Stochastic Coded Caching with Optimized Shared-Cache Sizes and Reduced Subpacketization

Adeel Malik, Berksan Serbetci, Petros Elia

Stochastic Coded Caching with Optimized Shared-Cache Sizes and Reduced Subpacketization

Adeel Malik, Berksan Serbetci, Petros Elia
Dept. of Communication Systems, EURECOM, Sophia Antipolis, 06410, France
{malik, serbetci, elia}@eurecom.fr

***Abstract*—This work studies the K-user broadcast channel with Λ caches, when the association between users and caches is random, i.e., for the scenario where each user can appear within the coverage area of – and subsequently is assisted by – a specific cache based on a given probability distribution. Caches are subject to a cumulative memory constraint that is equal to t times the size of the library. We provide a scheme that consists of three phases: the storage allocation phase, the content placement phase, and the delivery phase, and show that an optimized storage allocation across the caches together with a modified uncoded cache placement and delivery strategy alleviates the adverse effect of cache-load imbalance by significantly reducing the multiplicative performance deterioration due to randomness. In a nutshell, our work provides a scheme that manages to substantially mitigate the impact of cache-load imbalance in stochastic networks, as well as – compared to the best known state-of-the-art – the well-known subpacketization bottleneck by showing its applicability in deterministic settings for which it achieves the same delivery time – which was proven to be close to optimal for bounded values of t – with an exponential reduction in the subpacketization.**

***Index Terms*—Coded caching, shared caches, heterogeneous networks, femtocaching.**

I. Introduction

The ever-growing amounts of mobile data traffic have highlighted the need for innovative solutions that can provide service to an ever-increasing number of users while using restricted network bandwidth resources. Within this framework, cache-enabled wireless networks have emerged as a viable option that can transfigure the storage capabilities of the network nodes into a fresh and powerful resource.

The seminal work in [1] introduced the concept of coded caching, and revealed that an unbounded number of cache-aided users having different content requests can be served simultaneously by the aid of multicasting transmissions even with a bounded amount of network resources. Key to this approach is a novel and carefully designed cache placement algorithm. This delivery speedup is referred to as the *coding gain* – or equivalently as the Degrees-of-Freedom (DoF) – and it scales with the total storage capacity of the network. This same approach was shown to be information-theoretically optimal in [2], [3] for the shared-link broadcast channel.

The work is supported by the European Research Council under the EU Horizon 2020 research and innovation program / ERC grant agreement no. 725929 (project DUALITY).

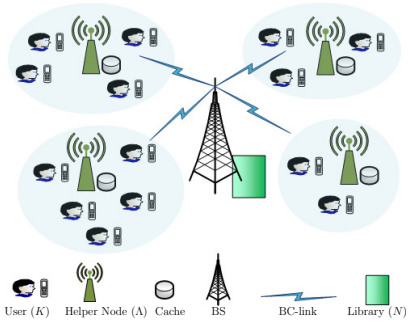

Fig. 1: An instance of a shared-cache network.

Many follow-up works have been studied since then, including the study of coded caching in D2D networks [4], in subpacketization-constrained settings [5]–[7], in settings with arbitrary popularity distributions [8]–[10], and in other settings as well [11]–[17].

A. Coded caching networks with shared caches

Even though coded caching offers massive gains for the shared-link broadcast channel, its applicability to more realistic settings requires further exploration. In the so-called shared-cache setting, different users are within the coverage area of different cache-aided base stations (or caches), and are forced to benefit from the same cache content stored in the cache that they are associated with. This shared-cache setting is of great importance, and it arguably represents a much more realistic scenario as having all users with their own caches dedicated to a certain content library is a fanciful assumption in practical wireless communication settings [15], [16]. Such realistic settings may include cache-enabled heterogeneous networks (HetNets), where a central transmitter (a base station, or a macrocell) needs to deliver content to a set of interfering users with the assistance of cache-enabled helper nodes (small base stations, or femtocells), where the cache content is available to all users (cost-free) within the coverage area of that node. An instance of a shared-cache network is shown in Figure 1.

An early work on this scenario can be found in [14], where each helper node is serving an equal number of users. The work in [15] removed this assumption, and characterized the optimal (under the uncoded cache placement) worst-case delivery time for the case when an arbitrary number of users is assisted by each cache, and when the cache placement is agnostic to the user-to-cache association. Subsequently,

the work in [16] extended the deterministic user-to-cache association setting in [15] to the stochastic network setting, and for the setting where the cache populations follow a given probability distribution. This work revealed the surprising fact that the cache-load imbalance could lead to the significant deterioration in coding gains. This same work also showed that a more balanced user-to-cache association performs better than the unbalanced one, leading to the fact that the multiplicative performance deterioration caused by the stochastic nature of the problem can be drastically reduced by carefully tuning the cache-load balance.

The work in [17] proposed a novel coded placement strategy by assuming that user-to-cache association during the placement phase is known, and showed that exploiting this extra knowledge yields additional coding gains. In a similar context, the work in [18] optimized the cache sizes as a function of the number of users served by each cache, and then proposed a novel coded caching scheme that outperforms the optimal scheme in [15].

At this point, we need to highlight that there is a direct relation between the aforementioned shared-cache setting and the so-called subpacketization bottleneck. In order to achieve the originally promised *theoretical coding gains* [1], each file in the content library must be partitioned into S unit-size subpackets, where S scales exponentially with number of cache states[1]. Subsequently, a subset of these subpackets is cached at different nodes depending on the cache-enabled device's cache state. The number of distinct cache states, denoted as Λ, is then subject to some physical limitations, i.e., a file cannot be partitioned into more subpackets than a certain threshold, hence forcing S to be less than some certain number, and inevitably forcing Λ to be less than a certain value. In a nutshell, Λ must be generally less than the total number of users since it is known that traditional coded caching techniques require file sizes that scale exponentially with Λ (cf. [6], [7].). In broad terms, reducing the number of cache states leads to a reduction in the coding gain, hinting out that the *subpacketization bottleneck* is in fact a major factor on the performance.

In this work, we aim to exploit cache-size differences and optimize the individual cache sizes to mitigate the cache-load imbalance bottleneck in stochastic shared-cache networks. To do so, we propose a coded caching scheme that optimizes the individual cache sizes based on each cache's load statistics, subject to a given cumulative cache capacity. We characterize the performance of our scheme and numerically verify its effectiveness in substantially ameliorating the impact of cache-load imbalance on the coding gain. We also show that for a deterministic user-to-cache association setting, our scheme achieves the same state-of-the-art (SoA) delivery time as of [18] with a significant (exponential) reduction in subpacketization, thus making our scheme more suitable than [18] to apply in HetNets in the finite file size regime.

[1]When adopting the cache placement strategy in [1] for Λ cache-enabled users, we refer the content to be placed in a single cache as a cache state.

B. Notations

Throughout this paper, we use the notation $[z] \triangleq [1, 2, \ldots, z]$, and we use $\mathbf{Q}\backslash\mathbf{P}$ to denote the set difference of $\mathbf{P}$ and $\mathbf{Q}$, which is the set of elements in $\mathbf{Q}$ but not in $\mathbf{P}$. We use $\mathcal{X}_k^{[n]} \triangleq \{\delta : \delta \subseteq [n], |\delta| = k\}$ and we use $\delta(i)$ to denote the ith element of δ.

II. Network Setting

We consider a heterogeneous cache-aided network setting which consists of a base station (BS) having access to a library of N unit-sized files $\mathcal{F} = \left\{F^1, F^2, \ldots, F^N\right\}$, as well as consists of Λ cache-enabled helper nodes (i.e., caches), and K receiving users. The BS delivers content via an error-free broadcast link of bounded capacity per unit of time to K users, with the assistance of helper nodes. We assume that users within the coverage area of a cache $\lambda \in [\Lambda]$ have direct access to the content stored at that cache. We consider the scenario of non-uniform cache population intensities where the number of users served by each cache may not be identical. For any cache $\lambda \in [\Lambda]$, let p_λ be the probability that a user appears in the coverage area of this λth cache-enable helper node, and let $\mathbf{p} = [p_1, p_2, \ldots, p_\Lambda]$ denote the cache population intensities vector, where $\sum_{\lambda\in[\Lambda]} p_\lambda = 1$. Without loss of generality, we assume that $p_1 \geq p_2 \geq \cdots \geq p_\Lambda$. At any given instance, we denote $\mathbf{V} = [v_1, v_2, \ldots, v_\Lambda]$ to be the cache population vector, where v_λ is the number of users having access to the content of cache $\lambda \in [\Lambda]$, and we let $\bar{\mathbf{V}} = K\mathbf{p} = [\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_\Lambda]$ be the expected cache population vector.

The size of each cache $\lambda \in [\Lambda]$ is the design parameter $M_\lambda \in (0, N]$ (measured in units of file), adhering to a cumulative sum cache-size constraint $\sum_{\lambda=1}^{\Lambda} M_\lambda = M_\Sigma$. For any cache $\lambda \in [\Lambda]$, we denote by $\gamma_\lambda \triangleq \frac{M_\lambda}{N}$ the normalized cache capacity, and subsequently we have the normalized cache capacity vector $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \ldots, \gamma_\Lambda]$. Consequently, the normalized cumulative cache-size constraint takes the form

$$\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t \triangleq \frac{M_\Sigma}{N}. \tag{1}$$

The communication process consists of three phases; the *storage allocation phase*, the *content placement phase* and the *delivery phase*. The storage allocation phase involves allocating the cumulative cache capacity M_Σ (or the normalized cumulative cache capacity t) to the caches subject to (1), a process that results in the aforementioned normalized cache capacity vector $\boldsymbol{\gamma}$. The content placement phase involves the placement of a portion of library-content, $\mathcal{Z}_\lambda$, in each cache $\lambda \in [\Lambda]$ — respecting its allocated cache capacity γ_λ — according to a certain placement strategy $\mathcal{Z} = [\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_\Lambda]$. We assume that the first two phases are aware of the cache population intensities $\mathbf{p}$. However, these two phases are oblivious to the actual requests generated during the delivery phase. The delivery phase begins after each user $k \in [K]$ appears within the coverage area of one of the caches, and requests a content file $F^{d_k} \in \mathcal{F}$, where d_k is the index of the file requested by user $k \in [K]$. As is common in coded caching works, we assume that each user requests a different file (i.e., worst-case).

Then for any *request vector* $\mathbf{d} = [d_1, \ldots, d_{v_1}, d_{v_1+1}, \ldots, d_K]$, using the knowledge of the content stored at each cache $\mathcal{Z}_\lambda \in \mathcal{Z}$, and the user-to-cache association, the BS delivers the content to the user.

A. Problem Definition

For a given normalized cache-size budget t, and cache population intensities vector $\mathbf{p}$, our goal is to design a content placement strategy $\mathcal{Z}$, and a delivery scheme for the system where a BS is serving K users with the help of Λ caches. Then our goal is to evaluate its performance in terms of the delivery time (delay), which corresponds to the time needed to complete the delivery of any request vector $\mathbf{d}$, where the time scale is normalized such that a unit of time corresponds to the optimal amount of time needed to send a single file from the transmitter to the receiver, had there been no caching and no interference. Given the random nature of our problem where at any given instance of the problem we may experience a different cache population vector $\mathbf{V}$, our measure of interest is the average delay

$$\overline{T}(t) = E_{\mathbf{V}}[T(\mathbf{V})] = \sum_{\mathbf{V}\in\mathcal{V}} P(\mathbf{V})T(\mathbf{V}), \tag{2}$$

where $T(\mathbf{V})$ is the time needed to complete the delivery of any request vector $\mathbf{d}$ corresponding to a specific cache population vector $\mathbf{V}$, where $\mathcal{V}$ is the set of all possible cache population vectors, and where $P(\mathbf{V})$ is the probability of observing the cache population vector $\mathbf{V}$.

III. Main Result

In this section, we first present our main results on the performance of the stochastic network setting described in Section II. After doing so, we will also discuss the applicability as well as the efficacy of our proposed scheme for a deterministic shared cache setting studied in [18] by showing that it provides an exponential reduction in the subpacketization. Our first result is the characterization of the achievable delivery time $T(\mathbf{V})$ for any cache population vector $\mathbf{V}$. Crucial to this characterization is the greatest common divisor (GCD) of vector $\bar{\mathbf{V}}$, which we denote by α, and the partition of $\mathbf{V}$ into a set $\mathcal{B}_V = [\mathbf{V}^1, \mathbf{V}^2, \ldots, \mathbf{V}^{\beta_\mathbf{V}}]$ of $\beta_\mathbf{V} = \max_{i\in[\Lambda]} \frac{\alpha v_i}{\bar{v}_i}$ vectors according to the partition algorithm presented in Algorithm 1. Each resulting partition vector $\mathbf{V}^j = [v_1^j, v_2^j, \ldots, v_\Lambda^j]$ will then satisfy $v_\lambda^j \leq \frac{\bar{v}_\lambda}{\alpha}$, where $\sum_{j\in[\beta_\mathbf{V}]} v_\lambda^j = v_\lambda$, $\forall \lambda \in [\Lambda]$. Under the assumption of a random user-to-cache association with cache population intensities vector $\mathbf{p}$ such that the expected cache population $\bar{v}_\lambda \in \bar{\mathbf{V}}$ is a non-negative integer for each cache $\lambda = [\Lambda]$, the achievable delay is given in the following theorem.

Theorem 1. *In the K-user, Λ-cache setting with a normalized cache budget t, and a random user-to-cache association with cache population intensities vector $\mathbf{p}$, the delivery time for any cache population vector $\mathbf{V}$*

$$T(\mathbf{V}) = \sum_{j\in[\beta_\mathbf{V}]} \frac{\sum\limits_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod\limits_{i=1}^{t+1} \frac{\bar{v}_{\tau(i)}}{\alpha} - \sum\limits_{\tau\in\mathcal{X}_{t+1}^{\mathcal{A}_j}} \prod\limits_{i=1}^{t+1} \left(\frac{\bar{v}_{\tau(i)}}{\alpha} - v_{\tau(i)}^j\right)}{\sum\limits_{\tau\in\mathcal{X}_t^{[\Lambda]}} \prod\limits_{i=1}^{t} \frac{\bar{v}_{\tau(i)}}{\alpha}} \tag{3}$$

is achievable if the expected cache population vector $\bar{\mathbf{V}}$ is a non-negative integer vector, where $\mathcal{A}_j \subseteq [\Lambda]$ is a subset of the set of caches, such that for each cache $\lambda \in \mathcal{A}_j$, $\frac{\bar{v}_\lambda}{\alpha} > v_\lambda^j$.

Proof. The proof is deferred to Section IV. □

With Theorem 1 in hand, we present our next result, which is the average delay $\overline{T}(t)$ corresponding to our stochastic network setting.

Theorem 2. *In the K-user, Λ-cache setting with a normalized cache budget t, and a random user-to-cache association with cache population intensities vector $\mathbf{p}$, the average delay of*

$$\overline{T}(t) = \sum_{\mathbf{V}\in\mathcal{V}} \frac{T(\mathbf{V})K!}{\prod_{\lambda\in[\Lambda]} v_\lambda} \prod_{\lambda\in[\Lambda]} p_\lambda^{v_\lambda} \tag{4}$$

is achievable if the expected cache population vector $\bar{\mathbf{V}}$ is a non-negative integer vector.

Proof. From the fact that the probability distribution $P(\mathbf{V})$ follows the well-known multinomial distribution with parameter $\mathbf{p}$, we have

$$P(\mathbf{V}) = \frac{K!}{\prod_{\lambda\in[\Lambda]} v_\lambda} \prod_{\lambda\in[\Lambda]} p_\lambda^{v_\lambda}. \tag{5}$$

Combining (3) with (5) allows us to obtain (4), which concludes the proof. □

Next, we see the applicability of our scheme in a similar setting, but with a fixed user-to-cache association, which was initially studied in [18].

Corollary 1. *In the K-user, Λ-cache setting with a normalized cache budget t, and a fixed user-to-cache association with cache population vector $\bar{\mathbf{V}}$, the proposed scheme in Section IV achieves the delivery time of*

$$T(\bar{\mathbf{V}}) = \alpha \frac{\sum\limits_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod\limits_{i=1}^{t+1} \frac{\bar{v}_{\tau(i)}}{\alpha}}{\sum\limits_{\tau\in\mathcal{X}_t^{[\Lambda]}} \prod\limits_{i=1}^{t} \frac{\bar{v}_{\tau(i)}}{\alpha}}, \tag{6}$$

which is same as of [18, equation (11)] and it requires the subpacketization rate of

$$S = \sum_{\tau\in\mathcal{X}_t^{[\Lambda]}} \prod_{j=1}^{t} \frac{\bar{v}_{\tau(j)}}{\alpha}, \tag{7}$$

which is α^t times less than the subpacketization rate of [18, equation (7)].

Proof. The analysis of our scheme is given in Section IV. In particular, the proof is straightforward from (10) and (13). □

Corollary 1 reveals the substantial benefits of our scheme compared to the SoA [18] as it achieves the same delivery

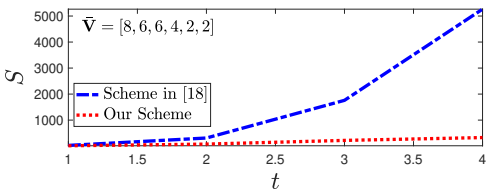

Fig. 2: Comparison of the required subpacketization.

time with a significantly reduced – with a factor of α^t – subpacketization. This exponential reduction in subpacketization is a crucial contribution as the subpacketization is a major bottleneck in the applicability of coded caching schemes [6], [16] especially in the finite file size regimes. To illustrate this gain, in Figure 2, we compare the required subpacketization of our scheme with the scheme in [18] for $\bar{\mathbf{V}} = [8, 6, 6, 4, 2, 2]$. We can see that even for a modest network consisting of an extremely small number of users, our scheme requires significantly less subpacketization.

Remark 1. *An interesting observation of our scheme is that α can be treated as a trade-off parameter between the subpacketization and the delivery time. For example, when $\bar{\mathbf{V}} = [20, 15, 15, 5, 5, 4]$, we have $\alpha = 1$. However, if we associate one virtual user to the Λ-th cache, α increases from 1 to 5, thus, reducing the subpacketization by a factor of 5^t with a modest increase in the delivery time. We leave the study of this trade-off for future work.*

IV. Placement and Delivery

In this section, we present the proof of Theorem 1. We describe the content placement and delivery strategies that can achieve the delay of (3).

A. Content Placement

The content placement scheme is based on the idea of assigning more storage capacity to the caches with high population intensities. We denote $\hat{\mathbf{V}} = [\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_\Lambda]$ as the *base cache population vector*, which is given as $\hat{\mathbf{V}} \triangleq \frac{\bar{\mathbf{V}}}{\alpha}$, where α is the GCD of the elements in $\bar{\mathbf{V}}$. For a base cache population vector $\hat{\mathbf{V}}$, we assume that there are $\hat{\Lambda} \triangleq \sum_{\lambda=1}^{\Lambda} \hat{v}_\lambda$ virtual caches such that each cache $\lambda \in [\Lambda]$ consists of $\hat{v}_\lambda$ virtual caches. We then use $\mathcal{C} = [1, 2, \ldots, \hat{\Lambda}]$ to denote the set of virtual caches, and $\mathcal{C}_\lambda = [\hat{v}_{\lambda-1} + 1, \ldots, \hat{v}_{\lambda-1} + \hat{v}_\lambda]$ (assuming $\hat{v}_0 \triangleq 0$) to denote the set of virtual caches that belongs to cache $\lambda \in [\Lambda]$. Let $\mathcal{Q}_t^{\mathcal{C}} \subseteq \mathcal{X}_t^{\mathcal{C}}$ be the set of all possible t-tuples of $\mathcal{C}$ such that for each tuple $\tau \in \mathcal{Q}_t^{\mathcal{C}}$, no two virtual caches $i, j \in \tau$ belong to the same cache $\lambda \in [\Lambda]$. Next, each content file $F^i \in \mathcal{F}$ is divided into $\left|\mathcal{Q}_t^{\mathcal{C}}\right|$ subpackets, and labeled as $F^i = \left\{F_\tau^i\right\}_{\tau \in \mathcal{Q}_t^{\mathcal{C}}}$. Then, the set of contents to be cached at each cache $\lambda \in [\Lambda]$ is given by

$$\mathcal{Z}_\lambda = \left\{\mathcal{Z}_{\hat{\lambda}} : \hat{\lambda} \in \mathcal{C}_\lambda\right\}, \tag{8}$$

where

$$\mathcal{Z}_{\hat{\lambda}} = \left\{F_\tau^i : F_\tau^i \in F^i, \tau \in \mathcal{Q}_t^{\mathcal{C}}, \hat{\lambda} \in \tau, F^i \in \mathcal{F}\right\}. \tag{9}$$

From the fact that for each cache λ we have $\hat{v}_\lambda$ virtual caches, we can conclude that for any t-tuple of caches $\tau \in \mathcal{X}_t^{[\Lambda]}$ there must be $\prod_{i=1}^{t} \hat{v}_{\tau(i)}$ t-tuples of virtual caches that belong to the set $\mathcal{Q}_t^{\mathcal{C}}$. Consequently, the number of t-tuples of $\mathcal{C}$ in the set $\mathcal{Q}_t^{\mathcal{C}}$ is given as

$$\left|\mathcal{Q}_t^{\mathcal{C}}\right| = \sum_{\tau \in \mathcal{X}_t^{[\Lambda]}} \prod_{j=1}^{t} \hat{v}_{\tau(j)}, \tag{10}$$

and the normalized cache capacity required at any cache $\lambda \in [\Lambda]$ is given as

$$\gamma_\lambda = \frac{|\mathcal{Z}_\lambda|}{N\left|\mathcal{Q}_t^{\mathcal{C}}\right|} = \frac{\sum_{\tau \in \mathcal{X}_t^{[\Lambda]} : \tau \ni \lambda} \prod_{j=1}^{t} \hat{v}_{\tau(j)}}{\sum_{\tau \in \mathcal{X}_t^{[\Lambda]}} \prod_{j=1}^{t} \hat{v}_{\tau(j)}}. \tag{11}$$

Thus, (11) yields our proposed storage allocation strategy. We can see that $\sum_{\lambda \in [\Lambda]} \gamma_\lambda = t$, as for each $\tau \in \mathcal{Q}_t^C$, the corresponding subpackets $\left\{F_\tau^i : F^i \in \mathcal{F}\right\}$ are placed in t caches. Thus, the content placement strategy satisfies the total caching budget constraint (1).

Algorithm 1 Cache Population Vector Partition

Input: $\mathbf{V}$ and $\bar{\mathbf{V}}$
Output: $\mathcal{B}_V$
Initialization: $\alpha \leftarrow GCD(\bar{\mathbf{V}})$, $\beta_{\mathbf{V}} \leftarrow \max_{i \in [\Lambda]} \frac{\alpha v_i}{\bar{v}_i}$, $\mathcal{B}_V \leftarrow \phi$
for j **from** 1 **to** $\beta_{\mathbf{V}}$ **do**
  for i **from** 1 **to** Λ **do**
    if $v_i > \frac{\bar{v}_i}{\alpha}$
      $v_i^j \leftarrow \frac{\bar{v}_i}{\alpha}$, $\quad v_i \leftarrow v_i - \frac{\bar{v}_i}{\alpha}$
    else
      $v_i^j \leftarrow v_i$, $\quad v_i \leftarrow 0$
    end if
  end for
  $\mathcal{B}_V \leftarrow [\mathcal{B}_V, [v_1^j, v_2^j, \ldots, v_\Lambda^j]]$
end for

B. Content Delivery

We will consider the worst-case delivery scenario where each user requests a different file. Once the BS is notified of the cache population vector $\mathbf{V}$ and the corresponding request vector $\mathbf{d}$, it commences delivery. We propose a delivery scheme that is completed in $\beta_{\mathbf{V}} = \max_{i \in [\Lambda]} \frac{v_i}{\hat{v}_i}$ rounds, where the content is delivered to at most $\hat{v}_\lambda$ users from each cache $\lambda \in [\Lambda]$ in each round. We divide the cache population vector $\mathbf{V}$ into a set of $\beta_{\mathbf{V}}$ vectors $\mathcal{B}_V = [\mathbf{V}^1, \mathbf{V}^2, \ldots, \mathbf{V}^{\beta_V}]$ based on the procedure described in Algorithm 1, such that for all $j \in [\beta_V]$, $\mathbf{V}^j = [v_1^j, v_2^i, \ldots, v_\Lambda^j]$ satisfies $v_\lambda^j \leq \hat{v}_\lambda$ and $\sum_{j \in [\beta_{\mathbf{V}}]} v_\lambda^j = v_\lambda$ for all $\lambda \in [\Lambda]$. In the following, we describe our delivery strategy for the two only possible cases after applying Algorithm 1.

Case 1: $\mathbf{V}^j = \hat{\mathbf{V}}$: In this case, the BS will serve $\hat{\Lambda}$ users based on the base cache population vector $\hat{\mathbf{V}}$. Let $\mathbf{U} = [u_1, u_2, \ldots, u_{\hat{\Lambda}}]$ denote the set of indices of users that corresponds to $\hat{\mathbf{V}}$, and $\mathbf{U}_\lambda = \{u_i\}_{i=\hat{v}_{\lambda-1}+1}^{\hat{v}_{\lambda-1}+\hat{v}_\lambda}$ be the set of users associated to cache $\lambda \in [\Lambda]$ (assuming $\hat{v}_0 = 0$). The corresponding request vector is $\mathbf{d}^{\hat{\mathbf{V}}} = \left[d_{u_1}, d_{u_2}, \ldots, d_{u_{\hat{\Lambda}}}\right]$.

Let $\mathcal{Q}^{\mathcal{C}}_{t+1} \subseteq \mathcal{X}^{\mathcal{C}}_{t+1}$ be the set of all possible $(t+1)$-tuples of $\mathcal{C}$ such that for each tuple $\tau \in \mathcal{Q}^{\mathcal{C}}_{t+1}$, no two virtual caches $i,\, j \in \tau$ belong to the same physical cache. Then, for each $(t+1)$-tuple $\tau \in \mathcal{Q}^{\mathcal{C}}_{t+1}$, the BS transmits the following XOR:

$$\mathcal{Y}_\tau = \oplus_{\hat{\lambda}\in\tau} F^{d_u}_{\tau\setminus\hat{\lambda}}. \tag{12}$$

The structure of $\mathcal{Y}_\tau$ allows to serve $t+1$ users simultaneously as each user can easily decode its required subpacket using the content $\mathcal{Z}_\lambda$ of its associated cache $\lambda \in [\Lambda]$. Let $\mathcal{Y} = \left\{\mathcal{Y}_\tau : \tau \in \mathcal{Q}^{\mathcal{C}}_{t+1}\right\}$ denote the set of all transmissions. In order to completely serve the request vector $\mathbf{d}^{\hat{\mathbf{V}}}$ corresponding to cache population vector $\hat{\mathbf{V}}$, the BS transmits $|\mathcal{Y}| = \left|\mathcal{Q}^{\mathcal{C}}_{t+1}\right| = \sum_{\tau\in\mathcal{X}^{[\Lambda]}_{t+1}} \prod_{j=1}^{t+1} \hat{v}_{\tau(j)}$ XORs in the set $\mathcal{Y}$. Thus the corresponding transmission delay is given as

$$T(\hat{\mathbf{V}}) = \frac{\sum_{\tau\in\mathcal{X}^{[\Lambda]}_{t+1}} \prod_{i=1}^{t+1} \hat{v}_{\tau(i)}}{\sum_{\tau\in\mathcal{X}^{[\Lambda]}_{t}} \prod_{i=1}^{t} \hat{v}_{\tau(i)}}. \tag{13}$$

Case 2: $\mathbf{V}^j \ni v^j_\lambda < \hat{v}_\lambda$ for some $\lambda \in [\Lambda]$: The delivery scheme for this case is exactly the same as for the case when $\mathbf{V}^j = \hat{\mathbf{V}}$. However, the number of users to be served in this case is less than $\hat{\Lambda}$, i.e., $|\mathbf{U}| < \hat{\Lambda}$. Hence, there may exists some subpackets in $\mathcal{Y}$ that does not serve any user, and the BS only transmits the subpacket $\mathcal{Y}_\tau \in \mathcal{Y}$ if it serves at least one user. Let $\mathcal{A}_j \subseteq [\Lambda]$ be the subset of caches such that for each cache $\lambda \in \mathcal{A}_j$, $v^j_\lambda < \hat{v}_\lambda$ holds. Then, for any cache population vector $\mathbf{V}^j$, the total number of subpackets $\mathcal{Y}_\tau \in \mathcal{Y}$ that will not serve any user is given by $\sum_{\tau\in\mathcal{X}^{\mathcal{A}_j}_{t+1}} \prod_{i=1}^{t+1} (\hat{v}_{\tau(i)} - v^j_{\tau(i)})$. Consequently, for any cache population vector $\mathbf{V}^j \ni v^j_\lambda \leq \hat{v}_\lambda \;\forall\; \lambda \in [\Lambda]$, the transmission delay $T(\mathbf{V}^j)$ is given as

$$T(\mathbf{V}^j) = \frac{\sum_{\tau\in\mathcal{X}^{[\Lambda]}_{t+1}} \prod_{i=1}^{t+1} \hat{v}_{\tau(i)} - \sum_{\tau\in\mathcal{X}^{\mathcal{A}_j}_{t+1}} \prod_{i=1}^{t+1} (\hat{v}_{\tau(i)} - v^j_{\tau(i)})}{\sum_{\tau\in\mathcal{X}^{[\Lambda]}_{t}} \prod_{i=1}^{t} \hat{v}_{\tau(i)}}. \tag{14}$$

Hence, the transmission delay $T(\mathbf{V})$ corresponding to the cache population vector $\mathbf{V}$ is equal to (3).

Example: Let us take the example of $K = N = 10$, $\Lambda = 4$, $t = 2$, and $\mathbf{p} = (0.4, 0.2, 0.2, 0.2)$. Then the expected cache population vector is $\bar{\mathbf{V}} = [4, 2, 2, 2]$, and consequently $\hat{\mathbf{V}} = [2, 1, 1, 1]$ ($\alpha = 2$ for $\bar{\mathbf{V}}$). The set of virtual caches is $\mathcal{C} = [1, 2, 3, 4, 5]$, where the virtual caches $\mathcal{C}_1 = [1, 2]$, $\mathcal{C}_2 = [3]$, $\mathcal{C}_3 = [4]$, and $\mathcal{C}_4 = [5]$ belong to the caches 1, 2, 3, and 4 respectively. Then we have nine 2-tuples in the set $\mathcal{Q}^{\mathcal{C}}_t = [(1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5)]$ and each file is divided into nine subpackets. The content placement at each cache is given as

$\mathcal{Z}_1 = \left\{F^i_\tau : \tau \in [(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)], i \in [N]\right\}$,
$\mathcal{Z}_2 = \left\{F^i_\tau : \tau \in [(1,3),(2,3),(3,4),(3,5)], i \in [N]\right\}$,
$\mathcal{Z}_3 = \left\{F^i_\tau : \tau \in [(1,4),(2,4),(3,4),(4,5)], i \in [N]\right\}$,
$\mathcal{Z}_4 = \left\{F^i_\tau : \tau \in [(1,5),(2,5),(3,5),(4,5)], i \in [N]\right\}$.

This placement leads to the normalized cache capacity allocation of $\gamma = [\frac{6}{9}, \frac{4}{9}, \frac{4}{9}, \frac{4}{9}]$. Now, let us move to the content delivery phase. Let us assume the case of $\mathbf{V} = [6, 2, 1, 1]$, where users 1 to 6 have access to cache 1 and request the content F^1, F^2, F^3, F^4, F^5, and F^6 respectively, users 7 and 8 have access to cache 2 and request F^7 and F^8 respectively, user 9 has access to cache 3 and requests F^9, and user 10 has access to cache 4 and requests F^{10}. The BS partitions the cache population vector into the set of $\beta_{\mathbf{V}} = 3$ vectors, and transmits the content in 3 rounds. The partition of $\mathbf{V}$ based on Algorithm 1 leads to $\mathcal{B}_V = [(2,1,1,1), (2,1,0,0), (2,0,0,0)]$, and the corresponding demand vectors are $\mathbf{d}^{\mathbf{V}^1} = [1, 2, 7, 9, 10]$, $\mathbf{d}^{\mathbf{V}^2} = [3, 4, 8]$, and $\mathbf{d}^{\mathbf{V}^3} = [5, 6]$. In the first round of delivery, the BS serves user 1 and 2 from cache 1, user 7 from cache 2, user 9 from cache 3, and user 10 from cache 4. For this round, we have $\mathbf{V}^1 = \hat{\mathbf{V}}$, thus, for the demand vector $\mathbf{d}^{\mathbf{V}^1} = [1, 2, 7, 9, 10]$, the BS transmits the following seven subpackets based on set $\mathcal{Q}^{\mathcal{C}}_{t+1} = [(1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5)]$,

$\mathcal{Y}_{1,3,4} = F^1_{3,4} \oplus F^7_{1,4} \oplus F^9_{1,3}$, $\mathcal{Y}_{1,3,5} = F^1_{3,5} \oplus F^7_{1,5} \oplus F^{10}_{1,3}$
$\mathcal{Y}_{1,4,5} = F^1_{4,5} \oplus F^9_{1,5} \oplus F^{10}_{1,4}$, $\mathcal{Y}_{2,3,4} = F^2_{3,4} \oplus F^7_{2,4} \oplus F^9_{2,3}$
$\mathcal{Y}_{2,3,5} = F^2_{3,5} \oplus F^7_{2,5} \oplus F^{10}_{2,3}$, $\mathcal{Y}_{2,4,5} = F^2_{4,5} \oplus F^9_{2,5} \oplus F^{10}_{2,4}$
$\mathcal{Y}_{3,4,5} = F^7_{4,5} \oplus F^9_{3,5} \oplus F^{10}_{3,4}$.

After this round user 1, 2, 7, 9, and 10 can successfully decode their required files. Then, in the second round of delivery, the BS serves users 3 and 4 from cache 1 and user 8 from cache 2. For this round, we have $\mathbf{V}^2 \neq \hat{\mathbf{V}}$, $\mathcal{A}_2 = [3, 4]$, and the demand vector $\mathbf{d}^{\mathbf{V}^2} = [3, 4, 8]$. Thus, the BS transmits the following seven subpackets based on set $\mathcal{Q}^{\mathcal{C}}_{t+1}$,

$\mathcal{Y}_{1,3,4} = F^3_{3,4} \oplus F^8_{1,4}$, $\mathcal{Y}_{1,3,5} = F^3_{3,5} \oplus F^8_{1,5}$
$\mathcal{Y}_{2,3,4} = F^4_{3,4} \oplus F^8_{2,4}$, $\mathcal{Y}_{2,3,5} = F^4_{3,5} \oplus F^8_{2,5}$
$\mathcal{Y}_{1,4,5} = F^3_{4,5}$, $\mathcal{Y}_{2,4,5} = F^4_{4,5}$, $\mathcal{Y}_{3,4,5} = F^8_{4,5}$.

After this round users 3, 4, and 8 can successfully decode their required files. Next, in the final round of delivery, the BS serves users 5 and 6 from cache 1. For this round, we have $\mathbf{V}^3 \neq \hat{\mathbf{V}}$, $\mathcal{A}_3 = [2, 3, 4]$, and the demand vector $\mathbf{d}^{\mathbf{V}^3} = [5, 6]$. We can see that subpacket $\mathcal{Y}_{3,4,5}$ will not serve any user, thus, the BS transmits the following six subpackets based on set $\mathcal{Q}^{\mathcal{C}}_{t+1} \setminus (3, 4, 5)$,

$\mathcal{Y}_{1,3,4} = F^5_{3,4}$, $\mathcal{Y}_{1,3,5} = F^5_{3,5}$, $\mathcal{Y}_{1,4,5} = F^5_{4,5}$
$\mathcal{Y}_{2,3,4} = F^6_{3,4}$, $\mathcal{Y}_{2,3,5} = F^6_{3,5}$, $\mathcal{Y}_{2,4,5} = F^6_{4,5}$.

After this round users 5 and 6 can successfully decode their required files. This completes the content delivery phase, which results in a delivery time of $T(\mathbf{V}) = \frac{20}{9}$.

V. Numerical Evaluation

We know from Theorem 2 that it is computationally expensive to numerically evaluate the exact average delay even for small system parameters. This is due to the fact that such evaluation would require the generation of the set $\mathcal{V}$ of all possible cache population vectors $\mathbf{V}$, which corresponds to the so-called weak composition problem, and where the

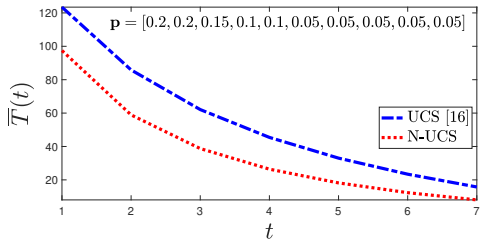

Fig. 3: $\overline{T}(t)$ from (15) (i.e., N-UCS) vs $\overline{T}_{uni}$ from (16) (i.e., UCS).

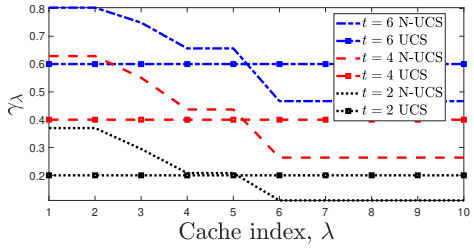

Fig. 4: Cache capacity allocations

cardinality of $\mathcal{V}$ is known to grow exponentially with system parameters K and Λ. Instead, we proceed to numerically evaluate our results by using the *sampling-based numerical* (SBN) approximation method, where we generate a large set $\mathcal{V}_1$ of randomly generated cache population vectors $\mathbf{V}$ based on cache population intensities $\mathbf{p}$, and approximate $\overline{T}(t)$ as

$$\overline{T}(t) \approx \frac{1}{|\mathcal{V}_1|} \sum_{\mathbf{V} \in \mathcal{V}_1} T(\mathbf{V}), \tag{15}$$

where $T(\mathbf{V})$ is given in (3). Then, the corresponding approximate performance is evaluated by comparing it with the achievable average delay for the uniform cache size from [16], which is given as

$$\overline{T}_{uni} \approx \frac{1}{|\mathcal{V}_1|} \sum_{\mathbf{V} \in \mathcal{V}_1} \sum_{\lambda=1}^{\Lambda-t} \mathbf{L}(\lambda) \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{16}$$

where $\mathbf{L} = sort(\mathbf{V})$ is the sorted (in descending order) version of the cache load vector $\mathbf{V}$.

Figure 3 compares the SBN approximation from (15) (i.e., our non-uniform cache size (N-UCS) scheme) with the SBN approximation from (16) (i.e., uniform cache size (UCS) scheme [16]) for $|\mathcal{V}_1| = 10000$, $K = 400$, $\Lambda = 10$, and $\mathbf{p} = [0.2, 0.2, 0.15, 0.1, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05]$, where $\mathcal{V}_1$ is generated based on cache population intensities vector $\mathbf{p}$. This figure highlights the significant gain that can be achieved by allocating the cache capacity according to the cache population intensities as our scheme significantly outperforms the uniform cache size based coded caching scheme. For the same parameter setup, Figure 4 compares the cache capacity allocations of our scheme (N-UCS) with uniform cache capacity allocations (UCS) for various capacity budgets t. This new figure illustrates how our scheme allocates cache capacity in proportion with cache population intensities.

VI. Conclusion

The work explored the coded caching problem in stochastic shared-cache networks, where each user can appear within the coverage area of one of Λ caches with a given probability, and in this context proposed a scheme that optimizes the storage allocation of caches under a cumulative cache-size constraint. The novel scheme alleviates the adverse effect of cache-load-imbalance by significantly ameliorating the detrimental performance deterioration due to randomness. Furthermore, for each and every instance of the coded caching problem, our scheme substantially alleviates – compared to the best known state-of-art — the well-known subpacketization bottleneck.

References

[1] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[2] K. Wan, D. Tuninetti, and P. Piantanida, "An index coding approach to caching with uncoded cache placement," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1318–1332, 2020.

[3] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Characterizing the rate-memory tradeoff in cache networks within a factor of 2," *IEEE Trans. Inf. Theory*, vol. 65, no. 1, pp. 647–663, Jan 2019.

[4] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless d2d networks," *IEEE Trans. Inf. Theory*, vol. 62, no. 2, pp. 849–869, Feb 2016.

[5] Q. Yan, M. Cheng, X. Tang, and Q. Chen, "On the placement delivery array design for centralized coded caching scheme," *IEEE Trans. Inf. Theory*, vol. 63, no. 9, pp. 5821–5833, Sep. 2017.

[6] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1176–1188, Jun. 2018.

[7] K. Shanmugam, M. Ji, A. M. Tulino, J. Llorca, and A. G. Dimakis, "Finite-length analysis of caching-aided coded multicasting," *IEEE Trans. Inf. Theory*, vol. 62, no. 10, pp. 5524–5537, Oct. 2016.

[8] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *IEEE Trans. Inf. Theory*, vol. 63, no. 2, pp. 1146–1158, Feb 2017.

[9] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 349–366, Jan 2018.

[10] B. Serbetci, E. Lampiris, T. Spyropoulos, and P. Elia, "Augmenting multiple-transmitter coded caching using popularity knowledge at the transmitters," in *Proc. 18th Int. Symp. on Mod. and Opt. in Mob., Ad Hoc, and Wireless Net. (WiOPT)*, Avignon, June 2020, pp. 1–8.

[11] A. Malik, B. Serbetci, E. Parrinello, and P. Elia, "Stochastic analysis of coded multicasting for shared caches networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Taipei, Taiwan, Dec. 2020, pp. 1–6.

[12] F. Brunero and P. Elia, "Fundamental limits of combinatorial multi-access caching," 2021. [Online]. Available: https://arxiv.org/pdf/2110.07426.pdf

[13] ——, "Unselfish coded caching can yield unbounded gains over symmetrically selfish caching," 2021. [Online]. Available: https://arxiv.org/pdf/2109.04807.pdf

[14] J. Hachem, N. Karamchandani, and S. Diggavi, "Coded caching for multi-level popularity and access," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 3108–3141, May 2017.

[15] E. Parrinello, A. Unsal, and P. Elia, "Fundamental limits of coded caching with multiple antennas, shared caches and uncoded prefetching," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2252–2268, Apr. 2020.

[16] A. Malik, B. Serbetci, E. Parrinello, and P. Elia, "Fundamental limits of stochastic shared-cache networks," *IEEE Trans. Commun.*, vol. 69, no. 7, pp. 4433–4447, 2021.

[17] K. Wan, D. Tuninetti, M. Ji, and G. Caire, "On the fundamental limits of fog-ran cache-aided networks with downlink and sidelink communications," *IEEE Transactions on Information Theory*, pp. 1–1, 2021.

[18] E. Parrinello and P. Elia, "Coded caching with optimized shared-cache sizes," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Visby, Sweden, Aug. 2019, pp. 1–5.