Computer Communications 116 (2018) 159–171

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Adapting caching to audience retention rate

Lorenzo Maggi*, Lazaros Gkatzikis, Georgios Paschos, Jérémie Leguay

Mathematical and Algorithmic Sciences Lab, France Research Center, Huawei Technologies France SASU, 92100 Boulogne-Billancourt, France

ARTICLE INFO

Keywords:
Cache replacement
Audience retention rate
Chunk
LRU

ABSTRACT

Rarely do users watch online contents entirely. We study how to take this fact into account to improve the performance of cache systems for video-on-demand and video-sharing platforms, in terms of traffic reduction on the core network. We exploit the notion of "audience retention rate" (ARR), introduced by mainstream online content platforms and measuring the popularity of different parts of the same video content. We first characterize the performance limits of a cache able to store parts of video files, when the popularity and the ARR of each file are available to the cache manager. We then relax the assumption of known popularity and we analyze the performance of a natural adaptation of Least Recently Used (LRU) cache replacement policy that operates on the first chunks of each file. We call it chunk-LRU. We prove that, under a weak assumption on the content popularity distribution, choosing smaller chunks allows to improve the performance of chunk-LRU policy, and we show numerically that even for a small number of chunks, the gains of chunk-LRU are almost optimal. Finally, we provide some guiding principles for chunk-LRU parameter design in real systems.

1. Introduction

Content Distribution Networks (CDN) and Video on Demand applications use network caches to store the most popular contents near the user and reduce backhaul bandwidth expenditure. The future projections for the cost of memory and bandwidth promote the use of caching to satisfy the ever-increasing network traffic [15]. Since the bandwidth saving potential of caching is restricted by the number of files that fit in the cache (the cache capacity), it is interesting to maximize the caching effectiveness under such a constraint. Here, we consider the use of *partial caching*, a technique according to which we may cache specific parts of files, instead of whole ones.

We focus on video files (or, simply, files) which represent a significant fraction of the global Internet traffic (64% according to [6]). Videos are the most representative example of contents that are only partially retrieved, since specific parts of a video file are viewed more often than others. Typically, the average user will "crawl" several videos before watching one in its entirety. Moreover, there exist several "uninteresting" videos that are typically abandoned very early. The above imply that most of the times it is not needed to cache the entire file. Fig. 1 shows the video watch-time from a trace of 7000 YouTube videos. The histogram emphasizes the fact that the vast majority of files is only partially watched, and motivates the design of caching algorithms that avoid caching rarely accessed file parts, e.g. the tail.

Optimization of caching is often based on file popularity. Storing the most popular files results in more *cache hits*, which decreases the impact on the traffic on the core network. Nevertheless, not all the parts of a file are equally popular [11]. Hence, a natural generalization of "store the most popular files" is to split the files into chunks and "store the most popular chunks" instead. To differentiate the popularity of each file chunk we use the metric of the *audience retention rate* (ARR) [24], which measures the popularity of different parts of the same file. Although it has never been exploited before, the ARR has many advantages: it is file specific, it is available in most content distribution platforms, e.g., YouTube [24], and it evolves very slowly over time, which facilitates its easy estimation.[1] The latter is not generally true for chunk popularity which are affected by the time-varying popularity of the corresponding file.

In this paper, we establish a link between the audience retention rate (ARR) and the efficiency of partial caching. Our approach is based on decomposing popularity into file popularity and ARR. More specifically, we address the following questions: (i) *How much bandwidth could we save via partial caching of video content by exploiting statistics on ARR* and (ii) *Is this gain achievable by practical caching algorithms?*

1.1. Related work

Partial caching techniques were first reported in the context of proxy caching, where it was proposed to store the file headers to improve latency performance [16]. To capture both latency and

* Corresponding author.
E-mail address: lorenzo.maggi@huawei.com (L. Maggi).

[1] The quasi-static nature of ARR relates to file particularities, e.g. a movie may become uninteresting towards the end.

https://doi.org/10.1016/j.comcom.2017.11.015
Received 31 May 2017; Received in revised form 13 November 2017; Accepted 24 November 2017

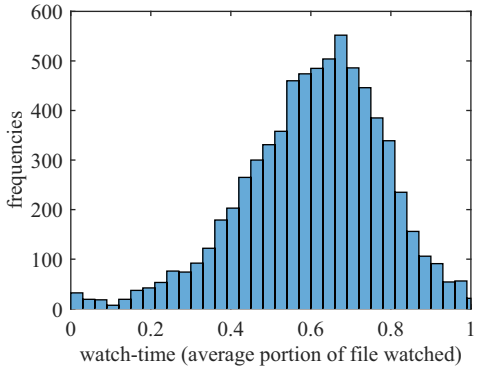

Fig. 1. Histogram of watch-time in YouTube (based on a data sample of 7000 video files from [26]). On average 60% of a file is watched.

bandwidth improvements, the work in [21] proposes to split the files into segments of exponentially increasing size. More generally, it is possible to cache specific chunks in order to capture the different popularity of sections within a file (a.k.a. internal popularity) [11,19].

Intuitively, infinitesimal chunking (e.g., at byte level) offers finer granularity and potentially leads to the optimal caching performance. However, tracking popularity at such fine granularity is impractical and leads to algorithms of prohibitively high complexity [25]. A series of works suggest to split each file into a small number of chunks and treat each chunk independently [1,21]. Alternatively, it is proposed to model internal popularity as a parametric k-transformed Zipf distribution [13,25]. Knowing the distribution type, simplifies the estimation task but still requires parameter estimations individually for each file. Moreover, deducing the optimal size and number of chunks is not straightforward. It was shown in [19] that restricting to n homogeneous chunks incurs a loss which is bounded by $O(n^{-2})$. Alternative heuristic approaches suggest that only a specific segment of each file should be cached and dynamically adjust its size. For instance, Chen et al.[5] propose a segmentation scheme where initially the whole object is cached but the segment size is gradually set equal to its estimated average watch-time. Similar adaptive strategies have been also considered for peer-to-peer networks [10], where starting from a small segment, the portion to be cached is increased according to the number of requests and watch-time. The caching of several segments of each file was proposed in [8], since users may be interested only in specific, non-contiguous parts of files. In this case the segment size has to be selected accordingly.

In the context of Dynamic Adaptive Streaming HTTP (DASH) video streaming, contents are split into chunks along two dimensions, i.e., time and encoding quality. Ye et al. [23] only consider the enconding dimension, thus tackling the problem of deciding which encoding layers should be cached so as to minimize backhaul traffic. The notion of *audience retention rate* (ARR), measuring the popularity of different parts of the same file, has been first introduced by Maggi et al. [14]. Yang et al. [22] extended its application in the context of coded caching. There, the ARR is supposed to be known by the cache manager. Instead, in our work we consider uncoded caching and we show how the classic Least Recently Used (LRU) caching policy can benefit from splitting files into chunks, even in the extreme case where the cache manager is oblivious to the ARR. Whereas we exploit audience retention rates to select which files to cache, in [12] the reverse problem of prefetching content so as to maximize retention rates is considered.

1.2. Main contributions

In this paper we first investigate a trace of YouTube data in [26] and we conclude that partial caching has a great potential to improve performance, mainly because (i) the average video watch-time is no more than 70%, and (ii) the size of a video is negatively correlated with its watch-time (see Section 2). Motivated by this, we harness the concept of ARR and we first study in Section 4 its impact on the theoretical gains that partial caching has on traditional caching systems, in terms of reduction of the traffic on the core network. Combining the theoretical analysis with the YouTube data, we show that in realistic settings the traffic reduction of partial caching over traditional caching may reach up to 50% if ARR and popularity were known for each file.

It is then interesting to investigate the benefits brought by partial caching in a setting where the content popularity and ARR are unknown. Thus, in Section 5, we derive the performance of a class of practical chunk-LRU (Least Recently Used) policies, which split files into different chunks, evict the chunk at the tail of files and perform the classic LRU scheme on the remaining chunks. Our analysis shows that chunk-LRU policies realize the gain of partial caching, and its performance can be further improved by tuning two essential parameters, namely the number of chunks and the size of the chunk at the tail of files. Hence, in Section 6 we gain intuition into the parameter design and we show that close-to-optimal performance can be attained with simple design principles in mind.

We resume our main technical contributions to the literature in the following:

- We formulate the traffic reduction optimization problem under the knowledge of ARR and provide a waterfilling algorithm to solve it efficiently. For the special case where users watch each video continuously until they abandon it, we derive the optimal waterfilling partial allocation in closed form. It consists of caching a compact interval $[0, \nu]$ of the file where ν is given in closed form.
- We consider a natural adaptation of LRU cache replacement algorithm to the scenario of partial viewing, which we call chunk-LRU and that operates on the first chunks of each file. We then build an analytical framework to relate the chunk-LRU performance to the ARR behavior, subject to the well-known Che's approximation for LRU performance [4].
- We provide a sufficient condition for ARR such that sub-splitting chunks is always beneficial for the chunk-LRU scheme.
- We provide simple hints for the design of chunk-LRU parameters in real systems, supported by numerical evaluations.

We remark that we choose to show the benefits of file chunking on LRU specifically for mainly three reasons. First, the analysis of LRU is tractable, thanks to Che's analytical approximation [4]. Second, it is widely used due to its simple and efficient implementation by means of a doubly linked list. Third, LRU serves as basis for several other more advanced recency replacement policies, such as LRU threshold, LRU*, LRU-hot, LRU-threshold, LRU-MIN, LRULSC, SB-LRU, SLRU and HLRU (see [2,18]).

2. Youtube video watch-time

In this section we examine YouTube access traces[2] in [26] in order to gather some useful statistics on the video watch-time, which for each file measures the portion ($\in [0; 1]$) watched by the users. Watch-times are crucial for caching: by employing partial caching we may avoid to cache rarely watched parts of videos and use the freed cache space to store more files.

Since most strategies try to cache the most popular files, first we investigate the relationship between average watch-time and file popularity. We classify video files into 10 groups according to their average daily views. Fig. 2 depicts the estimated probability density

[2] The dataset is publicly available and was crawled using the YouTube Data API in 2013. It contains information about 7000 files, including daily views, watch-time, duration, genre and title of each file.

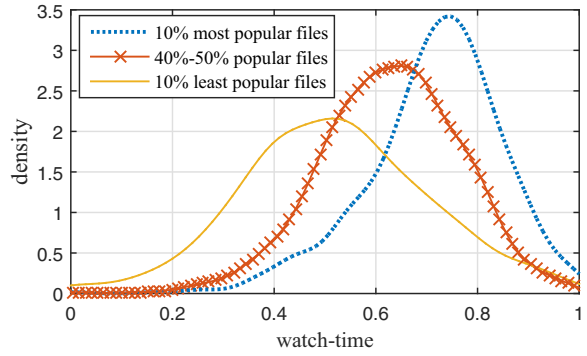

Fig. 2. Watch-time distribution for different classes of video popularity. The average watch-time of a video increases with its popularity.

function of watch-time for three representative groups, the 10% most popular videos, the 10% least popular, and the intermediate ones. Interestingly, we observe that *the more popular a video is, the higher the average watch-time*. However, even for the most popular ones, on average only 72% of each video is watched, which leaves room for caching optimization.

Next, we investigate the relationship between watch-time and file duration. The latter is a critical parameter for caching due to the cache capacity constraint which eventually determines caching performance. If longer videos are only partially watched, avoiding to cache their unwatched parts will yield a greater benefit. In Fig. 3, we depict with dots the YouTube data for the 20% most popular files. In order to identify how the watch-time is affected by the video duration and its popularity, we use locally weighted polynomial regression [7] to fit a smoothed surface to the corresponding data. Notice that the most beneficial regime for caching purposes corresponds to the upper left corner of the plot, namely highly popular videos of large size. We observe that in this region the average watch-time is around 0.7. In addition, independently of the video popularity, *watch-time decreases rapidly with video duration.*

We then group the available data to 10 classes according to their popularity and duration (≷200 s). We depict the details of the derived classes in Table 1, namely for each class we depict the average watch-time, the fraction of videos belonging to this class and its average duration in seconds. We observe that the *large and popular videos amount to a non-negligible percentage of* 5%. In addition, the average watch-time of large files is significantly smaller than that of smaller ones. To

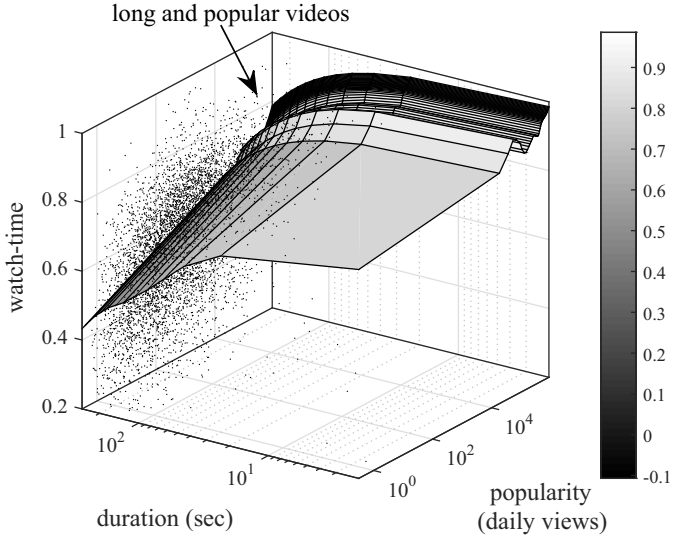

Fig. 3. Average watch-time is increasing with the popularity of files, but steeply decreasing with its duration.

Table 1
The characteristics of videos in [26], classified with respect to their size ("small" and "large"). These data will be used to derive realistic and class-specific AARs for our numerical evaluation.

Popularity duration	Small		
	Av. watch-time	Fraction of population	Av. duration (s)
Lowest	0.52	0.179	81
Low	0.6	0.162	112
Medium	0.64	0.153	128
High	0.67	0.152	130
Highest	0.72	0.145	124
Popularity duration	Large		
	Av. watch-time	Fraction of population	Av. duration (s)
Lowest	0.37	0.020	220
Low	0.47	0.036	220
Medium	0.57	0.045	223
High	0.60	0.047	222
Highest	0.65	**0.053**	235

precisely evaluate the impact of watch-time to caching, we use these data in the subsequent Sections 4 and 5 to quantify the theoretical maximum and the practically feasible caching performance.

3. System model

We consider a communication system where users download video files (or, simply, files) from the network. Let $\mathscr{M} = \{1, \cdots, M\}$ be the file catalog. Each file $i \in \mathscr{M}$ is of size S_i bytes. Content requests are generated according the well-known Independent Reference Model (IRM) [9], for which the file requests are independent of each other. We call p_i the probability that file i is requested, under the assumption that a file request has arrived. Equivalently, the sequence of file requests can be thought of as M independent homogeneous Poisson processes with intensity rate proportional to the probability vector $\{p_i\}_i$. For convenience of notation, we assume that the probabilities are in decreasing order, i.e., $p_1 \geq p_2 \geq \cdots \geq p_M$.

One cache of size C bytes is deployed in the network.[3] Whenever a requested file is found in the cache, the cache itself can directly serve the user. Otherwise, the file needs to be retrieved through the core network, which provides access to a central file content store containing the entire file catalog, see Fig. 4. Hence, caching can have a profound impact on the traffic reduction on the core network.

We next introduce the crucial concept of audience retention rate, that will be proven to have an intrinsic connection with the performance of partial caching.

3.1. Viewing behavior model: audience retention rate

The audience retention rate (ARR) $R_i(\tau)$ is defined by YouTube as the percentage of users that are still watching video i at the corresponding (normalized) instant τ, out of the overall number of views [24], see also Fig. 5. As it will become apparent, in our analysis the ARR has a prominent role in determining the caching performance.

Let us shed light on the definition of ARR by formally describing the typical viewing behavior of a typical video-on-demand user. A user may watch video file i from instant $a_i(1)$ up to $b_i(1)$, then she possibly skips to $a_i(2)$ and watches until $b_i(2)$, and so forth[4]. The (random) watched part W_i, which equals the minimum portion of file i that the user needs to download, is the union of all watch intervals j:

[3] Our analysis can be extended to a cache hierarchy by letting p_i express the probability that a request for file i is missed by the caches at all the child nodes [15].

[4] We remark that such intervals may also overlap, i.e., a user may rewind the video and watch a part of it multiple times. We assume that, if this occurs, then the user can directly retrieve the file portion that she has already watched from her terminal's cache.

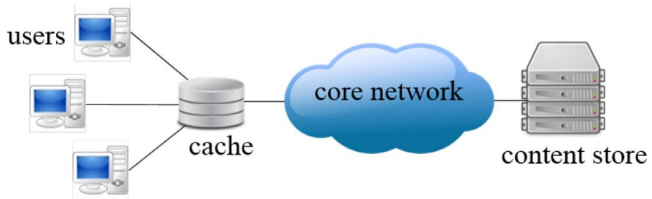

Fig. 4. System model.

$W_i = \cup_j [a_i(j); b_i(j)]$.

We call $|W_i|$ the (random) *watch-time* of user watching file i. For ease of notation, we consider a_i, $b_i \in [0; 1]$ as portions of the whole video file duration. The ARR[5] function $R_i(\tau)$ can be then formally defined as the probability that a user has watched the (normalized) instant τ of the file, i.e.,

$$R_i(\tau) = \Pr(\tau \in W_i), \qquad \tau \in [0; 1].$$

Alternatively, we may think of $R_i(\tau)$ as the fraction of users that watch the (normalized) instant τ of the file i.

We remark that, thanks to the definition of R_i, we can easily evaluate the average watch-time for file i as $\int_0^1 R_i(\tau)d\tau$.

In order to come up with a realistic ARR function, we will use the estimated parameters in Table 1 for our numerical investigations in Sections 4.3 and 5.4.

Next, we devise a realistic and more specific viewing behavior model and we derive its relationship to ARR.

3.1.1. Viewing abandonment model

This is a special instance of the viewing model presented above. It assumes that users always start watching each file i from its beginning, and they abandon it after a random time portion $b_i \in [0; 1]$. Hence, in this case the watched part W_i takes on the simple form $W_i = [0; b_i]$, thus b_i equals the watch-time. We call $\pi_i(.)$ the probability density distribution of the abandonment time variable b_i. The relationship between the abandonment distribution π_i and the ARR R_i is described by the expression:

$$R_i(\tau) = 1 - \int_0^\tau \pi_i(t)dt. \tag{1}$$

Hence, in this case the ARR $R_i(\tau)$ measures the *fraction of users with watch-time higher than* τ for the particular file i. We first observe from (1) that R_i is inherently non-increasing, with $R_i(0) = 1$. We also remark that, under the viewing abandonment assumption, the ARR R_i uniquely describes the random watch behavior $[0; b_i]$ of user via π_i. This observation does not hold though for the general case described in Section 3.1, where the same ARR R_i may result from an arbitrary distribution of watch behaviors.

In this paper we will specialize some of our results to the scenario where the viewing abandonment model holds.

4. Performance limits of partial caching

This section analyzes the performance limits of partial caching in the context of ARR. Our performance metric is core network traffic and we tackle the off-line problem of finding the optimal *static* (partial) file cache allocation.[6] In particular, we will compare the maximum network traffic saved by caching entire files versus caching arbitrary portions of each of those. In both cases it is idealistically assumed that the file popularity distribution $\{p_i\}_{i\in\mathscr{M}}$ and the ARR functions $\{R_i\}_{i\in\mathscr{M}}$ are perfectly known to the cache manager. This analysis serves as an upper bound for any cache replacement strategy with more limited information, as the one devised in Section 5.

Let us first formalize our problem. We define the *partial allocation* $Y_i \subseteq [0; 1]$ of file i to be the collection of (possibly) non-adjacent portions of file i, that are selected to be *permanently* stored in the cache. Subject to a partial allocation Y_i, any requests for the remaining portions $[0; 1] \setminus Y_i$ need to be served by the origin file store. Due to the specific ARR for this file, this happens with probability $\int_{[0;1]\setminus Y_i} R_i(\tau)d\tau$. Therefore, under a partial allocation vector $\boldsymbol{Y}$, we may express the expected traffic on the core network per request $B(\boldsymbol{Y})$ as

$$B(\boldsymbol{Y}) = \sum_{i\in\mathscr{M}} S_i p_i \int_{[0;1]\setminus Y_i} R_i(\tau)d\tau. \tag{2}$$

Considering the file size S_i and cache size C, a partial allocation vector $\boldsymbol{Y}$ is feasible whenever $\sum_{i\in\mathscr{M}} S_i \int_{Y_i} dx = C$. Our goal is to select a feasible vector $\boldsymbol{Y}$ that minimizes the incurred traffic $B_s(\boldsymbol{Y})$, i.e.,

$$\boldsymbol{Y}^* = \operatorname*{argmin}_{\boldsymbol{Y}} B(\boldsymbol{Y})$$
$$\text{s. t.} \begin{cases} \sum_{i\in\mathscr{M}} S_i \int_{Y_i} 1dx = C \\ Y_i \subseteq [0; 1] \end{cases} \tag{3}$$

If users always watch the whole file, i.e., $R_i(\tau) = 1$ for all $\tau \in [0; 1]$ and $i \in \mathscr{M}$, then the optimization (3) takes a simple form which is solved by the well-known *store-the-most-popular-files* policy. In this case, we would choose to fully store, $Y_i = [0; 1]$, the files of highest p_i up to the cache capacity and no portion of the rest, i.e. $Y_i = \emptyset$ otherwise. As indicated by the previous section however, in reality this is not the case, hence we expect $\boldsymbol{Y}^*$ to bring certain improvement, that we evaluate in Section 4.3.

Technically speaking, if we lift any assumption on the shape of the ARR, the best cache allocation should intuitively prescribe to partition all files at the finest granularity (at the byte level, say), order them according to their popularity, and fill the cache with the *most popular bytes*. We now provide an equivalent waterfilling characterization of the optimal partial file allocation $\boldsymbol{Y}^*$ to solve this problem. The main advantage of this formulation lies in the fact that it leads to an efficient algorithm to compute $\boldsymbol{Y}^*$, that we present in Section 4.2.

Theorem 1 (*Optimal allocation*). *The optimal partial file allocation* $\boldsymbol{Y}^*$ *can be expressed as*

$$Y_i^*(\mu) = \{\tau:\ p_i R_i(\tau) \geq \mu\} \quad \forall\ i \in \mathscr{M}, \tag{4}$$

where μ *is such that* $\sum_{i\in\mathscr{M}} S_i |Y_i^*(\mu)| = C$, *where* $|.|$ *is the size*[7] *of a subset of* $[0; 1]$.

Informally speaking, the water level μ determines a popularity threshold above which a byte of any file deserves to be stored in the cache.

4.1. Viewing abandonment model

In the special case of viewing abandonment model (see Section 3.1.1), we already observed that the ARR R_i is non-increasing for all $i \in \mathscr{M}$. This allows us to specialize our result in Theorem 1 as follows.

Corollary 1 (*Optimal allocation for viewing abandonment model*). *Consider the viewing abandonment model with strictly decreasing* R_i, *for all* $i \in \mathscr{M}$. *The optimal file allocations writes* $\boldsymbol{Y}^* = [0; \eta_i^*]$ *for all* $i \in \mathscr{M}$, *where*

[5] Our definition of ARR is in accordance with the definition of audience retention (or "engagement") rate by Wistia.com [20]. Youtube's ARR [24] actually counts the video rewinds as multiple views inside the same videos.

[6] We remark that in our analysis of the optimal traffic bandwidth $B(\boldsymbol{Y}^*)$ we assumed that the files $\boldsymbol{Y}^*$ are already present in the cache and we did not take into account the traffic needed to fill the cache. If we wish to incorporate this aspect, we could say that $B(\boldsymbol{Y}^*)$ is the expected traffic achieved asymptotically over a number of requests tending to infinity.

[7] Formally defined as the Lebesgue measure.

Fig. 5. Instance of audience retention rate (ARR) from YouTube.

$$\begin{cases} \eta_i^*(\mu) = \begin{cases} 1 & \text{if } p_i R_i(1) \geq \mu \quad (\mu \geq 0) \\ 0 & \text{if } p_i \leq \mu \\ R_i^{-1}(\mu/p_i) & \text{otherwise} \end{cases} \\ \sum_{i \in \mathscr{M}} S_i \eta_i^*(\mu) = C. \end{cases} \tag{5}$$

A remarkable observation here is that the optimum bandwidth performance is achieved by splitting every file in only two parts and caching the first one. We may determine the exact splits if the abandonment distribution is given. For instance, if π_i is truncated exponential one with parameter λ_i, i.e.,

$$\pi_i(\tau) = \frac{\lambda_i}{1 - e^{-\lambda_i}} e^{-\lambda_i \tau}, \qquad \tau \in [0; 1],$$

then the following holds.

Corollary 2 (*Optimal allocation for exponential viewing abandonment model*). *Under the exponential viewing abandonment model the optimal file allocations writes* $\boldsymbol{Y}^* = [0; \eta_i^*]$ *for all* $i \in \mathscr{M}$, *where*

$$\begin{cases} \eta_i^*(\mu) = \left[-\frac{1}{\lambda_i} \ln\left(\frac{\mu}{p_i}\left(1 - e^{-\lambda_i}\right) + e^{-\lambda_i} \right) \right]^+, \quad (\mu \geq 0) \\ \sum_{m=1}^{M} S_i \eta_i^*(\mu) = C. \end{cases} \tag{6}$$

4.2. Computation of optimal performance

To solve the optimization problem in (3), we observe that it can be expressed as a separable convex optimization problem with linear and box constraints. If we further assume that the functions R_i do not have any plateau, then the objective function becomes strictly convex, thus we can adapt the water-filling algorithm presented in [17, Section 7.2] to our scope in order to efficiently compute the optimal cache partial file allocation $\boldsymbol{Y}^*$. We defer the details of the algorithm to the Appendix, Section A.2. In few words, we iteratively compute the popularity threshold μ by solving a fixed-point equation (Step 2). Then, we compute the estimated cache occupation δ (Steps 3 and 4). Then, depending on whether δ exceeds the available cache capacity or not, we truncate the cache storing policy η to 0 or 1 (Step 5), until convergence.

4.3. Performance evaluation with real data

In order to evaluate the performance of the optimal partial allocation in a realistic scenario we utilize the average watch-time parameters shown in Table 1. In Fig. 6, we compare the core network traffic $\underline{B} = B_s(\boldsymbol{Y}^*)$ generated by the optimal partial caching strategy with the one produced by the most natural strategy prescribing to store the most popular files in their entirety. We observe that remarkable gains from partial caching are achieved for cache size ratios higher than 10^{-2} of the total catalog size, which we typically find in current CDN scenarios.

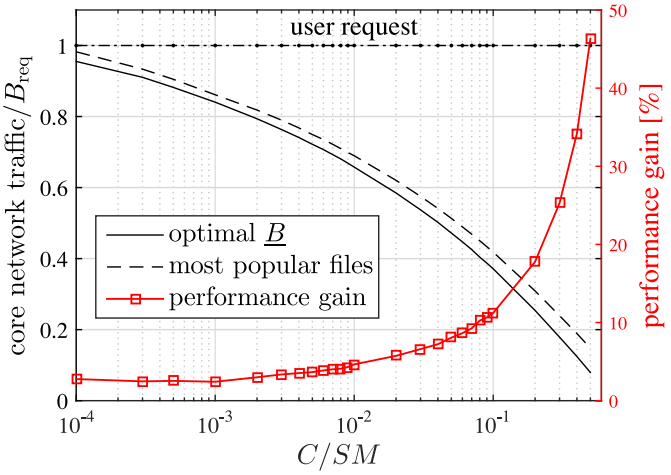

Fig. 6. Core traffic generated by the optimal partial caching strategy in a realistic scenario vs. the traffic produced by storing the most popular files in their entirety. We show in circled red line the resulting performance gain by using the first strategy. We utilized the parameters obtained via real data shown in Table 1. The file popularity distribution follows a Zipf law with parameter 0.8 [9]. S is denoted as the average file size. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We then show in Fig. 7 the optimal portion of files that should be stored according to the same optimal caching strategy, for different values of the cache size.

Interestingly, only very popular files are stored in their entirety, even for large cache sizes.

We finally remark that we will sometimes find convenient to normalize the core network traffic figures with respect to the number of bytes requested by users B_{req} per file request, which equals

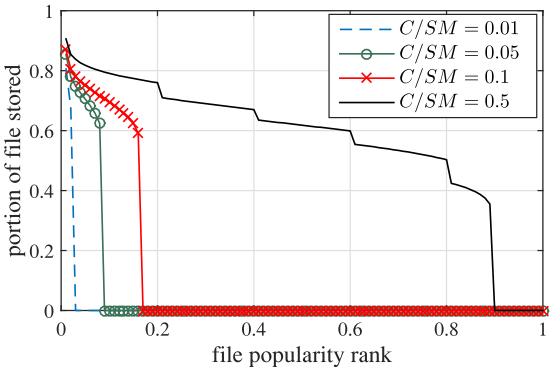

Fig. 7. Optimal portion of files that should be stored according to the same optimal caching strategy in Fig. 6. Given a certain C/SM, the file with file popularity x should be stored from its beginning up to portion y.

$$B_{\text{req}} = \sum_{i=1}^{M} S_i p_i \int_0^1 R_i(\tau)d\tau. \tag{7}$$

We notice that B_{req} is the minimum bandwidth per file request required to serve the users when no cache is deployed in the system.

5. Chunk-LRU: analysis

After analyzing the best performance that can only be achieved with full information on the system parameters, we turn to the study of a practical cache replacement scheme that shows good performance even when file popularity and ARR are unknown.

It is a widespread understanding that the Least Recently Used (LRU) cache replacement policy represents a good trade-off between hit-rate performance and implementation complexity in a real scenario where no statistics on file popularity are available to the cache manager. LRU operates in the following way: upon a new file request, if the file is not stored in the cache, then the least recently requested file is evicted from the cache and replaced with the newly requested one. Thus, LRU keeps track of file popularity by updating a recency table of file requests. Moreover, thanks to its short memory, LRU reacts quickly to variations in file popularity. In its simplest form though, each time a file is requested even only partially by a user and is not found in the cache, LRU would prescribe to cache it *in its entirety* (and to update the LRU recency table accordingly). Since users rarely watch video files entirely, as previously observed, such primitive form of LRU would generate extra-traffic in the core network and would waste precious cache space to store unpopular portions of files.

In the case of partial viewing, it is then natural to study a generalization of the classic LRU policy that operates on file *chunks*, instead of the whole file. We call it *chunk-LRU*, and it functions as follows. Each file is split into $N+1$ consecutive and non-overlapping chunks. According to chunk-LRU, if a chunk is requested by a user but not found in the cache, then it is retrieved from the content store and placed in the cache. If the cache is full, then the least recently requested chunk is evicted by the cache, in the classic LRU fashion. Finally, the user receives the requested chunk. We here study a simple generalization of this standard scheme, where the last (i.e., the $(N+1)$-th) chunk of each file, which is the least popular part under the assumption of decreasing ARR, is *never* be stored in the cache, even if requested by a user. Intuitively, this frees up space for *more* popular chunks of *less* popular files to be stored in the cache. We call ν the tail drop factor that pinpoints the position of the last chunk.

We now formally describe the chunk-LRU algorithm. Notice that the (normalized) file split is denoted as $[x_0 \equiv 0, x_1, \cdots, x_N \equiv \nu, x_{N+1} \equiv 1]$, and the ith chunk corresponds to the file portion $[x_{i-1}; x_i]$ (see also Fig. 8).

Chunk-LRU algorithm.

Step 1 (Initialization):
- **1.1**) Set the tail drop factor $\nu \in (0; 1]$
- **1.2**) Partition each file i into $N+1$ chunks of the form $[x_0 = 0; x_1], [x_1, x_2], \cdots, [x_{N-1}; \nu \equiv x_N], [x_N = \nu; x_{N+1} = 1]$, where $x_i \in [0; 1]$ (see Fig. 8)
- **1.3**) An initial chunk request recency vector is available

Step 2: A request for a packet of file $i \in \mathcal{M}$ belonging to its kth chunk $[x_{k-1}, x_k]$ arrives
- **2.1**) If $k = N+1$, then the request is handled by the core network and the cache is not updated (i.e., the tail is never cached)
- **2.2**) Else, if $1 \le k \le N$, then
  - **2.2.1**) If the requested chunk is stored in the cache, then the cache sends the packet to the user
  - **2.2.2**) If the requested chunk is not stored in the cache, then it is retrieved from the core network and then stored in the cache, after evicting the minimum number of least recently used chunks. Finally, the cache sends the packet to the user
- **2.3**) The recency vector of the chunks stored in the cache is updated in an LRU fashion. Return to Step **2**)

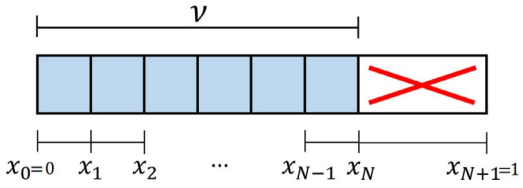

Fig. 8. File split into $N+1$ chunks. Only the first N are considered for chunk-LRU; the last one is never stored in the cache.

Remark 1. For the sake of analysis simplicity we assume that the chunk splitting, described by the variables $\mathbf{x}$ and ν, does not depend on the identity of the file. We leave the study of file-dependent split as a future extension.

Performing LRU on the first N chunks presents two main benefits. On the one hand, it reduces the extra-traffic on the core network caused for the retrieval of file portions that are not requested. For instance, whenever a user watches a file from its beginning up to portion b, only the first $\bar{k} = \min_k\{x_k \ge b\}$ chunks are downloaded. Hence, only the portion $x_k - b$ is stored in the cache without being accessed. On the other hand, we exploit the fact that the tail of a file is generally less popular than the rest [25]. Hence, by systematically discarding the tail of each file we avoid to evict from the cache the first chunks, which are likely to be more popular. Additionally, although this is not the focus of this paper, performing LRU on chunks would allow to keep track of the evolution of the popularity of each chunk. Nevertheless, the resulting benefits would be minor, since the ARR varies on a time scale much slower than the file popularity dynamics.

5.1. Chunk-LRU performance under viewing abandonment

After having described our chunk-LRU algorithm, we now turn to the analysis of its performance. To this purpose, in this section we will assume that *the viewing abandonment model holds* (see Section 3.1.1). Moreover, in order to come up with our analytical results we make the common simplifying assumption that all files have the same size $S = S_i$. This is well justified by the fact that we can break large files into equal size fragments, and perform chunk-LRU over the chunks of the file fragments.

We first observe that, under the viewing abandonment model (Section 3.1.1), the probability that the kth chunk of file i is requested by a user knowing that the user herself has already started watching file i equals $R_i(x_{k-1}) = \int_{x_{k-1}}^1 \pi_m(\tau)d\tau$. Since the requests for file i follow by assumption a Poisson process of intensity (proportional to) p_i, then the request process for the kth chunk is also Poisson with reduced intensity $p_i R_i(x_{k-1})$. Thus, thanks to an adaptation of the popular Che's approximation [4] we can already compute the hit rate for a specific chunk, i.e., the probability that a chunk is found in the cache when requested.

Let us elaborate on this. Che's approximation was originally proposed in [4] to compute the hit rate for files whose request successions follow independent Poisson processes. It approximates the characteristic time t_C, measuring the time that a file spends in the cache, as a constant. When shifting the request granularity from the file to the chunk level, the independence property of request streams is unavoidably lost. Nevertheless we can still rely on the intuition that when the cache size is significantly larger than the file size the characteristic

time of each chunk is approximately equal and constant, hence Che's approximation still holds, which has been shown valid in [15]. Therefore, the hit rate $h_{k,i}$ for the kth chunk of file i can be approximated as $h_{k,i} = 1 - e^{-p_i R_i(x_{k-1}) t_C}$, where the characteristic time t_C obeys the following relation [9]:

$$\frac{C}{S} = \sum_{k=1}^{N} \Delta x_k \sum_{i=1}^{M} h_{k,i}, \tag{8}$$

where $\Delta x_k = x_k - x_{k-1}$. Intuitively, expression (8) claims the equality between the number of items that can be cached (C/S) and the sum of file chunks (Δx_k), weighted by their probability of being found in the cache ($\sum_{i=1}^{M} h_{k,i}$).

Finally, we can derive the expected traffic per file request B_{cLRU} forwarded to the core network when the chunk-LRU cache replacement policy is employed. To this aim, we first observe that the expression $R_i(x_{k-1})(1-h_{k,i})$ measures the probability that chunk k of file i is requested but not found in the cache, under the assumption that file i has been requested (which occurs with probability p_i). Moreover, we notice that the average watch-time of the last chunk (which is never cached) equals $\int_{\nu}^{1} R_i(\tau)d\tau$. The expression of B_{cLRU} then follows:

$$B_{\text{cLRU}}(\mathbf{x}, \nu) = S \sum_{i=1}^{M} p_i \left(\sum_{k=1}^{N} R_i(x_{k-1})(1-h_{k,i})\Delta x_k + \int_{\nu}^{1} R_i(\tau)d\tau \right) \tag{9}$$

where $\mathbf{x} = \{x_1, \cdots, x_{N-1}\}$.

5.2. Benefits of chunk sub-splitting

We now focus on the impact of the chunk size on chunk-LRU performance, measured as the traffic generated at the core network B_{cLRU}. Intuitively speaking, increasing the number of chunks allows chunk-LRU to estimate the inner popularity of each file with finer granularity. Nevertheless, this does not prove the intuition, since modifying the chunk size also has an impact on the characteristic time t_C in a non-trivial way via the expression in (8).

Before stating the main result of this section, we first need to introduce some notation. We denote $\underline{t}_C$ and $\bar{t}_C$ as the characteristic times when only one chunk (i.e., $[0; \nu]$) and chunks of infinitesimal size dx (say, at the byte level) are employed, respectively. More formally, $\underline{t}_C$ and $\bar{t}_C$ are the unique roots of the two following equations:

$$\begin{aligned} \frac{C}{S} &= \nu \sum_{i=1}^{M} \left(1 - e^{-p_i \underline{t}_C}\right) \\ \frac{C}{S} &= \sum_{i=1}^{M} \int_0^{\nu} \left(1 - e^{-p_i R_i(x) \bar{t}_C}\right) dx, \end{aligned}$$

respectively. It is easy to see that $\underline{t}_C$ and $\bar{t}_C$ represent a lower and an upper bound for the characteristic time t_C, respectively. Next, we will say that the chunk split $\mathbf{x}'$ is a file sub-split with respect to the split $\mathbf{x}$ whenever $\mathbf{x} \subset \mathbf{x}'$. In other words, $\mathbf{x}'$ further splits the file in smaller chunks. We finally observe that if $\nu = \frac{C}{MS}$ then the cache can store all the first files up to their portion ν; hence, it is reasonable to constrain ν within the interval $\left[\frac{C}{MS}; 1\right]$.

We are now ready to prove that, under an assumption on the file popularity and ARR, any refinement of the chunk granularity produces a decrease in the expected traffic load on the core network.

Theorem 2 (*Sufficient condition for sub-splitting to be beneficial*). *Let* $\nu \in [\frac{C}{MS}; 1]$ *and let* $\mathbf{x}$ *be a file chunk split. Assume that*

$$\frac{d}{d\tau} \sum_{i=1}^{M} p_i R_i(\tau) e^{-p_i R_i(\tau) t_C} < 0, \qquad \forall\ t_C \in [\underline{t}_C; \bar{t}_C],\ \tau \in [0; 1] \tag{10}$$

Then, any file chunk sub-split $\mathbf{x}'$ *outperforms* $\mathbf{x}$ *in terms of traffic generated on the core network, i.e., the following holds:*

$$B_{\text{cLRU}}(\mathbf{x}', \nu) < B_{\text{cLRU}}(\mathbf{x}, \nu).$$

It easily follows from Theorem 2 that splitting each file into infinitesimal chunks is optimal. Clearly, this holds under the simplifying assumption that chunks can be managed without any traffic overhead. In Section 6, we discuss how to design the number of chunks under more realistic settings.

Finally, we remark that numerical experiments suggest that our sufficient condition (10) is very loose. More specifically, it generally holds for realistic popularity distributions and ARRs. It is not satisfied only in pathological cases where the distribution is extremely concentrated around few popular files and the cache size very small, near to the size of a single file.

5.3. Optimal performance of chunk-LRU

In this section we focus on the computation of the best performance of chunk-LRU, optimized over the chunk size and tail drop factor ν. We will utilize it as a benchmark for the performance evaluation of practical chunk-LRU policies in realistic scenarios in Section 5.4.

In order to come up with the best performance achievable by chunk-LRU we need to find the solution of the following optimization problem:

$$\underline{B}_{\text{cLRU}} = \min_{N, \mathbf{x}, \nu, t_C} B_{\text{cLRU}}(\mathbf{x}, \nu)$$
$$\text{s. t.} \begin{cases} \dfrac{C}{S} = \displaystyle\sum_{k=1}^{N} \Delta x_k \sum_{i=1}^{M} 1 - e^{-p_i R_i(x_{k-1}) t_C} \\ \dfrac{C}{MS} \le \nu \le 1 \\ 0 = x_0 \le x_1 \le \cdots \le x_{N-1} \le x_N = \nu. \end{cases} \tag{11}$$

It follows from Theorem 2 that, if condition (10) holds, then the bandwidth utilization of any file chunk split $\mathbf{x}$ and $\nu \in [\frac{C}{MS}; 1]$ is lower bounded by the performance $\underline{B}_{\text{cLRU}}(\nu)$ of the infinitesimal split (say, at the byte level). This greatly simplifies the formulation of (11) in a two-variable constrained optimization problem (see Eq. (12)). Below we formalize this result.

Corollary 3 (*Performance bound for chunk-LRU*). *Assume that condition (10) holds. For any file chunk split* $\mathbf{x}$ *and tail drop factor* ν*, the traffic performance* $B_{\text{cLRU}}(\mathbf{x}, \nu)$ *is lower bounded by the performance* $\underline{B}_{\text{cLRU}}$ *of the infinitesimal chunking approach:*

$$\underline{B}_{\text{cLRU}} \le B_{\text{cLRU}}(\mathbf{x}, \nu),$$

where $\underline{B}_{\text{cLRU}}$ *is computed as*

$$\underline{B}_{\text{cLRU}} = \min_{\nu, t_C} \sum_{i=1}^{M} \int_0^{\nu} p_i R_i(x) e^{-p_i R_i(x) t_C} dx + \int_{\nu}^{1} p_i R_i(\tau) d\tau$$
$$\text{s. t.} \begin{cases} \dfrac{C}{S} = \displaystyle\sum_{i=1}^{M} \int_0^{\nu} \left(1 - e^{-p_i R_i(x) t_C}\right) dx \\ \dfrac{C}{MS} \le \nu \le 1. \end{cases} \tag{12}$$

We stress the fact that $\underline{B}_{\text{cLRU}}$ is the lowest core network traffic achievable by a chunk-LRU cache replacement policy.

Thanks to the formulation in (12), we can prove the following two intuitive results via standard Lagrangian optimization techniques. First, if users never watch video files in their entirety, then it is always optimal to never cache a non-negligible portion of file, i.e., $\nu^* < 1$.

Corollary 4. *If* R_i *is continuous and* $R_i(1) = 0$ *for all* $i \in \mathcal{M}$ *then the optimal* $\nu^* < 1$.

Finally, as intuition suggests, if all users watch the whole video file then the best chunk-LRU policy is actually the standard LRU.

Corollary 5. *If $R_i(\tau) = 1$ for all $\tau \in [0; 1]$, $i \in \mathscr{M}$ then splitting files into chunks does not improve LRU traffic performance.*

5.4. Numerical evaluations of chunk-LRU performance

In this section we evaluate numerically the traffic performance on the core network of the proposed class of chunk-LRU cache replacement policies. In all simulations we considered file size and chunks of equal size, in order to restrict our focus on the two most impacting parameters on the chunk-LRU performance, i.e., the number of chunks N and the tail drop factor ν. As in Section 4, we consider the ARR scenario shown in Table 1, estimated from the real Youtube dataset from [26]. We show our results[8] in Fig. 9. For comparison purposes, we also display the optimal performance $\underline{B}$ under full information that we derived in Section 4, that represents a performance bound for *any* cache replacement policy under partial viewing assumption.

We first notice that, as hinted by Theorem 2, the traffic generated by chunk-LRU decreases as the number N of chunks increases ($N = 4, 20$). The infinitesimal chunk size limit ($N = \infty$) is shown to achieve optimal performance $\underline{B}_{\text{cLRU}}$, as claimed in Corollary 3. Notably, chunk-LRU performs close to its optimal performance even with a limited number of chunks ($N = 20$, but also $N = 4$). On the other hand, as expected, not splitting the file and setting $\nu = 1$ (1-chunk-LRU) is a poor choice in the presence of partial viewing behavior. In fact, the traffic generated by retrieving parts of file that are not requested by the users outweighs the obtained benefits through cache hits even for medium-size caches. This explains why the traffic generated by 1-chunk-LRU can be even higher than the one without any cache deployed.

The best tail drop factor $\nu^* = \nu^*(N)$ used to produce Fig. 9 is optimized for each value of N and cache size C, as shown in Fig. 10. We notice that ν^* is closely related to average watch-time, since it captures the portion of files with the lowest popularity which need to be systematically discarded from the cache. For small cache sizes, simulations show that the optimal value ν^* is lower than the watch-time: in fact, to compensate for the reduced cache size, low values of ν allow to squeeze in the cache a significant amount of different - and popular - file headers.

Nevertheless, we remark that in order to compute the optimal value of $\nu^*(N)$ one should be aware of all system parameters, i.e., content popularity and abandonment distribution. Since this is clearly not the case in real systems, we should expect that a sub-optimal value of ν is chosen in reality. Therefore, the lines in Fig. 9 for different values of N should be regarded as performance lower bounds for chunk-LRU policies operating on N chunks.

Motivated by this, in Section 6 we tackle this issue by showing the sensitivity of chunk-LRU performance with respect to ν, and by providing sensible advice on the design of ν in real systems.

6. Chunk-LRU: principles for parameter design

In the previous sections we investigated the impact of content chunking on caching performance. We first computed the performance limit $\underline{B}$ of any cache replacement policy, then we analyzed the performance of chunk-LRU, being a natural adaptation of LRU operating on chunks of files, rather than on the whole files.

In this final section we aim at providing some hints on the practical design of the parameters defining chunk-LRU. In particular, we will discuss the choice of the number of chunks N and of the tail drop factor ν.

[8] The traffic performance is normalized w.r.t. the number of bytes effectively requested by users B_{req} per file request (see Eq. (7)).The chunk-LRU policies have chunks with equal size.

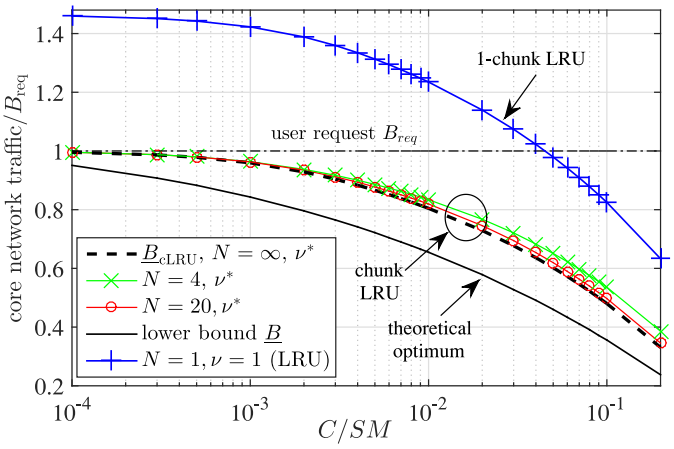

Fig. 9. Normalized core network traffic generated by chunk-LRU for different number of chunks vs. the theoretical optimum B and vs. standard LRU. The optimal $\nu^* = \nu^*(N)$ is computed for each value of N and cache size C, as depicted in Fig. 10. We also evaluate the performance achieved when the sub-optimal value of $\nu = 1$ is utilized. The file popularity distribution follows a Zipf law with parameter 0.8 [9].

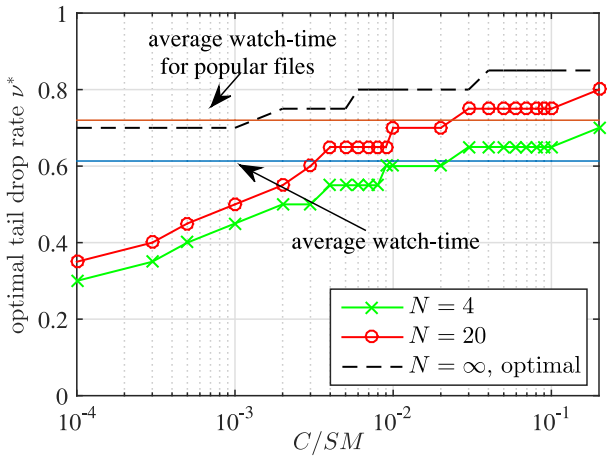

Fig. 10. Optimal tail drop factor ν^* for different number of chunks $N = 4, 20, \infty$. We notice that the optimal $\nu^*(N)$ is within a neighborhood of the average watch-time of 0.61.

factor ν.

6.1. Number of chunks N

Firstly, we discuss a fundamental performance/complexity trade-off faced when designing the number of chunks N. Corollary 3 claims that, according to our model, it is always beneficial to increase the number of chunks to decrease the traffic on the core network. However, in practice, infinitesimal chunking (say, at the byte level) suffers from the two following limitations on complexity and overhead.

(i) *Complexity*: It is well known that LRU cache replacement policy can be implemented with complexity $O(1)$; in other words, increasing the number of chunks does not affect the amount of operations needed to handle one chunk. However, increasing the number of chunks N causes the complexity of chunk-LRU *per unit of time* to scale linearly with N.

To tackle this issue, we can suppose that the available processing/memory resources constrain the number of chunks within some maximum value $N_{\max}$, i.e., $N \leq N_{\max}$.

(ii) *Overhead*: Chunking introduces overhead due to encoding and data encapsulation. For instance, HTTP streaming also impose file segmentation of equal size, and segmentation introduces an overhead per chunk, which increases the overall file size. More specifically, it was recently shown that dividing a DASH segment into fragments could

improve latency performance, but at the cost of an additional overhead of up to 20% [3]. Finally, an encapsulation overhead has to be considered if the chunking is performed at sub-MTU (Maximum transmission unit) scale, i.e., chunks smaller than 1500 bytes. In this case, if a chunk size of K times smaller than the MTU is selected, then since TCP-IP packets carry a header of 66 bytes, an additional overhead of $\frac{66}{\frac{1500}{K}} = 4.4K\%$ is imposed.

From the discussion above it should be clear that chunking at very fine granularity, i.e., setting N arbitrarily large, is not desirable in practice.

In order to provide some guiding principles on the parameter design of the number of chunks N in real systems, we make henceforth the simplifying assumption that each chunk is appended with a header of invariable size δS. Moreover, since in ABS it is common practice to split each file in chunks of equal duration, thus here we assume equally sized chunks. In this case, the original expression of the expected traffic generated on the core network in (9) becomes

$$\begin{aligned} B^{\delta}_{\text{cLRU}}(N,\nu) &= S\sum_{i=1}^{M} p_i\left(\sum_{k=1}^{N} R_i\left(\frac{(k-1)\nu}{N}\right)(1-h_{k,i})\left(\frac{\nu}{N}+\delta\right)\right. \\ &\quad \left. + \int_{\nu}^{1} R_i(\tau)d\tau\right) \\ \text{s. t. } \frac{C}{S} &= \sum_{k=1}^{N}\left(\frac{1}{N}+\delta\right)\sum_{i=1}^{M} 1-e^{-p_i R_i\left(\frac{(k-1)\nu}{N}\right)t_C} \\ N &\le N_{\max}. \end{aligned} \tag{13}$$

We observe from Fig. 11 that two different regimes arise for the choice of N, depending on the relative size of the overhead δ with respect to the whole file size. When the overhead is negligible (Fig. 11, $\delta \le 10^{-3}$) it is beneficial to split the file into as many chunks as possible in order to minimize the traffic on the core network (i.e., $N = N_{\max}$). As we will see in the following, this choice also has a beneficial impact on the choice of ν. On the other hand, if the overhead size is non-negligible with respect to the whole file size ($\delta > 10^{-3}$), then the traffic depends in a non-monotonic fashion on the number N of chunks.

6.2. Tail drop factor ν

Turning now our attention to the design of the tail drop factor ν, we display in Fig. 12 the dependence of the performance of the chunk-LRU scheme with respect to ν for different values of number of chunks N and cache size. We can first distinguish two different regimes for the design of ν. If the number of chunks is sufficiently high ($N > 50$ in this case), the performance of chunk-LRU has very limited sensitivity with respect to the choice of ν in a left neighborhood of 1: in fact, *the fine granularity of chunk splitting already prevents the tail of files not to be cached, if not popular*. In this case, setting $\nu = 1$ appears to be a near-optimal choice.

However, for values of $N \le 50$ the choice $\nu = 1$ is largely sub-optimal. This highlights the fact that, *in the "small N" regime, dropping the last portion of each file helps making up for the poor granularity of file chunking*.

In conclusion, by comparing Figs. 11 and 12 we can distinguish two different regimes for parameter design, only depending on the size of the chunk overhead δ. If δ is sufficiently small ($\le 1\%$ of the whole file size), then opting for the maximum allowed number of chunks $N = N_{\max}$ and the maximum tail drop factor ($\nu = 1$, i.e., all chunks can be stored in the cache) is a good design choice. In fact, this does not incur a significant performance loss (see, e.g., the curves with $\delta = 10^{-3}$ in Fig. 11 and $N = 50$ in Fig. 12) and it is *oblivious to all system parameters*, i.e., popularity distribution p_i and ARR R_i.

On the other hand, if the chunk overhead is non negligible (e.g., $> 1\%$ of the whole file size), then from Fig. 11 a reasonable choice for N appears to be in a range between 10 and 20. In this case, the choice of the tail drop factor ν should be refined ($\nu < 1$). We suggest that in this case, to gain further insight in the optimization problem in (13), the shape of probability distribution p and the ARR R should be somehow estimated *offline*. In fact, we firstly remark that the optimal ν^* is not strictly a function of the popularity of each file, but only of the rank-dependent popularity p_i of the i-th most popular file, for each i (see Eq. (13)). It has been shown in [9] that such rank-dependent relation depends on the class of traffic and is slowly varying over time, hence it is easily predictable offline. Secondly, we argue the ARR functions R_i vary on a much slower time scale than that of file popularity, which greatly facilitates its offline estimation. For such two reasons, we claim that an offline estimation of p and R may suffice to refine the choice of ν. We leave more in-depth analysis of such interesting scenario to future investigations.

7. Conclusions

In this paper, we shed light on the intrinsic connection between the caching traffic performance and the audience retention rate (ARR), which measures the popularity of different portions of the same video file. We first derive the performance limits of partial caching when ARR is known by the cache manager. Then we analyze the performance of a natural adaptation of the classic LRU scheme that operates on chunks of file, called chunk-LRU. This prescribes to split each file into chunks and to apply LRU on the chunks, while never storing the last one. We formally prove that sub-splitting is beneficial if chunk overhead is not considered. In more realistic scenarios, we suggest that if the overhead is non negligible then the optimal number of chunks is finite, and the tail drop factor helps making up for the poor granularity of file chunking.

The introduction of ARR in caching decisions opens up new interesting research directions. ARR is generally available in online video

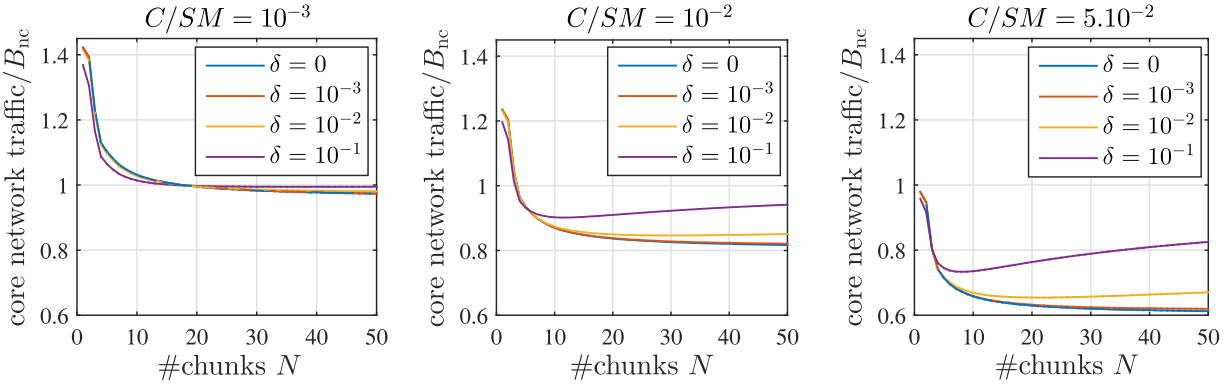

Fig. 11. Traffic on the core network vs. number of chunks, with tail drop factor $\nu = 1$ (all chunks are cached), for different values of the overhead size and cache size. The file size is $S = 1$. All chunks are assumed to be of equal size.

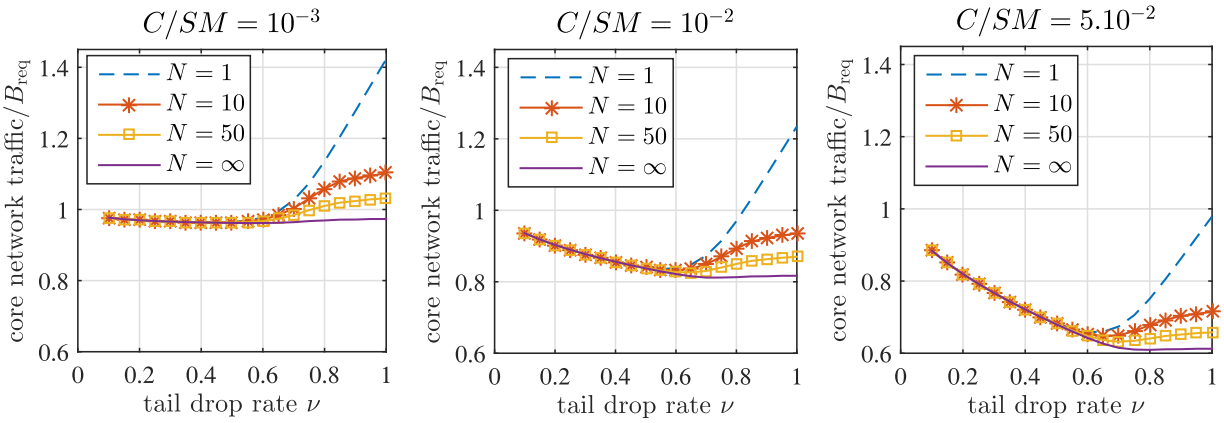

Fig. 12. Normalized core network traffic vs. tail drop factor ν, for different number of chunks N and cache size C.

distribution systems and does not evolve over time. Thus, it can be used to decompose the problems of file popularity estimation and optimal chunking without loss of optimality. In this context, the generalization of existing caching mechanisms so as to optimally exploit the benefits of partial caching is an interesting topic for future study.

Appendix

A1. Proof of Theorem 1

Proof. As a first step, let us define $f_i(\tau)$: $[0; 1] \to [0; 1]$ as a one-to-one function such that the permuted ARR function $R_i'(\tau)$: $=R_i(f_i^{-1}(\tau))$ is non decreasing. The function f_i is a permutation function that orders the file parts in order of decreasing popularity, such that $f_i(\tau) < f_i(\tau')$ if and only if $R_i(\tau) > R_i(\tau')$.[9] Then, R_i' is the outcome of such permutation. As a second step, we reformulate the optimization problem in (3) as

$$\boldsymbol{Y}^* = \underset{\boldsymbol{Y}}{\operatorname{argmax}} \sum_{i \in \mathscr{M}} S_i \int_{Y_i} p_i R_i(\tau) d\tau$$
$$\text{s. t.} \begin{cases} \sum_{i \in \mathscr{M}} S_i \int_{Y_i} 1 d\tau = C \\ Y_i \subseteq [0; 1] \end{cases} \tag{14}$$

We can recast the bandwidth saving optimization problem in (14) in terms of the permuted engagement rates R_i' and by considering only right intervals of 0 of the kind $Y_i = [0; \eta_i]$, as follows:

$$\max_{\boldsymbol{\eta} \in \mathbb{R}^M} \sum_{i \in \mathscr{M}} p_i S_i \int_0^{\eta_i} R_i'(\tau) d\tau$$
$$\text{s. t.} \begin{cases} \sum_{i \in \mathscr{M}} \eta_i S_i = C \\ \eta_i \in [0; 1]. \end{cases} \tag{15}$$

In fact, it is not profitable to consider a larger search domain, e.g., more complicated subsets $\boldsymbol{Y}$ of $[0; 1]^M$: for any collection of subsets $\boldsymbol{Y}$ it is possible to replace Y_i with the interval $\left[0; \int_{Y_i} d\tau\right]$ with a strict increase of the objective function while the feasibility is still preserved. We can further simplify (15) by defining the function $R_i''(\tau) = p_i R_i'(\tau)$, as follows:

$$\min_{\boldsymbol{\eta} \in \mathbb{R}^M} \sum_{i \in \mathscr{M}} \int_0^{\eta_i} -R_i''(\tau) d\tau$$
$$\text{s. t.} \begin{cases} \sum_{i \in \mathscr{M}} \eta_i = C \\ \eta_i S_i \in [0; S_i]. \end{cases} \tag{16}$$

We notice that $\frac{d}{d\eta_i} \int_0^{\eta_i} - R_i''(\tau) d\tau = -p_i R_i'(\eta_i)$, which is non-decreasing in η_i. Thus we recognize in (16) a convex optimization problem with linear and box constraints, where the objective function is separable in the optimization variables $\boldsymbol{\eta}$. It is known that such kind of problems can be solved via a classic water-filling technique (see [17, Chapter 6]): more specifically, there exists a positive "water level" μ such that the optimal portions $\boldsymbol{\eta}^*(\mu)$ can be computed as

[9] We notice that such f_i always exists, even though is not unique, since it can arbitrarily break the ties among equally popular parts of a single file, and it is in general discontinuous.

$$\begin{cases} \eta_i^*(\mu) = \begin{cases} 1 & \text{if } \min\limits_{\tau\in[0;1]} R_i^{'}(\tau) \geq \mu \\ 0 & \text{if } \max\limits_{\tau\in[0;1]} R_i^{'}(\tau) \leq \mu \\ R_i^{'-1}(\mu) & \text{else} \end{cases} \\ \sum\limits_{i\in\mathscr{M}} S_i \eta_i^*(\mu) = C \end{cases} \tag{17}$$

By rewriting (17) in terms of R_i', we obtain the expressions:

$$\begin{cases} \eta_i^* = \begin{cases} 1 & \text{if } p_i \min\limits_{\tau\in[0;1]} R_i'(\tau) \geq \mu \\ 0 & \text{if } p_i \max\limits_{\tau\in[0;1]} R_i'(\tau) \leq \mu \\ R_i^{'-1}(\mu/p_i) & \text{else} \end{cases} \\ \sum\limits_{i\in\mathscr{M}} S_i |Y_i^*| = C. \end{cases}$$

and we can finally claim that

$$Y_i^* = f_i^{-1}([0;\eta_i^*]) = \{\tau:\ p_i R_i(\tau) \geq \mu\} \quad \forall\ i \in \mathscr{M}.$$

The thesis follows. □

A2. Waterfilling algorithm

Algorithm to compute the optimal stored portion η_i^* for each content i.

Input: Audience retention rate R_i for all contents i, content popularity distribution $\{p_i\}_i$, cache size C, size of video files $\{S_i\}_i$.

Step 1 (Initialization) Let $k = 0$, $C^{(0)}$: $=C$, $\mathscr{M}^{(0)}$: $=\mathscr{M}$, $\mathscr{M}_a^{\mu}$: $=\varnothing$, $\mathscr{M}_b^{\mu}$: $=\varnothing$. Define R_i' as a strictly decreasing extension of R_i over the whole real axis, i.e., $R_i'(\tau) = R_i(\tau)$ for all $\tau \in [0;1]$ and R_i' is strictly decreasing over $\mathbb{R}$.

Step 2 Estimate the optimal popularity threshold $\mu^{(k)}$ according to the modified ARR R' by solving the fixed-point equation: $\sum_{i\in\mathscr{M}^{(k)}} S_i [R_i']^{-1}(\mu^{(k)}) = C^{(k)}$.

Step 3 Compute the set of contents whose estimated stored portion:

- is negative, i.e., $\{m:\ [R_i']^{-1}(\mu^{(k)}) < 0\}$: $=\mathscr{M}_-^{\mu(k)}$
- exceeds 1, i.e., $\{m:\ [R_i']^{-1}(\mu^{(k)}) > 1\}$: $=\mathscr{M}_+^{\mu(k)}$
- is within $[0;1]$, i.e., $\{m:\ 0 \leq [R_i']^{-1}(\mu^{(k)}) \leq 1\}$: $=\mathscr{M}^{\mu(k)}$

Step 4 Compute the estimated cache occupation $\delta(\mu^{(k)})$: $\delta(\mu^{(k)}) = \sum_{i\in\mathscr{M}_+^{\mu(k)}} S_i + \sum_{i\in\mathscr{M}^{\mu(k)}} S_i [R_i']^{-1}(\mu^{(k)})$.

Step 5

- If the estimated cache occupation equals the available cache memory ($\delta(\mu^{(k)}) = C^{(k)}$) or $\mathscr{M}^{\mu(k)} = \varnothing$ then set $\mu = \mu(k)$, $\mathscr{M}_-^{\mu} = \mathscr{M}_-^{\mu} \cup \mathscr{M}_-^{\mu(k)}$, $\mathscr{M}_+^{\mu} = \mathscr{M}_+^{\mu} \cup \mathscr{M}_+^{\mu(k)}$, $\mathscr{M}^{\mu} = \mathscr{M}^{\mu(k)}$. Go to Step 6 and terminate.
- Else, if the estimated cache occupation exceeds the available cache memory ($\delta(\mu^{(k)}) > C^{(k)}$) then set $C^{(k+1)}$: $=C^{(k)}$. Compute $\mathscr{M}^{(k+1)}$: $=\mathscr{M}^{(k)} \setminus \mathscr{M}_-^{\mu(k)}$, and update $\mathscr{M}_-^{\mu}$: $=\mathscr{M}_-^{\mu} \cup \mathscr{M}_-^{\mu(k)}$, k: $=k+1$. Go to Step 2.
- Else, update the remaining available cache memory as $C^{(k+1)} = C^{(k)} - \sum_{i\in\mathscr{M}_+^{\mu(k)}} S_i$ and set $\mathscr{M}^{(k+1)}$: $=\mathscr{M}^{(k)} \setminus \mathscr{M}_+^{\mu(k)}$, $\mathscr{M}_+^{\mu}$: $=\mathscr{M}_+^{\mu} \cup \mathscr{M}_+^{\mu(k)}$, k: $=k+1$. Go to Step 2.

Step 6 (Termination) Set the optimal stored portion $\eta_i^* = 0$ for all $i \in \mathscr{M}_-^{\mu}$; $\eta_i^* = 1$ for all $i \in \mathscr{M}_+^{\mu}$; $\eta_i^* = [R_i']^{-1}(\mu)$ for all $i \in \mathscr{M}^{\mu}$.

Return optimal stored portion η_i^* for all contents i.

A3. Proof of Proposition 1

Proof. Since R_i is already strictly decreasing, then we can consider $f_i(\tau) = \tau$ and $R_i' = R_i$. Moreover, in this case $\min_\tau R_i(\tau) = 0$ and $\max_\tau R_i(\tau) = 1$. The thesis easily follows. □

A4. Proof of Corollary 2

Proof. Define

$$\widetilde{R}_i^{-1}(\tau) = -\frac{1}{\lambda_i} \ln\left(\tau\left(1 - e^{-\lambda_i}\right) + e^{-\lambda_i}\right).$$

We notice that $\widetilde{R}_i^{-1}(\mu/p_i) = R_i^{-1}(\mu/p_i)$ when $0 < \mu \le p_i$ and $\widetilde{R}_i^{-1}(\mu/p_i) < 0$ whenever $p_i > \mu$. Then, we can rewrite (5) as

$$\begin{cases} \eta_i^* = [\widetilde{R}_i^{-1}(\mu/p_i)]^+ \\ \sum_{i \in \mathscr{M}} S_i \eta_i^* = C. \end{cases}$$

The thesis easily follows. □

A5. Proof of Theorem 2

Proof. Let us first introduce the function

$$\xi^{(t_C)}(\tau) = \sum_{i=1}^{M} p_i R_i(\tau) e^{-p_i R_i(\tau) t_C}.$$

We then define $\mathscr{I}(f)|_{\mathbf{x}}$, where f is a continuous function defined over $\mathbb{R}$, the integral approximation of f via Riemann sums of the type:

$$\mathscr{I}(f)|_{\mathbf{x}} = \sum_{k=1}^{N} f(x_{k-1}) \Delta x_k.$$

We notice that if f is increasing (decreasing) then $\mathscr{I}(f)|_{\mathbf{x}} < (>) \mathscr{I}(f)|_{\mathbf{x}'}$ for any sub-splitting $\mathbf{x}'$. We can now rewrite $B_{\mathrm{cLRU}}(\mathbf{x}, \nu)$ as (compare with (9))

$$\begin{aligned} B_{\mathrm{cLRU}}(\mathbf{x}, \nu) &= \mathscr{I}\left(\xi^{(t_C)}\right)|_{\mathbf{x}} \\ \text{s. t. } M\nu - \frac{C}{S} &= \mathscr{I}\left(h^{(t_C)}\right)|_{\mathbf{x}} \end{aligned}$$

where $h^{(t_C)}(\tau) = \sum_{i=1}^{M} e^{-p_i R_i(\tau) t_C}$. Since $h^{(t_C)}(\tau)$ is increasing in τ, it easily follows from an induction argument that the value of characteristic time for any chunk splitting is found within $[\underline{t}_C; \bar{t}_C]$.

Consider now a sub-splitting $\mathbf{x}'$ with associated characteristic time t_C'. Since $h^{(t_C)}(\tau)$ is increasing, then $\mathscr{I}(h^{(t_C)})|_{\mathbf{x}'} > \mathscr{I}(h^{(t_C)})|_{\mathbf{x}}$. Also, since $\mathscr{I}(h^{(t_C')})|_{\mathbf{x}'} = \mathscr{I}(h^{(t_C)})|_{\mathbf{x}}$, and $h^{(t)}(\tau)$ is decreasing in t then $t_C' > t_C$. We then have

$$\begin{aligned} B_{\mathrm{cLRU}}(\mathbf{x}, \nu) &= \mathscr{I}\left(\xi^{(t_C)}\right)|_{\mathbf{x}} > \mathscr{I}\left(\xi^{(t_C')}\right)|_{\mathbf{x}} > \mathscr{I}\left(\xi^{(t_C')}\right)|_{\mathbf{x}'} \\ &= B_{\mathrm{cLRU}}(\mathbf{x}', \nu) \end{aligned}$$

where the second inequality follows from the fact that $\xi^{(t)}(\tau)$ is decreasing in τ for any value t of the characteristic time. The thesis is proven. □

A6. Proof of Corollary 4

Proof. The derivative with respect to ν of the objective function in (12) in the direction along which the constraint is satisfied writes

$$q(\nu) = -\sum_{i=1}^{M} \left(1 - e^{-p_i R_i(\nu) t_C}\right) p_i R_i(\nu) + \int_0^{\nu} \sum_{i=1}^{M} p_i^2 R_i^2(\tau) e^{-p_i R_i(\tau) t_C} d\tau \frac{\sum_{i=1}^{M} 1 - e^{-p_i R_i(\nu) t_C}}{\int_0^{\nu} \sum_{i=1}^{M} p_i R_i(\tau) e^{-p_i R_i(\tau) t_C} d\tau} \tag{A.18}$$

Let us calculate $q(1 - d\nu)$, which equals

$$d\nu \left(\frac{A + B\, d\nu}{C + D\, d\nu} \sum_{i=1}^{M} p_i \left| R_i'(1) \right| - d\nu \sum_{i=1}^{M} p_i^2 \left| R_i'(1) \right|^2 \right).$$

Since $A = \int_0^{\nu} \sum_{i=1}^{M} p_i^2 R_i^2(\tau) e^{-p_i R_i(\tau) t_C} d\tau > 0$ and $B = \int_0^{\nu} \sum_{i=1}^{M} p_i R_i(\tau) e^{-p_i R_i(\tau) t_C} d\tau > 0$, then $q(1 - d\nu) > 0$ and thesis is proven. □

A7. Proof of Corollary 5

Proof. We first observe that, if $R_i(\tau) = 1$, then for all ν we have $B_{\mathrm{cLRU}}([0; \nu], \nu) = B_{\mathrm{cLRU}}(\mathbf{x}, \nu)$ for any chunk splitting $\mathbf{x}$. Then it suffices to prove that $q(\nu) < 0$ holds for all $\nu \in (0; 1)$, i.e., that the following expression holds:

$$\left(\sum_{i=1}^{M} 1 - e^{-p_i t_C} \right) \sum_{i=1}^{M} p_i^2 e^{-p_i t_C} - \sum_{i=1}^{M} \left(1 - e^{-p_i t_C}\right) p_i \sum_{i=1}^{M} p_i e^{-p_i t_C} < 0.$$

The thesis follows. □

References

[1] K. Agrawal, T. Venkatesh, D. Medhi, A dynamic popularity-based partial caching scheme for video on demand service in IPTV networks, Proceedings of COMSNETS'14 (2014) 1–8, http://dx.doi.org/10.1109/COMSNETS.2014.6734888.

[2] W. Ali, S.M. Shamsuddin, A.S. Ismail, A survey of web caching and prefetching, Int. J. Adv. Soft Comput. Appl. 3 (1) (2011) 18–44.

[3] N. Bouzakaria, C. Concolato, J.L. Feuvre, Overhead and performance of low latency live streaming using MPEG-DASH, Proceedings of the Fifth International Conference on Information, Intelligence, Systems and Applications, IISA 2014, IEEE, 2014, pp. 92–97.

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

[5] S. Chen, H. Wang, X. Zhang, B. Shen, S. Wee, Segment-based proxy caching for internet streaming media delivery, IEEE Multimed. 12 (3) (2005) 59–67. ISSN 1070-986X. http://doi.ieeecomputersociety.org/10.1109/MMUL.2005.56 .
[6] Cisco, Cisco visual networking index: forecast and methodology, 2014, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html. 2014–2019.
[7] W.S. Cleveland, Robust locally weighted regression and smoothing scatterplots, J. Am. Stat. Assoc. 74 (368) (1979) 829–836.
[8] U. Devi, R. Polavarapu, M. Chetlur, S. Kalyanaraman, On the partial caching of streaming video, Proceedings of the IEEE IWQoS, 2012, (2012), pp. 1–9, http://dx.doi.org/10.1109/IWQoS.2012.6245982.
[9] C. Fricker, P. Robert, J. Roberts, A versatile and accurate approximation for LRU cache performance, Proceedings of the Twenty-fourth International Teletraffic Congress (ITC 24), (2012), pp. 1–8.
[10] M. Hefeeda, O. Saleh, Raffic modeling and proportional partial caching for peer-to-peer systems, IEEE/ACM Trans. Netw. 16 (6) (2008) 1447–1460. ISSN 1063-6692. doi:10.1109/TNET.2008.918081 .
[11] K.W. Hwang, D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, V. Misra, K.K. Ramakrishnan, D.F. Swayne, Leveraging video viewing patterns for optimal content placement, Proceedings of IFIP Conference on Networking, IFIP'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 44–58. ISBN 978-3-642-30053-0.
[12] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, N. Shahmehri, Bandwidth-aware prefetching for proactive multi-video preloading and improved HAS performance, Proceedings of the Twenty-third ACM international conference on Multimedia, ACM, 2015, pp. 551–560.
[13] S.-H. Lim, Y.-B. Ko, G.-H. Jung, J. Kim, M.-W. Jang, Inter-chunk popularity-based edge-first caching in content-centric networking, IEEE Commun. Lett. 18 (8) (2014) 1331–1334. ISSN 1089–7798. doi:10.1109/LCOMM.2014.2329482 .
[14] L. Maggi, L. Gkatzikis, G. Paschos, J. Leguay, Adapting caching to audience retention rate: which video chunk to store? (2015), arXiv preprint arXiv:1512.03274.
[15] J. Roberts, N. Sbihi, Exploring the memory-bandwidth tradeoff in an information-centric network, Proceedings of ITC, (2013), pp. 1–9.
[16] S. Sen, J. Rexford, D. Towsley, Proxy prefix caching for multimedia streams, Proceedings of the IEEE INFOCOM'99, 3 (1999) 1310–1319, http://dx.doi.org/10.1109/INFCOM.1999.752149.
[17] S.M. Stefanov, Separable Programming: Theory and Methods, vol. 53, Springer Science & Business Media, 2013.
[18] J. Wang, A survey of web caching schemes for the internet, ACM SIGCOMM Comput. Commun. Rev. 29 (5) (1999) 36–46.
[19] L. Wang, S. Bayhan, J. Kangasharju, Optimal Chunking and partial caching in information-centric networks, Comput. Commun. 61 (2015) 48–57.
[20] Wistia, 2016, http://wistia.com/doc/audience-engagement-graph.
[21] K.-L. Wu, P. Yu, J. Wolf, Segmentation of multimedia streams for proxy caching, IEEE Trans. Multimed. 6 (5) (2004) 770–780. ISSN 1520–9210. doi:10.1109/TMM.2004.834870 .
[22] Q. Yang, M.M. Amiri, D. Gündüz, Audience retention rate aware coded video caching, Proceedings of the 2017 IEEE International Conference on Communications Workshops (ICC Workshops), IEEE, 2017, pp. 1189–1194.
[23] Z. Ye, F. De Pellegrini, R. El-Azouzi, L. Maggi, T. Jimenez, Quality-aware dash video caching schemes at mobile edge, Proceedings of the 2017 Twenty-ninth International, Teletraffic Congress (ITC 29), 1 IEEE, 2017, pp. 205–213.
[24] YouTube, 2016, http://support.google.com/youtube/answer/1715160?hl=en-GB.
[25] J. Yu, C.T. Chou, Z. Yang, X. Du, T. Wang, A dynamic caching algorithm based on internal popularity distribution of streaming media, Multimed. Syst. 12 (2) (2006) 135–149.
[26] M. Zeni, D. Miorandi, F. De Pellegrini, YOUStatanalyzer: a tool for analysing the dynamics of YouTube content popularity, Proceedings of the of VALUETOOLS 13, ICST, 2013, pp. 286–289.