Online Coded Caching

Ramtin Pedarsani, Mohammad Ali Maddah-Ali, *Member, IEEE*, and Urs Niesen, *Member, IEEE*

***Abstract*—We consider a basic content distribution scenario consisting of a single origin server connected through a shared bottleneck link to a number of users each equipped with a cache of finite memory. The users issue a sequence of content requests from a set of popular files, and the goal is to operate the caches as well as the server such that these requests are satisfied with the minimum number of bits sent over the shared link. Assuming a basic Markov model for renewing the set of popular files, we characterize approximately the optimal long-term average rate of the shared link. We further prove that the optimal online scheme has approximately the same performance as the optimal offline scheme, in which the cache contents can be updated based on the entire set of popular files before each new request. To support these theoretical results, we propose an online coded caching scheme termed *coded least-recently sent* (LRS) and simulate it for a demand time series derived from the dataset made available by Netflix for the Netflix Prize. For this time series, we show that the proposed coded LRS algorithm significantly outperforms the popular least-recently used caching algorithm.**

***Index Terms*—Coded caching, content distribution, online scheme.**

I. Introduction

THE demand for video streaming services such as those offered by Netflix, YouTube, Amazon, and others is growing rapidly. This places a significant burden on networks. One way to mitigate this burden is to place memories into the network that can be used to cache files that users may request. In this paper, we investigate how to optimally use these caches. In particular, we are interested in *online* algorithms for this problem, in which the operations of the caches have to be performed on the fly and without knowledge of future requests.

The online caching problem (also known as the paging problem in the context of virtual memory systems) has a long history, dating back to the work by Belady in 1966 [1]. This problem has been investigated both for systems with a single cache [1]–[13] as well as for systems with multiple distributed caches [14]–[16]. One solution to the caching problem that is popular in practice and for which strong optimality guarantees can be proved [2], [5], [9], [11]–[13] is the *least-recently used* (LRU) eviction policy. In LRU, each cache is continuously updated to hold the most recently requested files, allowing it to exploit the temporal locality of content requests.

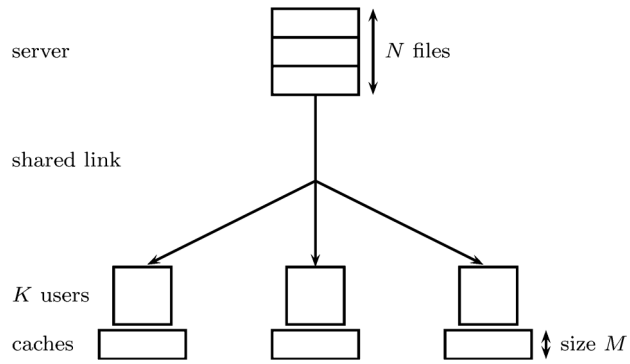

Fig. 1. Caching system considered in this paper. A server containing N files of size F bits each is connected through a shared link to K users each with a cache of size MF bits. In the figure, $N = K = 3$ and $M = 1$.

The figure of merit adopted by the papers mentioned so far is the cache-miss rate (or page-fault rate in the context of the paging problem), sometimes weighted by the file size. This cache-miss rate is used as a proxy for the network load. For systems with a *single* cache, the weighted cache-miss rate and the network load are indeed proportional to each other, and hence minimizing the former also minimizes the latter. However, this proportionality no longer holds for systems with *multiple* caches. For such systems with multiple caches, a fundamentally different so-called *coded caching* approach is required. This coded caching approach has been recently introduced in [17]–[19] for the *offline* caching problem.

In this paper, we investigate *online* coded caching, focusing on a basic content distribution scenario consisting of a single origin server connected through a shared (bottleneck) link to a number of users each equipped with a cache of finite memory (see Fig. 1). The users issue a sequence of content requests from a set of popular files, and the goal is to operate the caches as well as the server such as to satisfy these requests with the minimum number of bits sent over the shared link. We consider the case where the set of popular files evolve according to a Markov model and users select their demand uniformly from this set.

We approximately characterize the optimal long-term average rate of the shared link for this setting. We show further that the optimal online scheme performs approximately the same as the optimal offline scheme. This is perhaps surprising, since in the offline scheme caches are allowed to be updated in an offline fashion each time the set of popular files changes,

Manuscript received February 08, 2014; revised October 13, 2014; accepted December 24, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor T. Javidi. The work of U. Niesen was supported in part by AFOSR under Grant FA9550-09-1-0317. This paper was presented in part at the International Conference on Communications, June 2014.

R. Pedarsani is with the University of California, Berkeley, CA 94706 USA (e-mail: ramtin@eecs.berkeley.edu).

M. A. Maddah-Ali is with Bell Labs, Alcatel-Lucent, Holmdel, NJ 07733 USA (e-mail: mohammadali.maddah-ali@alcatel-lucent.com).

U. Niesen was with Bell Labs, Alcatel-Lucent, Holmdel, NJ 07733 USA. He is now with Qualcomm's New Jersey Research Center, Murray Hill, NJ 07974 USA (e-mail: urs.niesen@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2394482

whereas in the online scheme caches are updated in an online fashion based solely on the limited observations they have through the sequence of requests.[1]

To evaluate the gain of coded caching in practical scenarios, we propose an online coded caching scheme termed *coded least-recently sent* (LRS) and simulate it on a demand time series derived from the dataset made available by Netflix for the Netflix Prize. For this time series, we show that the proposed coded LRS algorithm significantly outperforms the baseline LRU algorithm.

The remainder of this paper is organized as follows. Section II provides background information on coded caching. Section III formally introduces the problem setting. Section IV contains the main results. The proof of these results are provided in Section V.

II. Background on Coded Caching

Coded caching, recently introduced in [17]–[19], is a novel approach to the distributed caching problem. It can achieve a significant reduction in network load by creating and exploiting coded multicasting opportunities between users with different demands. We make essential use of the offline coded caching scheme from [18] in the present paper. Therefore, we now briefly overview that algorithm and illustrate it with an example.

The setting in [18] is the offline version of the one depicted in Fig. 1 in Section I. In particular, a single origin server is connected to K users through a shared link. There is a fixed set of $N \geq K$ files of length F bits, and each user has a memory of size MF bits with $M \leq N$. The cache memories are prefetched in an *offline* fashion during a placement phase (during a period of low network load, say the early morning) so as to minimize the peak load $R(M,N,K)F$ over the shared link during a later delivery phase (say in the evening) during which each user requests a single file. We refer to the normalized peak load $R(M,N,K)$ as the peak rate.

The offline coded caching scheme proposed in [18] achieves, for sufficiently large F, a peak rate of

$$R(M,N,K) \triangleq K \cdot (1-M/N) \cdot \frac{N}{KM}\left(1-(1-M/N)^K\right) \tag{1}$$

which is shown to be within a constant factor of the optimal rate.

In the placement phase of the algorithm in [18], each user caches a random subset of MF/N bits of each of the N files. In the delivery phase, the server sends an appropriate linear combination of those bits over the shared link to enable all users to recover the requested files, as is illustrated in the following toy example.

Example 1 (Decentralized Caching Scheme [18]): Consider the caching problem with $N = 2$ files say A, B and $K = 2$ users, each with a cache of size MF. In the placement phase, each user caches $MF/2$ bits of each file independently at random, satisfying the memory constraint. We partition

$$A = (A_\emptyset, A_1, A_2, A_{1,2})$$

where $A_\mathcal{S}$ denotes the bits of file A that are stored at the users in the set $\mathcal{S}$, and similarly for B. For F sufficiently large, the size of the subfile $A_\mathcal{S}$ tends to $(M/2)^{|\mathcal{S}|}(1-M/2)^{2-|\mathcal{S}|}F$ bits by the law of large numbers.

In the delivery phase, suppose that users one and two request files A and B, respectively. User one already has access to parts A_1 and $A_{1,2}$ of its requested file and needs $A_\emptyset$ and A_2, which are not cached its memory. Similarly, user two already has access to parts B_2 and $B_{1,2}$ of its requested file and needs $B_\emptyset$ and B_1. The server can then satisfy these user requests by sending $A_\emptyset$, $B_\emptyset$, and $A_2 \oplus B_1$ over the shared link, where $\oplus$ denotes the XOR operation applied element-wise to A_2 and B_1 treated as vectors of bits.

Observe that user one has B_1 stored in its cache. From this and the output $A_2 \oplus B_1$ of the shared link, user one can recover the desired file part A_2. Similarly, user two has A_2 stored in its cache and can use this to recover the desired file part B_1 from the output $A_2 \oplus B_1$ of the shared link. Thus, using the contents of their caches and the outputs of the shared link, both users can recover all the required file parts.

The rate over the shared link is

$$\left(\frac{M}{2}\right)(1-M/2) + 2(1-M/2)^2 = R(M,2,2)$$

where $R(M,N,K)$ is defined in (1). While here the delivery phase is explained for a specific set of user requests, one can verify that this rate is achievable for all other possible requests as well. ◊

The main gain from using this scheme derives from the coded multicasting opportunities between users with different demands. These coded multicasting opportunities are created in the placement phase and are exploited in the delivery phase. As the size M of the cache memories increases, this coded multicasting gain increases as well. This gain, called the *global* gain in [17], [18], is captured by the factor

$$\frac{N}{KM}\left(1-(1-M/N)^K\right)$$

in $R(M,N,K)$. There is a second, *local*, gain deriving from having part of the requested file available in a user's local cache. This local gain is captured by the factor

$$(1-M/N)$$

in $R(M,N,K)$. As is shown in [17]–[19], this local gain is usually less significant than the global coded multicasting gain.

III. Problem Setting

We consider a content distribution system with a server connected through a shared, error-free link to K users as illustrated in Fig. 1 in Section I. At time $t \in \mathbb{N}$, each user k requests a file $d_t(k)$ from a time-varying set $\mathcal{N}_t$ of popular files with cardinality $N \geq K$. The K user's requests, collectively denoted

[1]This definition of an offline caching scheme differs from the definition adopted by other papers in the caching literature such as [2]. In those papers, an offline scheme is one that has noncausal knowledge of the entire sequence of demands.

by the vector d_t, are chosen uniformly at random without replacement from $\mathcal{N}_t$. Each file has size F bits, and each user is equipped with a cache memory of size MF bits.

The content distribution system operates as follows. At the beginning of each time slot t, the users reveal their requests d_t to the server. The server, having access to the database of all the files in the system, responds by transmitting a message of size R_tF bits over the shared link. Using their cache contents and the message received over the shared link, each user k aims to reconstruct its requested file $d_t(k)$.

The goal is to design the actions of the users and the server such as to minimize the long-term average rate $\bar{R}$ of the system, i.e.,

$$\bar{R} \triangleq \limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}(R_t), \tag{2}$$

while satisfying the memory and reconstruction constraints. Observe that the rate $\bar{R}$ is the long-term average load $\bar{R}F$ over the shared link normalized by the file size F. In order to obtain a rate $\bar{R}$ independent of the file size and to simplify the analysis, we allow the file size F to be as large as needed.

In this paper, we are interested in *online* caching schemes, which place additional restrictions on the actions of the server and the caches. In such online schemes, the cache content at user k at the beginning of time slot t is a function of the cache content at the same user at the previous time $t-1$, the output of the shared link at time $t-1$, and the requests $d_1, d_2, \ldots, d_{t-1}$ up until time $t-1$. In particular, the cache content may *not* be a function of the outputs of the shared link at times prior to $t-1$. Furthermore, for an online scheme, the message sent by the server over the shared link at time t is a function of only the demands d_t and the cache contents of the users at that time. We define $\bar{R}^\star$ to be the long-term average rate over the shared link of the optimal online caching scheme.

Example 2 (LRU): A popular online caching scheme is LRU. In this scheme, each user caches M entire files. When a user requests a file that is already contained in its cache, that request can be served out of the cache without any communication from the server. When a user requests a file that is not contained in the cache, the server sends the entire file over the link. The user then evicts the least-recently requested (or used) file from its cache and replaces it with the newly requested one.

Observe that this is a valid online caching strategy. Indeed, the content of a user's cache at the beginning of time slot t is a function of the cache content at time $t-1$, the output of the shared link at time $t-1$, and the past requests (in order to determine which file was least-recently used). Moreover, the message sent by the server at time t is only a function of the demands d_t and the cache contents of the users at that time (in order to decide if a file needs to be transmitted at all). We will adopt LRU as the baseline scheme throughout this paper. ◊

We next provide a formal description of the dynamics of the set of popular files $\mathcal{N}_t$. The initial set $\mathcal{N}_1$ consists of N distinct files. The set $\mathcal{N}_t$ at time t evolves from the set $\mathcal{N}_{t-1}$ at time $t-1$ using an arrival/departure process. With probability $1-p$, there is no new arrival and $\mathcal{N}_t = \mathcal{N}_{t-1}$. With probability p, there is a new arrival, and the set $\mathcal{N}_t$ is constructed by choosing a file uniformly at random from $\mathcal{N}_{t-1}$ and replacing it with a new, so far unseen, file. Note that this guarantees that $|\mathcal{N}_t| = N$ for all t.

Remark 1: In our model, all users request from the same set of popular files uniformly at random; thus, they have the same demand distribution at each time slot. While each individual user might have different preferences and demand distribution, the caches that we consider are not actually at the users' end, but they are internal nodes in the network such that each cache supports a large group of users. Thus, the demand distribution from the cache's point of view is the average demand of a group of users, which will be almost identical among all the caches if the number of users supported by each cache is large.

Example 3 (Popular-File Dynamics): Consider a toy system with $N = 2$ popular files. A possible evolution of the set $\mathcal{N}_t$ of popular files is as follows.

$t = 1$: The initial set of popular files is $\mathcal{N}_1 = \{B, C\}$.
$t = 2$: There is an arrival. The file C is randomly chosen and replaced with the new file D, so that $\mathcal{N}_2 = \{B, D\}$.
$t = 3$: There is no arrival, and $\mathcal{N}_3 = \mathcal{N}_2 = \{B, D\}$.

◊

IV. Main Results

We start by introducing a new, online, coded caching algorithm in Section IV-A. A simplified variant of this algorithm is shown in Section IV-B to have performance close to the optimal online caching scheme. Section IV-C provides simulation results comparing the proposed coded caching algorithm to the baseline LRU scheme for an empirical demand time series derived from the Netflix Prize database.

A. An Online, Coded Caching Algorithm

In this section, we propose an online version of the caching algorithm in [18], which we term *coded least-recently sent* (LRS). The coded LRS algorithm is presented formally in the listing Algorithm 1. The statement of the algorithm uses the shorthand $[K]$ for $\{1, 2, \ldots, K\}$.

Algorithm 1 The coded LRS caching algorithm for time t.

procedure DELIVERY
  for $s = K, K-1, \ldots, 1$ **do**
    for $\mathcal{S} \subset [K] : |\mathcal{S}| = s$ **do**
      Server sends $\oplus_{k\in\mathcal{S}} V_{\mathcal{S}\setminus\{k\}}(k)$
    end for
  end for
end procedure
procedure CACHE UPDATE
  for $k, k' \in [K]$ **do**
    if $d_t(k')$ is not partially cached at user k **then**
      User k replaces the least recently sent file in its cache with a random subset of MF/N' bits of file $d_t(k')$
    end if
  end for
end procedure

Algorithm 1 consists of a delivery procedure and a cache update procedure. We now explain those two procedures in detail.

The delivery procedure is formally similar to the delivery procedure of the decentralized caching algorithm in [18]. $V_{\mathcal{S}}(k)$ denotes the bits of the file $d_t(k)$ requested by user k cached exclusively at users in $\mathcal{S}$. In other words, a bit of file $d_t(k)$ is in $V_{\mathcal{S}}(k)$ if it is present in the cache of every user in $\mathcal{S}$ and if it is absent from the cache of every user outside $\mathcal{S}$. The XOR operation $\oplus$ in Line 4 is to be understood as being applied element-wise to $V_{\mathcal{S}}(k)$ treated as a vector of bits. If those vectors do not have the same length, they are assumed to be zero padded for the purposes of the XOR. We thus see that the delivery procedure of the coded LRS algorithm consists of sending one linear file combination for each subset $\mathcal{S}$ of users. It is worth pointing out that, whenever a requested file is not cached at any user, then $V_{\emptyset}(k)$ is equal to the entire requested file, and hence when $\mathcal{S} = \{k\}$ in Line 3 the delivery procedure sends in this case the entire requested file uncoded over the shared link.

Consider next the cache update procedure. In each time slot t, the users maintain a list of

$$N' \triangleq \alpha N$$

partially cached files for some $\alpha \geq 1$. The parameter α can be chosen to optimize the caching performance; a simple and reasonable choice is $\alpha \in (1, 2]$. At time t, after the delivery procedure is executed, the caches are updated as follows. If a requested file $d_t(k')$ of *any* user k' is not currently partially cached, *all* users evict the least-recently used file and replace it with MF/N' randomly chosen bits from file $d_t(k')$. This is feasible since the uncached file $d_t(k')$ was sent uncoded over the shared link during the delivery procedure. Note that this update procedure guarantees that the number of partially cached files remains N', and that the same files (but not necessarily the same bits) are partially cached at each of the K users. Furthermore, the update procedure is helpful to keep track of changes in $\mathcal{N}_t$ for *all* users. In other words, the LRS eviction rule allows all users to learn the distribution of the popular files over time.

We illustrate the proposed coded LRS algorithm with an example. This example also illustrates that the rate of the proposed scheme can be related to the rate $R(M, N, K)$ defined in (1) of the decentralized caching algorithm from [18].

Example 4: (Coded LRS): We consider again a system with $N = 2$ popular files and assume the same popular-file dynamics as in Example 3 in Section III. Assume there are $K = 2$ users with a cache memory of $M = 1$. Let $\alpha = 3/2$ so that each user caches a fraction 1/3 of the bits of $N' = \alpha N = 3$ files. We assume that initially each user partially caches the files $\{A, B, C\}$.

- $t = 1$: The set of popular files is $\mathcal{N}_1 = \{B, C\}$. Assume the users request $d_1 = (B, C)$. Both of the requested files are partially cached at the users. In the delivery procedure, the server sends $B_{\emptyset}$, $C_{\emptyset}$, and $B_2 \oplus C_1$. For F sufficiently large, so that each of these file parts has close to expected size (as discussed in Example 1 in Section II), this results in a rate of

$$\left(\frac{M}{N'}\right)(1 - M/N') + 2(1 - M/N')^2 = R(M, N', 2) = \frac{10}{9}$$

  where $R(M, N, K)$ is as defined in (1). Since all of the requested files are already partially cached, the set of cached files stays the same in the cache update procedure. In other words, each user still partially caches $\{A, B, C\}$.
- $t = 2$: The set of popular files changes to $\mathcal{N}_2 = \{B, D\}$. Assume the users request $d_2 = (B, D)$. Here, file B is partially cached at the users but file D is not. The server sends $B_{\emptyset}$, $D_{\emptyset}$, and $B_2 \oplus D_1$. Since D is not cached at any of the users, we have in this case that $D_{\emptyset} = D$ and $D_1 = \emptyset$. Hence, the transmission of the server is equivalently $B_{\emptyset}$, D and B_2. This results in a rate of

$$\left(\frac{M}{N'}\right)(1 - M/N') + (1 - M/N')^2 + 1 = R(M, N', 1) + 1 = \frac{15}{9}.$$

  Since A is the LRS file, it is evicted from each cache and replaced by a random third of the file D. The new set of partially cached files is $\{B, C, D\}$.
- $t = 3$: The set of popular files stays the same $\mathcal{N}_3 = \{B, D\}$. Assume the users request $d_3 = (D, B)$, both of which are now partially cached at the users. The server now sends $B_{\emptyset}$, $D_{\emptyset}$, and $B_2 \oplus D_1$. Unlike the previous time step, D_1 is now no longer empty, and the resulting rate is

$$R(M, N', 2) = \frac{10}{9}$$

  as calculated before. The set of partially cached files stays the same, namely $\{B, C, D\}$.

$\Diamond$

It is worth comparing the proposed coded LRS algorithm with the well-known LRU algorithm described in Example 2 in Section III. Both of them are online algorithms. However, there are three key differences. First, coded LRS uses a coded delivery procedure whereas the transmissions in LRU are uncoded. Second, coded LRS caches many (N') partial files whereas LRU caches fewer (M) whole files. Third, coded LRS uses a least-recently sent eviction rule, taking into account the files requested by all users jointly, compared to the least-recently used eviction rule, taking into account only the files requested by every user individually. The impact of these differences will be explored in more detail later.

B. Theoretical Results

The main result of this paper is the following theorem for the setting described in Section III with N files and K users each with a cache of size M.

Theorem 1: The long-term average rate $\bar{R}^{\star}$ of the optimal online caching scheme satisfies

$$\frac{1}{12}R(M, N, K) \leq \bar{R}^{\star} \leq 2R(M, N, K) + 6$$

where

$$R(M, N, K) \triangleq K \cdot (1 - M/N) \cdot \frac{N}{KM}\left(1 - (1 - M/N)^K\right).$$

The proof of Theorem 1 is presented in Section V. The upper bound in Theorem 1 results from the analysis of a simplified version of the proposed coded LRS caching scheme, showing that this algorithm is approximately optimal. For the lower bound, we use the rate of the optimal offline scheme, whose rate is approximately $R(M, N, K)$.

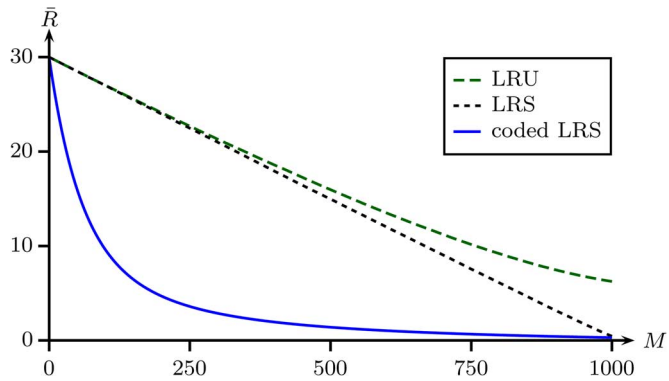

Fig. 2. Performance of various caching approaches for a system with $N = 1000$ popular files, $K = 30$ users, and arrival probability $p = 0.1$. The figure plots the long-term average rate $\bar{R}$ over the shared link as a function of cache memory size M for LRU (dashed green line), uncoded LRS (dashed black line), and the proposed coded LRS (solid blue line).

The theorem implies that the rate of the optimal online caching scheme is approximately the same as the rate of the optimal offline scheme. Recall that, in an offline scheme, the cache memories are given access to the entire set of popular files each time it changes. Moreover, these cache updates are performed offline, meaning that the data transfer needed to update and maintain the caches is not counted towards the load of the shared link. In contrast, in the online scenario, caches are updated based on the limited observations they have through the sequence of demands. Furthermore, the cache updates are performed through the same shared link and therefore affect the average rate. Theorem 1 thus indicates that these significant restrictions for the online problem have only a small effect on the rate compared to the offline scheme.

We now compare the proposed coded LRS scheme to the baseline LRU scheme. The performances of these two schemes are shown in Fig. 2 for a system with $N = 1000$ popular files, $K = 30$ users, and arrival probability $p = 0.1$. As is visible from the figure, coded LRS provides significant gains over LRU both for small and large memory sizes. For example, for $M = 250$ (meaning that the cache is large enough to hold 1/4 of the popular files), LRU results in a rate of 22.7 (meaning that we need to send the equivalent of 22.7 files over the shared link on average), whereas coded LRS results in a rate of 3.6. Similarly, for $M = 1000$, LRU results in a rate of 6.2, whereas coded LRS results in a rate of 0.3.

As mentioned in Section IV-A, the three main differences between coded LRS and LRU are coded delivery, partial caching, and LRS eviction. To get a sense of the impact of these three differences, Fig. 2 also depicts the performance of the uncoded LRS scheme. In this scheme, M whole files are cached and uncoded delivery is used (as in LRU); however, the LRS eviction rule is used (unlike in LRU).

Comparing (uncoded) LRS to LRU, we see that the two schemes perform quite similarly for small and moderate values of M, say $0 \leq M \leq 750$. For large values of M, say $750 < M \leq 1000$, LRS provides a significant improvement over LRU. This is because when M is close to the number of popular files N, the rate is dominated by the arrival of new popular files, and LRS eviction allows the caches to learn these new files with fewer cache misses than LRU.

Comparing uncoded LRS to coded LRS, we see that only when M is very close to the number of popular files N are the performances of the schemes similar. For all other values, coded LRS significantly outperforms uncoded LRS. This implies that, except for large values of M, the main gain of the coded LRS scheme derives from the partial caching of many files and from the coded delivery.

C. Empirical Results

We now evaluate the performance of the proposed coded LRS and the baseline LRU schemes for a real-life time series of demands. Note that our simulation setting does exactly coincide with the theoretical model that we consider; though, it demonstrates similar behavior in the evolution of popular files, and supports and completes our theoretical results in Section IV-B. This time series is derived from the dataset made available by Netflix for the Netflix Prize as follows. Each entry in the Netflix dataset consists of a user ID, a movie ID, a time stamp, and a rating that the user gave to the movie at the specified time. We are not interested in the rating here, but would like to use the time of rating a movie as a proxy for the time of viewing that movie.

This approach is problematic for old movies, which users may rate long after they have seen them. However, it is reasonable to expect that the rating time is close to the viewing time for recently released movies. To ensure that this is the case, we selected all user ratings in the database from the year 2005 (the last full year for which ratings are available), and kept only those that are for movies released in either 2004 or 2005. The resulting filtered time series contains about 10^7 user ratings for 1948 unique movies.

To validate this approach, Fig. 3 plots the number of ratings for the two most-rated movies ("Troy" and "National Treasure") as a function of time measured in weeks. The movie "Troy" was released on DVD on January 4, 2005 (at which time it was likely also available on Netflix), corresponding to week 1. The movie "National Treasure" was released on DVD on May 3, 2005, corresponding to week 18. As can be seen from Fig. 3, the number of ratings increases strongly on the DVD release week, stays relatively high for a number of weeks, and then drops. This suggests that the rating time is indeed a valid proxy for the viewing time when applied to recently released movies. It also suggests that the model of time-varying popular files described in Section III and used for the theoretical analysis in Section IV-B is a reasonable model for the viewing behavior of users.

Fig. 4 compares the performance of the proposed coded LRS scheme to the baseline LRU scheme for the Netflix demand time series for a system with $K = 30$ caches (each here corresponding to many users that are attached to it). The figure shows that coded LRS again significantly outperforms LRU. In particular, for a cache size of $M = 100$, LRU achieves a rate of 10.9 compared with a rate of 4.7 for coded LRS.

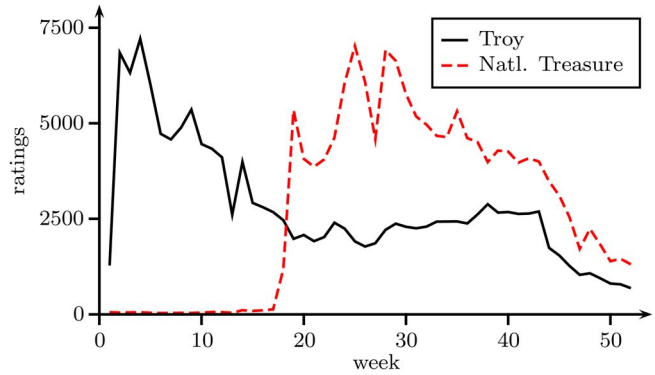

Fig. 3. Number of ratings in the Netflix database for two movies ("Troy" and "National Treasure") as a function of week in 2005.

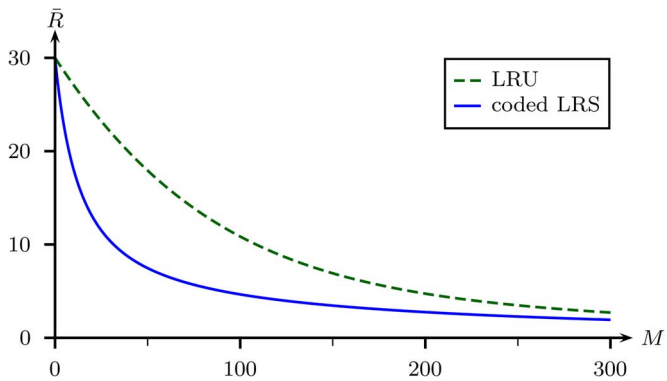

Fig. 4. Performance of various caching approaches for the Netflix demand time series and a system with $K = 30$ caches. The figure plots the long-term average rate $\bar{R}$ over the shared link as a function of cache memory size M for LRU (dashed green line) and the proposed coded LRS (solid blue line).

V. Proof of Theorem 1

To prove Theorem 1, we establish an upper bound (Section V-A) and a lower bound (Section V-B) on the optimal long-term average rate $\bar{R}^\star$.

A. Upper Bound in Theorem 1

For the upper bound on $\bar{R}^\star$, we analyze a simplified version of the proposed coded LRS scheme. We refer to this simplified scheme as *coded random eviction*. In coded random eviction, if Y_t files are requested by users at time t that are not currently partially cached, then Y_t of the cached files are randomly chosen, evicted from *all* cache memories, and replaced with MF/N' randomly chosen bits from each of the Y_t newly requested files. Observe that this guarantees that the same collection of files is partially cached at each user. The remainder of the algorithm is the same as coded LRS as listed in Algorithm 1. In particular, the coded random-eviction algorithm partially caches $N' = \alpha N$ files with $\alpha = 1.4$ at each user, where N is the cardinality of the set of popular files $\mathcal{N}_t$.

Recall that, according to the system model described in Section III, at each time slot t, the users request K randomly chosen files from the set of popular files $\mathcal{N}_t$ without replacement. At any time t, not all files in $\mathcal{N}_t$ may be partially cached at the users. As in the previous paragraph, we denote by Y_t the (random) number of uncached files requested by the users at time t. By definition, Y_t takes value in set $\{0, 1, \ldots, K\}$.

The delivery procedure in Algorithm 1 transmits these Y_t files uncoded over the shared link. To send the remaining $K - Y_t$ files that are partially cached at the users, the delivery procedure of Algorithm 1 uses coding. This requires a rate of $R(M, \alpha N, K - Y_t)$ as described in Section II, and with $R(M, N, K)$ as defined in Theorem 1 and in (1). Thus, the rate R_t over the shared link at time t is

$$\begin{aligned} R_t &= R(M, \alpha N, K - Y_t) + Y_t \\ &\leq R(M, \alpha N, K) + Y_t. \end{aligned}$$

The long-term average rate $\bar{R}$ of coded random eviction is therefore upper bounded by

$$\begin{aligned} \bar{R} &= \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(R_t), \\ &\leq R(M, \alpha N, K) + \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(Y_t). \end{aligned} \tag{3}$$

To prove Theorem 1, we show that the first term in (3) is approximately $R(M, N, K)$ and that the second term is upper bounded by a constant. This second upper bound is perhaps surprising, since Y_t itself can take any value up to K and is hence not upper bounded by a constant independent of the problem parameters.

We start with the analysis of the second term in (3). Let the random variable X_t denote the number of files in $\mathcal{N}_t$ that are partially stored in the caches at the beginning of time slot t. Note that X_t takes value in $\{0, 1, \ldots, N\}$. Conditioned on X_t, the random variable Y_t has expected value

$$\mathbb{E}(Y_t \mid X_t) = K(1 - X_t/N).$$

Therefore

$$\begin{aligned} \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(Y_t) &= \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left(\mathbb{E}(Y_t | X_t)\right) \\ &= \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} K\left(1 - \mathbb{E}(X_t)/N\right) \\ &= K\left(1 - \frac{1}{N} \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(X_t)\right). \end{aligned} \tag{4}$$

In what follows, we investigate the random process $\{X_t\}_{t\in\mathbb{N}}$.

Lemma 2: $\{X_t\}_{t\in\mathbb{N}}$ is a Markov process and has a unique stationary distribution $\pi = (\pi_0, \pi_1, \ldots, \pi_N)$. Moreover

$$\liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(X_t) \geq \mathbb{E}(X)$$

where X is distributed according to π.

Proof: Due to the nature of the random-eviction algorithm, and due to the memoryless arrivals and departures to the set of popular files $\mathcal{N}_t$, $\{X_t\}$ is a Markov process. It is easy to see that this Markov process has a single ergodic recurrent class consisting of the states $\{\lceil K/2 \rceil - 1, \lceil K/2 \rceil, \ldots, N\}$ and has transient states $\{0, 1, 2, \ldots, \lceil K/2 \rceil - 2\}$. Therefore, $\{X_t\}$ has a unique stationary distribution π.

Since X_t can only take non-negative values, from Tonelli's theorem [20] and the properties of lim inf, we have

$$\begin{aligned}\liminf_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}(X_t) &= \liminf_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\sum_{x=0}^{N}x\mathbb{P}(X_t=x)\\ &= \liminf_{T\to\infty}\sum_{x=0}^{N}x\frac{1}{T}\sum_{t=1}^{T}\mathbb{P}(X_t=x)\\ &\geq \sum_{x=0}^{N}x\liminf_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{P}(X_t=x).\end{aligned}$$

Since

$$\lim_{T\to\infty}\mathbb{P}(X_t=x)=\pi_x$$

by ergodicity, we obtain from the Cesàro-mean theorem that

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{P}(X_t=x)=\pi_x.$$

This implies that

$$\sum_{x=0}^{N}x\liminf_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{P}(X_t=x)=\sum_{x=0}^{N}x\pi_x=\mathbb{E}(X),$$

completing the proof. ∎

Applying Lemma 2 to (4) yields that

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}(Y_t)\leq K\left(1-\mathbb{E}(X)/N\right). \tag{5}$$

To establish the upper bound on Theorem 1, it thus remains to lower bound $\mathbb{E}(X)$. This is done in the next lemma.

Lemma 3: Let X be as in Lemma 2. Then

$$K\left(1-\mathbb{E}(X)/N\right)\leq\frac{1}{(1-1/N)(1-1/\alpha)}.$$

Proof: To analyze $\mathbb{E}(X)$, we will need a more detailed understanding of the random process $\{X_t\}$. We define two auxiliary processes W_t, and U_t, both for $t\in\mathbb{N}$. W_t is the number of randomly evicted files from the caches at the end of time slot t that are in $\mathcal{N}_t$. In other words, W_t counts the number of wrongly evicted files. U_t is the number of files that were correctly cached at the end of time slot t but are no longer popular at time $t+1$; that is, $U_t=1$ if at the end of time slot t, there is a departure from the set $\mathcal{N}_t$ of popular files and that the departing file is partially stored in the caches at the end of time slot t (i.e., after the cache update).

Using these auxiliary processes, we can write the following update equation for process X_t:

$$X_{t+1}=X_t+Y_t-W_t-U_t. \tag{6}$$

In words, (6) states that the number of correctly cached files at time $t+1$ is equal to the number X_t of correctly cached files at time t, plus the number of newly requested and cached files Y_t, minus the number of wrongly evicted files W_t, minus the number of files U_t that were correctly cached at the end of time slot t but are no longer popular at time $t+1$.

Example 5: Consider a scenario with $K=3$ users and $N=N'=3$ files. Let $\mathcal{N}_1=\{C,D,E\}$ and assume at time $t=1$ the files $\{A,B,C\}$ are partially cached. Then $X_1=1$, since the overlap is file C. Assume the users request (C,D,E) at time $t=1$. Then $Y_1=2$, since two uncached files D and E are requested. To accommodate the two new files, we randomly evict two cached files. Assume those files are B and C so that the cached files at the end of time slot t are $\{A,D,E\}$. Then $W_1=1$, since file C in $\mathcal{N}_1$ is evicted from the caches. Finally assume that file D is randomly selected to depart from $\mathcal{N}_1$ and is replaced by the new file F, so that $\mathcal{N}_2=\{C,E,F\}$. Then $U_t=1$, since the departed file D is cached at the end of time slot t. Finally, $X_2=1$, since file E is both popular and cached. This satisfies

$$X_2=X_1+Y_1-W_1-U_1.$$

◊

To establish a lower bound on $\mathbb{E}(X)$ we use the update (6) instead of directly computing the stationary distribution π, which is not tractable. Assume that the process $\{X_t\}$ is started in steady-state.[2] In other words, X_t has distribution π for every $t\in\mathbb{N}$. Then, taking expectations on both sides of (6), we have

$$\mathbb{E}(X_{t+1})=\mathbb{E}(X_t)+\mathbb{E}(Y_t-W_t)-\mathbb{E}(U_t).$$

Since $\mathbb{E}(X_t)=\mathbb{E}(X)=\mathbb{E}(X_{t+1})$, this simplifies to

$$\mathbb{E}(U_t)=\mathbb{E}(Y_t-W_t). \tag{7}$$

We now calculate the two expectations in (7). We start with the left expectation. Consider the number of correctly cached files X'_{t+1} at the end of time slot t just after the caches are updated but before any departures from the set of popular files $\mathcal{N}_t$. Note that

$$X'_{t+1}=X_{t+1}+U_t.$$

Conditioned on X'_{t+1}, we obtain

$$\mathbb{P}\left(U_t=1|X'_{t+1}\right)=p\frac{X'_{t+1}}{N}$$

since a departure happens with probability p and since there are N files out of which X'_{t+1} are popular. Now, $U_t\in\{0,1\}$ so that

$$\begin{aligned}\mathbb{E}(U_t)&=\mathbb{E}\left(\mathbb{E}\left(U_t|X'_{t+1}\right)\right)\\ &=\mathbb{E}\left(\mathbb{P}\left(U_t=1|X'_{t+1}\right)\right)\\ &=p\frac{\mathbb{E}\left(X'_{t+1}\right)}{N}\\ &=p\frac{\mathbb{E}(X_{t+1})+\mathbb{E}(U_t)}{N}\\ &=p\frac{\mathbb{E}(X)+\mathbb{E}(U_t)}{N}.\end{aligned}$$

[2]We point out that this assumption is merely a proof technique. The actual caching system itself may not be started in steady state.

Solving for $\mathbb{E}(U_t)$ yields

$$\mathbb{E}(U_t) = \frac{p}{1-p/N}\frac{\mathbb{E}(X)}{N}. \tag{8}$$

We then consider the right expectation in (7). Conditioned on Y_t and X_t, the random variable W_t has the expected value

$$\mathbb{E}(W_t|X_t, Y_t) = Y_t\frac{X_t}{\alpha N}$$

since Y_t files are evicted and X_t of the αN partial files in memory are correctly cached. Thus,

$$\mathbb{E}(Y_t - W_t|X_t, Y_t) = Y_t\left(1 - \frac{X_t}{\alpha N}\right)$$

and

$$\begin{aligned}\mathbb{E}(Y_t - W_t|X_t) &= \mathbb{E}(Y_t|X_t)\left(1 - \frac{X_t}{\alpha N}\right)\\ &= K\left(1 - \frac{X_t}{N}\right)\left(1 - \frac{X_t}{\alpha N}\right).\end{aligned}$$

Finally, the right expectation in (7) can be evaluated as

$$\begin{aligned}\mathbb{E}\left(\mathbb{E}(Y_t - W_t|X_t)\right) &= \mathbb{E}\left(K\left(1 - \frac{X_t}{N}\right)\left(1 - \frac{X_t}{\alpha N}\right)\right)\\ &= \mathbb{E}\left(K\left(1 - \frac{X}{N}\right)\left(1 - \frac{X}{\alpha N}\right)\right).\end{aligned} \tag{9}$$

For ease of notation, define

$$\begin{aligned}\bar{x} &\triangleq \frac{1}{N}\mathbb{E}(X),\\ \sigma^2 &\triangleq \frac{1}{N^2}\mathrm{var}(X),\\ \tilde{p} &\triangleq \frac{p}{1-p/N}\end{aligned}$$

and

$$\beta \triangleq \frac{1}{\alpha}.$$

Substituting (8) and (9) into (7) and rearranging then yields

$$K\beta\bar{x}^2 - \left(\tilde{p} + K(1+\beta)\right)\bar{x} + K(1+\beta\sigma^2) = 0. \tag{10}$$

This is a quadratic equation in the expected value $\bar{x}$ of X/N. In Appendix A, we show that the solutions to this quadratic equation can be lower bounded as

$$\bar{x} \geq \frac{\tilde{p} + 2K\beta - \tilde{p}(1+\beta)/(1-\beta)}{2K\beta}. \tag{11}$$

Observe that, crucially, this lower bound does not depend on the variance σ^2 of X/N. Using this lower bound on $\bar{x}$, we obtain after some algebra

$$\begin{aligned}K\left(1 - \mathbb{E}(X)/N\right) &= K(1-\bar{x})\\ &\leq \frac{\tilde{p}}{1-\beta}\\ &= \frac{p}{(1-p/N)(1-1/\alpha)}\\ &\leq \frac{1}{(1-1/N)(1-1/\alpha)}\end{aligned}$$

concluding the proof of Lemma 3. ■

Applying Lemma 3 to (5) and substituting into (3) shows that

$$\bar{R} \leq R(M, \alpha N, K) + \frac{1}{(1-1/N)(1-1/\alpha)}.$$

In Appendix B, we show that $R(M, \alpha N, K)$ is upper bounded as,

$$2R(M, N, K) + \frac{\alpha - 1}{1 - \alpha/2}. \tag{12}$$

for $\alpha < 2$. Hence,

$$\bar{R} \leq 2R(M, N, K) + \frac{1}{(1-1/N)(1-1/\alpha)} + \frac{\alpha-1}{1-\alpha/2}.$$

Setting $\alpha = 1.4$, we obtain for $N \geq 7$

$$\bar{R} \leq 2R(M, N, K) + 6.$$

On the other hand, for $N \leq 6$, we trivially have

$$\bar{R} \leq 6 \leq 2R(M, N, K) + 6.$$

Since the long-term average of the optimal scheme is less than or equal to $\bar{R}$, this implies

$$\bar{R}^\star \leq \bar{R} \leq 2R(M, N, K) + 6$$

thus concluding the proof of the upper bound in Theorem 1. ■

B. Lower Bound in Theorem 1

We consider an offline scenario in which all cache memories are fully aware of the set of popular files $\mathcal{N}_t$. In addition, at the beginning of each time slot t, before users decide on their requests, the caches are given full access to all the files in $\mathcal{N}_t$ to update their stored content at no cost. However, the cache memories are not aware of future requests. Clearly, the rate of the optimal scheme for this offline setting is a lower bound on the optimal rate for the online setting.

This offline problem is in fact equal to the prefetching problem investigated in [17]–[19], where it is shown that the instantaneous rate, and therefore also the long-term average rate, is lower bounded by $(1/12)R(M, N, K)$. Thus

$$\bar{R}^\star \geq \frac{1}{12}R(M, N, K)$$

as needed to be shown. ■

VI. Conclusion and Discussion

We considered a content distribution system with one server connected through a shared link to K users. We addressed an online caching problem in which users demand a sequence of files from a set of popular files at each time slot, where the set of popular files is changing according to a Markov model. We characterized approximately the optimal long-term average rate of the shared link, and showed that the performance of the optimal online scheme is close to the performance of the optimal offline scheme. Furthermore, we proposed an online coded caching algorithm termed coded least-recently sent (LRS), and demonstrate its great advantage over the conventional LRU scheme via simulations.

We point out that, while we focused here on the basic network configuration depicted in Fig. 1, the coded caching approach can also be applied to more general configurations such as tree networks with caches at the leaves (see [18]) or networks with hierarchical caches (see [21]). In addition, the cache updating scheme proposed in this paper can be used in conjunction with the delivery schemes designed for non-asymptotic file sizes, where each file can be split into only a small number of subfiles and for delay sensitive applications, such as video streaming (both discussed in [22]).

The caching problem is related to the index coding problem [23], [24] (or, equivalently [25], the network coding problem [26]). Indeed, after the cache content has been placed or updated, the following delivery phase in response to a specific set of demands and for fixed cache content induces an index coding problem. The main challenge in caching compared to index coding is that in the cache updating phase, the system is not aware of the upcoming sequence of demands. Therefore, the cache updating must be designed to simultaneously facilitate exponentially many possible sequences of index coding problems, one for each possible future demand. Given that the index coding problem is known to be computationally hard to solve even approximately [27], it is perhaps surprising that the solution to the caching problem proposed in this paper approximately achieves the optimum average rate. For further discussion on the connection between caching and index coding, the reader is referred to [17, Sec. VIII].

Appendix A
Proof of (11)

Set

$$\begin{aligned} a &\triangleq \beta K \\ b &\triangleq -(\tilde{p}+(1+\beta)K) \\ c &\triangleq (1+\beta\sigma^2)K. \end{aligned}$$

Then, (10) can be written as $a\bar{x}^2+b\bar{x}+c=0$ with solutions $(-b\pm\sqrt{b^2-4ac})/2a$. Since $\bar{x}$ is the average of a real random sequence and satisfies the above quadratic equation, this equation has real solutions. In this case, the smaller solution is with the negative sign. Thus

$$\begin{aligned} \bar{x} &\geq \frac{-b-\sqrt{b^2-4ac}}{2a} \\ &= \frac{\tilde{p}+(1+\beta)K-\sqrt{\left(\tilde{p}+(1+\beta)K\right)^2-4\beta(1+\beta\sigma^2)K^2}}{2\beta K} \\ &\geq \frac{\tilde{p}+(1+\beta)K-\sqrt{\left(\tilde{p}+(1+\beta)K\right)^2-4\beta K^2}}{2\beta K} \end{aligned}$$

where, for the last inequality, we have used that $\beta \geq 0$.

Now

$$\begin{aligned} \left(\tilde{p}+(1+\beta)K\right)^2-4\beta K^2 &= \left((1-\beta)K+\frac{1+\beta}{1-\beta}\tilde{p}\right)^2 \\ &\quad +\left(1-\left(\frac{1+\beta}{1-\beta}\right)^2\right)\tilde{p}^2 \\ &\leq \left((1-\beta)K+\frac{1+\beta}{1-\beta}\tilde{p}\right)^2 \end{aligned}$$

where we have again used $\beta \geq 0$. Thus, $\bar{x}$ is lower bounded as

$$\bar{x} \geq \frac{\tilde{p}+2\beta K-\tilde{p}(1+\beta)/(1-\beta)}{2\beta K}.$$

Appendix B
Proof of (12)

Assume first that $N/M \geq 1/(2-\alpha)$. Then

$$\begin{aligned} R(M,\alpha N,K) &= (\alpha N/M-1)\left(1-(1-M/(\alpha N))^K\right) \\ &\leq (\alpha N/M-1)\left(1-(1-M/N)^K\right) \\ &\leq 2(N/M-1)\left(1-(1-M/N)^K\right) \end{aligned}$$

where the last inequality holds since

$$\alpha N/M-1 \leq 2(N/M-1)$$

for $N/M \geq 1/(2-\alpha)$ and $\alpha < 2$. Assume then that $N/M < 1/(2-\alpha)$. Then

$$\begin{aligned} R(M,\alpha N,K) &= (\alpha N/M-1)\left(1-(1-M/(\alpha N))^K\right) \\ &\leq \alpha N/M-1 \\ &\leq \frac{\alpha-1}{1-\alpha/2}. \end{aligned}$$

Combining those two inequalities shows that

$$R(M,\alpha N,K) \leq 2R(M,N,K)+\frac{\alpha-1}{1-\alpha/2}.$$

References

[1] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.

[2] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, pp. 202–208, Feb. 1985.

[3] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *J. Algorithms*, vol. 12, pp. 685–699, Dec. 1991.
[4] L. A. McGeoch and D. D. Sleator, "A strongly competitive randomized paging algorithm," *Algorithmica*, vol. 6, pp. 816–825, 1991.
[5] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive paging with locality of reference," in *Proc. ACM STOC*, 1991, pp. 249–259.
[6] A. R. Karlin, S. J. Phillips, and P. Raghavan, "Markov paging," in *Proc. IEEE FOCS*, Oct. 1992, pp. 208–217.
[7] N. Young, "The *k*-server dual and loose competitiveness for paging," *Algorithmica*, vol. 11, no. 6, pp. 525–541, 1994.
[8] P. Cao and S. Irani, *Cost-Aware WWW Proxy Caching Algorithms*, pp. 193–206, Dec. 1997.
[9] M. Chrobak and J. Noga, "LRU is better than FIFO," in *Proc. ACM-SIAM SODA*, Jan. 1998, pp. 78–81.
[10] N. E. Young, "On-line file caching," in *Proc. ACM-SIAM SODA*, Jan. 1998, pp. 82–86.
[11] E. Torng, "A unified analysis of paging and caching," *Algorithmica*, vol. 20, pp. 175–200, Feb. 1998.
[12] E. Koutsoupias and C. H. Papadimitriou, "Beyond competitive analysis," *SIAM J. Comput.*, vol. 30, pp. 300–317, Jul. 2000.
[13] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz, "On the separation and equivalence of paging strategies," in *Proc. ACM-SIAM SODA*, Jan. 2007, pp. 229–237.
[14] B. Awerbuch, Y. Bartal, and A. Fiat, "Distributed paging for general networks," in *Proc. ACM-SIAM SODA*, Jan. 1996, pp. 574–583.
[15] F. M. auf der Heide, B. Vöcking, and M. Westermann, "Caching in networks," in *Proc. ACM-SIAM SODA*, Jan. 2000, pp. 430–439.
[16] X. Li, C. G. Plaxton, M. Tiwari, and A. Venkataramani, "Online hierarchical cooperative caching," *Theory Comput. Syst.*, vol. 39, pp. 851–874, Nov. 2006.
[17] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.
[18] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. Netw.*, vol. PP, no. 99, 2014, DOI: 10.1109/TNET.2014.2317316.
[19] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *arXiv:1308.0178 [cs.IT]*, Aug. 2013.
[20] D. H. Fremlin, *Measure Theory*, Torres Fremlin, 2001, vol. 2.
[21] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. Diggavi, "Hierarchical coded caching," *Art. ID arXiv:1403.7007 [cs.IT]*, Mar. 2014.
[22] U. Niesen and M. A. Maddah-Ali, "Coded caching for delay-sensitive content," *Art. ID arXiv:1407.4489 [cs.IT]*, 2014.
[23] Y. Birk and T. Kol, "Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2825–2830, Jun. 2006.
[24] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1479–1494, Mar. 2011.
[25] M. Effros, S. El Rouayheb, and M. Langberg, "An equivalence between network coding and index coding," *arXiv:1211.6660 [cs.IT]*, Nov. 2012.
[26] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Apr. 2000.
[27] M. Langberg and A. Sprintson, "On the hardness of approximating the network coding capacity," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 1008–1014, Feb. 2011.

Ramtin Pedarsani received the B.Sc. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2009, and the M.Sc. degree in Communication Systems from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 2011. He is currently working toward the Ph.D. degree at the University of California, Berkeley, CA, USA.

His research interests include coding and information theory, queueing theory, networking, and signal processing.

Mohammad Ali Maddah-Ali (M'14) received the B.Sc. degree from Isfahan University of Technology, Isfahan, Iran, and the M.A.Sc. degree from the University of Tehran, Tehran, Iran, both in electrical engineering, and the Ph.D. degree from the University of Waterloo, Waterloo, ON, Canada, in 2007.

From 2002 to 2007, he was with the Coding and Signal Transmission Laboratory (CST Lab), Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, working toward the Ph.D. degree. From March 2007 to December 2007, he was with the Wireless Technology Laboratories, Nortel Networks, Ottawa, ON, Canada, in joint program between CST Lab and Nortel Networks. From January 2008 to August 2010, he was a Postdoctoral Fellow with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CAU, USA. Since September 2010, he has been with Bell Laboratories, Alcatel-Lucent, Holmdel, NJ, USA, as a Communication Network Research Scientist. His research interests include wireless communications, content delivery networks, and multiuser information theory.

Dr. Maddah-Ali was the recipient of several awards, including Natural Science and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship.

Urs Niesen (S'03–M'09) received the M.S. degree from the École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, in 2005, and the Ph.D. degree from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2009.

From 2009 until 2014, he was a Member of Technical Staff with Bell Labs, Alcatel-Lucent, Holmdel, NJ, USA. Currently, he is with Qualcomm's New Jersey Research Center, Murray Hill, NJ, USA. His research interests are in the general area of information theory with applications in networking and communications.