Coding for Caching: Fundamental Limits and Practical Challenges

Mohammad Ali Maddah-Ali and Urs Niesen

ABSTRACT

Caching is an essential technique to improve throughput and latency in a vast variety of applications. The core idea is to duplicate content in memories distributed across the network, which can then be exploited to deliver requested content with less congestion and delay. The traditional role of cache memories is to deliver the maximal amount of requested content locally rather than from a remote server. While this approach is optimal for single-cache systems, it has recently been shown to be significantly suboptimal for systems with multiple caches (i.e., cache networks). Instead, cache memories should be used to enable a coded multicasting gain. In this article, we survey these recent developments. We discuss both the fundamental performance limits of cache networks and the practical challenges that need to be overcome in real-life scenarios.

INTRODUCTION

Over the last decade, the composition of cellular traffic has shifted from being mainly voice to being mainly content. Unlike voice, which is generated in real time just ahead of transmission, content is typically generated well ahead of transmission. This pre-generation of content allows it to be cached throughout the network (either directly in mobile phones or in small cell base stations) during periods of low network utilization. During periods of high network utilization, this cached content can then be used to reduce network load. Such caching of content is expected to become a central component of future cellular network architectures [1].

Fundamental limits on the performance of systems with a single cache were developed in the computer science community mainly in the 1980s and '90s. The main gain of caching in such single-cache systems derives from the local delivery of content from a nearby cache. However, this theory of single-cache systems does not apply to systems with multiple caches (i.e., cache networks). It is precisely such cache networks that are relevant in the context of next-generation cellular systems.

It is only very recently that a fundamental understanding of such cache networks has been obtained. This magazine article surveys these recent developments. We start by introducing a basic model for a cache network, consisting of a single server connected through a broadcast channel to a number of caches. An analysis of this network reveals that, in addition to the local delivery gain available in single-cache systems, cache networks allow a second, global, caching gain. This global caching gain is exploited through coded multicast transmissions from the server that are simultaneously useful for several users. Unlike the local gain, this global gain scales with the number of caches in the network. An information-theoretic argument shows that there are no other caching gains scaling with the system parameters. Thus, this global gain is a fundamental quantity for the basic cache network, and coded multicast transmission is a fundamental technique for this network.

These results demonstrate that coding plays a key role in the optimal operation of cache networks. However, their derivation relies on a number of stylized assumptions (e.g., simultaneous user demands, uniform content popularity, offline cache updating, and broadcast network topology). We discuss how these assumptions can be relaxed. We review a flexible decentralized coded caching approach that allows for caches to be populated independent of each other. Building on this decentralized coded caching scheme, we discuss how to handle scenarios with asynchronous user demands, nonuniform content popularity, and online cache updating. We also provide a brief overview of how these results extend to more general network topologies. Finally, we discuss a recent video streaming prototype using coded caching, demonstrating the applicability of these concepts in a real-life setting.

CODING FOR CACHING: FUNDAMENTAL LIMITS

To highlight the fundamental nature of coding for caching, we focus on a basic network model introduced in [2]. This network consists of a server connected through a shared bottleneck link to K users, as shown in Fig. 1. The server holds N files (e.g., movies) each of size F bits. Each user is equipped with an isolated cache of size MF bits for some real number $M \in [0, N]$. For simplicity of exposition, we assume $N \geq K$.

This setting can model a wireless network with a transmitter (e.g., a WiFi access point or a cellular base station) and several users (cell phones, laptops), all sharing the common wire-

The traditional role of cache memories is to deliver the maximal amount of requested content locally rather than from a remote server. While this approach is optimal for single-cache systems, it has recently been shown to be significantly suboptimal for systems with multiple caches (i.e., cache networks). Instead, cache memories should be used to enable a coded multicasting gain.

Mohammad Ali Maddah-Ali is with Nokia Bell Labs; Urs Niesen is with the Qualcomm New Jersey Research Center.

0163-6804/16/$25.00 © 2016 IEEE

This setting can model a wireless network with a transmitter (e.g., a WiFi access point or a cellular base station) and several users (cellphones, laptops), all sharing the common wireless medium. The caching could take place directly in the end-user equipment. Alternatively, the caching could take place in femtocells close to the users.

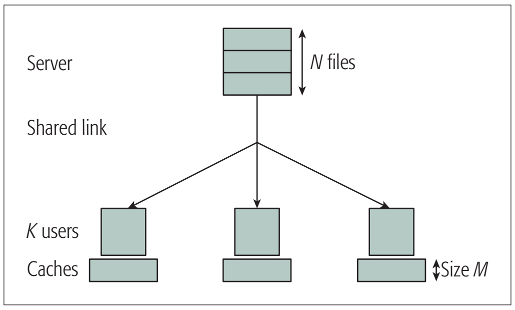


Figure 1. A basic cache network from [2]: A server with a library of N files is connected to K users through a shared link. Each user has access to an isolated cache holding up to the equivalent of M files. The objective is to design the placement phase (how to populate the caches) and the delivery phase (what to send from the server) in order to minimize the load of the shared link. In the figure, $N = K = 3$ and $M = 1$.

less medium. The caching could take place directly in the end-user equipment. Alternatively, the caching could take place in femtocells close to the users. The basic cache network setting can also model a wired network with several caches connected to a common server. In this wired scenario the shared link models a bottleneck along the path between the server and the users. As we see later, the intuition developed for this basic cache network carries over to more complicated network topologies.

We assume that the system operates in two phases: a *content placement phase* and a *content delivery phase*. The placement phase is executed when traffic is low, and network resources are cheap and abundant (e.g., in the early morning). In this phase, each cache prefetches data from the central server subject to the memory limit of MF bits. The key assumption is that the system is not aware of the users' future demands, and therefore, the placement phase cannot be a function of those demands. The delivery phase occurs after the placement phase, when traffic is high and network resources are scarce and expensive (e.g., in the evening). At the beginning of this phase, each user reveals its request for one of the N files. The server is informed of these K requested files. In response, the server sends RF bits (or the equivalent of R files) over the shared link. The number R is called the *rate* of the server transmission. From the server transmission and its local cache content, each user needs to be able to recover its requested file.

We can design *both* the content placement and delivery phases. The objective is to minimize the rate R with which every possible set of user demands can be satisfied. The constraints are the memory limit during content placement and the recovery requirement during content delivery.

As a baseline for future comparison, we start by reviewing a conventional uncoded caching scheme.

Example 1 (*uncoded caching*). In this solution, each user caches the first M/N fraction of the bits of each file during the placement phase. Since there are a total of N files, this respects the memory constraint at each user. The motivation for this content placement is that the system should be ready for any possible set of demands. Thus, each user should dedicate the same fraction of its memory to each file.

During the delivery phase, each user requests one of the N files. Recall that each user already has a fraction M/N of its requested file stored in its local cache. Thus, the server only needs to transmit the remaining $1 - M/N$ fraction of the requested file for the user to be able to decode. Since there are K users in the system, the rate of this scheme adds up to

$$R_U(M) \triangleq K \cdot (1 - M/N). \qquad (1)$$

For a specific example, consider the case with $K = N = 2$ users and files, and with memory size $M = 1$. Then $R_U(M) = 1$.

The rate-memory trade-off $R_U(M)$ for this caching scheme has two terms. The first term, K, is the rate of the network in the absence of caches. The second factor, $1 - M/N$, appears because an M/N fraction of the requested file is available locally. We call this second factor in Eq. 1 the *local caching gain*.

Since neither the placement phase nor the delivery phase of this scheme uses coding, we refer to it as *uncoded caching*. In this uncoded scheme, the role of caching is to deliver part of the requested files *locally*.

We have selected the uncoded caching scheme in Example 1 for simplicity of exposition. There is a long list of other uncoded caching approaches developed for different applications and scenarios. Perhaps the best known of these are least recently used (LRU) and least frequently used (LFU).

All of these uncoded caching schemes share the following basic properties:
- The main role of caching is to deliver part of the content locally. As a consequence, maximizing the hit rate (i.e., the chance of local delivery) is often chosen as the objective.
- For isolated private caches, each user can only derive a caching gain from its own cache.
- Caches that observe statistically similar demands should cache similar content.

These common sense principles lead to efficient solutions for systems with a *single* cache. However, as we see next, these principles do not carry over to networks with *multiple* caches. Indeed, we show that local delivery achieves only a small fraction of the gain that cache networks can offer. Cache memories, if populated correctly, can offer additional gains that scale with the size of the network. We explain the main idea with a toy example from [2].

Example 2 (*coded caching*, $K = N = 2$, $M = 1$). Consider a system with $K = 2$ users, each with a cache large enough to store one file so that $M = 1$. Assume that the server has $N = 2$ files, A and B. We split each file into two non-overlapping subfiles of equal size, $A = (A_1, A_2)$ and $B = (B_1, B_2)$. In the placement phase, instead of placing the same content in all caches, we place different content pieces at the users' caches as shown in Fig. 2. In particular, user one caches (A_1, B_1) and user two caches (A_2, B_2).

For the delivery phase, let us consider a generic case in which user one requests file A and user two requests file B as depicted in Fig. 2a. From the placement phase, user one already has subfile A_1 but is missing A_2, and user two already has subfile B_2 but is missing B_1. Thus, similar to the uncoded approach, half of the requested file can be delivered locally. The server could send the missing parts A_2 and B_1 with total rate of 1. However, as we shall see, the particular pattern of content placement creates an opportunity for reducing the transmission rate through coding.

We note that A_2 required at user one is available in cache two. Similarly, B_1 required at user two is available in cache one. Unfortunately, the two users are unable to exchange these file parts since their caches are isolated. Instead, the server can exploit this situation by transmitting $A_2 \oplus B_1$ over the shared link, where $\oplus$ denotes bitwise XOR. User one can recover A_2 from the received signal $A_2 \oplus B_1$ and its cache content B_1. Similarly, user two can recover B_1 from the received signal $A_2 \oplus B_1$ and its cache content A_2. This is shown in Fig. 2a. Since the coded signal $A_2 \oplus B_1$ is simultaneously useful for both users, the load of the shared link is reduced by a factor of 2 compared to the uncoded approach. Note that this multicasting opportunity is created for the two users even though they ask for different files.

As shown in Fig. 2, although the content placement does not depend on the future requested files, the above coded multicasting opportunity is available for all four possible demand tuples. The required rate is 0.5 in each case. Thus, the rate of the coded scheme in this example is $R_C(M) = 0.5$. This compares to the rate $R_U(M) = 1$ for the uncoded caching scheme from Example 1 with the same parameters.

Example 2 shows that the role of caching is not limited to local delivery. In particular, careful placement of content not only allows local delivery of part of the content but in addition creates multicasting opportunities through coding. In Example 2, this coded multicasting opportunity reduces the load of the shared bottleneck link by an additional factor of 2.

The rate-memory trade-off for coded caching for arbitrary number of files N, number of uses K, and local cache size M has been derived in [2]. For ease of exposition we state the result here for the case $K \leq N$. It is shown that for local cache size of $M \in \{0, N/K, 2N/K, \ldots, N\}$, coded caching achieves the rate

$$R_C(M) \triangleq K \cdot (1 - M/N) \cdot \frac{1}{1 + KM/N}. \qquad (2)$$

For general $0 \leq M \leq N$, the lower convex envelope of these points is achievable.

Comparing the rate expression $R_C(M)$ in Eq. 2 for the coded scheme with the rate expression $R_U(M)$ in Eq. 1 for the uncoded scheme, we see that the first two factors are the same. This means that both caching schemes enjoy the gain of local delivery quantified by the factor $1 - M/N$. However, the coded scheme alone enjoys an additional gain of a factor

$$\frac{1}{1 + KM/N}.$$

This additional factor quantifies the gain of creating and exploiting coded multicasting opportunities.

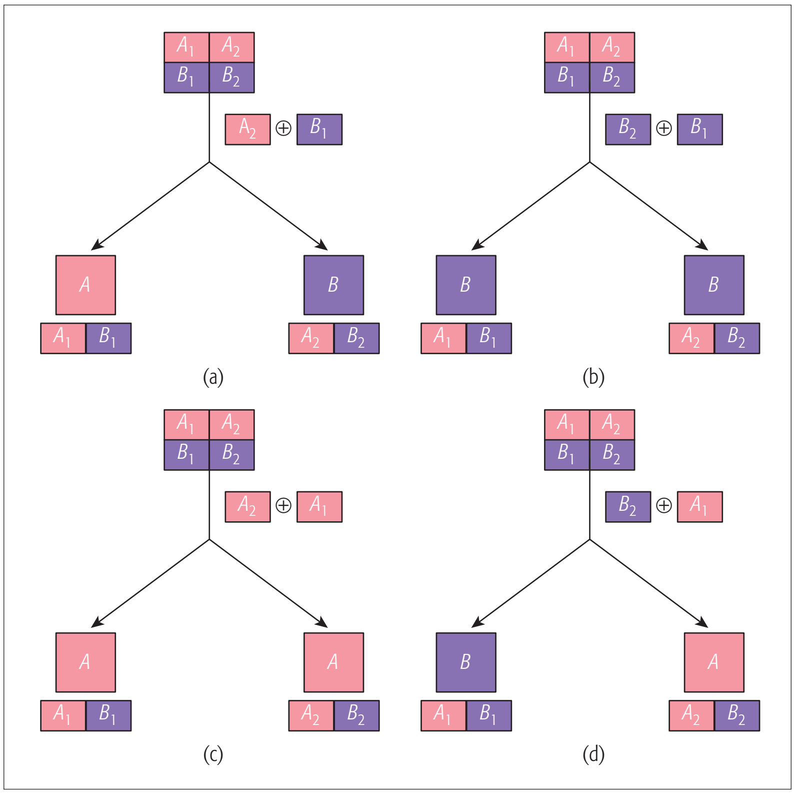


Figure 2. (Centralized) coded caching strategy for $K = 2$ users, $N = 2$ files, and cache size $M = 1$. Each file is split into two subfiles of size 1/2 (e.g., $A = (A_1, A_2)$). User 1 caches (A_1, B_1) and user two caches (A_2, B_2). The delivery phase uses coding to satisfy two user demands with a single transmission. This coded-multicasting opportunity is available simultaneously for all four possible demand tuples shown in the four subfigures: a) demand (A, B); b) demand (B, B); c) demand (A, A); d) demand (B, A).

This gain is enabled by deliberately placing content in order to allow the server to satisfy $1 + KM/N$ (partial) user demands with a single coded multicast transmission. Thus, these users enjoy the gain of multicasting even though they ask for *different* demands. This coded multicasting opportunity is available simultaneously for every one of the N^K possible user demand vectors.

Coding gain is global

Unlike the local caching gain, which derives from the local cache at a single user, the coding gain inherently derives from several caches at several users. Motivated by this observation, we refer to the coding gain as the global caching gain.

Observe that the local caching gain

$$1 - M/N \approx \frac{1}{1 + M/N}$$

is significant when the *local* cache size M is comparable to the size of the entire content N. In contrast, the global caching gain

$$\frac{1}{1 + KM/N}.$$

is significant when the *cumulative* cache size KM is comparable to the size of the entire content N. Thus, by employing coding, we make approximately K times better use of the available memory resources.

Coding also results in better use of the com-

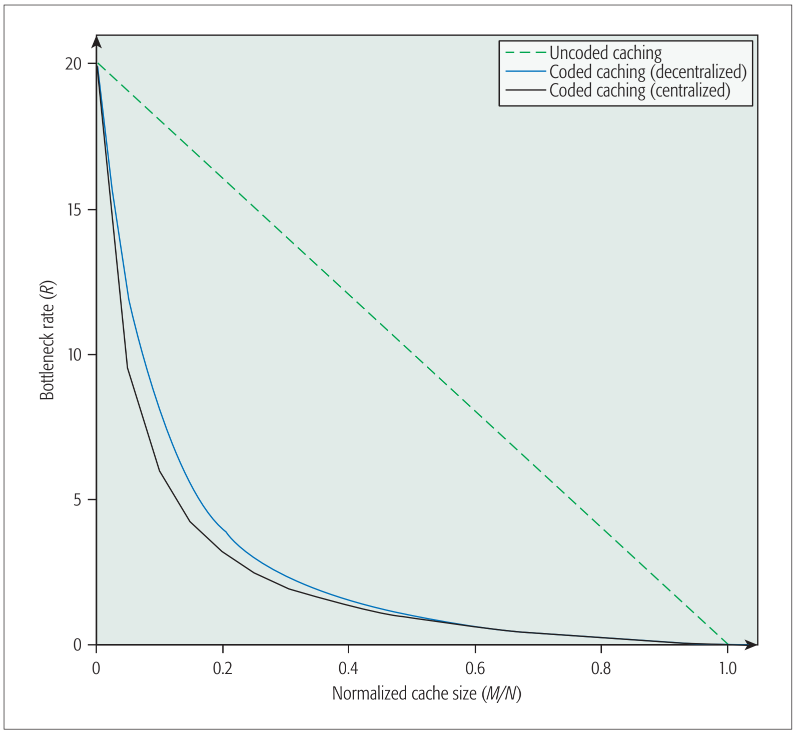


Figure 3. Rate R in the delivery phase vs. the normalized memory size M/N for $K = 20$ users from [2, 3]. The figure compares the performance of the coded caching schemes (centralized and decentralized) to that of conventional uncoded caching.

munication bandwidth. Note that for the uncoded scheme, the rate $R_U(M)$ in Eq. 2 increases linearly with the number of users K. On the other hand, for large numbers of users, $R_C(M)$ in Eq. 2 can be approximated by $R_C(M) \approx N/M$. In other words, for the coded scheme the load of the bottleneck link does not increase (significantly) with K. Thus, by employing coding, we make approximately K times better use of the available bandwidth resources. We summarize these two observations as follows:

Coding gain scales with network size

Unlike the gain of local delivery, the global caching gain due to coding scales with the number of caches in the network.

Figure 3 shows the rate-memory trade-off for $K = 20$ users for both the uncoded and coded schemes. To compare the two schemes, let us focus on the case where each user has space to cache half of the content (i.e., $M/N = 0.5$). In this case uncoded caching reduces the load of the shared link from 20 files down to the equivalent of 10 files. In contrast, for the (centralized) coded caching scheme, the load of the bottleneck link is less than the equivalent of just a single file.

It can be shown that the rate-memory trade-off $R_C(M)$ achieved by coded caching is within a constant factor of the information-theoretic optimum for all values of the problem parameters [2]. This implies that the local and global caching gains quantified above are *fundamental*: there are no other gains that scale with the system parameters.

Remark: The coded caching problem described in this section is related to the network coding [4] and index coding [5] problems. We refer the reader to [2] for a discussion of this connection.

Coding for Caching: Practical Challenges

The analysis of the basic cache network indicates that coding plays a crucial role in the operation of cache networks and in attaining their fundamental performance limits. To apply coded caching in real systems, several practical challenges need to be addressed. These include handling asynchronous demands, dealing with nonuniform content distributions, updating cache contents online, delivering delay-sensitive content such as video streaming, and dealing with more general network topologies, among others. These issues are currently subject to extensive research activity in both academia and industry. Here we present some of the solutions.

A key tool to deal with several of these challenges is a more flexible content placement scheme developed in [3]. This scheme uses decentralized content placement, in which the placement phase is independent of the number and identity of users. We explain the main idea using a toy example from [3]. We then outline how this decentralized scheme can help address some of the practical challenges raised above.

Example 3 (*decentralized coded caching*, $K = N = 3$). Consider the caching problem with $N = 3$ files, say A, B, C, and with $K = 3$ users, each with a cache of size MF bits. In the placement phase, each user caches MF/N bits of each file uniformly at random, independently of other caches. Clearly this placement satisfies the memory constraints. The most important feature of this placement scheme is that each cache is populated independent of the number and identity of the other users in the system.

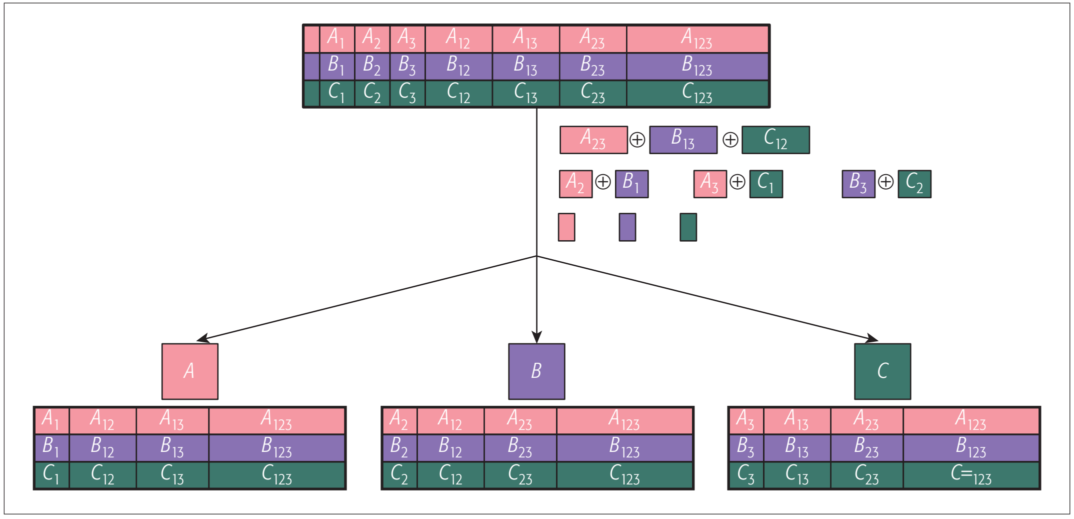
In the delivery phase, consider a generic request tuple with users one, two, and three requesting files A, B, and C, respectively. Let us partition file A into eight subfiles,

$$A = (A_\varnothing, A_1, A_2, A_3, A_{12}, A_{13}, A_{23}, A_{123}),$$

where A_S denotes the bits of file A that are stored exclusively at users in the set $\mathcal{S}$, and similarly for B and C. Based on this partitioning, the content of each cache can be expressed as shown in Fig. 4. For delivery, the server uses greedy linear coding as follows. It first sends the coded packet $A_{23} \oplus B_{13} \oplus C_{12}$, which is simultaneously useful to all three users (as is easily verified by examining the cache contents). It then sends coded packets that are useful for two users, $A_2 \oplus B_1$ for users one and two, $A_3 \oplus C_1$ for users one and three, and $B_3 \oplus C_2$ for users two and three. Finally, the server sends $A_\varnothing$, $B_\varnothing$, and $C_\varnothing$, each useful to only one user.

Even though the placement phase is independent of the number of users K, the rate of this decentralized coded caching scheme can be shown to be within a constant factor of universally optimal for any number of users K [3]. The rate of the decentralized coded caching scheme is also very close to the rate of the centralized coded caching scheme, as can be seen in Fig. 3.

The universality property of the decentral-



A_1 A_2 A_3 A_{12} A_{13} A_{23} A_{123}
B_1 B_2 B_3 B_{12} B_{13} B_{23} B_{123}
C_1 C_2 C_3 C_{12} C_{13} C_{23} C_{123}
$A_{23} \oplus B_{13} \oplus C_{12}$
$A_2 \oplus B_1$
$A_3 \oplus C_1$
$B_3 \oplus C_2$
A
B
C
A_1 A_{12} A_{13} A_{123}
B_1 B_{12} B_{13} B_{123}
C_1 C_{12} C_{13} C_{123}
A_2 A_{12} A_{23} A_{123}
B_2 B_{12} B_{23} B_{123}
C_2 C_{12} C_{23} C_{123}
A_3 A_{13} A_{23} A_{123}
B_3 B_{13} B_{23} B_{123}
C_3 C_{13} C_{23} $C{=}_{123}$

Figure 4. Decentralized coded caching scheme for $K = 3$ users, $N = 3$ files, and cache size M. In the placement phase, each user caches $MF/3$ bits of each file independently and uniformly at random. The delivery phase uses greedy linear coding.

Before attempting delivery, the server identifies the set of active users sharing a bottleneck link. It then employs the delivery scheme of decentralized coded caching for that set of users. The efficiency of this approach is guaranteed by the universality of the decentralized content placement.

ized scheme is the key to address the problem of asynchronous user requests. It also makes the decentralized scheme a key ingredient in handling online and nonuniform demands, as briefly discussed next.

Asynchronous Demands and Dynamic Networks: User demands for content are typically asynchronous. Furthermore, in some applications such as cellular networks, users may move from one "network" (i.e., one cell) to another. As a consequence, the number and identity of users sharing a bottleneck link can change continuously. The universality of the decentralized coded caching scheme allows us to efficiently handle these cases. Before attempting delivery, the server identifies the set of active users sharing a bottleneck link. It then employs the delivery scheme of decentralized coded caching for that set of users. The efficiency of this approach is guaranteed by the universality of the decentralized content placement.

Nonuniform Demands: Both the centralized and decentralized coded caching schemes are approximately optimal for content with uniform popularity [6]. However, in practice, the content files often have popularities varying over several orders of magnitude. For such nonuniform distributions, [6] suggests splitting the set of files into several groups with similar popularity within each group. A fraction of memory is then dedicated to each group. The placement and delivery phases of the decentralized coded caching scheme are then applied within each group of files. We note that the number of users requesting files from a particular file group is random and not known during the placement phase. Nevertheless, the universality of the decentralized caching scheme guarantees the efficiency of this file grouping approach. It is shown in [7, 8] that just two groups are sufficient to achieve performance within a constant factor of optimality.

Online Cache Updating: Not only is some content more popular than others; the set of popular files is also continuously changing. To maintain efficiency, the cache contents should be adapted accordingly. This is usually done by employing online caching schemes, in which the cache contents are modified at the end of the delivery phase. While the popular least recently used (LRU) cache updating rule is approximately optimal for single-cache systems, it can be significantly suboptimal for cache networks [9]. In an alternative approach proposed in [9], the server has two delivery modes. It uses coded delivery for the requested files that are already partially cached. For the files that are not partially cached, the server uses uncoded delivery. Each cache makes use of such uncoded delivery by evicting some old file parts and replacing them with randomly chosen parts of the newly delivered file. Since the number of requests from the cached files are random, the universality of decentralized coded caching is again used here.

The ideas developed for the basic cache network topology can be extended to other networks such as tree networks [3], hierarchical cache networks [10], device-to-device communication [11], cache-aided interference channels [12, 13], and channels with erasures [14] (Fig. 5).

To demonstrate the gains of caching in a practical setting, [15] describes a prototype implementation applying coded caching to the problem of video streaming. A screenshot of this prototype is shown in Fig. 6a. In addition to the techniques described so far, several other issues arise in this context, such as protocol overhead, state synchronization, delay constraints, and computational limitations, as discussed next.

Protocol Overhead: Header information is required for the users to know which file bits are coded together in each received packet. In order for this protocol overhead to be manageable, each file is split into chunks of size 10 kB. For example, video3.flv[3689] in Fig. 6a refers to chunk 3689 of file video3.flv. The decentralized content placement and delivery schemes described earlier are then applied at the chunk level rather than at the bit level. For example, line 2 in Fig. 6a shows the decoding operation from chunk video3.flv[3691] XORed with chunk video2.flv[5452].

State Synchronization: In order to form the correct coded transmissions, the server needs to know the state of the user caches. A convenient way to achieve this state synchronization is to use a pseudo-random number generator for the decentralized content placement. If a user with already populated cache connects to a server, it can simply communicate the seed number of the pseudo-random number generator used for the content placement to the server. From this seed, the server can then recompute the entire state of the user cache.

Delay Constraints: Video streaming applications have tight delay constraints, typically on the order of 30 s. To meet these constraints, instead of requesting entire files, users instead request individual file chunks. At the server, the requested chunks are kept in an ordered queue according to their delivery deadline. Coding opportunities are only exploited among the chunks in the queue. Whenever the deadline is up for a particular chunk, it gets coded with other chunks in the queue and delivered to the users.

Computational Limitations: Finding the optimal coding opportunities among the chunks in the queue is computationally intractable. Instead, a greedy coding scheme is used, which has complexity that is only linear in the queue length.

A trace of the coding gain (defined as the ratio of the rate of uncoded and the rate of coded caching) corresponding to Fig. 6a is shown in Fig. 6b. In this trace, the number of users is increased from one to four with an interval of 30 s. Each time a new user joins, the coding gain increases as the system is able to exploit coding opportunities among a larger number of users. Also shown in dotted lines are the theoretical values of the coding gain. The figure indicates that the additional protocol overhead and other losses due to delay constraints and computational limitations are modest in this scenario.

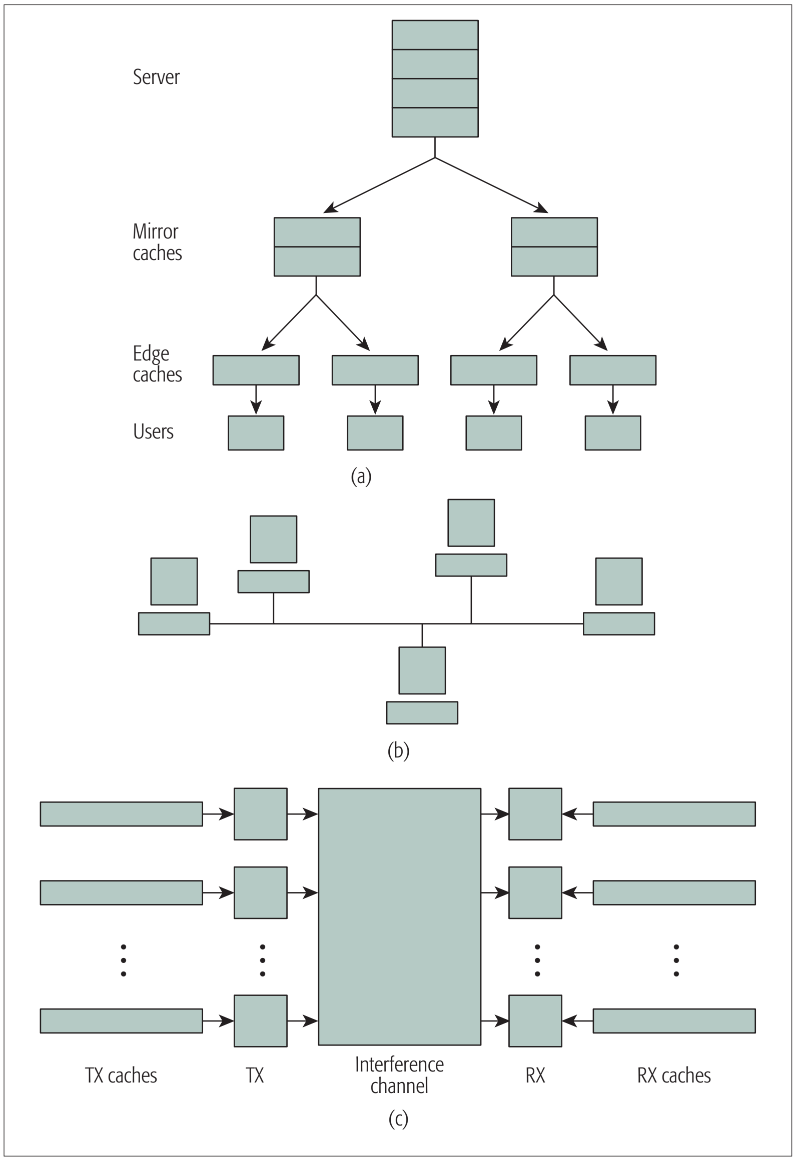


Figure 5. Other cache network topologies: a) hierarchical cache networks [10]; b) device-to-device [11]; c) cache-aided interference channels [12, 13].

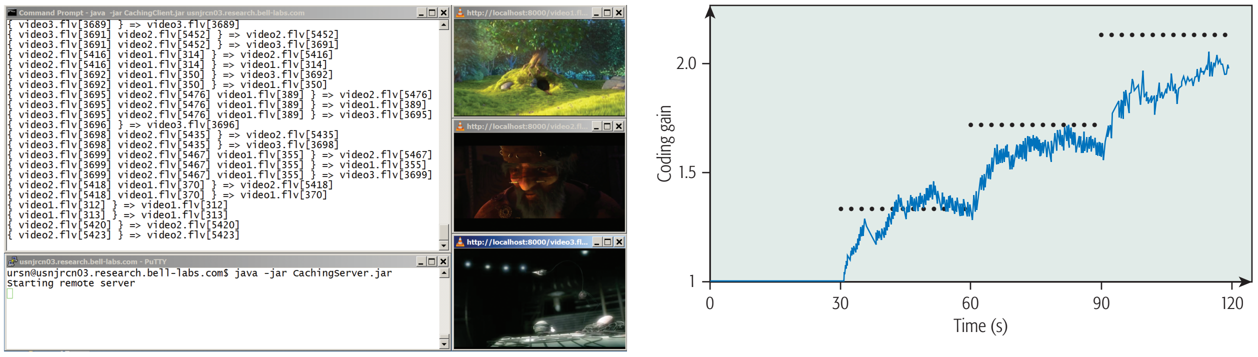
Open Problems

The results presented so far demonstrate the promise of coded caching. There are many open problems left to be investigated.

Sharpening of Approximations: Characterizing the exact optimal rate-memory trade-off or at least sharpening the approximations is of interest. We have seen that uncoded content placement combined with linearly coded content delivery is sufficient for achieving a rate within a constant factor of optimal for the canonical broadcast cache network. One important open question is if nonlinear codes are needed to exactly achieve the optimal rate-memory trade-off. Another open question concerns the value of coded content placement. In [2], it is shown that placing coded content in the user caches can improve the rate for small cache sizes. However, it is still unknown if coded content placement is required to exactly achieve the rate-memory trade-off for large cache sizes. Conversely, it is known that the current bounds are not tight in general and need to be improved.

General Cache Networks: Characterizing (either exactly or approximately) the rate-memory trade-off for general cache networks is of great interest, but expected to be difficult. The reason is that the related multiple-unicast problem over general networks is known to be challenging; for example, nonlinear coding can offer unbounded performance improvement, and Shannon inequalities are insufficient to prove optimality. Alternatively, one may focus on specific topologies that are relevant in practical settings (e.g., layered networks) or restrict the class of solutions (e.g., only allowing linear codes or only allowing coding at some of the network nodes).

Impact of Other Constraints: As we have seen in the context of the video streaming prototype, in many applications we may encounter additional constraints. In particular, we have seen constraints on the protocol overhead, state synchronization overhead, delivery delay, and computational resources. Other such constraints are on the number of sub-packets per file and the number of disk reads at either the server or the caches. Here we have presented heuristic solutions to handle some of these constraints; deriving fundamental limits of caching under these additional constraints is an important, largely open, question.



```
{ video3.flv[3689] } => video3.flv[3689]
{ video3.flv[3691] video2.flv[5452] } => video2.flv[5452]
{ video3.flv[3691] video2.flv[5452] } => video3.flv[3691]
{ video2.flv[5416] video1.flv[314] } => video2.flv[5416]
{ video2.flv[5416] video1.flv[314] } => video1.flv[314]
{ video3.flv[3692] video1.flv[350] } => video3.flv[3692]
{ video3.flv[3692] video1.flv[350] } => video1.flv[350]
{ video3.flv[3695] video2.flv[5476] video1.flv[389] } => video2.flv[5476]
{ video3.flv[3695] video2.flv[5476] video1.flv[389] } => video1.flv[389]
{ video3.flv[3695] video2.flv[5476] video1.flv[389] } => video3.flv[3695]
{ video3.flv[3696] } => video3.flv[3696]
{ video3.flv[3698] video2.flv[5435] } => video2.flv[5435]
{ video3.flv[3698] video2.flv[5435] } => video3.flv[3698]
{ video3.flv[3699] video2.flv[5467] video1.flv[355] } => video2.flv[5467]
{ video3.flv[3699] video2.flv[5467] video1.flv[355] } => video1.flv[355]
{ video3.flv[3699] video2.flv[5467] video1.flv[355] } => video3.flv[3699]
{ video2.flv[5418] video1.flv[370] } => video2.flv[5418]
{ video2.flv[5418] video1.flv[370] } => video1.flv[370]
{ video1.flv[312] } => video1.flv[312]
{ video1.flv[313] } => video1.flv[313]
{ video2.flv[5420] } => video2.flv[5420]
{ video2.flv[5423] } => video2.flv[5423]
```

```
ursn@usnjrcn03.research.bell-labs.com$ java -jar CachingServer.jar
Starting remote server
```

Figure 6. Video-streaming prototype from [15]: a) screenshot showing the decoding process; b) trace of coding gain for caches storing half the file library ($M/N = 0.5$). The number of users K changes from 1 to 4 in 30 second intervals.

Cache-Aided Wireless Communication: For the wireless broadcast channel, we have seen that caching can offer a coded multicasting gain. For other wireless scenarios, caching can offer additional gains. For the interference channel, caching at the transmitters offers interference cancellation and interference alignment gains [12]. When caches are also present at the receivers, we can in addition exploit the coded multicasting gain seen earlier [13]. Cataloguing what caching gains are available in other wireless scenarios is of interest. Characterizing the capacity region (or at least the degree of freedom region) of cache-aided multi-user wireless channels is also an important open question. System-level evaluations (either analytical or based on simulations) taking into account the backhaul load, queue stability, and control signal overhead could provide a better understanding of the performance of cache-aided wireless networks in practice.

Conclusions

In this article, we have argued that coding plays a crucial role in the efficient operation of cache networks. In particular, coding can offer an improvement of network load that scales with the network size. We have also demonstrated that this gain is achievable in the presence of practical system constraints and for different network topologies. Finally, we have presented numerous open problems, both theoretically and practically motivated, that need to be addressed.

References

[1] N. Golrezaei *et al.*, "Femtocaching: Wireless Video Content Delivery through Distributed Caching Helpers," *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1107–15.

[2] M. A. Maddah-Ali and U. Niesen, "Fundamental Limits of Caching," *IEEE Trans. Info. Theory*, vol. 60, May 2014, pp. 2856–67.

[3] M. A. Maddah-Ali and U. Niesen, "Decentralized Coded Caching Attains Order-Optimal Memory-Rate Tradeoff," *IEEE/ACM Trans. Net.*, vol. 23, Aug. 2015, pp. 1029–40.

[4] R. Ahlswede *et al.*, "Network Information Flow," *IEEE Trans. Info. Theory*, vol. 46, Apr. 2000, pp. 1204–16.

[5] 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. Info. Theory*, vol. 52, June 2006, pp. 2825–30.

[6] U. Niesen and M. A. Maddah-Ali, "Coded Caching with Nonuniform Demands," *Proc. IEEE INFOCOM Wksps*., Apr. 2014, pp. 221–26.

[7] M. Ji *et al.*, "On the Average Performance of Caching and Coded Multicasting with Random Demands," *Proc. IEEE ISWCS*, Aug. 2014, pp. 922–26.

[8] J. Zhang, X. Lin, and X. Wang, "Coded Caching Under Arbitrary Popularity Distributions," *Proc. ITA*, Feb. 2015, pp. 98–107.

[9] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online Coded Caching," *Proc. IEEE ICC*, June 2014, pp. 1878–83.

[10] N. Karamchandani *et al.*, "Hierarchical Coded Caching," *Proc. IEEE ISIT*, June 2014, pp. 2142–46.

[11] M. Ji, G. Caire, and A. F. Molisch, "Fundamental Limits of Caching in Wireless D2D Networks," *IEEE Trans. Info. Theory*, vol. 62, Feb. 2016, pp. 849–69.

[12] M. A. Maddah-Ali and U. Niesen, "Cache-Aided Interference Channels," *Proc. IEEE ISIT*, June 2015, pp. 809–13.

[13] N. Naderializadeh, M. A. Maddah-Ali, and A. S. Avestimehr, "Fundamental Limits of Cache-Aided Interference Management," *Proc. IEEE ISIT*, July 2016.

[14] R. Timo and M. Wigger, "Joint Cache-Channel Coding over Erasure Broadcast Channels," *Proc. IEEE ISWCS*, Aug. 2015.

[15] U. Niesen and M. A. Maddah-Ali, "Coded Caching for Delay-Sensitive Content," *Proc. IEEE ICC*, June 2015, pp. 5559–64.

Biographies

Mohammad Ali Maddah-Ali [S'03, M'08] (mohammad.maddah-ali@nokia.com) received his B.Sc. degree from Isfahan University of Technology, his M.A.Sc. degree from the University of Tehran, and his Ph.D. degree from the University of Waterloo. From 2008 to 2010, he was a post-doctoral fellow at the University of California at Berkeley. Since September 2010, he has been with Bell Labs, New Jersey. He received an NSERC post-doctoral fellowship (2007), a best paper award from IEEE ICC (2014), and the IEEE Communications and Information Theory Societies Joint Paper Award (2015).

Urs Niesen [S'03, M'09] (urs.niesen@ieee.org) received his M.S. degree from the School of Computer and Communication Sciences at the École Polytechnique Fédérale de Lausanne in 2005 and his Ph.D. degree from the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in 2009. From 2009 until 2014, he was a member of technical staff at Bell Labs, Alcatel-Lucent. Currently, he is with the Qualcomm New Jersey Research Center.