 slot each.

C. OPT_{off} is optimal

We observe that OPT_{off} incurs optimal number of cache misses (by definition). Thus, for optimality, we only need to prove that there does not exist a policy which costs less damage than OPT_{off} in choosing retention times for files without exceeding the optimal number of cache misses. We report the optimality proof in [20] due to space constraints.

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

D. Simulation Results: 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 candidate policies (namely, LRU, FIFO, RND and FiF) write a file exactly for the time till it is not evicted. For the OPT_{off} policy, we write all files for the optimal retention durations as calculated above. Our goal is to analyze and demonstrate the optimality of OPT_{off} against the well-known caching policies for two different settings.

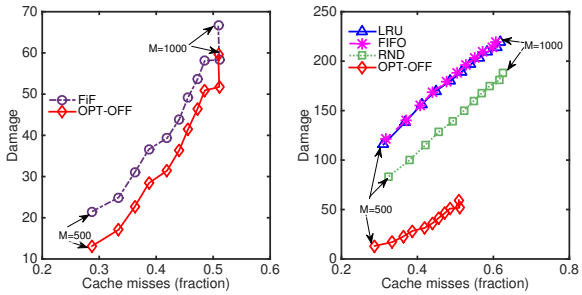

Fig. 1. Cache damage vs. cache miss under IRM with a constant cache size ($B = 300$) and increasing number of files ($M = 500$ to 1000).

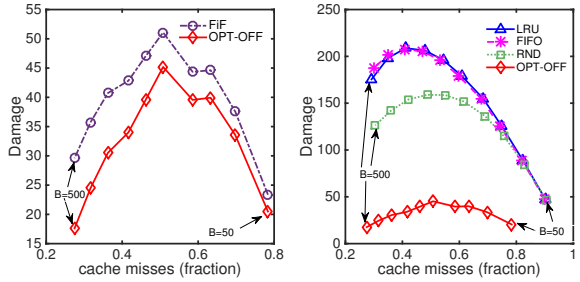

Fig. 2. Cache damage vs. cache miss under IRM with a constant number of files ($M = 800$) but with increasing cache size ($B = 50$ to 500).

1) Varying number of files: We perform various iterations by keeping cache size to a constant $B = 300$ and varying the number of files from 500 to 1000 in steps of 50 (with each point averaged over 500 iterations). The file request string is generated using a Poisson process modulated with linear popularity $\lambda_m = m$ and fed to the cache implementing LRU, FIFO, RND, FiF and OPT_{off} caching policies.

Figure 1 shows that as the number of files increase, both the fraction of the cache misses as well as cache damage increase as expected. The first figure shows that OPT_{off} achieves strictly less damage than FiF (for the same fraction of cache misses) thus demonstrating its optimality. From the second figure, we note that OPT_{off} gives *4-10 fold* damage reduction over traditional caching policies– LRU, RND, FIFO – thus justifying the need for a damage-aware working policy in caches.

2) Varying cache sizes: This case differs from the previous in that now we set the number of files to a constant (800) and vary the cache size in the interval 50 to 500 in steps of 50 (averaging over 500 iterations).

Figure 2 shows that as the cache size increases, the fraction of cache misses decreases, as expected. However, the damage first increases and then decreases with increasing cache size, and this transition happens at different values of cache size for different policies. While this may seem counterintuitive at first, it is reasonable because we observe that a bigger cache gives the provision of writing more files in cache (hence lower total cache misses) each with a lower retention time thus causing less damage. This behavior is particularly true for the offline caching when the request string is *known* to all policies and may not be true in general. Finally, we note

that OPT_{off}, while achieving lesser damage than FiF, gives *2-15 fold* less damage compared to LRU, FIFO and RND.

We next generate a second set of file request string using a Poisson process but modulated with a ZipF popularity distribution [15] with exponent $\alpha = 1.2$ (see graphs in Appendix [20]); our results exhibit similar trend as linear popularity but with higher damage reductions (*30-80 fold*). We observe behaviour with varying α to conclude that OPT_{off} shows high damage reductions if the file popularities get more skewed which is known to be true for trace-driven data [12].

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 cache misses given by the FiF policy. The case of offline caching lend insights to motivate the online caching problem where the arrival requests are not know apriori. This we discuss next.

III. Flash-aware Optimal Online Caching for Large Caches

We approach the online caching problem in two stages. In this Section, we formulate the optimization problem to find optimal retention times for *large caches* (caches with no capacity constraint) wherein we minimize cache damage subject to a constraint on network delay. A large cache assumption implies that there are no evictions and the cache misses are only because of the files getting expired. In Section IV, we extend the results obtained here to analyze a finite cache where we obtain optimal eviction sequence for a cache miss when the cache is full. Note that, the problem of jointly optimizing over all possible retention times and eviction decisions remains an open problem.

We first state the system model for the online caching which applies to Sections III-IV.

A. System Model

1) Traffic model: File request string is i.i.d. where the requests arrive according to the Independent Reference Model (IRM) [14], [15] which assumes that: (1) all requests are for a fixed collection of M files; (2) the probability that file i is requested is p_i which is static (does not vary with time) and independent of past or future requests.

We assume that the interarrival times of files are exponentially distributed and the arrival process for files conform to an independent and homogeneous Poisson process. Let $\{X_n(m)\}$ denote the i.i.d exponential interarrival time ($X(m) \sim exp(\lambda_m)$) between the n^{th} and $n + 1^{th}$ arrival of a file m. Under IRM, the probability of requesting file m arriving with rate λ_m is given by, $p_m = \frac{\lambda_m}{\sum_{j=1}^{M} \lambda_j}, \ \forall m$.

2) Cost of fetching and writing a file: The total cost of fetching and writing file m upon a cache miss consists of delay cost $\delta(m)$ and retention (writing) cost $f(m)$. The total cost per miss on file m is denoted by $c(m) = \delta(m) + f(m)$.

- Delay cost, $\delta(m)$: For every cache miss, fetching the requested file from the server results in a known, determinimistic delay cost which can be thought of as the transmission delay to obtain the file from the server based on the time of the day, current server workload, or available channel bandwidth, etc.
- Retention cost, $f(m)$: Retention cost is incurred due to flash memory damage. While there are only empirical relationships known about flash memory damage as a function of memory retention times, we outline two desirable properties for constructing a suitable damage function: (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 [6]–[8]; (2) Damage function, f, is a complicated, non-linear function with $f(0) = 0$. Based on these properties, we choose a *convex increasing polynomial* as a damage function satisfying both (1)-(2); the various coefficients of the polynomial serve as a handle in adjusting the slope of the curve in various regions. In the event of a cache miss on a file, a one-shot retention cost is incurred (see Definition 1). The cumulative *retention cost* is defined as the sum of all one-shot retention costs.

  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, i.e., $f(Z) \in \mathbb{R}^+$ where f 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 $a_0 = 0$.*

The retention time for file m is assumed to be distributed as an exponential random variable $\mathcal{R}(m)$ with parameter μ_m, $m = 1, 2, \ldots, M$ to keep the problem tractable. Moreover, a random retention time captures that writing a file in memory with a precise retention of R leaves a non-zero probabilty of finding the file in cache after R.

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 [21]. 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 it is well known that reading the disk is order of magnitudes faster than writing and erasing [21].

After system model, we formulate the problem next.

B. Problem formulation for finding optimal retention times

The optimal online policy for large caches is defined as:

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 Δ.*

To find the optimal online policy, we start with obtaining 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 the i.i.d. exponential

retention times ($\sim exp(\mu)$) of arrivals $n = 1, 2, \ldots$ for the single file. Let I_n be the indicator variable defined as follows:

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

Note that $I_n = 1$ corresponds to the event $X_n > \mathcal{R}_n$. Thus, $\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} I_n = \mathbb{P}(X_n > \mathcal{R}_n) = p_{miss} = \frac{\mu}{\lambda+\mu}$. Similarly, the probability of a cache hit is, $p_{hit} = \frac{\lambda}{\lambda+\mu}$. 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 the file otherwise. Thus, the *expected damage*, D, can be expressed as:

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

which is true since I_n is independent of the retention time $\mathcal{R}_{n+1}$. Similarly the expected delay constraint can be expressed as $p_{miss} \times \delta \leq \Delta$. When $\mathcal{R}_{n+1} \sim exp(\mu)$ and $f(x) = x$, then $\mathbb{E}_{\mathcal{R}}[\mathcal{R}_{n+1}] = 1/\mu$, which is independent of n. Thus, for a single file, the goal is to minimize $p_{miss} \times \frac{1}{\mu}$ subject to the constraint that $p_{miss} \times \delta \leq \Delta$.

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

$$\underset{\mu_m \in \boldsymbol{\mu}}{\text{minimize}} \sum_{m=1}^{M} p_{miss}(m) \times p_m \times \mathbb{E}_{\mathcal{R}} \left[f(\mathcal{R}(m)) \right] \quad (3a)$$

$$\text{subject to } \sum_{m=1}^{M} p_{miss}(m) \times p_m \times \delta(m) \leq \Delta \quad (3b)$$

Define $q_m := \lambda_m/(\mu_m + \lambda_m)$, 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 defined in (1)) in the objective of formulation (3). The objective becomes:

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

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

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

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

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

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

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

Thus, the final formulation from the objective in (6) and the constraint in (7a) becomes:

Damage Formulation

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

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

$$0 \leq q_m \leq 1, \forall m \quad (7c)$$

The constraint in formulation (7) poses upper and lower bounds on the value of q_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 proof in the Appendix of [20]). We use a MATLAB convex program solver (with $q_m = \lambda_m / \sum_i \lambda_i$ as the initial values) to solve (7) and report the results in Figure 3.

C. Simulations: Trade-off with various damage functions

We study the delay-damage trade-offs obtained from using three kinds of polynomial damage functions (linear, quadratic and cubic) on Poisson arrivals modulated with ZipF popularities ($\lambda_m = 1/m^\alpha$, $\alpha = 1.2$). We assume a unit delay for fetching files ($\delta(m) = 1, \forall m$). Note that, with a unit delay we have $\Delta = \epsilon$, where ϵ denotes the cache-miss fraction. 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 more room for writing files with lower retention values thus incurring less damage. We also observe that the damage increases with increasing number of files for the same value of ϵ. Also, as the number of files increase, the value of the damage quickly decreases.

Remark: 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) [9], [11] and is thus not considered in this work. Our work is concerned with finding optimal caching policies *given* any polynomial damage function.

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

In this Section, we use the same model as defined in Section III with the only difference that now the cache size is finite, denoted by B. Files are written in the cache with the optimal retention times obtained in Section III. We aim to obtain the optimal file to evict on every cache miss when the cache is full using only the knowledge of the past requests and cache contents. We formulate and solve the problem of finding optimal eviction sequence as a sequential decision making problem using the Markov Decision Process (MDP).

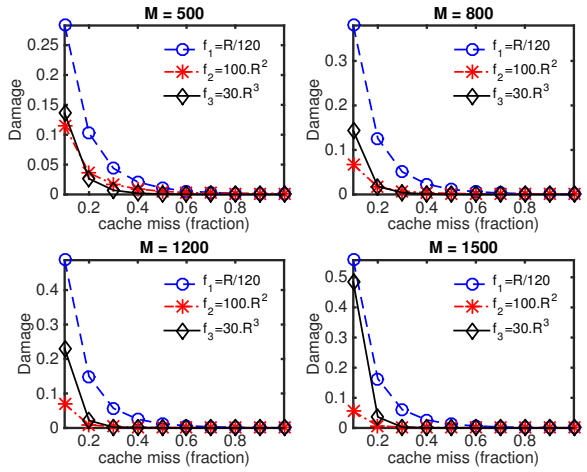

Fig. 3. The figure shows the objective function values for poisson arrivals modulated with ZipF $\alpha = 1.2$ when plotted against increasing (allowed) fraction of cache misses, ϵ, for a unit delay.

We assume that the evictions are *optional*, i.e., the policy may not evict a stored file upon a cache miss (in which case we say that the requested file itself is instantaneously evicted). Files leave the cache either because they were evicted or because their retention time expired. A file whose retention time expires is said to *depart* from the cache.

Our work is a significant generalization of [14] 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 [14] 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: note that the minimization is performed over different file sets in both the cases, hence the optimal solutions in [14] is not a solution to our problem and vice versa. Moreover, adding retention times to 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 to obtain a discrete time Markov chain (DTMC) from the continuous Markov process. Let $n = 1, 2, \dots N$ denote the time indices corresponding to the state transitions marked by file arrivals and file departures. Let $\mathbf{S}(n)$ be a state in the Markov Chain denoted by a 3-tuple, $\mathbf{S}(n) = \{S(n), R(n), D(n)\}$, where $S(n)$ is the set of files in the cache, $R(n)$ denotes the file requested at time n and $D(n)$ is the first file departing at time n. Note that the time distribution of the first file to depart at the n^{th} transition will be distributed as the sum of the i.i.d. exponentials for all files each with parameter μ_j (as obtained in Section III).

We assume that a transition is either due to a file arrival or a file departure and *not* both. For a file arrival, $D(n) := 0$ and for a file departure, $R(n) := 0$. Thus, the states of the MDP are of the form $\{S(n), R(n), 0\}$ or $\{S(n), 0, D(n)\}$.

We explain the cache state transitions as follows. When file $D(n)$ departs from the cache $S(n)$, the cache becomes $S(n) - D(n)$. If there is a file arrival which results in a cache hit (i.e., $R(n) \in S(n)$) then the cache content at time $n+1$ is the same as that at time n (i.e., $S(n+1) = S(n)$). 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., $S(n+1) = S(n) + R(n)$; (2) If the cache is full, then, the state at time $n+1$ is $S(n) + R(n) - U(n)$ where $U(n), U(n) \in S(n) + R(n)$ is the random variable denoting the file evicted on n^{th} arrival on a full cache. Formally,

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

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

2) Markovian Policy: It is easy to see the state $\mathbf{S}(n+1)$ only depends on state $\mathbf{S}(n)$ and $U(n)$. 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 $\pi = \{\pi_1, \pi_2, \pi_3, \dots\}$, where each π_n is a mapping from state $\mathbf{S}(n)$ to the evicted file in $\{0, 1, \dots, M\}$, *i.e.*, $U(n) = \pi_n(\mathbf{S}(n))$. We define $U(n) := 0$ when: (1) no eviction decision needs to be made (i.e., $R(n) \in S(n)$); (2) there is a cache miss and $U(n)$ refers to a file not present in cache or request (i.e., $R(n) \notin S(n)$ and $U(n) \notin S(n) + R(n)$). Let $\pi_n(u, \mathbf{S}(n))$ be the probability that policy π evicts file u in state $\mathbf{S}(n)$ on n^{th} arrival, where $u \in \mathcal{M}$, then $\pi_n(u, \mathbf{S}(n))$ satisfies the following properties:

$$\sum_{u=0}^{u=M} \pi_n(u, \mathbf{S}(n)) = 1,$$
$$\pi_n(u, \mathbf{S}(n)) = 0 \ \ \forall u > 0 \text{ if } R(n) \in S(n),$$
$$\pi_n(u, \mathbf{S}(n)) = 0 \ \ \forall u : u \notin S(n) + R(n), R(n) \notin S(n),$$

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 $\hat{p}_r = \lambda_r / \sum_{m=1}^{M} (\lambda_m + \mu_m)$ and due to a departure of file d is $\tilde{p}_d = \mu_d / \sum_{m=1}^{M} (\lambda_m + \mu_m)$. 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 $\mathbf{1}[.]$ be the indicator function then we derive the state transition probabilities as follows:

$$\mathbf{P}_\pi[U(n) = u | \mathbf{S}(n)] = \pi_n(u, \mathbf{S}(n)), u = 0, 1, \dots, M \quad (9)$$

$$\begin{aligned} &\mathbf{P}_\pi[S(n+1) = \tilde{S}, R(n+1) = r, D(n+1) = 0 | \mathbf{S}(n), U(n)] \\ &= \hat{p}_r \times \mathbf{P}_\pi[S(n+1) = \tilde{S} | \mathbf{S}(n), U(n)] \\ &= \hat{p}_r \times \mathbf{1}[T(\mathbf{S}(n), U(n)) = \tilde{S}] \end{aligned} \quad (10)$$

$$\mathbf{P}_\pi[S(n+1)=\tilde{S}, R(n+1)=0, D(n+1)=d|\mathbf{S}(n)]$$
$$=\tilde{p}_d \times \mathbf{P}_\pi[S(n+1)=\tilde{S}|\mathbf{S}(n)] \quad (11)$$

Equations (9)-(10) follow since IRM file arrivals are independent of the state of the cache and the time of the request. Equations (10)-(11) apply for every $(\tilde{S}, r, d) \in \mathbf{S}(\mathbf{n}+\mathbf{1})$.

4) Cost function: A one-shot cost $c(m)$ for file m (as in Section III) is incurred on every cache miss. The expected cost for the horizon of length N under the policy π becomes:

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

The average cost over the horizon of N discrete time steps under policy π is given by, $J_c(\pi) = \limsup_{N\to\infty} \frac{\sum_{m=1}^{M}(\lambda_m+\mu_m)}{N+1} J_c(\pi, N)$. 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.

B. The Optimal Eviction Policy

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

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

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

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:

$$\begin{aligned} V_{N+1}(S, r, 0) &= \mathbf{1}_{\{r\in S\}}\mathbb{E}_{R^*}[V_N(S, R^*, 0)] \\ &+ \mathbf{1}_{\{r\notin S, |S|<B\}}\left(c(r) + \mathbb{E}_{R^*}[V_N(S+r, R^*, 0)]\right) \\ &+ \mathbf{1}_{\{r\notin S, |S|=B\}} \\ &\left(c(r) + \min_{u\in S+r} \mathbb{E}_{R^*}[V_N(S+r-u, R^*, 0)]\right), \\ &+ \mathbf{1}_{\{r\in S\}}\mathbb{E}_{D^*}[V_N(S, 0, D^*)] \\ &+ \mathbf{1}_{\{r\notin S, |S|<B\}}\left(c(r) + \mathbb{E}_{D^*}[V_N(S+r, 0, D^*)]\right) \\ &+ \mathbf{1}_{\{r\notin S, |S|=B\}} \\ &\left(c(r) + \min_{u\in S+r} \mathbb{E}_{D^*}[V_N(S+r-u, 0, D^*)]\right) \end{aligned} \quad (12)$$

$$\begin{aligned} V_{N+1}(S, 0, d) &= \mathbb{E}_{R^*}(V_N(S-d, R^*, 0)) \\ &+ \mathbb{E}_{D^*}(V_N(S-d, 0, D^*)) \end{aligned} \quad (13)$$

We explain the different terms in (12) as follows:

- $V_{N+1}(S, r, 0)$ is the value of the objective when optimal action is taken in the state $S, r, 0$ at time $n = 0$ to minimize the cost over the horizon $[0, N+1]$. 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 $[0, N]$ due to a file request R^* (or a departure D^*) at time $n = 0$ is given by $\mathbb{E}_{R^*}[V_N(S, R^*, 0)]$ (or $\mathbb{E}_{D^*}[V_N(S, 0, D^*)]$).
- The second and fifth terms differ from the above in that the file r requested at $n = 0$ 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 $[0, N]$ by $c(r)$.
- The third and sixth terms capture the case when the cache is full at $n = 0$ and there is a cache miss upon request thus leading to a file eviction. The expected cost over the horizon $[0, N]$ is thus obtained by minimizing over all possible evictions, i.e., $u \in S + r$.

Equation 13 represents file d departing from the cache at time $n = 0$. Here no cost is incurred over the horizon $[0, N+1]$ 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 $n = 1$.

By inspecting carefully, we can see that in the Dynamic Programming Equation (12)-(13), we *only* need to optimize the term: $\min_{u\in S+r} \mathbb{E}_{R^*}[V_N(S+r-u, R^*, 0)]$

The optimal eviction policy that achieves the above turns out to be the policy given in the following Theorem 2:

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

$$v = \arg\min_{u\in S+r} \{p(u)c(u)\} \quad (14)$$

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

Proof: Due to space constraints we provide a sketch of the proof; see the full version in the Appendix of [20]. The proof proceeds by induction on $N = 0, 1, \ldots$. Let $v := \arg\min\{u \in S + r : p(u)c(u)\}$, which implies, $\mathbb{E}[V_N(S + r - v, R^*, 0)] = \min_{u\in S+r} \mathbb{E}[V_N(S+r-u, R^*, 0)]$. At each step of induction, we want to show that

$$\mathbb{E}[V_N(S+r-u, R^*, 0)] - \mathbb{E}[V_N(S+r-v, R^*, 0)] \geq 0,$$

For the basis step, after a few algebraic manipulations, we show that $\mathbb{E}[V_0(S+r-u, R^*, 0)] - \mathbb{E}[V_0(S+r-v, R^*, 0)] = 2(p(u)c(u) - p(v)c(v))$ which is ≥ 0 by the definition of v.

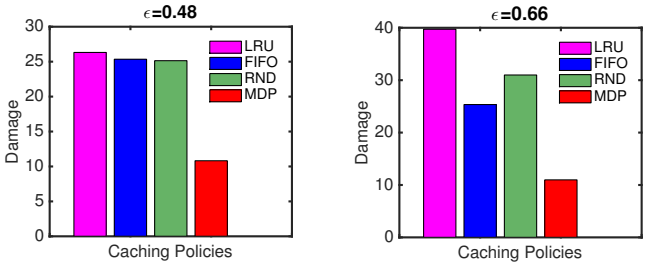

Fig. 4. Damage values for LRU, FIFO, RND and MDP with Poisson arrivals: (1) linear popularity, $\epsilon = 0.48$, (2) ZipF popularity, $\epsilon = 0.66$.

For the induction step at $N+1$, we assume that the claim is true for N, i.e. $\mathbb{E}[V_N(S+r-u,R^*,0)] - \mathbb{E}[V_N(S+r-v,R^*,0)] \geq 0$. The analysis of the induction step turns out to be significantly involved; for the detailed proof, see Appendix of the full-version in [20]. Let $(1-p_C)$ denote the steady-state probability that the cache is full, then the final step of induction requires:

$$p(v)\mathbb{E}[V_N(S+r-u,R^{**},0)]- \\ [(1-p_C)(p(v)-p(u))-p(u)]\mathbb{E}[V_N(S+r-v,R^{**},0)] \geq 0$$

which is true if the inequality $p(v) \geq (1-p_C)(p(v)-p(u))-p(u))$ implies that $2p(u) \geq p_C(p(u)-p(v))$. It does, since $2p(u) = p(u)+p(u) \geq p(u)-p(v) \geq p_C(p(u)-p(v))$. Also, we have $\mathbb{E}[V_N(S+r-u,R^*,0)]-\mathbb{E}[V_N(S+r-v,R^*,0)] \geq 0$ by induction. Hence, the claim.

Theorem 2 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.

C. Simulations

Recall that the caching policies (LRU, FIFO and RND) *assume* that the files are written in 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 two data-sets with Poisson arrivals, one with linear popularity ($\lambda_m = m$) and other with ZipF popularity ($\lambda_m = 1/m^\alpha$) with $\alpha = 1.2$. We consider unit delay and fix a value for ϵ in (7). Further, we simulate the MDP-policy by writing files in cache for the optimal retention time computed from the solution of (7) with cost $c(m) = f(m)$ for the chosen value of ϵ. Upon a cache miss, we evict 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, the MDP-policy outperforms other policies by giving a *2-4 fold* damage reduction, thus agreeing with the analytical result (derived in Theorem 2). Moreover, we note that LRU and RND, which are being actively considered for deploying in CCN caches, perform very poor in terms of damage.

V. 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.

One possible direction for future work is to consider temporal correlations in file request arrivals. Another direction is to extend the problem to a network of caches. Finally, it is an open problem to devise a framework that jointly optimizes over both retention time durations (i.e., all possible distributions) and eviction sequences.

Acknowledgements

This material is based upon work partially supported by the National Science Foundation under Grant No. 1456887 and 1422153. We thank Professor Tong Zhang for useful discussions on flash memory.

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*.

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

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

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

[8] 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.

[9] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing nand flash-based ssds via retention relaxation," *Target*.

[10] 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.

[11] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System performance for Enterprise Applications," in *IEEE HPCA*, 2012.

[12] 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.

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

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

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

[16] 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*.

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

[18] 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*.

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

[20] S. Shukla and A. Abouzeid, "On Designing Optimal Memory Damage Aware Caching Policies for Content-Centric Networks," https://www.ecse.rpi.edu/homepages/abouzeid/preprints/damage-aware-caching.pdf, 2015.

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