Model-based Design and Analysis of Cache Hierarchies

Amr Rizk
Technische Universität Darmstadt
amr.rizk@kom.tu-darmstadt.de

Michael Zink
University of Massachusetts, Amherst
zink@ecs.umass.edu

Ramesh Sitaraman
University of Massachusetts, Amherst
ramesh@cs.umass.edu

***Abstract*—The caching of web and video content is a key component of today's Internet. Caches are often organized as hierarchies where one cache can retrieve and store content from other caches. Cache hierarchies have three objectives: decreasing the traffic served by the content provider's origin (called origin offload), decreasing the cost of transporting content over the WAN (called midgress reduction), and providing a faster response time for users downloading the content. While there is a large body of research on individual caches, much less is known about cache hierarchies. To gain insights into cache hierarchies, we develop an analytical model that can be applied to arbitrary cache hierarchies described by a directed acyclic graph. We show that our model predicts hit rates and other metrics for upper layer caches within cache hierarchies significantly better than prior work. We use our model to shed light on fundamental questions of cache hierarchy design such as how to size individual caches and how to store and route content in a cache hierarchy. Using real-world CDN traces and implementable cache eviction policies, we provide insights that are applicable to real-world cache hierarchy designs.**

I. Introduction

Caching plays an important role in today's Internet. Many of the most popular applications like video streaming, the provisioning of web content, and social networks heavily depend on caching. Very popular applications and services need to be based on cache hierarchies to make them scalable. One example is video streaming where every major provider (e.g., Netflix, Hulu, YouTube) makes either use of their own cache hierarchy or uses one offered by Content Distribution Network (CDN) providers like Akamai or Limelight. In addition, the increasing popularity of Information Centric Networks (ICN) [1], [27] also motivates the investigation of the performance of cache hierarchies. In ICN, routers serve as caches, and based on the topology of the network and the routing policies, quite complex cache hierarchies can be constructed.

Cache hierarchies involve three major players. *Content providers* offer content on *origin servers*. The content providers employ a *CDN* who operates the cache hierarchy that consists of a network of caches, including *edge caches* that serve users and *parent caches* that serve other edge caches. A request from a user is first routed to the closest edge cache. If the edge cache does not contain the requested content, the request is routed to one or more parent caches. If the content is not found in the parent caches either, the request is routed to the origin that serves the content. Content follows the reverse path of the request and is often cached by all the caches on the way back.

Cache hierarchies are designed with three goals in mind: *i)* the *performance* experienced by users must be enhanced by serving the requested content from an edge cache or from a parent cache as "close" to the user as possible; *ii) Midgress* traffic, which is the traffic between parent or edge caches within the hierarchy, must be reduced, since it represents an operational overhead for the CDN; *(iii)* Traffic sent between the origin and the cache hierarchy must be reduced, as it incurs cost for the content providers. The reduction in origin traffic due to the cache hierarchy is referred to as *origin offload*.

While there is a large body of work on the analysis of single caches, there is much less work on the analysis of cache hierarchies. Although cache hierarchies have been in use for a few decades there are significant challenges in the design and operation of these systems. In current practice, cache hierarchies are designed in an ad-hoc fashion, since there is a lack of models that allow for an analysis of such hierarchies. In this paper, we derive a model for cache hierarchies that allows for their analysis. The goal of our work is to provide designers and operators with tools that will allow them to evaluate the performance of alternative cache hierarchies. The models we present in this paper are evaluated by a series of simulations using both synthetic and real-world CDN traces. In this paper we make the following contributions:

- **Model:** We introduce a new model for hit rate calculation in cache hierarchies that builds on the work of Che et al. [4] and Martina et al. [12], [16] by applying an *iterative* approach. We show that our approach models the hit rate and other metrics for cache hierarchy significantly better than prior work.
- **Analyzing Cache Hierarchies:** Based on the improved model we analyze cache hierarchies and how midgress, origin offloading, and performance vary with cache size. We show that *i)* single cache hit rates always have diminishing returns but that is not generally true for cache hierarchies where not all caches have diminishing returns; *ii)* the midgress of a hierarchy increases with the imbalance of the edge cache sizes for uniform requests while the origin traffic stays relatively unchanged; and *iii)* performance in terms of average hop-to-hit or average response time has diminishing returns with cache size.
- **Content request forwarding:** We analyze different approaches for content request forwarding and show that *i)* geo(graphic) splitting of requests does worse than object-

This work is supported in part by NSF grant CNS-1413998 and by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 – MAKI.

ISBN 978-3-901882-94-4

based splitting for midgress and origin offloading; *ii)* object-based splitting behaves like a single unified cache; *iii)* geo-splitting does better than object-based splitting when hop-to-hit count is taken into consideration; and *iv)* hybrid splitting strikes a balance between midgress reduction and hop-to-hit count performance.

II. Background on Cache Hierarchies

A. Cache Hierarchy

We start with a general description of cache networks and introduce terms that are specific for this work. Our description follows common descriptions of such networks as, e.g., given in [21], [22]. We model a cache hierarchy as a directed acyclic graphs (DAG) $G = (V, E)$ with $V = \{v_1, ..., v_n\}$, and $E \subseteq V \times V$. Each node of the DAG represents an origin, parent, or edge cache. Edge (k, v) represents the fact that v can download content from k. Note that DAGs can be topologically sorted into levels. Let L be the number of levels, and H_l be the number of caches on level l. We can represent vertices in the DAG by a tuple $\langle l, u\rangle$ where the first subscript $l \in \{1, \ldots, L\}$ denotes the level while the second subscript $u \in \{1, \ldots, H_l\}$ is the cache index. The cache size of node $\langle l, u\rangle \in V$ is denoted $C_{l,u}$ and the total cache budget of a cache hierarchy is expressed as $C_{tot} = \sum_{l=1}^{L} \sum_{u=1}^{H_l} C_{l,u}$.

Each DAG contains a set S of one or more origin servers on which all N equal sized objects $O = \{o_1, ..., o_N\}$ are stored. Besides the origin servers, the remaining caches in V are divided in two subgroups: edge caches and parent caches. Edge caches can download from parent caches or origins, but do not feed other caches, i.e., there are no outgoing edges from an edge cache. Parent caches download from other parent caches or origins. Origins store original versions of the content and so have no incoming edges in the DAG. Further, we associate a value with each edge that describes the (average) latency between two caches.

Figures 1a – 1d show a set of typical cache hierarchies as they exist in today's CDN networks. The routing policy is expressed by the adjacency matrix of the hierarchy. In case of request stream splitting as in Sect.V-C the entries of the adjacency matrix encode the splitting algorithm, i.e., in terms of probabilities, in case of random splitting or in terms of ordered percentiles of the popularity distribution, in case of popularity based splitting. When a request for object o_i arrives at cache v, a *hit* is generated if the object is stored on the cache, otherwise it is a *miss* and the request is forwarded according to the routing policy. In Sect. V-C, we show how our model captures that clients, edge, and parent caches may belong to different geographical **regions** $R = r_1, ...r_n$. The region concept is important in cache networks, since it impacts the performance at the client and the midgress.[1]

B. Cache Replacement Strategies

Since the storage capacity of an individual cache is usually much smaller than the combined size of all objects in a universe U, it fills up quickly. In case of a cache miss it *a)* has to be decided if the requested object should be cached to serve further requests, and *b)* if so, which of the currently stored objects should be removed to free the space to cache the new object. The first part is described as **admission strategy**, while the second part is described as **eviction strategy**. The admission strategy decides if a requested object that is currently not cached should be cached or if the request should be passed on to the next level cache without caching the object locally. Caching every object that is requested by a client is not necessarily the best approach since popularity distributions of content like video have a long-tail (see Sect. II-D) and caching so-called "one-hit-wonders" [15] will not generate cache hits.

In the literature there exists a large body of work on cache eviction strategies [18] with FIFO, LFU, and LRU being the most prominent ones. Here, we briefly introduce LRU, since it is the strategy which we will use throughout this paper. In the case of LRU, the object that has been requested the longest time ago will be evicted, which makes the algorithm easy to implement and cheap in terms of computational overhead.

C. Che's Approximation

Next, we describe Che's approximation which is a decoupling technique that enables the analysis of caches implementing different caching replacement strategies such as LRU [4], FIFO and Random replacement [16]. The main idea of this approximation [4] lies in the notion of a constant characteristic time T_c that describes the time needed to evict an arbitrary object that has just been requested. This time T_c is assumed to be constant and independent of the object index which was shown to be asymptotically true under fairly general conditions [11]. Hence, given Che's approximation we find that for N distinct objects arriving at a cache of size C according to Poisson processes with rates λ_i where $i \in \{1, \ldots, N\}$, the probability of an object i to be in the cache at a certain time t equals the probability of having at least one prior request in the time span $(t - T_c, t]$. For Poisson arrivals this is simply $p_{o,i} = 1 - e^{-\lambda_i T_c}$, which also equals the object hit probability p_i. In $p_{o,i}$ the first subscript stands for occupancy, while the second stands for the object index. Note that the popularity of object i is given as $\hat{\lambda}_i = \frac{\lambda_i}{\sum_{j=1}^{N} \lambda_j}$ with $\sum_{i=1}^{N} \hat{\lambda}_i = 1$. Now to determine the characteristic time T_c the approximation enforces that the sum of the average object occupancies equals the cache size C, which is expressed as

$$\sum_{i=1}^{N} \left(1 - e^{-\lambda_i T_c}\right) = C. \tag{1}$$

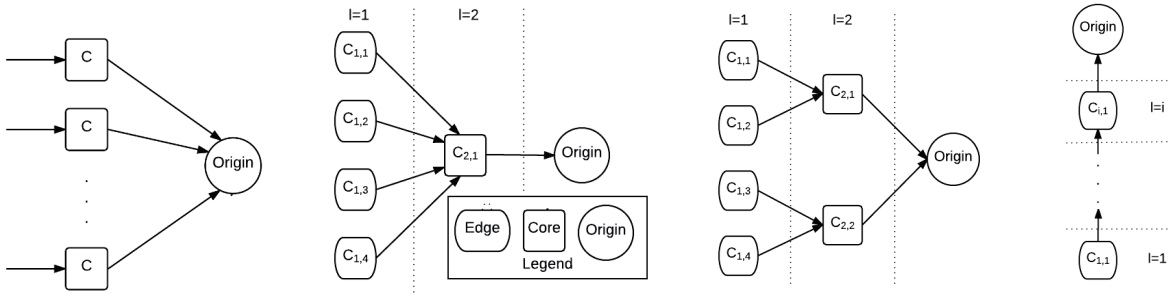
Equation (1) relates the probability of an object to be in cache (occupancy) to the cache size C in the sense of expectation. It is justified in [11] for different object popularity distributions such as the often observed Zipf distribution. Hence, to obtain the average hit rate of an edge cache, one only needs to weight the object hit probabilities correspondingly [4], i.e.,

$$p_{1,1} = \sum_{i=1}^{N} \hat{\lambda}_i \left(1 - e^{-\lambda_i T_c}\right). \tag{2}$$

D. CDN Trace

To analyze the quality of our proposed model, we performed trace-based simulations where we used a data set (see [13] for further details) collected in June 2014 from Akamai which operates one of the world's largest CDNs. This

[1] Midgress is the traffic between parent and/or edge caches within the cache hierarchy., i.e., it is traffic that is neither sent to a user or sent from an origin. From a CDN perspective, midgress is purely an overhead to be minimized.



(a) One-level (b) Tree (c) Binary Tree (d) Line

Fig. 1: One-level (1a), tree (1b), binary tree (1c) and line (1d) hierarchies.

anonymized dataset was collected from a large cross-section of actual users around the world. The dataset includes unique video object names and lengths, and identifies geographical regions from where the requests originate. Overall, the dataset contains traces from 5 million video sessions originating from 200 thousand unique clients who were served by 1294 video servers from around the world. We consider a 24 hour excerpt from this trace that includes requests for more than 70 thousand distinct videos. The analysis of this dataset reveals an approximately Zipf-distributed video popularity, i.e., $\hat{\lambda}_i$ in the model, in addition to a significant portion of objects that are requested only once (so-called "one-hit-wonders").

III. Modelling and Analysis of Cache Hierarchies

A. Basic approach

We consider cache hierarchies represented as DAGs as exemplified in Fig. 1 with no traffic replication. If not mentioned otherwise, all caches apply LRU and misses are forwarded to parent caches triggering replacements at every cache which produces a miss (i.e., leave copy everywhere). We observe that the following approximation by Martina et al. [16] significantly improves the object hit rate estimates in parent caches in comparison to [4]. This approximation is based on tracking an object hit at a parent cache back to a corresponding arrival. Consider for simplicity Fig. 1d and assume a hit of object i at parent cache $C_{2,1}$ at time t: Object i is in cache $C_{2,1}$ at time t if a request arrives at cache $C_{1,1}$ in $(t-T_{c,2}, t-T_{c,1}]$ since requests arriving in $(t-T_{c,1}, t]$ get filtered by the edge cache with characteristic time $T_{c,1}$. This approximation assumes $T_{c,2} > T_{c,1}$, implying monotonically increasing cache sizes along the path. Here, it is also assumed that the arrivals to the second cache are characterized by a Poisson process, which generally does not hold as this depends on the state of the first cache and resembles in general an ON-OFF type of process. However, this assumption helps determine the characteristic time $T_{c,2}$ of the parent cache using a parameterized version of (1). Together with this approximation, the object hit rate at cache $C_{2,1}$ is given in [16] as

$$p_{i,2,1} = \left(1 - e^{-\lambda_{i,2}(T_{c,2}-T_{c,1})}\right) \mathbb{1}_{\{T_{c,2}>T_{c,1}\}} \tag{3}$$

where $\mathbb{1}_x$ is the indicator function and $\lambda_{i,2}$ is the Poisson rate of object i within the input stream to the parent cache $C_{2,1}$. Now, we turn to our model where we use, first, the notion of the object miss rate $m_{i,l,u} = 1 - p_{i,l,u}$ at an individual cache $C_{l,u}$, and second, the object miss rate $m_{i,l} = f(m_{i,l,u})$ at a cache level l. A good approximation for the overall system hit rate P_{sys} that consist of L cache levels is given by

$$p_{sys} = 1 - \sum_{i=1}^{N} \hat{\lambda}_i \prod_{l=1}^{L} m_{i,l}. \tag{4}$$

On each level we calculate the average object miss rate as

$$m_{i,l} = \sum_{u=1}^{H_l} m_{i,l,u} \frac{\sum_i \hat{\lambda}_i m_{i,l,u}}{\sum_u \sum_i \hat{\lambda}_i m_{i,l,u}}. \tag{5}$$

Note that (5) differs in two ways from the literature, e.g. [12]: First, the resulting object hit rate probability at an intermediate cache is different from the formulation in [12]. Second, and most importantly, considering cache hierarchies that are mapped to directed acyclic graphs allows us to substitute a layer of caches feeding into one parent cache by an equivalent system with corresponding approximate object miss rates $m_{i,l}$. In Sect. V, we will show two implications of cache and input request heterogeneity on the hierarchy design using this formulation. Next, we will present an iterative method to approximate individual cache hit rates in a hierarchy.

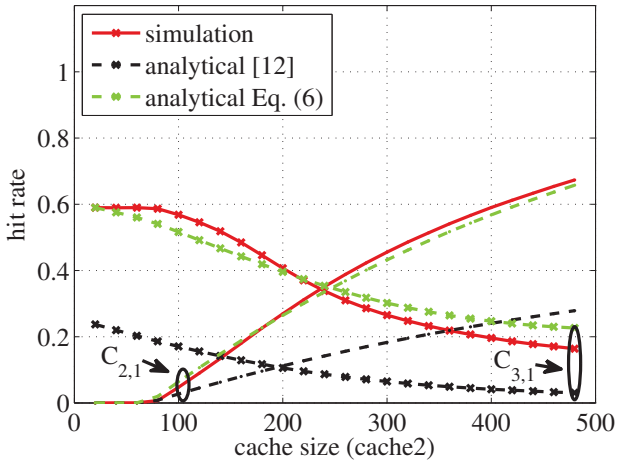
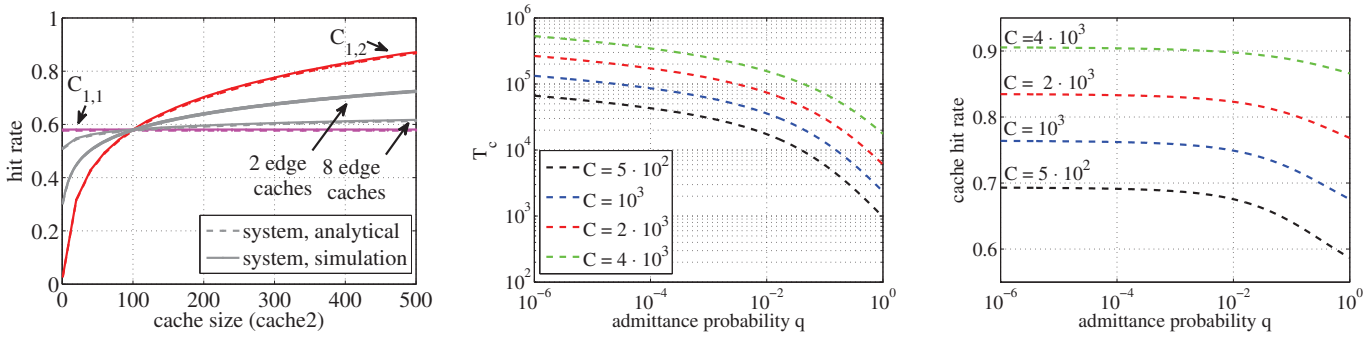


Fig. 2: Individual cache hit rates for a cache line of $L = 3$ LRU caches. Fixed edge cache size $C_{1,1} = 100$ objects. Parent caches possess a cache budget of $C^{\Sigma} = C_{2,1} + C_{3,1} = 500$ objects. Equation (6) provides better parent cache hit rates estimates than using [12].

B. Iterative cache hit rate calculation

Starting from the system hit rate formulation in (5) and the approximation in Sect. II-C we provide accurate cache hit rate



(a) Diminishing return of increased edge cache size (b) Characteristic time T_c vs. q (c) Diminishing return of q

Fig. 3: (3a) Diminishing returns for LRU caches: Hit rate behavior of an entire edge level consisting of 2 or 8 LRU caches. (3b)-(3c) Diminishing returns for probabilistic LRU: Nonlinear impact of changing the admittance probability q on the characteristic time (3b) and the cache hit rate (3c). $N = 10^4$ objects.

estimates for the individual caches of a considered hierarchy. We estimate the cache hit rate at cache level l as

$$\begin{aligned}\hat{p}_l &= 1 - \frac{1-\hat{p}_{sys,l}}{1-\hat{p}_{sys,l-1}},\\ &= 1 - \frac{\sum_{i=1}^{N}\hat{\lambda}_i\prod_{k=1}^{l} m_{i,k}}{\sum_{i=1}^{N}\hat{\lambda}_i\prod_{k=1}^{l-1} m_{i,k}},\end{aligned} \tag{6}$$

where $\hat{p}_{sys,l}$ denotes an estimated system hit rate of an equivalent hierarchy consisting of the first l levels. We illustrate the applicability of the method using the following example: Consider a $L = 3$ level cache line hierarchy as in Fig. 1d. We estimate the cache hit rate at the cache of level 2 as

$$\hat{p}_2 = 1 - \frac{\sum_{i=1}^{N}\hat{\lambda}_i e^{-\lambda_i T_{c,1}\left(1-e^{-\lambda_i T_{c,1}}\right)} e^{-\lambda_i T_{c,2} e^{-\lambda_i T_{c,1}}}}{\sum_{i=1}^{N}\hat{\lambda}_i e^{-\lambda_i T_{c,1}}}. \tag{7}$$

Further, for hierarchies that contain multiple caches per level such as Fig. 1b we use (5) to calculate the average object miss rates at the input of the next level cache and plug this into (7). Finally, we use this method to estimate the individual cache hit rates of parent caches to evaluate the efficiency of individual caches within a hierarchy.

Individual cache hit rate estimation: Consider a line cache hierarchy as shown in Fig. 1d with number of levels $L = 3$. We depict individual cache hit rates (for the intermediate and the parent cache) in Fig. 2 showing that the analytical approximation accurately follows the behavior observed in simulations. We consider a cache budget C^{Σ} for the upper levels, i.e., $C^{\Sigma} = C_{2,1}+C_{3,1}$, while fixing the edge cache size. While the approach from [12] models the system and first level hit rates quite well, the model shows some discrepancies for intermediate cache hit rates. Interestingly, the model from [12] approximates the *individual object hit probabilities* at the parent cache quite well. As shown in the figure the model we propose in this section provides much closer *cache hit rates* for parent caches compared to [12].

IV. Understanding Cache Mechanisms

In this section, we examine how performance metrics of hierarchies vary with allocated cache sizes. We consider the aspects of diminishing returns of cache hit rates and the impact of cache size imbalance on midgress. In addition, we validate our model with results from trace-based simulations.

A. On the diminishing returns of cache hit rates

Next, we will analytically show a diminishing return phenomenon, i.e., a diminishing impact of increased cache size on improving the cache hit rate. The results also shed light on the impact of cache size heterogeneity in cache hierarchies.

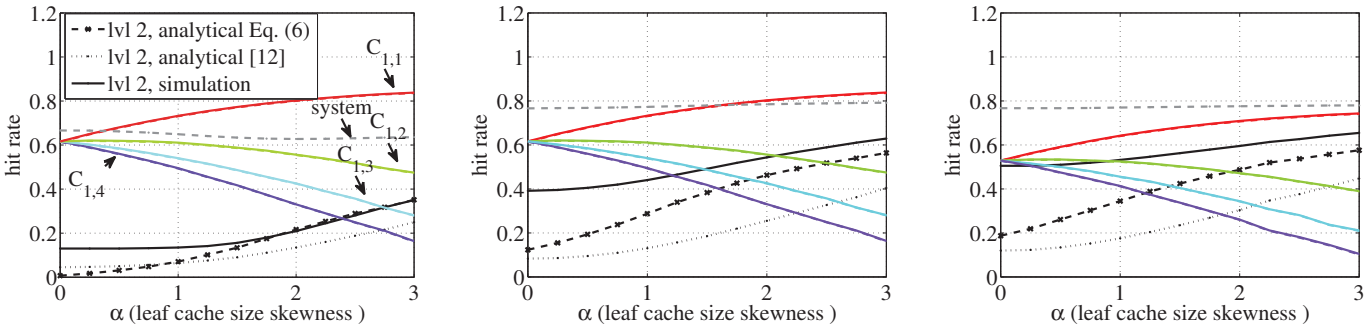
Diminishing returns at LRU caches: First, we consider a single LRU cache as regarded in Sect. II-C and drop unnecessary subscripts.

Proposition. *Given an LRU cache of variable size C with an IRM input stream containing N objects each with a popularity $\hat{\lambda}_i$ for $i \in N$. Increasing the cache size C has diminishing marginal return on the cache hit rate.*

Proof: In the following we provide a sketch of the proof. The following derivations make use of the object miss probability $m_i = 1 - p_i$ and the average cache miss probability m at cache C. From the convexity of the object miss probability $e^{-\lambda_i T_c}$ with respect to T_c we directly see the diminishing return for the sum $\sum_i \hat{\lambda}_i m_i$, e.g., in (5). In a last step, we need to characterize the relationship between the characteristic time T_c and the cache size C. Using a Taylor series expansion of the exponential function we rewrite (1) as

$$\sum_{k=1}^{\infty}(-1)^{k+1}\frac{\sum_{i=1}^{N}\lambda_i^k}{k!}T_c^k = C. \tag{8}$$

To relate T_c and the cache size C we resort to Vieta's theorem that establishes a relationship between the roots of a polynomial and its coefficients. In our case, we consider the polynomial in (8) and the coefficients $\frac{\sum_{i=1}^{N}\lambda_i^k}{k!}$. First, we make the case using a finite odd power k in (8) and conjecture that the observed analytical behavior remains unchanged for the infinite series in (8). Note that the coefficients of the polynomial decrease rapidly with $\frac{\sum_{i=1}^{N}\lambda_i^k}{k!}$ and that the derivation assumes *any* finite cut of the series expansion (8) such that we safely neglect this error in the neighborhood of the root T_c. We are interested in the behavior of the roots of the polynomial in (8) with increasing cache size C and will show in the following that this polynomial possesses only one real root T_c that is positive and increases with the cache size C as $T_c \sim C^{\gamma}$ with $\gamma > 0$. For an increasing C one can show that the discriminant of the polynomial (8) is negative such that (8) has only *one* real root. Together with the condition $\sum_i \hat{\lambda}_i = 1$ and that the



(a) $\sum_i C_{1,i} = 500$, $C_{2,1} = 100$ (b) $\sum_i C_{1,i} = 500$, $C_{2,1} = 300$ (c) $\sum_i C_{1,i} = 300$, $C_{2,1} = 300$

Fig. 4: Cache hit rates at a simple 2 level hierarchy (see Fig. 1b) with skewed level 1 cache sizes. Edge caches have a Zipfian cache size distribution with parameter α.

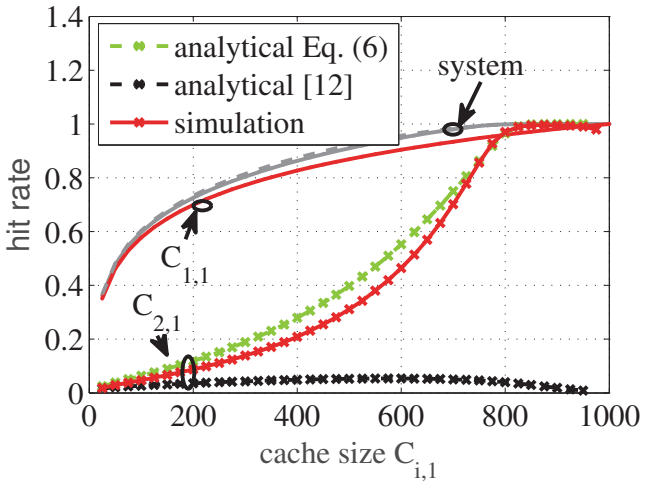


Fig. 5: Diminishing returns for edge caches and the entire system when simultaneously increasing all cache sizes $C_{i,1}$.

coefficients $\frac{\sum_{i=1}^{N}\lambda_i^k}{k!}$ are monotonically decreasing in k one can analyze the behavior of the real root and the pairs of complex roots of (8) with increasing cache size C using the system of equations provided by Vieta's theorem. After some algebra that we omit here due to space constraints we find that the real root of (8) behaves asymptotically as $T_c \sim C^{\gamma}$. Hence, together with the formulation for the average cache hit rate (2) we clearly see the diminishing returns of an increasing cache size on the cache hit rate. ■

Figure 3a shows the diminishing return within cache levels. Here, we depict the hit rate of an entire edge level of 2 (respectively 8) LRU caches, where we increase the size of one cache while keeping the rest equally constant for comparison. Note that the input request stream to all caches is homogeneous. From (5) it is evident that here too a diminishing return on cache size exists for the entire cache level. Note that the rate of diminishing returns of the entire cache level depends, however, on the number of parallel caches (plotted here for 2 and for 8 edge caches) and on the request input streams (specifically on the number of objects N in each stream). Figure 5 further shows the diminishing return on the entire system hit rate if *all* cache sizes are increased. Note that in this case this behavior does not simply carry over to individual caches such as the parent cache $C_{2,1}$.

Diminishing returns for probabilistic LRU caches: Next we consider probabilistic LRU caches (qLRU) that generalize known LRU caches [14] and show that the concept of diminishing returns also extends to this type of caches. A qLRU cache operates like an LRU cache except when it needs to replace an object after a cache miss. Here, it admits the new object (evicting the least recently used one) only with probability q. From [12] we can write the hit probability for an object i in a qLRU cache with $q \in (0, 1]$ as

$$p_i = \frac{(1 - e^{-\lambda_i T_{q,c}})q}{1 - (1 - e^{-\lambda_i T_{q,c}})(1 - q)} \tag{9}$$

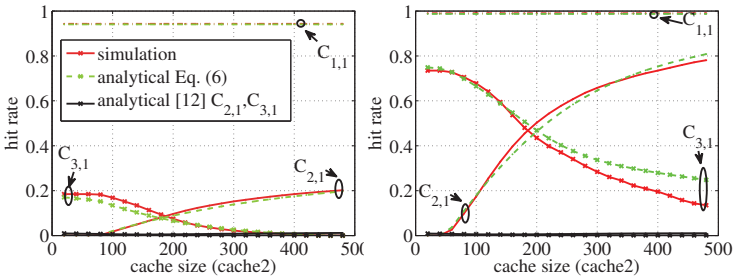
with the *characteristic time* $T_{q,c}$ being the solution of the fixed point equation

$$\sum_{i=1}^{N} \frac{(1-e^{-\lambda_i T_{q,c}})q}{1-(1-e^{-\lambda_i T_{q,c}})(1-q)} = C. \tag{10}$$

Given Poissonian arrivals it becomes clear from (10) which impact qLRU has on the caching performance. Setting the admittance probability $q = 1$ leads to the standard LRU case. Decreasing the admittance probability $q < 1$ leads to characteristic times $T_{q,c}$ that approximately correspond to caches with larger capacity, which explains that qLRU with $q < 1$ achieves a higher hit rate than LRU. This positive impact is approximately limited to the relation $N \geq \frac{C}{q}$ yielding a diminishing return on q. Hence, the boost by qLRU to caching performance is stronger when the number of objects N is large compared to the cache capacity C, as shown in Fig. 3b and 3c. Here, the diminishing return is apparent as the characteristic time and the cache hit rate experience a sharp bend for increasing q. Also note the impact of the relative size C/N of the cache size C compared to the object library N leading to a staggered behavior of the hit rate as in Fig. 3c.

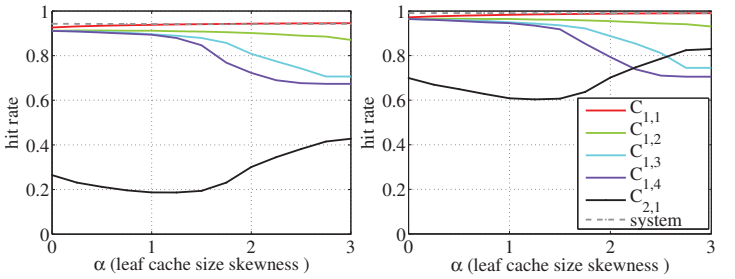
B. The impact of cache size imbalance on hierarchy midgress

In the following, we show that appropriately dimensioned parent caches mitigate the impact of heterogeneous cache sizes on the midgress. Simulation and analytical results for a simple 2-level hierarchy as depicted in Fig. 1b are shown in Fig. 4. In this figure, the hit rates for the four edge caches and the single parent cache are shown. In this particular case, we investigate the individual hitrate of all caches in the hierarchy for a skewed cache size distribution of the edge caches. Fig. 4 shows



(a) Cache line with one hit wonders (b) Cache line without one hit wonders

Fig. 6: Individual cache hit rates for a cache line of $L = 3$ LRU caches with (6a) and without (6b) one-hit-wonders. The edge cache size is fixed to $C_{1,1} = 100$ objects while the parent caches possess a cache budget of $C^{\Sigma} = C_{2,1} + C_{3,1} = 500$ objects. The individual cache hit rates using (6) and the one obtained through trace-based simulations are quite close.



(a) $\sum C_{1,i} = 100$ w/ 1 hit wonders (b) $\sum C_{1,i} = 100$ w/o 1 hit wonders

Fig. 7: Cache hit rates for a trace-based simulation with a two level hierarchy (Fig. 1b), edge cache budget and skewed (Zipf) edge cache sizes.

cache and system hit rates against the skewness parameter α of the edge cache distribution, i.e., for $\alpha = 0$ the caches are all of the same size while for increasing $\alpha > 0$ the discrepancy between the cache sizes increases according to a Zipf distribution. Figs. 4a - 4c show that the analytical estimate of the cache hit rate on intermediate levels, e.g., level two in this case, follows the simulation result more closely than the formulation from [12]. Given homogeneous inputs, the simulation and analytical results show that the midgress increases with an increase in the skewness of leaf cache sizes. In addition to heterogeneity in cache size, the formulae in Sect. III can also be used to optimize the overall system hit rate for heterogeneous input streams, i.e., for different numbers of objects within the different input streams (not shown here).

C. Trace-based model validation

In this section, we describe the results from trace-based simulations to evaluate the quality of our model in the case of real-world requests logged in a large CDN (see Sect. II-D for a description of the data set). Figure 6 shows parent cache hit rates derived using our model in comparison to trace-based simulation and a reference calculated using [12]. Since the trace contains a large set of one-hit-wonders, we compare the hit rates for the case with and without filtering the one-hit-wonders. We look at both cases, since techniques that prevent the caching of such one-hit-wonders based on bloom filters have been proposed, e.g., in [15]. We note that to the best of our knowledge this is the first work that validates cache hierarchy models using large-scale real world traces. Figures 6a and 6b show the results for a 3-cache line hierarchy with and without one hit wonders, respectively. The results show that in both cases the hit rates obtained through our model and the trace-based simulations are very close.

The results of a trace-based simulation in the case of a tree hierarchy (as shown in Fig. 1b) are shown in Fig. 7. The cache sizes at the level $l = 1$ are skewed as described in Sect. IV-B. In contrast to the trace-based simulation described above, here we split the requests from the trace according to regions r_1 to r_4 where requests from one region are assigned to one edge cache. We use the following distribution of the requests to regions as *observed* in the trace: r_1: (14% of total requests), r_2: (13%), r_3: (69%), and r_4: (1%). For the sake of simplicity we assume one parent cache. The edge caches in this example resemble entire caching systems while the parent cache represents a backbone caching system. For our analysis it is sufficient to represent this system through the simple cache topology described above. The results are very similar to the ones shown in Fig. 4.

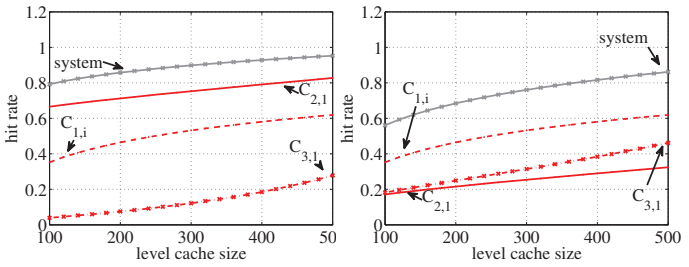
V. Optimizing Cache Hierarchies

In this section, we consider the optimization of three aspects of cache hierarchies, i.e., *(i)* origin offloading, *(ii)* average hop-to-hit count which relates to the object retrieval latency, and *(iii)* request stream splitting between local and global content.

A. Origin Offloading/System Hit Rate

Next, we analyze the impact of the hierarchy design on the origin offloading, i.e., the system hit rate. From the model in Sect. III we know that the origin offloading for a cache hierarchy of multiple levels is based on the object miss rates $m_{i,l}$ at the different levels. In Sect. IV-A we have shown how the level miss rate $m_{i,l}$ relates to the number of parallel caches, their individual size and their individual inputs. This allows us to optimize the origin offloading given a cache hierarchy.

Next, we consider the question of optimizing the hierarchy design for given homogeneous input request streams. Here, we examine the two topologies shown in Fig. 1b and Fig. 1c with overall $H_1 = 8$ edge caches. In the first scenario, there is one parent cache serving all eight edge caches, while in the second scenario there are two parent caches that serve each half of the edge caches (four). The overall cache capacity of the parent level, i.e., $l = 2$, is kept identical for comparison. The result of this analysis is given in Fig. 8. A comparison between Fig. 8a and Fig. 8b shows that a concentration of the cache capacity at the intermediate level $l = 2$, i.e., one cache on level 2, increases the overall system hit rate (gray line) and reduces midgress. Reducing midgress and increasing system hit rate supports the content provider's and CDN provider's objectives but might negatively impact the performance observed by users since some requests may be served from a further away cache increasing average latency.



(a) 1 cache on level 2 (Fig. 1b) (b) 2 caches on level 2 (Fig. 1c)

Fig. 8: Concentration of cache capacity (here on level 2) leads to a higher overall system hit rate and lower midgress.

B. Optimization of Cache hierarchies for latency

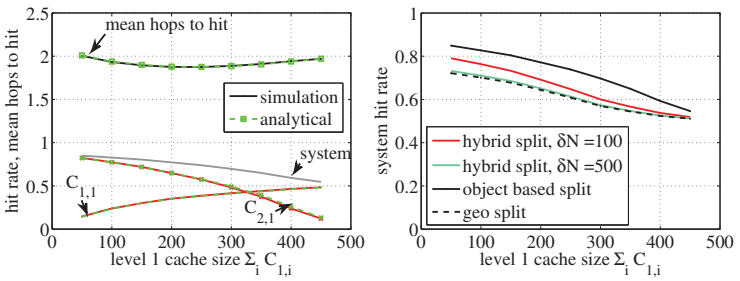
Next, we consider tradeoffs in cache hierarchies that arise when optimizing for object retrieval latency and for cache hit rate, respectively. We will use an elementary example topology from Sect. II to convey the message of this section. However, the techniques discussed in the following are easily expanded to DAGs as described in the previous sections. Here, we consider the optimization of the allocation of a cache budget to a given hierarchy, e.g., as depicted in Fig. 1b with $H_1 = 8$ edge caches. Our optimization goal is to minimize the average number of hops a request has to traverse to produce an object hit, which in turn minimizes the average content retrieval latency assuming stationary hop latency distributions. Next, we assume a fixed cache size budget of $C^\Sigma = H_1 C_{1,1} + C_{2,1}$ where $C_{2,1}$ is the cache size of level 2. For a given number of requests N_{rq} of a given object we consider the number of requests relayed through each cache level l, $N_{rq,l}$. We omit the obvious step (as illustrated in Sect. III-A) of object weighting by $\hat{\lambda}_i$ at the end. We capture the average number of hops W that an object traverses as

$$\mathsf{E}[W] = \sum_{l=1}^{L} \frac{lN_l}{N_{rq}} = \frac{N_{rq,1}+2N_{rq,2}+3N_{rq,3}}{N_{rq}}, \tag{11}$$

given

$$\begin{aligned} N_{rq,1} &= N_{rq} p_1(C_{1,1}) \\ N_{rq,2} &= N_{rq}(1-p_1(C_{1,1}))p_2(C^\Sigma - H_1 C_{1,1}) \\ N_{rq,3} &= N_{rq}(1-p_1(C_{1,1}))(1-p_2(C^\Sigma - H_1 C_{1,1})), \end{aligned}$$

where the first equation resembles the homogeneity of the edge caches. The second equation resembles (5), while the third equation is derived from the overall system hit rate. Note that (11) can be rewritten in terms of the hit rates p_i at the different levels only. Even for this simplified optimization problem a closed form solution is not viable due to the evaluation of the fixed point equation in (1). However, since the analytical model from Sect. III is shown to capture the behavior of LRU cache hierarchies well enough, we use this analytical model to numerically find the optimal cache budget allocation that minimizes the average number of hops-to-hit $\mathrm{E}[W]$. Fig. 9a shows how the average hop count $\mathrm{E}[W]$ evolves for different allocations of cache sizes. The figure also includes the individual cache hit rates of an edge cache and the parent cache. Equation (11) can be easily augmented using weights each describing the (average) hop latency such that the solution of this optimization problem is a cache budget allocation that minimizes the average object retrieval time.



(a) Reducing the average hops to hit (b) Impact of cache splitting

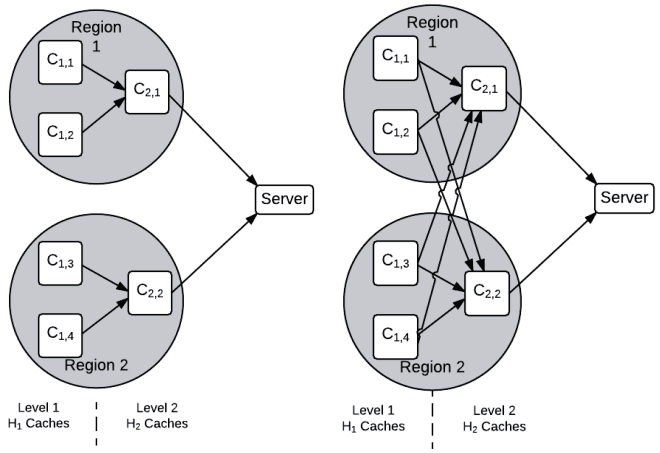
Fig. 9: (9a) Analytical optimization of the cache size allocation to reduce the average hops (latency) for object retrieval. (9b) Comparison of cache hit rates for miss stream splitting between parent caches, i.e., geo split, object-based split, and hybrid split. Parameters for both figures: $C^\Sigma = 500$ objects, Poisson requests with $N = 10^3$ objects.

C. Split Caching

Our model from Sect. III utilizes the notion of DAGs to allow forwarding requests to different parents within a cache hierarchy. In general, clients direct their content requests to a specific edge cache and, in the case of a miss, this edge cache forwards the request to the parent cache which is higher up in the hierarchy. Besides this straightforward approach other alternatives for request forwarding exist, which we describe as *split caching*. Next, we introduce and analyze the performance of three approaches that we denote *geo*, *object-based*, and *hybrid* split. The results of this analysis will provide general design guidelines for request forwarding in cache hierarchies:

1) Geo Split Caching: This version follows the most common approach in today's CDNs. Here, all requests from a lower-layer cache in the hierarchy will always be directed to a fixed higher level cache. An example for geo-based caching is shown in Fig. 10a. This approach has the benefit that content is always served from the "closest" cache located in the same region, which can be important for low-latency applications. Hence, for homogeneous request streams over N objects with popularity $\hat{\lambda}_i$ of object i, the system hit rate under geo split is captured by the framework in Sect. II-C using (1). Geo split caching is easy to implement and does not require any additional complexity, since the forwarding path from requests from level $l-1$ to level l is predefined.

2) Object-based Split Caching: In this approach there is no fixed one-to-one mapping between lower- and higher level caches. Requests are directed based on the content itself, i.e., as shown in Figure 10b, some object requests remain local while other object requests are forwarded globally to other regions. It can be shown that under a uniformly distributed strategy of assigning object requests to parent caches together with consistent assignment of objects to caches the resulting popularities within the split streams, i.e., the stream going from an edge cache to a parent cache, remain Zipfian. The advantage of the object-based split approach is the resulting increased ratio of cache capacity to requested objects at the parent layer. Assuming an object universe of size N of requested objects at the edge caches, each parent cache in Fig. 10b receives requests for only $N/2$ of the objects (assuming an object is assigned to either cache with equal probability). This has a strong impact on increasing parent and system hit rate.



(a) Geo split caching (b) Object based split caching

Fig. 10: Split caching.

Obviously, inter-region connections will come with the cost of a higher object retrieval latency since at each edge cache $N/2$ of the objects will experience a higher inter-region average delay. Hence, for homogeneous request streams over N objects with popularity $\hat{\lambda}_i$ of object i, the system hitrate in the object-based split case can be computed using the framework in the previous sections such as Sect. III, however, with the condition

$$C = \sum_i^{N/2} \left(1 - e^{-\lambda_i' T'}\right). \tag{12}$$

Object-based split caching provides a superior hit rate when compared to geo split caching as seen in Fig. 9b. The reason behind this can be directly obtained from the comparison of (12) and (1). In (12) a higher characteristic time T' is achieved, thus a higher hit rate, due to the fact that the sum goes only over $N/2$ terms. In other words, the ratio of parent cache size to the number of objects that can be requested from that cache increases significantly. Further, the popularity distribution $\hat{\lambda}_i'$ of the combined miss stream entering the parent nodes is approx. a Zipf distribution that comprises only $N/2$ objects. Object-based caching introduces some complexity at the caches, since they have to keep a list of object to parent assignments.

3) Hybrid Split Caching: Next, we introduce a hybrid splitting method which provides a combination of geo-split caching and object-based split caching. Hybrid split caching models the fact that some objects possess geographical locality that arises either due to geographical popularity or as a performance constraint, e.g., on the retrieval latency. Here, a fraction of objects δN, with $\delta \in (0, 1)$, remains local while the remaining objects are split similar to the object-based split caching case. Consider the following example which is based on the object popularity: We deploy geo-splitting for the hottest δN objects in the input streams of the edge caches while all colder objects are split randomly between parent caches as described above. The impact of hybrid splitting can be clearly seen in

$$C = \sum_i^{\delta N + (1-\delta)N/2} \left(1 - e^{-\lambda_{i,\delta} T_\delta}\right), \tag{13}$$

i.e., in the increased number of terms in the sum compared to (12), but also in the changed popularity distribution $\hat{\lambda}_{i,\delta}$.

Figure 9b depicts a comparison of the system hit rate for geo, object-based, and hybrid split caching. We assume a hierarchy as depicted in Fig. 10b with a cache size budget C^Σ for the entire system. We vary the fraction of the cache size budget assigned to the edge level on the x-axis and give every edge cache an equal share. Further, we assume homogeneous request streams at the edge caches. Here, we model locality using different *local fractions* δ. Figure 9b shows that the hit rate of hybrid split lies always between the corresponding curves of geo and object-based split. We observe diminishing returns as outlined in Sect. IV, first, in the gain in hit rate when switching from geo split to object-based split caching increasing level 1 cache size. A diminishing behavior is also observed when varying the local fraction δ, i.e., by keeping a small fraction of the hottest objects local, the system hit rate deviates at the beginning significantly from the result of object-based split. Note that this is mainly due to the skewness of the object popularity distribution. Note that hybrid splitting introduces further complexity (if compared to object-based splitting described in Sect. V-C2), since the leaf caches have to keep track of the popularity of the content requested by the clients to decide where to route a request that resulted in a local cache miss.

VI. Related Work

Despite the fact that individual caches have been extensively studied in the past, e.g., [5], [20], many critical aspects of caching hierarchies have not yet been fully understood. In this section, we review related work on caching mechanisms and hierarchies highlighting the distinctions to our work. The analysis of caching hierarchies with traditional caching algorithms, e.g., LRU and its variants or FIFO, resorts usually to approximations, e.g., for Poisson request processes in the seminal work by Che et al. [4]. This was further extended for a broader class of conditions in [11]. These approximations accurately map the cache dynamics using a single primitive denoted as *characteristic time*. The use of approximations circumvents the computationally exhaustive exact analysis of caches, such as LRU. An exact analysis suffers from Markovian state space explosion under a high numbers of objects, e.g., as in CDNs [15] and, especially, under the paradigm of Information-Centric networking (ICN) [26].

The work in [16] presents a unified methodology to analyze the performance of caches running various algorithms such as LRU, qLRU, LFU and FIFO. In [16] and [17] the authors provide an excellent starting point for the analysis of hierarchies as they consider approximations for the concatenation of LRU caches. In our work, we extend some results from [16] for parent cache hit rates and show corresponding results in Sect. III. The work in [10] provides a framework for analyzing TTL-based caches with exogenous renewal arrivals. For the TTL model, which assigns expiration timers to objects, the authors provide closed-form expressions for single cache metrics under stationary and ergodic input, as well as, an iterative method to approximate the hit rate in cache networks assuming renewal input at every cache. The TTL abstraction has been shown to approximate various caching algorithms such as LRU, Random and FIFO pretty well [12] under the notion of a characteristic time which relates the object occupation probabilities to the cache size using a fixed-point equation. The authors of [21] provide an iterative method to analyze caching networks under the assumption of exogenous Poisson input at every cache. The authors of [9] provide a hybrid method

for evaluating the performance of TTL cache networks that is based on approximating the first two moments of the request processes and fitting the processes to hyper/shifted exponential renewal processes. Finally, exact results for feedforward TTL cache networks are provided in [2] for object requests that are characterized by Markov arrival processes. The authors of [7] formulate a framework for optimizing the aggregate utility of TTL caching networks which is based on the individual cache utility derived from the object hit rates.

The rise of ICN [1], [3], [24], [26] has revived interest in caching networks, respectively, hierarchies. In the ICN context, the authors of [19] show in-network caching schemes that use probabilistic LRU to reduce cache content redundancy in a given caching network. The authors of [6] use a TTL based approach to analyze and dimension the internal pending interest tables of ICN caches. The work in [23] exploits different graph properties such as betweenness and degree centrality to distribute cache capacity. The authors of [8] discuss the feasibility of an incremental deployment of ICN and use trace-based simulations to analyze the quantitative benefits of ICN showing that simple caching architectures can provide most of the gain that is attributed to ICN. A further trace-driven analysis of ICN caching algorithms in [25] has shown that a combination of the Least-frequently used (LFU) object replacement and the probabilistic admission (which is analyzed in Sect. IV) yields the highest cache hit rates in the considered caching network. In Sect. IV, we shed light on the root cause for the improvement in cache hit rate under probabilistic admission.

VII. Conclusions and Future Work

In today's Internet the caching of web and video content is a key component. Such caches are often organized in hierarchies as it is the case for CDNs and they are used to decrease the traffic served by the content provider's origin, decrease the cost of transporting content over the WAN, and providing a faster response time for users downloading the content. In this paper, we presented a new analytical model that can be applied to arbitrary hierarchies characterized as directed acyclic graphs to determine the hit rate of individual caches. Through comparison with simulations and real-world trace data we show that our model predicts midgress traffic, and hit rates of parent caches more accurately than existing analytical models, while predicting origin offload and single cache hierarchies equally well. In addition, we show how this analytical model can be used to determine performance characteristics of certain cache hierarchies and the trade-offs between different configurations of these hierarchies.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[2] D. S. Berger, P. Gland, S. Singla, and F. Ciucu. Exact analysis of TTL cache networks. *Performance Evaluation*, 79:2 – 23, 2014. Special Issue: Performance 2014.

[3] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks (extended version). *Computer Communications*, 36(7):758 – 770, 2013.

[4] H. Che, Y. Tung, and Z. Wang. Hierarchical web caching systems: modeling, design and experimental results. *IEEE JSAC*, 20(7):1305–1314, Sep 2002.

[5] A. Dan and D. Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. *SIGMETRICS Perform. Eval. Rev.*, 18(1):143–152, Apr. 1990.

[6] M. Dehghan, B. Jiang, A. Dabirmoghaddam, and D. Towsley. On the analysis of caches with pending interest tables. In *Proc. of ACM ICN*, pages 69–78, 2015.

[7] M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. Tay. A utility optimization approach to network cache design. In *Proc. of IEEE INFOCOM*, April 2016.

[8] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: Incrementally deployable icn. In *Proc. of ACM SIGCOMM*, pages 147–158, 2013.

[9] N. C. Fofack, M. Dehghan, D. Towsley, M. Badov, and D. L. Goeckel. On the performance of general cache networks. In *Proc. of VALUETOOLS*, pages 106–113, 2014.

[10] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks*, 65:212–231, 2014.

[11] C. Fricker, P. Robert, and J. Roberts. A versatile and accurate approximation for lru cache performance. In *Proc. of ITC*, pages 8:1–8:8, 2012.

[12] M. Garetto, E. Leonardi, and V. Martina. A unified approach to the performance analysis of caching systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 1(3):12:1–12:28, May 2016.

[13] D. K. Krishnappa, M. Zink, and R. K. Sitaraman. Optimizing the video transcoding workflow in content delivery networks. In *Proc. of the ACM Multimedia Systems Conference*, MMSys '15, pages 37–48, New York, NY, USA, 2015. ACM.

[14] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *Proc. of IEEE IPCCC*, pages 445–452, 2004.

[15] B. M. Maggs and R. K. Sitaraman. Algorithmic Nuggets in Content Delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[16] V. Martina, M. Garetto, and E. Leonardi. A unified approach to the performance analysis of caching systems. In *Proc. of IEEE INFOCOM*, pages 2040–2048, April 2014.

[17] N. B. Melazzi, G. Bianchi, A. Caponi, and A. Detti. A general, tractable and accurate model for a cascade of lru caches. *IEEE Communications Letters*, 18(5):877–880, 2014.

[18] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, Dec. 2003.

[19] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *Proc. of ICN Workshop on Information-centric Networking*, pages 55–60, 2012.

[20] P. Rodriguez and E. W. Biersack. Bringing the web to the network edge: Large caches and satellite distribution. *Mobile Networks and Applications*, 7(1):67–78, 2002.

[21] E. J. Rosensweig, J. Kurose, and D. Towsley. Approximate models for general cache networks. In *Proc. of IEEE INFOCOM*, pages 1–9, March 2010.

[22] E. J. Rosensweig, D. S. Menasche, and J. Kurose. On the steady-state of cache networks. In *Proc. of IEEE INFOCOM*, pages 863–871, April 2013.

[23] D. Rossi and G. Rossini. On sizing ccn content stores by exploiting topological information. In *Proc. of IEEE INFOCOM Workshops*, pages 280–285, March 2012.

[24] L. Saino, I. Psaras, and G. Pavlou. Hash-routing schemes for information centric networking. In *Proc. of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 27–32, 2013.

[25] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig. Trace-driven analysis of icn caching algorithms on video-on-demand workloads. In *Proc. of ACM CoNEXT*, pages 363–376, 2014.

[26] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie. Design and evaluation of the optimal cache allocation for content-centric networking. *IEEE Transactions on Computers*, 65(1):95–107, 2016.

[27] G. Zhang, Y. Li, and T. Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16):3128 – 3141, 2013.