Coded Caching with Heterogeneous User Profiles

Su Wang and Borja Peleato,
Purdue University
West Lafayette, IN 47907
Email:{wang2506,bpeleato}@purdue.edu

***Abstract*—Coded caching has been proven to be a useful technique for reducing traffic in networks with point-to-multipoint links. The key idea is to pre-fetch popular content at the end users during off-peak hours, and encode transmissions when resources are scarce in such a way that different users can obtain different information from the same packet.**

Prior works have proposed placement and delivery algorithms to address a wide range of scenarios with varying file popularities, file sizes, and cache capacities. Both centralized and distributed algorithms have been proposed, and their performance limits have been characterized in terms of peak and average transmission rates. However, existing works have focused on the case where all the end users present an identical distribution of requests; in other words, the popularity of a given file is identical for all users. This assumption is overly simplistic in modern networks, where network operators often build detailed individual profiles on each user.

This paper proposes and compares the peak rate of three coded caching schemes when the end users can be classified into distinct groups with different distribution of demands. Specifically, the first scheme ignores the differences between user profiles, the second performs independent caching and delivery of each class of files, and the third ignores the similarities between user profiles. The second scheme derives a method to partition the cache between widely popular files and files that may only be requested by a subset of the users. Our analysis yields some counter-intuitive results.

I. Introduction

Modern market segmentation has become increasingly precise and specific [1]–[3]. Based on user actions, algorithms have been able to classify people into distinct groups, each of which has a specific set of interests. It follows that their requests on an interconnected medium, e.g. sites such as Netflix, would logically fall into specific discrete and overlapping sets. Application of grouping requests, therefore, may help reduce transmission rates.

Caching has always been a popular method to reduce network congestion. By pre-fetching the most popular content during off-peak hours and storing it in the end user's memories, the network can reduce the total information to be sent when the users request that content. This gain is commonly known as the local caching gain, since it depends on the hit rates on the local cache of the end users. However, recent research has found that the overall transmission rate in point-to-multipoint networks can be further reduced through a careful design of the segments cached and a coded delivery mechanism [4]. This gain depends not just on the local information cached at each user, but also on the segments shared by user subgroups. For this reason, it is known as global caching gain.

In [4], Maddah-Ali and Niesen designed a centrally coordinated scheme to populate user caches prior to user demands such that coded-multicasting opportunities are available simultaneously for all possible requests. For situations without a central coordinating server, a decentralized coded caching scheme that caches a random subset of bits from each file was proposed in [5]. Additional research on coded caching has extended it to topics such as device-to-device caching [6], hierarchical caching [7], online caching [8], caching with distributed storage [9], distinct file sizes [10], and caching with nonuniform demands [11], [12]. In particular, the latter address a scenario with heterogeneous file popularities by partitioning files into groups such that, within each group, the files have approximately equal popularities. However, is assumes that the popularity of each file is the same for every user.

Save for a few simple cases [13], existing research has yet to explore the effect of different users assigning different popularities to the same file. This paper groups the users into distinct heterogeneous profiles. Different profiles will periodically request different files but, sometimes, they will request the same file. The files that are likely to be requested by all user profiles are termed common files and the other files are termed unique files. For instance, college students are likely to request information from their college's website, young children are likely to request cartoon flash games, and both are likely to request cat pictures. Therefore, user caches should be divided in a similar manner to take advantage of coded caching and subsequent multicasting opportunities.

Intuitively, a disproportionate amount of cache capacity should be devoted for common files to take advantage of better multicasting opportunities. However, since common files can be better multicast, their impact to peak rate is less significant than that of unique files. When users request unique files, opportunities for multicasting are scarcer; the server needs to transmit more of the file uncoded. Investigating an effective and efficient cache allocation and delivery scheme for heterogeneous user profiles thus becomes a valuable endeavor.

The paper will be organized as follows: Section II – system model, Section III – proposed schemes, Section IV - results, Section V – simulations, and Section VI - conclusion.

II. System model

The system consists of a single server connected to K users through a shared, error-free, broadcast link. The server stores a

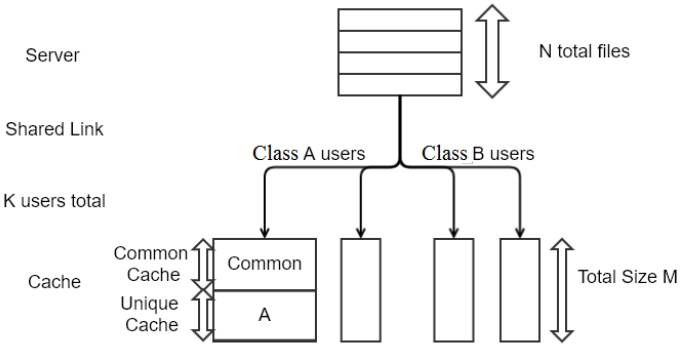

FIGURE 1: System model with two classes, A and B, with two users each. Cache are split between common and unique files.

total of N files, each of size F bits, and each user has access to a cache of MF bits, with $M \in [0, N]$. The N files are further divided into common and unique files. Common files may be requested by any user but unique files are only of interest to a specific class of users. Similarly, the K users are grouped into G classes according to the files that they may request. For simplicity, this paper assumes that the number of users and unique files is the same for every class, and that they do not overlap. Hence, there are $\frac{K}{G}$ users per class and $N = N_c + GN_u$, where N_c and N_u represent the number of common files and the number of unique files **per class**, respectively. This scenario is illustrated in Figure 1 with only two classes.

The problem is split into placement and delivery phases. In the placement phase, the central server carefully coordinates content storage among users. Each user's common cache stores uncoded segments from exclusively common files, while the unique cache stores segments from unique files exclusively, also uncoded. In the delivery phase, each user requests a file, either a common or unique file relevant to its class. The goal of this paper is to propose placement and delivery schemes to minimize the peak rate, defined as the maximum number of bits that the server needs to transmit for the worst possible set of requests.

This paper adapts the centralized coded caching scheme proposed by Maddah-Ali and Niesen [4], henceforth referred to as MN's scheme, to heterogeneous user classes. It is therefore important to have a good understanding of that scheme before we proceed any further. MN's scheme features a centralized server containing all files and connected to all users through an error-free broadcast link. All files are assumed to be of the same size and all users are assumed to have the same cache capacity.

In the placement phase, MN's scheme splits each file into $\binom{K}{t}$ distinct segments, where $t = \frac{KM}{N}$. Each segment is cached by a distinct set of t users, which results in each user caching $\binom{K-1}{t-1}$ segments from each file. In addition to the local caching gain derived from caching these segments, this placement provides opportunities for a coded message to satisfy the file requests of multiple users in the delivery phase, resulting in a global caching gain. Specifically, this scheme is able to satisfy any vector of requests by transmitting at most $\binom{K}{t+1}$ messages of size $\binom{K}{t}$ bits. The messages are themselves the bitwise XOR of $t+1$ segments. The peak rate (normalized by the file size F) becomes

$$R_{MN}(K,t) = \frac{\binom{K}{t+1}}{\binom{K}{t}} \tag{1}$$

$$= \frac{K-t}{t+1}. \tag{2}$$

Should the server only receive requests from m users, then the peak transmission rate with MN's scheme becomes

$$R(K,M,N,m) = \frac{\binom{K}{t+1} - \binom{K-m}{t+1}}{\binom{K}{t}}, \tag{3}$$

where $t = KM/N$, as was shown in [9].

When t is not an integer, the combinatorial expressions present in Equation (3) cannot be applied. This paper focuses on asymptotic and high level trends, not on the integer effects. Therefore, we adopt Stirling's approximation, $n! = \sqrt{2\pi n}(\frac{n}{e})^n$, to simply the expression analytically [14] and obtain a continuous relaxation of the problem. All subsequent results will refer to the continuous relaxed problem.

III. Proposed Schemes

The paper will consider three schemes for dealing with heterogeneous user profiles:

- Scheme 1: Treat all files as common. Ignore the different user profiles and have every user cache segments from every file, even those that it would never request. This scheme sacrifices local caching gain for the benefit of the global caching gain. Assuming that MN's scheme is used for the placement and delivery, the peak rate for this scheme is given by

$$R^{(1)}_{\text{peak}} = \frac{K - t_1}{t_1 + 1}, \tag{4}$$

  where $t_1 = \frac{KM}{N}$ and $N = N_c + GN_u$ denotes the total number of files (common and unique).
- Scheme 2: Split the caching and delivery of common and unique files. Each user devotes a fraction x of its cache to store segments from common files and $(1-x)$ to unique files. The placement and delivery of common files is done according to MN's scheme, with the N_c common files distributed across all users and MFx bits of cache per user. The placement and delivery of unique files within the users in each class also follows MN's scheme, with N_u files and $MF(1-x)$ bits of cache capacity per user. The delivery phase is independent for each class of files, never combining file segments from different classes in the same message. If α out of the K/G users in each class

request unique files[1], the peak data rate for this scheme is given by

$$R^{(2)}(x,\alpha) = R_c(x,\alpha) + GR_u(x,\alpha) \quad (5)$$
$$= \frac{\binom{K}{t_c+1} - \binom{G\alpha}{t_c+1}}{\binom{K}{t_c}} + G\frac{\binom{\frac{K}{G}}{t_u+1} - \binom{\frac{K}{G}-\alpha}{t_u+1}}{\binom{\frac{K}{G}}{t_u}},$$

where $t_c = K\frac{Mx}{N_c}$ and $t_u = \frac{K}{G}\frac{M(1-x)}{N_u}$. The two components in this expression correspond to the rate required to deliver common files, $R_c(x,\alpha) = R(K, Mx, N_c, G\alpha)$, and that required to deliver unique files, $R_u(x,\alpha) = R(\frac{K}{G}, M(1-x), N_u, \frac{K}{G} - \alpha)$. The fraction x will be optimized numerically so as to minimize the peak rate. Despite users only store the files that they are presumed to request, x might favor unique files over common files (or vice versa), so the local caching gain is still being sacrificed in favor of global caching gain.

- Scheme 3: Treat all files as unique. Ignore the fact that some files can be requested by all user classes and have an independent placement and delivery for each class of users. This scheme sacrifices global caching gain for the benefit of local caching gain, which is maximized. Assuming that MN's scheme is used for the placement and delivery, the peak rate for this scheme is given by

$$R^{(3)}_{\text{peak}} = G\frac{\frac{K}{G} - t_3}{t_3 + 1}, \quad (6)$$

where $t_3 = \frac{K}{G}\frac{M}{N_c+N_u}$.

The goal of this paper is to compare these three schemes for different scenarios and parameter values. It is clear that when there is no cache, *i.e.* $M = 0$, all three schemes yield a peak rate of K. Similarly, when M is large enough, all files can be cached by all users and all three schemes offer peak rate 0. Our analysis will therefore focus on intermediate values of M.

IV. Results

First, we compare Schemes 1 and 3, since they have the simplest peak rate expressions.

Theorem 1. *Scheme 1 offers lower peak rate than Scheme 3 for small M, and vice versa. Specifically,*

$$R^{(1)}_{\text{peak}} \le R^{(3)}_{\text{peak}} \quad \Leftrightarrow \quad M \le N_c - \frac{GN_u}{K}. \quad (7)$$

Proof. This theorem can be proved by simple, albeit tedious, manipulation of Equations (4) and (6). □

Corollary 1. *It is often beneficial for users to cache segments from undesired files, to increase multicasting opportunities. The loss in local caching gain is more than compensated by the gain in global caching gain.*

In Schemes 1 and 3, the number of users requesting common versus unique files is irrelevant, since segments from both files can be encoded together. In Scheme 2, however, it plays a major role. We now intend to show that in order to compute the peak rate, we only need to consider the case where the subdivision is the same for all classes.

Lemma 1. *Let $R_u(x,\alpha)$ represent the peak rate component associated to the delivery of unique files, as defined in Equation (5). If $\frac{K}{G} \ge \alpha > \beta$, then*

$$R_u(\beta+1) - R_u(\beta) \ge R_u(\alpha) - R_u(\alpha-1), \quad (8)$$

where x is omitted for simplicity.

Proof. This result follows from the fact that the rate is submodular in the number of requests, but we prove it anyway. Assume that $\alpha > \beta$. Then Equation (8) can be written as

$$\frac{\binom{\frac{K}{G}-\beta}{t_u+1} - \binom{\frac{K}{G}-\beta-1}{t_u+1}}{\binom{\frac{K}{G}}{t_u}} \ge \frac{\binom{\frac{K}{G}-\alpha+1}{t_u+1} - \binom{\frac{K}{G}-\alpha}{t_u+1}}{\binom{\frac{K}{G}}{t_u}} \quad (9)$$

or, equivalently,

$$\binom{\frac{K}{G} - (\beta+1)}{t_u} \ge \binom{\frac{K}{G} - \alpha}{t_u}, \quad (10)$$

which is true, since binomial coefficients increase monotonically with the number of elements. □

Theorem 2. *If the number of users of each class $\frac{K}{G}$ (identical for all classes) is sufficiently large, then there exists a number $\alpha \in (0, \frac{K}{G})$ such that the peak rate for Scheme 2 is achieved when every class has α users requesting unique files.*

Proof. Let α_i represent the number of users from class i requesting unique files, and assume that $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_G)$ maximizes the rate, given by

$$R(\boldsymbol{\alpha}) = R_c\left(\sum_{i=1}^{G}\alpha_i\right) + \sum_{i=1}^{G} R_u(\alpha_i), \quad (11)$$

where R_c and R_u have been defined by Equation (5) and we omit x for simplicity.

Without loss of generality, assume that $\alpha_1 > \alpha_2$ and let $\boldsymbol{\beta} = (\alpha_1 - 1, \alpha_2 + 1, \alpha_3, \ldots, \alpha_G)$. Then

$$R(\boldsymbol{\beta}) - R(\boldsymbol{\alpha}) = R_u(\alpha_1 - 1) - R_u(\alpha_1) + R_u(\alpha_2 + 1) - R_u(\alpha_2).$$

Using Lemma 1 we realize that $\boldsymbol{\beta}$ achieves a rate at least as high as $\boldsymbol{\alpha}$ with less variance across the coefficients. For large enough K (*i.e.* using the continuous relaxation of the problem), we can conclude that iterative application of this process would yield a uniform set of coefficients achieving the peak rate.

□

Assuming an optimal splitting of the cache x, the peak rate in Scheme 2 can be defined as

$$R^{(2)}_{\text{peak}} = \min_x \max_\alpha R^{(2)}(x,\alpha). \quad (12)$$

We are now ready to compare Scheme 2 with the other two.

[1]We will later prove that, when the number of users is sufficiently large, the peak rate happens when the number of users requesting unique files is the same for every class.

Theorem 3. *Scheme 2 offers lower peak rate than Scheme 3 for large M. Specifically,*

$$R^{(2)}_{\text{peak}} \leq R^{(3)}_{\text{peak}} \qquad \Leftarrow \qquad M \geq \frac{G}{G-1}\frac{K+1}{K}N_u. \quad (13)$$

Proof. If $x = 1 - \frac{N_u}{M}$, each user stores all the unique files that it might request. The worst case α is therefore $\alpha = 0$. Observe that

$$\min_x \max_\alpha R^{(2)}(x,\alpha) \leq \max_\alpha R^{(2)}_{\text{peak}}\left(1-\frac{N_u}{M},\alpha\right) \quad (14)$$

$$= R^{(2)}_{\text{peak}}\left(1-\frac{N_u}{M},0\right) \quad (15)$$

$$= K\frac{N_u+N_c-M}{K(M-N_u)+N_c}. \quad (16)$$

After some rearrangement, Equation (6) can be written as

$$R^{(3)}_{\text{peak}} = KG\frac{N_c+N_u-M}{KM+G(N_c+N_u)}. \quad (17)$$

Imposing equality between the last two equations yields Equation (13). □

Corollary 2. *For small enough M, Scheme 1 yields lower rate than Schemes 2 and 3. For large enough M, Scheme 2 offers the lowest rate of the three. In some cases, there is a range of intermediate M values for which Scheme 3 has lower rate than the other two.*

V. Simulations

Figure 2 illustrates the above results for a system with $N_c = 64$ common files, $N_u = 64$ unique files for each class and 8 users per class. The plots show the peak rate with all three schemes as a function of the cache size M for different number of classes G. As Corollary 2 stated, it is better to treat all files as common when the cache size is small, regardless of the number of classes. Having every user store segments from every file seems counterintuitive, since the number of unique files grows with the number of classes, but the multicasting gain more than compensates for the loss in local caching.

As the number of classes grows, the peak rates become larger, due mainly to the increase in the number of files. Additionally, it can be seen that Scheme 3 (all unique) falls behind the others, reaching a point when it is never the preferred option. Again, this is somewhat counterintuitive; it seems a good idea to deal with each class independently when the number of classes is large but, in terms of peak rate, it is not.

Figure 3 analyzes in more detail the effect of the number of classes and how the cache should be distributed. As before, it assumes $N_c = 64$ common files and $N_u = 64$ unique files per class, but it also fixes the cache capacity at $M = 64$. Therefore, each user has enough capacity to cache half of the files that it could request. The left plots correspond to the case with 16 users per class, and the right plots to the case with a single user per class, so as to isolate the behavior of strictly local vs global gains. Both cases yield similar results: Scheme 2 (Split) provides significantly lower peak rate than

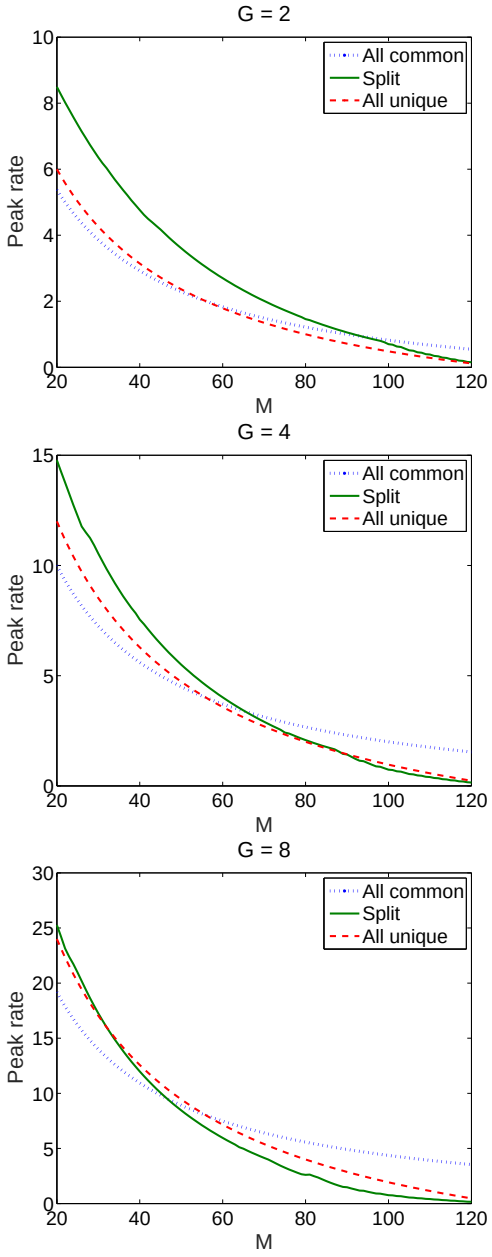

FIGURE 2: Peak rate vs cache size (M) for $N_c = 64$, $N_u = 64$, and 8 users per class.

the other two as the number of classes G grows. It does this by progressively decreasing the portion of the cache devoted to common files, as shown by the bottom plots.

It would seem intuitive to favor caching common files, since they present better multicasting opportunities. This is usually the case if we aim to reduce the average rate, but in terms of peak rate, it is better to do the opposite. Since unique files are more costly to transmit (no or little multicasting) we need to devote a larger proportion of the cache to them.

Finally, Figure 4 studies the performance of all three schemes when the total number of files stays constant, $N = 2048$, but their type changes with G so that there are always

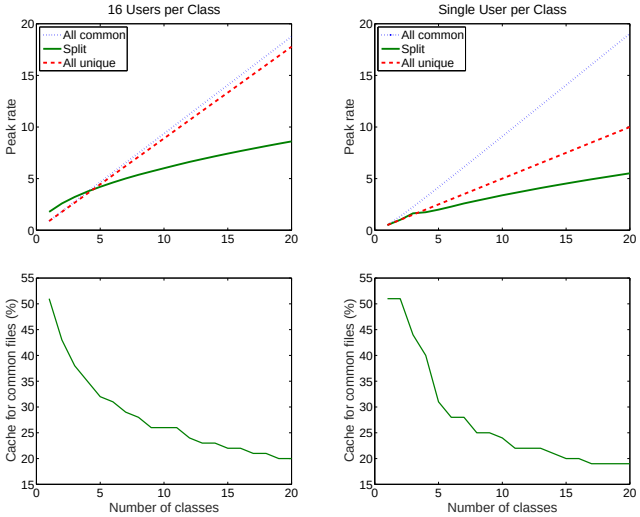

FIGURE 3: *Top* - Peak rate behavior of the three schemes for a system with $N_c = 64$, $N_u = 64$, $M = 64$ and 16 (left) or 1 (right) users per class. *Bottom* - Cache distribution in Scheme 2 (Split) that results in the minimum peak rate.

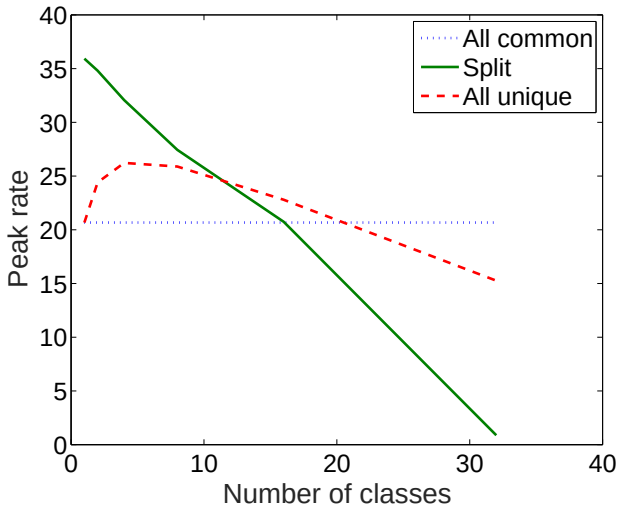

FIGURE 4: Peak rate behavior of the three schemes for a system with fixed number of users, $N = 2048$, and $N_u(G) = N_c(G)$, $M = 64$, and $K = 64$.

the same number of common and unique files per class. The number of users and cache capacity per user is also kept constant at $K = 64$ and $M = 64$, respectively. This can be understood as refining the user profiles, subdividing them into more classes and realizing that some common files are only of interest to a specific class of users. Scheme 1 treats all files as common and is therefore oblivious to the number of classes. Scheme 3 treats all files as unique, which gives it an edge when the number of classes is large and most files are indeed unique. Scheme 2 seeks an optimal splitting of the cache and independent delivery of common and unique files. As the system becomes increasingly precise and capable of segmenting users into more classes, Scheme 2 again outperforms its counterparts.

VI. Conclusion

More efficient caching is possible under powerful servers capable of segmenting users into heterogeneous classes. By evaluating local and global caching gain tradeoffs, three implementations of MN's coded caching were developed, each fulfilling a unique niche based on cache size and quantity of groups to minimize peak rate.

Scheme 1, caching all files as common files, lends itself well to situations with both small cache size and limited user segmentation. Scheme 2, treating common and unique files separately, provides better performance for cases involving large caches and/or a significant number of user classes. Finally, Scheme 3 is useful in situations with a small number of classes and intermediate cache sizes.

Possible extensions could be to investigate average rate instead of peak rate, nonuniform class populations, and overlapping groups in which users can be members of multiple classes.

References

[1] M. Pazzani and D. Billsus, "Learning and revising user profiles: The identification of interesting web sites," Machine learning, vol. 27, no. 3, pp. 313–331, 1997.

[2] S. Goel, J. M. Hofman, and M. I. Sirer, "Who does what on the web: A large-scale study of browsing behavior." in Proceedings of the Sixth International AAAI Conference on Weblogs and Social Media. IAAA, 2012.

[3] R. Kumar and A. Tomkins, "A characterization of online browsing behavior," in Proceedings of the 19th international conference on World wide web. ACM, 2010, pp. 561–570.

[4] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," IEEE Transactions on Information Theory, vol. 60, no. 5, pp. 2856–2867, 2014.

[5] ——, "Decentralized coded caching attains order-optimal memory-rate tradeoff," IEEE/ACM Transactions on Networking (TON), vol. 23, no. 4, pp. 1029–1040, 2015.

[6] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," IEEE Communications Magazine, vol. 51, no. 4, pp. 142–149, 2013.

[7] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. N. Diggavi, "Hierarchical coded caching," IEEE Transactions on Information Theory, vol. 62, no. 6, pp. 3212–3229, 2016.

[8] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," IEEE/ACM Transactions on Networking (TON), vol. 24, no. 2, pp. 836–845, 2016.

[9] T. Luo, V. Aggarwal, and B. Peleato, "Coded caching with distributed storage," arXiv preprint arXiv:1611.06591, 2016.

[10] J. Zhang, X. Lin, C.-C. Wang, and X. Wang, "Coded caching for files with distinct file sizes," in 2015 IEEE International Symposium on Information Theory (ISIT). IEEE, 2015, pp. 1686–1690.

[11] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," IEEE Transactions on Information Theory, vol. 63, no. 2, pp. 1146–1158, 2017.

[12] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," IEEE Transactions on Information Theory, vol. 64, no. 1, pp. 349–366, 2018.

[13] C.-H. Chang and C.-C. Wang, "Coded caching with full heterogeneity: Exact capacity of the two-user/two-file case," in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019.

[14] J. Dutka, "The early history of the factorial function," Archive for history of exact sciences, vol. 43, no. 3, pp. 225–249, 1991.