SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

EURECOM
Sophia Antipolis

Stochastic Coded Caching Networks: A Study of Cache-Load Imbalance and Random User Activity

Adeel MALIK

Dissertation submitted in partial satisfaction
of the requirements for the degree of
Doctor of Philosophy
in
Information and Communication Engineering

Thesis Supervisor: **Petros ELIA**

Defense date: 30 March 2022

Abstract

Motivated by index coding and its ability to exploit receivers' side information to create coded multicasting opportunities for users requesting different files, the concept of coded caching for cache-aided wireless networks introduced the idea of a combinatorial clique-based cache placement algorithm and a synergistic delivery scheme that enables transmitting independent content to multiple users at a time. The magnificent ability of coded caching for simultaneous delivery of an unlimited number of user requests, with a limited delay has strengthened the importance of transfiguring the storage capabilities of the network nodes (access points, helper nodes, etc.) into a fresh and powerful network resource that can complement the limited amount of network bandwidth resources to provide services to an ever-increasing number of users. This information-theoretic aspect of cache-aided networks (i.e., coded caching), has attracted significant interest from the research community. Consequently, several extensions of the coded caching in a wide range of network settings have been studied to prove that coded caching offers multiplicative gains which scale with the total storage capacity of the network. However, an overwhelming majority of these coded-caching studies focus on the deterministic information-theoretic frameworks and it is of utmost importance to transcend the boundary and study coded caching in more realistic wireless communication networks that are random in nature. In this context, this thesis elevates coded caching from their purely information-theoretic framework to a stochastic setting where the stochasticity of the networks originates from the heterogeneity in users' request behaviors.

We first study the stochastic shared-cache setting, where the association between users and shared 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-enabled helper node based on a given probability distribution (i.e., *stochastic cache population intensities*). This setting is crucial to the development of larger, realistic coded caching networks as it captures promising scenarios such as the promising scenario of heterogeneous networks. We establish the exact performance limits of coded caching in such networks. In the scenario where the delivery involves K users and Λ cache-enabled helper nodes, our work provides a statistical analysis of the average performance of such networks, identifying in closed form, the exact optimal average delivery time. To insightfully capture this delay, we derive easy-to-compute closed-form analytical bounds that prove tight in the limit of a large number of shared caches Λ. Under the premise that more balanced user-to-cache associations perform better than the unbalanced ones, we reveal the exact extent of cache-load imbalance bottleneck of coded caching in such stochastic network settings and show that imbalance in cache population densities can lead to the vanishing of the coding gain. To mitigate the impact of the cache-load imbalance bottleneck of coded caching in stochastic shared-cache networks, we propose three techniques that are very effective in resolving this bottleneck.

Then, we study the subpacketization-constrained (state-constrained) coded caching networks with *heterogeneous user activity*. The subpacketization-constrained setting is the operational reality for practical coded caching networks. It is known that to benefit from the exceptional gains of coded caching, it requires each user to be allocated to their own specifically-designed cache state (cache content), which effectively requires files to be of astronomical sizes. Thus given any reasonable constraint on the file sizes, the number of distinct cache states Λ is effectively forced to be less than the number of users K that must be shared among the users, which makes this setting related to the shared-cache setting. One main difference between these two settings is that in the shared-cache setting, the user-to-cache state associations are restricted by proximity constraints between users and helper nodes, while, in the subpacketization-constrained setting, the user-to cache state association strategies can be treated as a design parameter as each user has its own cache. Therefore, in subpacketization-constrained settings, association strategies can be optimized to mitigate the impact of the cache-load imbalance bottleneck of coded caching in stochastic networks.

In this context, we study the K-user cache-aided broadcast channel with a limited number of cache states, and explore the effect of user-to-cache state association strategies in the presence of arbitrary user activity levels; a combination that strikes at the very core of the coded caching problem and its crippling subpacketization bottleneck. We first present a statistical analysis of the average worst-case delay performance of such subpacketization-constrained coded caching networks, and provide computationally efficient performance bounds as well as scaling laws for any arbitrary probability distribution of the user activity levels. Subsequently, we propose a novel user-to-cache state association algorithm that leverages the knowledge of the statistics of user activity. Next, we follow a data-driven approach that exploits the prior history on user activity levels and correlations, in order to predict interference patterns, and thus better design the caching algorithm. This optimized strategy is based on the principle that users that overlap more, interfere more, and thus have higher priority to secure complementary cache states. This strategy is proven here to be within a small constant factor from the optimal.

In summary, this thesis highlights that stochasticity in the cache-aided networks can lead to the vanishing of the gains of coded caching. We determine the exact extent of the cache-load imbalance bottleneck of coded caching in stochastic networks, which has never been explored before. Our work provides techniques to mitigate the impact of this bottleneck for the scenario where the user-to-cache state associations are restricted by proximity constraints between users and helper nodes such as in shared-cache setting as well as for the scenario where user-to-cache state associations strategies are considered as a design parameter such as in subpacketization-constrained setting.

Acknowledgements

It is a well-known fact that the journey to a Ph.D. is not a smooth ride, and it comes with many ups and downs. The way one can handle the down moments of this journey decides the extent of the ups. Therefore, it becomes crucial for a doctoral student to have the guidance and support to achieve this beautiful outcome. In this context, I believe that I am blessed to have consistent support throughout my studies from my mentors, colleagues, friends, and family. Therefore, I believe that my thesis is not the outcome of only my efforts but also reflects the guidance and support I have received throughout my studies.

The first person I would like to thank is my advisor Prof. Petros Elia, who gave me an opportunity to complete my doctoral studies under his supervision and provided me with the utmost guidance as well as support throughout my studies. His expertise on the topic has helped me to complete this thesis in time as well as has improved my research skill that will become crucial to my professional life.

Next, I would like to express my gratitude to my co-author as well as my office mate Dr. Berksan Serbetci for his helpful suggestions every time I ran out of ideas to solve my research problem. Without his help, it would have been a lot more difficult to complete this work.

I am also grateful to all the committee members, Prof. Giovanni Neglia, Prof. Antti Tölli, Prof. Dirk Slock, Prof. Mari Kobayashi, and Prof. Thrasyvoulos Spyropoulos for their precious time in evaluating my dissertation and for providing the insightful comments. It was an honor to present my work to them.

Last but not the least, I am thankful to my siblings and friends for their love and support.

Dedicated to my Amma Sain (ZuliKhan Bibi) and my Abba Sain (Malik Muhammad Aslam). Their endless love and support has made me come this far.

Contents

List of Figures

Notations, Acronyms, and Abbreviations

Notations

Following is the list of notations used throughout the dissertation.

- $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ denote the sets of naturals, integers, and reals, respectively.
- We use the notation $[n] \triangleq [1, 2, \ldots, n], \forall n \in \mathbb{Z}^{+}$.
- 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 $\mathcal{X}_k^{[n]} \triangleq \{\delta : \delta \subseteq [n], |\delta| = k\}$.
- We use $\mathbf{P}(i)$ to denote the ith element of $\mathbf{P}$.
- We use the following asymptotic notation:
  - $f(x) = O(g(x))$ will mean that there exist constants a and c such that $f(x) \leq ag(x), \forall x > c$.
  - $f(x) = o(g(x))$ will mean that $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$.
  - $f(x) = \Omega(g(x))$ will be used if $g(x) = O(f(x))$.
  - $f(x) = \omega(g(x))$ will mean that $\lim_{x \to \infty} \frac{g(x)}{f(x)} = 0$.
  - $f(x) = \Theta(g(x))$ will be used if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.
- We use the term polylog(x) to denote the class of functions $\bigcup_{k \geq 1} O((\log x)^k)$ that are polynomial in $\log x$.
- $\oplus$ will denote the bit-wise XOR operation.
- $\binom{n}{k}$ denotes the n-choose-k operation for some $n, k \in \mathbb{N}$, $n \geq k$.
- Unless otherwise stated, we assume logarithms to have base 2.

Additionally, in each chapter we will introduce more notation that will remain relevant to that chapter.

Acronyms and Abbreviations

The acronyms and abbreviations used throughout the dissertation are specified in the following.

MIMO	Multiple-Input-Multiple-Output
MAN	Maddah-Ali and Niesen
BC	Broadcast Channel
BS	Base Station
SINR	Signal-to-Interference-plus-Noise Ratio
RSSI	Received Signal Strength Indication
HetNet	Heterogeneous Networks
D2D	Device to Device
XOR	Exclusive OR
SoA	State of the Art
NUB	Numerical Upper Bound
NLB	Numerical Loweer Bound
AUB	Analytical Upper Bound
ALB	Analytical Loweer Bound
SBN	Sampling-Based Numerical approximation
GCD	Greatest Common Divisor
N-UCS	Non-Uniform Cache Size
UCS	Uniform Cache Size

Chapter 1

Introduction to Cache-aided Networks

1.1 Why Caching?

The mobile data traffic is growing unprecedentedly and exponentially [3], and with each passing day, we are heading towards the limits of existing wireless networks, which rely on limited bandwidth resources. This ever-growing amount of mobile data traffic drives the need for innovative solutions that can provide service to an ever-increasing number of users and do so with a limited amount of network bandwidth resources.

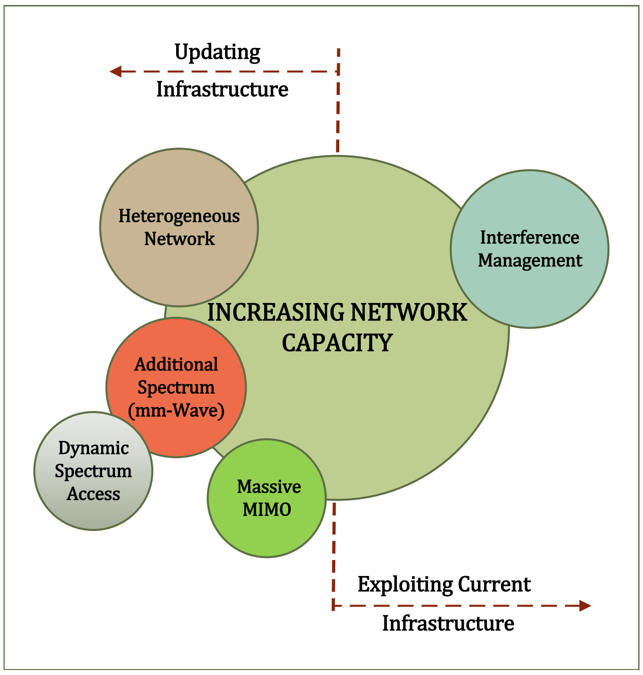


Figure 1.1: Some of the well-known traditional techniques to increase the network capacity.

In this context, the well-known traditional approaches (Figure 1.1) to increase the network capacity may involve the following approaches:

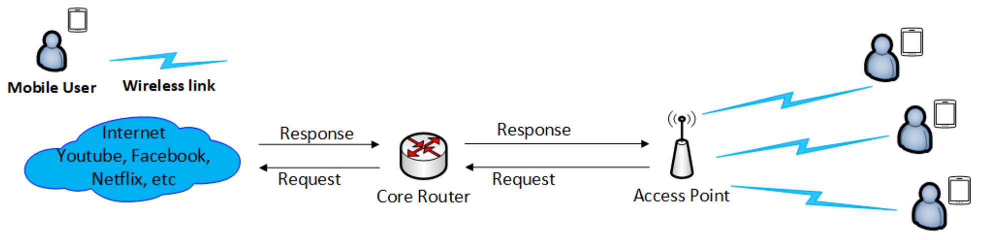
1. The deployment of heterogeneous networks in which many micro base stations are deployed along with the macro base station to improve the total spectral efficiency.

2. The use of additional spectrum and along with it efficient use of the spectral resources.

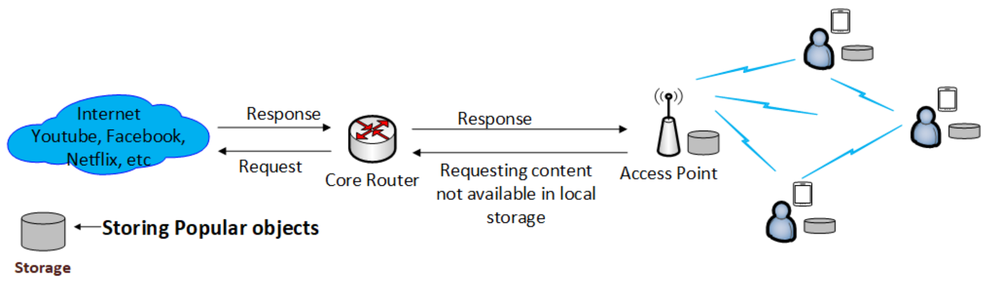
3. Multiple-Input-Multiple-Output (MIMO) communication systems in which transmitters and receivers are equipped with multiple antennas to exploit diversity and multiplexing gains.

4. Additional techniques that efficiently manage interference in order to maximize capacity.

However, these traditional approaches either come with high additional costs or do not increase the capacity to the required scale. In this context, cache-aided wireless networks have emerged as a viable option that can transfigure the storage capabilities of the network nodes (access points, helper nodes, etc.) into a fresh and powerful network resource. The main idea of cache-aided wireless networks is to bring content closer to the users by proactively storing a subset of the content library during the off-peak hours at the network edge, including at wireless communication stations [4–6] as well as on end-user devices [7–10] such that upon the user's request the content is provided locally by neighboring cache-aided network edge, without or with minimally utilizing the backhaul communication. Figure 1.2 offers an illustration of a generalized wireless network with or without caching.



(a) Wireless network without caching



(b) Wireless network with caching

Figure 1.2: A generalized wireless network with and without caching.

1.2 Coded Caching

The cache-aided wireless networks have the limitation of finite storage capacity, and one cannot cache the entire content library at the local cache. Thus, deciding which content needs to be stored at which cache plays a crucial role in the overall performance of the network. Generally caching is based on the idea that storing data can allow a receiving node to have easy access to *its own* desired file [4–10], however, recent work has shown the powerful effects of exploiting the existence of the aforementioned desired file *at the caches of other receiving users*. In interference-limited scenarios — such as in downlink settings exemplified by the broadcast channel where each user has access to their own cache and requires their own distinct file — the findings by Maddah-Ali and Niesen (MAN) in [1] suggest that a proper use of caching can allow for single multicast transmissions to simultaneously serve many users each having their own distinct demands. This breakthrough in the way caching is perceived, is based on the ideas of index coding which tells us that when the stored content in one user's cache overlaps with other users' requests, one can design multicast transmissions (in the form of XORs or other linear combinations of desired data), that allow for rapid delivery of any possible set of demands.

Motivated by index coding and its ability to exploit receivers' side information to create coded multicasting opportunities for users requesting different files, the seminal work by MAN in [1] has introduced the concept of *coded caching*. This work revealed that — under some theoretical assumptions, and in the presence of a deterministic information-theoretic broadcast framework — the use of caching at the receivers can allow the simultaneous delivery of an unlimited number of user-requests, with a limited delay. This astounding conclusion was achieved by carefully designing a combinatorial clique-based cache placement algorithm, and a synergistic delivery scheme that enables transmitting independent content to multiple users at a time. Let us now proceed with a brief introduction to the coded caching algorithm.

1.2.1 Coded Caching Algorithm by MAN [1]

The original setting in [1] considers a unit-capacity single-stream broadcast channel (BC), where a transmitting base station (BS) has access to a library (catalog) $\mathcal{F} = \left[F^1, F^2, \dots, F^N\right]$ of N unit-sized files, and serves K receiving users each equipped with a cache of size equal to the size of M files, or equivalently equal to a fraction $\gamma \triangleq \frac{M}{N} \in \left[\frac{1}{K}, \frac{2}{K}, \dots, 1\right]$ of the library. An instance of this cache-aided wireless network is shown in Figure 1.3.

The communication process consists of two phases; the *placement phase* and the *delivery phase*. During the placement phase, each user's cache is filled with the content from the library, and this phase is oblivious to the upcoming file demands in delivery phase. The delivery phase begins with users simultaneously requesting one distinct file each, and continues with the BS delivering this content to the users. The work in [1] provides a novel

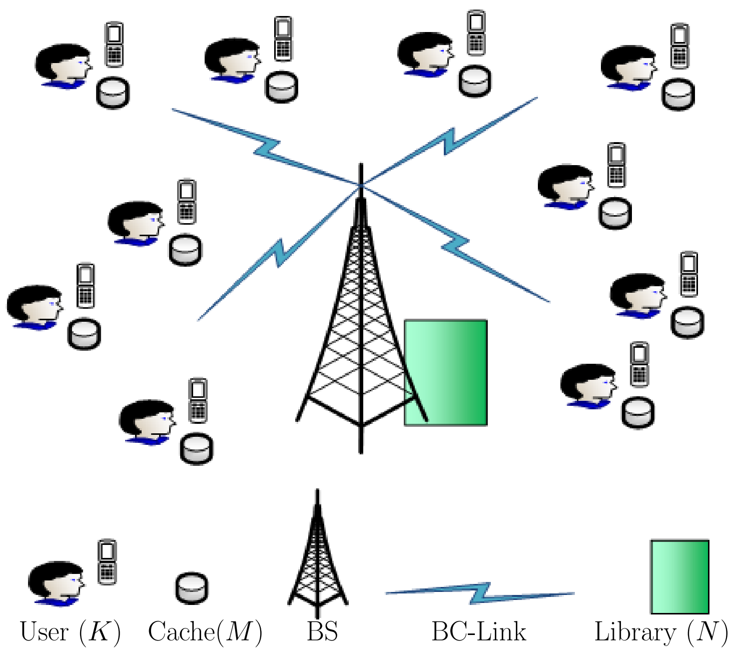


Figure 1.3: An instance of a cache-aided wireless network.

placement and delivery scheme that can serve any set of K simultaneous requests with a worst-case delivery time of $\frac{K(1-\gamma)}{1+K\gamma} \approx \frac{1}{\gamma}$. This ability to serve a theoretically ever-increasing number of users with a bounded delay, is a direct result of exploiting the cache-enabled multicasting opportunities that allow for delivery to $K\gamma+1$ users at a time. 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 (i.e., $K\gamma$). Let us now see how this scheme works.

Content Placement During the placement phase, each file $F^i \in \mathcal{F}$ is partitioned into $\binom{K}{t}$ distinct equisized subpackets, where $t \triangleq K\gamma$ for some $t \in [K]$. Then, index each subpacket of a file by a distinct subset $\tau \subseteq [K]$ of size t. The set of indexed subpackets corresponding to file $F^i \in \mathcal{F}$ is given by $\{F^i_\tau : \tau \subseteq [K], |\tau| = t\}$. The set of content to be stored at the cache of user $k \in [K]$ is given by

$$\mathcal{Z}_k = \left\{F^i_\tau : i \in [N], k \in \tau, \tau \subseteq [K], |\tau| = t\right\}.$$

Consequently, each user $k \in [K]$ cache a total of $|\mathcal{Z}_k| = N\binom{K-1}{t-1}$ subpackets, which abides by the cache-size constraint since $N\frac{\binom{K-1}{t-1}}{\binom{K}{t}} = M$.

Definition 1.1 (Cache State). *When adopting the above mentioned cache placement strategy for K cache-enabled users, we refer the content to be placed in a single cache as a cache state*[1].

[1]The cache state defines the content stored at the cache of a certain user. Two users sharing the same cache state must store the exact same content in their cache. As we will see later, having fewer cache-states generally implies a smaller DoF.

Content Delivery The delivery phase begins after each user $k \in [K]$ requests a distinct content file $F^{d_k} \in \mathcal{F}$ (i.e., worst-case), where $d_k \in [N]$ is the index of the file requested by user $k \in [K]$. Then, for a *demand vector* $\mathbf{d} = [d_1, d_2, \ldots, d_K]$, the BS sequentially transmits the following bit-wise XOR coded subpackets:

$$X_\delta = \oplus_{k \in \delta} F^{d_k}_{\delta \backslash \{k\}}, \quad \forall \delta \subseteq [K], |\delta| = t+1, \tag{1.1}$$

where for each subset $\delta \subseteq [K]$ of $t+1$ users, the corresponding bit-wise XOR coded subpacket X_δ is created in such a way that any user $k \in \delta$ can decode a subpacket $F^{d_k}_{\delta \backslash \{k\}}$ of its requested file by removing the t interfering subpakcets in X_δ using the content stored in its cache. Consequently, each bit-wise XOR coded subpacket X_δ simultaneously serves $t+1$ users.

Definition 1.2 (Delivery Time). *The delivery time (delay) corresponds to the time needed to complete the delivery of any file-demand 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.*

It is straightforward to see that after BS finishes the transmission of $\binom{K}{t+1}$ bit-wise XOR coded subpackets, each user $k \in [K]$ can easily decode $\binom{K-1}{t}$ missing subpackets of its requested file. Thus, the worst-case delivery time of MAN coded caching algorithm is given as

$$T_{MAN}(K, N, M) = \frac{\binom{K}{t+1}}{\binom{K}{t}} = \frac{K-t}{1+t} = \frac{K(1-\gamma)}{1+K\gamma}. \tag{1.2}$$

This worst-case delivery time in (1.2) achieved by MAN coded caching algorithm was shown to be information-theoretically optimal within a factor of 2 in [11]. Later, in [12], it was shown that under the assumption of uncoded cache placement, the MAN's algorithm is exactly optimal, where the uncoded cache placement means any content placement strategy that caches the bits of the content library without applying any coding. Furthermore, the work in [13] removed the worst-case assumption and characterized the performance of coded caching for the scenarios where users can have identical content requests. The delivery time of MAN coded caching algorithm for any demand vector $\mathbf{d}$ is given as

$$T_{MAN}(K, N, M, d) = \frac{\binom{K}{t+1} - \binom{K-I(\mathbf{d})}{t+1}}{\binom{K}{t}}, \tag{1.3}$$

where $I(\mathbf{d})$ denotes the number of distinct requests in $\mathbf{d}$.

Example: Let us take the example of $K = N = 3$, and $M = 1$. Then, each file F^i is partitioned into three $\left(i.e., \binom{K}{t}\right)$ distinct equisized subpackets $[F^i_1, F^i_2, F^i_3]$ and the content placement at each user is given as

$$\mathcal{Z}_1 = \left\{F^1_1, F^2_1, F^3_1\right\},$$

$$\mathcal{Z}_2 = \left\{F_2^1, F_2^2, F_2^3\right\},$$
$$\mathcal{Z}_3 = \left\{F_3^1, F_3^2, F_3^3\right\}.$$

Now, let us move to the content delivery phase. Let us assume the demand vector $\mathbf{d} = [1, 2, 3]$, where user 1 requests file F^1, user 2 requests file F^2, and user 3 requests file F^3. Then, the BS sequentially transmits the following three bit-wise XOR coded subpackets:

$$X_{1,2} = F_2^1 \oplus F_1^2$$
$$X_{1,3} = F_3^1 \oplus F_1^3$$
$$X_{2,3} = F_3^2 \oplus F_2^3.$$

After receiving $X_{1,2}$ and $X_{1,3}$, user 1 can easily decode the subpacket F_2^1 and F_3^1 by removing the interference (i.e., F_1^2 and F_1^3) using its cached content. Following the same approach user 2 and user 3 can also decode the content of their interest. Then, the delivery time corresponding to this example is $T_{MAN}(3, 3, 1) = 3 \times \frac{1}{3} = 1$.

1.2.2 Extensions of Coded Caching

The outstanding gains of MAN's coded caching scheme have attracted significant interest, and consequently, several extensions of the basic coded caching setting have been studied. Such works include the study of coded caching for arbitrary file popularity distributions [14–16], for decentralized system models [2,17], for various topology models [18–20], for MIMO broadcast channels [21,22], for PHY-based coded caching [20–29], for a variety of heterogeneous networks (HetNets) [30,31], for D2D networks [32,33], and for other settings as well [34–43]. These studies are just the tip of the iceberg, as there exists a plethora of literature on the extensions of coded caching. The focus of this thesis is mainly on the shared-cache networks, therefore, in the following, we describe in detail the application as well as the related work of coded caching for shared-cache networks. Similarly, as we proceed with the thesis, we will highlight the coded caching work relevant to our study.

1.3 Coded Caching for Shared-Cache Networks

Pivotal to the development of larger, realistic coded caching networks is the so-called *shared-cache setting*, where different users are forced to benefit from the same cache content. Such a promising scenario can be found in the context of cache-aided heterogeneous networks [4], where a central transmitter (a base station) delivers content to a set of interfering users, with the assistance of cache-enabled helper nodes that serve as caches to the users. An instance of such a network is illustrated in Figure 1.4. Such networks capture modern trends that envision a central base-station covering a larger area, in tandem with a multitude of smaller helper nodes each covering smaller cells. In this scenario, any user that appears in a particular small cell, can benefit from the cache-contents of the helper node covering that cell.

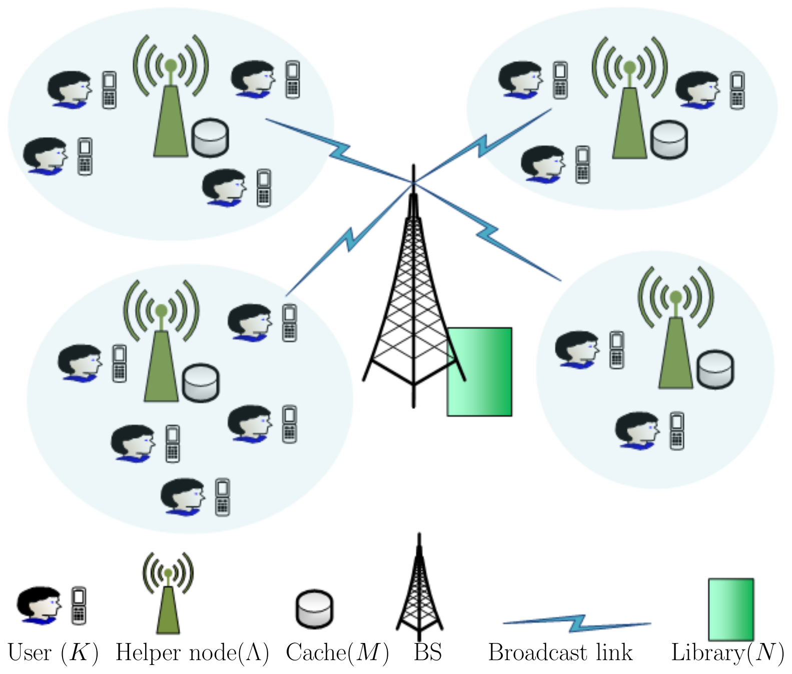


Figure 1.4: An instance of a cache-aided heterogeneous network.

In the context of coded caching, an early work on this scenario can be found in [30], which employed the uniform user-to-cache association assumption where each helper node is associated to an equal number of users. This assumption was removed in [31], which — under the assumption that content cache placement is uncoded as well as agnostic to the user-to-cache association — identified the exact optimal worst-case delivery time (i.e, the case of users' demand vector which requires the longest delivery time), as a function of the user-to-cache association profile that describes the number of users served by each cache. A similar setting was studied in [44] for the case of non-distinct requests. Let us now see how the topology-agnostic (i.e., the user-to-cache association is unknown during the content placement) coded caching works in the shared-cache networks.

1.3.1 Topology-Agnostic Coded Caching Algorithm for Shared-Cache Networks

Consider a BS having access to a library $\mathcal{F} = \left[F^1, F^2, \dots, F^N\right]$ of N equisized files, delivers content via a broadcast link to K receiving users, with the assistance of Λ cache-enabled helper nodes. Each helper node $\lambda \in [\Lambda]$ is equipped with a cache of storage capacity equal to the size of M files, thus being able to store a fraction $\gamma = \frac{M}{N} \in \left[\frac{1}{\Lambda}, \frac{2}{\Lambda}, \dots, 1\right]$ of the library. Each such helper node (cache) can assist in the delivery of content to any number of receiving users. The communication process consists of three phases; the *content placement phase*, the *user-to-cache association phase*, and the *delivery phase*.

Content Placement The first phase involves the placement of library-content in the caches, and it is oblivious to the outcome of the next two phases. The placement strategy comes directly from MAN's algorithm in Section 1.2.1, where now we have Λ cache states instead of K.

User-to-Cache Association The second phase is when each user is assigned to exactly one cache from which it can download content at zero cost. For any cache $\lambda \in [\Lambda]$, we denote by v_λ the number of users that are assisted by it, and we consider the *cache population vector* $\mathbf{V} = [v_1, \dots, v_\Lambda]$. Additionally we consider the sorted version $\mathbf{L} = [l_1, \dots, l_\Lambda] = sort(\mathbf{V})$, where $sort(\mathbf{V})$ denotes the sorting of vector $\mathbf{V}$ in descending order. We refer to $\mathbf{L}$ as a *profile vector*, and we note that each entry l_λ is simply the number of users assisted by the λ-th most populous (most heavily loaded) cache.

Content Delivery The delivery phase begins after each user $k \in [K]$ requesting a distinct content file $F^{d_k} \in \mathcal{F}$ (i.e., worst-case), where d_k is the index of the file requested by user $k \in [K]$. In this phase the BS is aware of the content of the caches, as well as aware of which cache assists each user. Then, for any demand vector $\mathbf{d} = [d_1, d_2, \dots, d_K]$, the BS delivers the content to the users following a multi-round delivery scheme proposed in [31], which is optimal for this setting under the assumption of uncoded cache placement. The main idea is that for any profile vector $\mathbf{L}$, the BS delivers the content in l_1 rounds. In each round, BS adopts the delivery strategy in [13] and delivers the content to at most 1 user from each cache $\lambda \in [\Lambda]$. This multi-round delivery scheme introduces — for any $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$ — a worst-case delivery time of

$$T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{1.4}$$

where $t = \Lambda\gamma$. For the case when Λ divides K and when $\mathbf{L}$ is uniform (i.e., $l_\lambda = \frac{K}{\Lambda}$, $\forall \lambda \in [\Lambda]$), this delay takes the form of

$$T_{min} = \frac{K}{\Lambda} \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} = \frac{K(1-\gamma)}{1+\Lambda\gamma}, \tag{1.5}$$

which indicates that coded caching offers a multiplicative gain of $\Lambda\gamma + 1$ in this uniform setting as at each transmission $\Lambda\gamma + 1$ users are served simultaneously. From (1.4), we can see that for any non-uniform $\mathbf{L}$, the associated delay $T(\mathbf{L})$ will exceed T_{min}. This is due to the fact that when more users are associated with the same cache then these users do not have the ability to jointly receive a multicasting message that can be useful to them. We know that in practical wireless networks there could be many phenomenons that can cause randomness in $\mathbf{L}$, such as users' mobility, heterogeneity in user activity, etc. In this thesis, we will analyze the actual gain of coded caching in such stochastic settings.

It is interesting to note that to a certain extent, this same shared-cache setting also applies to the scenario where each user requests multiple files (see for example [45–47]). In addition, this setting is of great importance because it has direct relation with the so-called subpacketization bottleneck of coded caching [21, 48]. Let us now see in detail what is the subpacketization bottleneck of coded caching and how it is equivalent to the shared-cache setting.

1.3.2 Subpacketization Bottleneck of Coded Caching and Its Equivalence to Shared-Cache Setting

Recall from Section 1.2.1 that in order to achieve the originally promised *theoretical coding gains* [1], each file in the content library must be partitioned into

$$P = \binom{\Lambda}{\Lambda\gamma}.$$

unit-size subpackets, where Λ denotes the number of distinct cache states (see Definition 1.1) and P scales exponentially with Λ. 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 Λ is then subject to some physical limitations, i.e., a file cannot be partitioned into more subpackets than a certain threshold, hence forcing P 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 K since it is clear that original coded caching techniques [1] require file sizes that scale exponentially with Λ (cf. [21, 48–53]).

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. This is due to fact that in order to maintain the ability to jointly exploit multicasting opportunities, users must be associated to complementary cache states that are carefully designed and which cannot be identical. However, in the finite file size regime, the subpacketization bottleneck forces us to the state-limited scenario (where $\Lambda \ll K$). The basic problem with the state-limited scenario in coded caching is simply the fact that if two or more users are forced to share the same cache state (i.e., the same content in their caches), then these users generally do not have the ability to jointly receive a multicasting message that can be useful to all. In a hypothetical scenario where coded caching is applied across a city of, let's say, one million mobile users, one would have to assign each user with one of Λ cache states (Λ independent caches), where Λ would probably be forced to be in the double or perhaps triple digits.

It is straightforward to see that the shared-cache setting is directly related to the unavoidable subpacketization bottleneck because this bottleneck can force the use of a reduced number of distinct cache states that must be inevitably shared among the many users. One can see that both of the above isomorphic settings are of utmost importance as they capture promising scenarios (such as the heterogeneous network scenario) as well as operational realities for coded caching (namely, the subpacketization constraint).

Chapter 2

Problem Statement and Main Contribution

2.1 Motivation: Coded Caching in Stochastic Networks

Since the introduction of the idea of coded caching by MAN [1], there have been a plethora of studies to prove that coded caching offers multiplicative gains which scale with the total storage capacity of the network in a wide range of network settings. However, an overwhelming majority of these coded-caching studies focus on the deterministic information-theoretic frameworks which are mostly based on idealistic assumptions such as synchronized content requests, deterministic typologies, etc. An attempt to evaluate coded caching in the absence of such assumptions often leads to the bottlenecks [21, 39, 48, 54] of coded caching and these bottlenecks have become major obstacles in the practical implementation of coded caching. One such example which we have already discussed in the previous chapter is the subpacketization bottleneck. We know that practical wireless networks are stochastic by nature. Therefore, it is necessary to shift the focus from deterministic information-theoretic settings to analyzing the coded caching in stochastic settings in order to better understand the impacts of the heterogeneity present in wireless networks. Indeed heterogeneity can take many forms, such as in the users' request behaviors, file popularity, users' mobility patterns, etc.

Analyzing the coded caching in the presence of such heterogeneity can help us understand the actual gains of the coded caching in stochastic network settings. In addition, these studies are poised to also play a vital role in resolving the bottlenecks of coded caching. Let us see an example of how incorporating the users' request behavior in coded caching can help us resolve the subpacketization bottleneck.

Example 2.1. *Let us assume that we want to design a coded caching system for $K = 200$ cache-enabled users, each with a normalized storage capacity of $\gamma = 0.1$. Now consider that we study the users' request behavior and find* *a pattern that users are divided into 4 groups where each group consists of 50 distinct users and at any given instance of time only 50 users belonging to the same group request a file while the other 150 users are inactive. With this knowledge, for each group, we can and should design a separate coded caching system. As at any instance of the time only one of the group is active, we have that the corresponding worst-case delivery time from* (1.2) *takes the form* $\frac{50(1-0.1)}{1+5} = \frac{45}{6} = 7.5$, *and the required subpacketization rate is* $P = \binom{50}{5} \approx 2.12 \times 10^6$. *However, if do not exploit this knowledge, and design a single coded caching system for all* $K = 200$ *users, then at any instance of time, when only 50 users are requesting, the corresponding worst-case delivery time from* (1.3) *takes the form* $\frac{\binom{200}{21}-\binom{150}{21}}{\binom{200}{20}} = 8.558$, *and the required subpacketization is* $P = \binom{200}{20} \approx 1.61 \times 10^{27}$. *We can see that by exploiting the users' request behavior allows us to not only reduce the delivery time but also allows us to reduce the required subpacketization rate by a factor of* 7.6×10^{20}. *In practice, users' request behavior will be stochastic, and we may not see such clear patterns as we have seen for this simplistic example. However, this example is a motivation to explore the patterns in users' request behavior that can be exploited in coded caching.*

In this thesis, we elevate the coded caching networks from their purely information-theoretic framework to a stochastic setting, where the stochasticity of the networks originates from the heterogeneity in user's request behavior. In particular, we focus on incorporating the two stochastic phenomena in coded caching networks. The first phenomenon is the *stochastic cache population intensities* in the shared-cache setting, where each user can appear within the coverage area of a specific cache-enabled helper node based on a given probability distribution. The second phenomenon is the *heterogeneous user activity* in subpacketization-constrained coded caching networks, where at any given instance of the time a user may or may not request a file from the content library.

Recall our discussion from Section 1.3.2, where we showed that in the context of coded caching, the shared-cache setting is isomorphic to the subpacketization-constrained setting. However, at this point, we need to highlight two important observations. The first observation is that under the aforementioned stochastic phenomena, the utility of results of one setting to the other setting would be limited to under certain conditions, which we will discuss in detail in the coming chapters. The second observation is that these two settings differ in the context of the extent of freedom in deciding the user-to-cache association, which determines the extent of coding gain we can achieve by exploiting the multicasting opportunities. One can see that the freedom of deciding the user-to-cache association would be very limited in the shared-cache setting as compared to the subpacketization-constrained setting. In this thesis, we study the impact of these two stochastic phenomena on the performance of coded caching. We also present techniques to exploit the freedom of deciding the user-to-cache state association to achieve better performance under these stochastic settings.

2.2 Thesis Outline

In terms of contribution, this thesis is divided into two parts. The first part of the thesis (Chapter 3 and Chapter 4) focuses on coded caching in a stochastic shared-cache setting when cache population intensities are stochastic. In particular, we study 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. In Chapter 3, we analyze the performance of coded caching in such stochastic settings and identify the cache-load imbalance bottleneck of coded caching due to the stochastic cache population intensities. Then, in Chapter 4, we identify the techniques that can mitigate the impact of cache-load imbalance bottleneck of coded caching in the stochastic shared-cache network. The second part of the thesis (Chapter 5) focuses on the subpacketization-constrained coded caching in a stochastic setting when user activity is random. In particular, we study the K-user cache-aided broadcast channel with a limited number of cache states, and explore the effect of user-to-cache state association strategies in the presence of arbitrary user activity levels; a combination that strikes at the very core of the coded caching problem and its crippling subpacketization bottleneck. Then, we conclude this thesis in Chapter 6 where we discuss the main results that emerge from our study as well as we identify the possible future directions that could be the continuation of this work.

In the following, we present the preview of the main results of the thesis.

2.3 Preview of Results

Part 1: Coded Caching in Shared-Cache Networks with Stochastic Cache Population

In the first part of this thesis, we consider the shared-cache coded caching setting where a BS having access to a library of N equisized files, delivers content via a broadcast link to K receiving users, with the assistance of Λ cache-enabled helper nodes. Each helper node $\lambda \in [\Lambda]$ is equipped with a cache of normalized storage capacity γ. For any cache $\lambda \in [\Lambda]$, let p_λ be the probability that a user can appear in the coverage area of λth cache-enabled helper node such that $\mathbf{p} = [p_1, p_2, \cdots, p_\Lambda]$, where $\sum_{\lambda \in [\Lambda]} p_\lambda = 1$, denotes the cache population intensities vector. This setting implies that different broadcast sessions would experience user populations that differently span the spectrum of cache states. In the most fortunate of scenarios, a transmitter would need to deliver to a set of K users that uniformly span the Λ states, while in the most unfortunate of scenarios, a transmitter would encounter K users that happen to have an identical cache state. Both cases are rare instances of a stochastic process, which we explore in order to identify the exact optimal performance of such systems.

Identifying the Cache-Load Imbalance Bottleneck of Coded Caching due to the Stochastic Cache Population Intensities

- In Section 3.3.1, we characterize, in closed form, the exact optimal average delivery time $\overline{T}^*(\gamma)$ optimized over all placement and delivery schemes under the assumption of uncoded cache placement and under the assumption that each user can appear in the coverage area of any particular cache-enabled helper node with equal probability (i.e., statistically uniform cache population intensities), where the average is over all possible user-to-cache associations. We show that the optimal performance $\overline{T}^*(\gamma)$ corresponds to the uncoded cache placement scheme in Section 1.2.1 and the multi-round delivery scheme in Section 1.3.1. To simplify the numerical interpretation of $\overline{T}^*(\gamma)$, we propose analytical bounds that can be calculated efficiently in Section 3.3.2. Then, in Section 3.3.3, we provide the asymptotic analysis of the optimal performance $\overline{T}^*(\gamma)$ for this statistically uniform random user-to-cache association setting, in the limit of large Λ, and show that the optimal average delivery time scales as

$$\overline{T}^*(\gamma)=\begin{cases}\Theta\left(\frac{T_{min}\Lambda\log\Lambda}{K\log\frac{\Lambda\log\Lambda}{K}}\right) & \textit{if } K\in\left[\frac{\Lambda}{\text{polylog}(\Lambda)},o\left(\Lambda\log\Lambda\right)\right]\\ \Theta\left(T_{min}\right) & \textit{if } K=\Omega\left(\Lambda\log\Lambda\right),\end{cases}\tag{2.1}$$

  where $T_{min}=\Theta\left(\frac{K(1-\gamma)}{1+\Lambda\gamma}\right)$ is the delivery time corresponding to the deterministic uniform user-to-cache association setting.

- In Section 3.4, we extend our analysis to the scenario where cache population intensities are following a non-uniform distribution and propose analytical bounds on the average delivery time $\overline{T}(\gamma)$ which corresponds to the uncoded cache placement scheme in Section 1.2.1 and the multi-round delivery scheme in Section 1.3.1. In order to gain some simple and insightful form of the performance in the presence of non-uniform cache population intensities, we provide asymptotic analysis of the $\overline{T}(\gamma)$ under the assumption that cache population intensities $\mathbf{p}$ follow the Zipf distribution and show that the delivery time scales as

$$\overline{T}^*(\gamma)=\begin{cases}\Theta\left(T_{min}\Lambda\right) & & \alpha>1\\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K}+\frac{\Lambda^2}{(\log\Lambda)^2}}\right)\text{ and} & \Omega\left(T_{min}\frac{\Lambda}{\log\Lambda}\right) & \alpha=1\\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K}+\Lambda^{2\alpha}}\right)\text{ and} & \Omega\left(T_{min}\Lambda^{\alpha}\right) & 0.5<\alpha<1\\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K}+\Lambda\log\Lambda}\right)\text{ and} & \Omega\left(T_{min}\sqrt{\Lambda}\right) & \alpha=0.5\\ O\left(T_{min}\left(\sqrt{\Lambda+\frac{\Lambda^2}{K}}\right)\right)\text{ and} & \Omega\left(T_{min}\Lambda^{\alpha}\right) & \alpha<0.5,\end{cases}\tag{2.2}$$

where $\alpha > 0$ is the Zipf exponent that determines the skewness of the probability distribution.

The scaling laws of the performance reveal interesting insights about the impact of association randomness. In particular, it enables us to quantify the extent of multiplicative performance deterioration $G(\gamma) \triangleq \frac{\overline{T}^*(\gamma)}{T_{min}}$ experienced in this random setting compared to the deterministic uniform user-to-cache association setting. For example, under statistically uniform cache population intensities, when $K = \Theta(\Lambda)$, the performance deterioration $G(\gamma)$ is unbounded and scales exactly as $\Theta\left(\frac{\log \Lambda}{\log\log \Lambda}\right)$, whereas when K increases, this deterioration gradually reduces, and ceases to scale when $K = \Omega(\Lambda \log \Lambda)$.

The impact of association randomness becomes prominent under non-uniform cache population intensities. The scaling laws reveal to what extent the performance deterioration increases with α (i.e., the skewness in cache population intensities). We can see that in some cases there is no coded caching gain, e.g., the performance deterioration $G(\gamma)$ scales as $\Theta(\Lambda)$ for $\alpha > 1$. It also reveals that unlike the case of uniform cache population intensities – where the deterioration can be avoided as long as $K = \Omega(\Lambda \log \Lambda)$ – the existence of skewness in cache population intensities can lead to an unbounded deterioration irrespective of the relation between K and Λ.
This work on coded caching in stochastic shared-cache networks resulted in the following publications:

[55] *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.*

[56] *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.*

Resolving Cache-Load Imbalance

Our stochastic analyses of the coded caching in shared-cache networks revealed the detrimental impact of the user-to-cache association's randomness on the delivery time. It highlights the importance of incorporating the knowledge of the cache population intensities while designing the placement and delivery schemes as being unaware of the severeness of this non-uniformity may lead to vanishing coding gain, and the system may eventually need to confine itself to the local caching gain. Therefore, in Chapter 4, we focus on techniques to mitigate this impact. In particular, we studied the following three techniques to mitigate the impact of the cache-load imbalance bottleneck of coded caching in stochastic shared-cache networks.

Load-Balancing

In Section 4.1, we aim to reduce the detrimental impact of the user-to-cache association's randomness on the delivery time by using load-balancing methods that introduce a certain element of choice in this association. Such choice can exist naturally in different scenarios, like in the wireless cache-aided heterogeneous network setting, where each user can be within the communication range of more than one cache helper node. In particular, we focus on two load-balancing methods which will prove to allow for unbounded gains.

- First, we consider a randomized load-balancing method ϕ_r which, for any given user, picks $h \geq 2$ candidate caches at uniformly random, and then associates each such user with the least loaded cache among these h caches. We show that under the assumption of uniform cache population intensities and randomized load-balancing method ϕ_r, the optimal average delivery time scales as

$$\overline{T}^*_{\phi_r}(\gamma)=\begin{cases}\Theta\left(T_{min}\frac{\Lambda\log\log\Lambda}{K\log h}\right) & \textit{if } K=o\left(\frac{\Lambda\log\log\Lambda}{\log h}\right)\\ \Theta\left(T_{min}\right) & \textit{if } K=\Omega\left(\frac{\Lambda\log\log\Lambda}{\log h}\right).\end{cases} \tag{2.3}$$

- Randomized load-balancing method may not apply when the choice is limited by the geographical proximity. To capture this limitation, we consider the proximity-bounded load-balancing approach ϕ_p where each user benefits from the least loaded cache among h neighboring caches. We show that under the assumption of uniform cache population intensities along with proximity-bounded load-balancing ϕ_r, the optimal average delivery time scales as

$$\overline{T}^*_{\phi_p}(\gamma)=\begin{cases}\Theta\left(\frac{T_{min}\Lambda\log\frac{\Lambda}{h}}{hK\log\frac{\Lambda\log\frac{\Lambda}{h}}{hK}}\right) & \textit{if } K\in\left[\frac{\Lambda}{h\,\text{polylog}(\frac{\Lambda}{h})},o\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)\right]\\ \Theta\left(T_{min}\right) & \textit{if } K=\Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right).\end{cases} \tag{2.4}$$

Our analysis reveals that in the practical scenario where we are given a choice to associate a user to the least loaded cache among a randomly chosen group of h *neighboring* helper nodes, the performance deterioration stops scaling as early as $K=\Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)$. An even more dramatic improvement can be seen when the aforementioned neighboring/proximity constraint is lifted. It shows that load-balancing, when applicable, can play a crucial role in significantly reducing the performance deterioration due to random user-to-cache association.
This work on load-balancing in coded caching is part of the following publication:

[55] *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.*

Cache Size Optimization

In Section 4.2, we aim to optimize the individual cache sizes to mitigate the cache-load imbalance bottleneck in stochastic shared-cache networks. The main intuition is to assign more storage capacity to the caches with high population intensities. In this context, we propose a coded caching scheme that optimizes the individual cache sizes based on cache load statistics, subject to a given cumulative cache capacity. For a random user-to-cache association for any cache population intensities vector **p**, we characterize the achievable average delay of our scheme. The scaling laws of the performance under this setting is a work in progress. However, we numerically verify the effectiveness of this setting in substantially ameliorating the impact of cache-load imbalance on the coding gain.

We also show the applicability of our scheme in a similar setting, but with a deterministic user-to-cache association, which was initially studied in [57]. We show that for a deterministic user-to-cache association setting, our scheme achieves the same state-of-the-art delivery time as of [57] with a significant (exponential) reduction in subpacketization, thus making our scheme more suitable than [57] to apply in the finite file size regime.
This work on cache size optimization in stochastic shared-cache networks resulted in the following publication:

[58] *A. Malik, B. Serbetci, and P. Elia, "Stochastic coded caching with optimized shared-cache sizes and reduced subpacketization," in IEEE Int. Conf. Commun. (ICC), Seoul, South Korea, May, 2022.*

Two-Layered Shared-Cache Networks

In Section 4.3, we aim to mitigate the adverse effect of the cache-load imbalance bottleneck by adding an additional layer of cache in the shared-cache setting. In particular, we consider Λ-helper nodes, K-user shared-cache setting with each user equipped with a normalized storage capacity γ_u and each helper node equipped with the normalized storage capacity of γ. The main idea is that in addition to a cache-enabled helper node in each cell (i.e., the coverage area of a helper node), each user is also equipped with its own storage capacity. This will enable the overpopulated (and thus under-resourced) cell to have higher collective storage capacity compared to the less populated cells, and eventually alleviate the detrimental performance deterioration due to randomness.

- In Section 4.3, we propose a content placement and delivery scheme for this setting, and for the case of statistically uniform cache population intensities, we show that the average delivery time scales as

$$\overline{T}(\gamma,\gamma_u) = \begin{cases} \Theta\left((1-\gamma)\frac{1-\gamma_u-\gamma}{\gamma_u}\right) & \textit{if } \gamma_u \geq \frac{2-2\gamma}{1+c} \\ \Theta\left(\frac{c(1-\gamma-\gamma_u)}{c\gamma_u+\gamma} - \frac{\gamma\gamma_u(c-1)^2}{4c\gamma_u+4\gamma}\right) & \textit{otherwise} \end{cases}, \tag{2.5}$$

where

$$c = \begin{cases} \frac{\log \Lambda}{\log\left(\frac{\Lambda}{K}\log \Lambda\right)} & \textit{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda \log \Lambda\right)\right] \\ \frac{K}{\Lambda} & \textit{if } K = \Omega\left(\Lambda \log \Lambda\right). \end{cases} \tag{2.6}$$

We also show that for this setting, the best-case delivery time $T_{min}(\gamma, \gamma_u)$ corresponding to a deterministic uniform user-to-cell association scales as

$$T_{min}(\gamma, \gamma_u) = \begin{cases} \Theta\left((1-\gamma)\frac{1-\gamma_u-\gamma}{\gamma_u}\right) & \textit{if } \gamma_u \geq \frac{2-2\gamma}{1+c} \\ \Theta\left(\frac{c(1-\gamma-\gamma_u)}{c\gamma_u+\gamma} - \frac{\gamma\gamma_u(c-1)^2}{4c\gamma_u+4\gamma}\right) & \textit{otherwise}, \end{cases} \tag{2.7}$$

where $c = \frac{K}{\Lambda}$.

The scaling of the $\overline{T}(\gamma, \gamma_u)$ and $T_{min}(\gamma, \gamma_u)$ helps us understand the impact of the additional layer in mitigating the adverse effect of cache-load imbalance. We can identify the memory regimes for the additional cache layer that can completely nullify the impact of the cache-load imbalance such that the multiplicative gap between average delivery time and best-case delivery time is equal to one. For example, when $\gamma_u \geq \frac{2-2\gamma}{1+\frac{\log \Lambda}{log\left(\frac{\Lambda}{K}\log \Lambda\right)}}$, irrespective of K and Λ, the average delivery time $\overline{T}(\gamma, \gamma_u)$ scales in the same order as the best-case delivery time $T_{min}(\gamma, \gamma_u)$.
This work on two-layered stochastic shared-cache networks has been submitted for the following publication:

A. Malik, B. Serbetci, and P. Elia, "Resolving cache-load imbalance bottleneck of stochastic shared-cache networks," in IEEE Wireless Communications and Networking Conference (WCNC) Apr. 2022.

PART 2: Subpacketization-Constrained Coded Caching Networks with Heterogeneous User Activity

In the second part of this thesis, we analyze a subpacketization-constrained (i.e., state-constrained) coded caching network of K cache-aided users with normalized cache capacity γ when users have different activity levels. The subpacketization constraint forces users to store the content from one of the Λ distinct cache states. The basic problem with the state-limited scenario (where $\Lambda \ll K$) in coded caching is simply the fact that if two or more users are forced to store the same content in their cache (i.e., share the same cache state), then these users generally do not have the ability to jointly receive a multicasting message that can be useful to all. Such state-limited scenarios inherit a large deterioration in coding gains compared to the originally promised theoretical coding gains [1]. What we additionally learn from our study in Part 1 (Chapter 3) is that if users are assigned states at random, then this randomness imposes an additional *unbounded* performance deterioration that is a result of *unfortunate* associations where too many users share the same cache state. It is intuitive that in order to maximize the multicasting opportunities, users that are correlated in terms of their activity should be associated with different cache states. That is why the task of user-to-cache state association is of utmost importance for this subpacketization-constrained coded caching setting. The focus of this part of the thesis is to explore the effect of user-to-cache state association strategies in the presence of arbitrary user activity levels.

Statistical Approach

In Section 5.3, we study the statistical approach for the subpacketization-constrained coded caching network, when the user activity levels follow an arbitrary probability distribution $\mathbf{p} = [p_1, p_2, \ldots, p_K]$, where p_k denotes the probability that user $k \in [K]$ requests a file from the content library. The association between users and cache states is subject to an arbitrary association strategy which is defined by a matrix $\mathbf{G} = [0,1]^{\Lambda \times K}$, of which the (λ, k) element $g_{\lambda,k}$ takes the value 1 if user k is storing the content of cache state $\lambda \in [\Lambda]$, else $g_{\lambda,k} = 0$.

- In Section 5.3.1, we present a statistical analysis of the average worst-case delay performance of such subpacketization-constrained coded caching networks and provide computationally efficient bounds on the average delivery time $\overline{T}(\mathbf{G})$ for a given user-to-cache association strategy $\mathbf{G}$. We provide the asymptotic analysis of the $\overline{T}(\mathbf{G})$ for a given association strategy $\mathbf{G}$ and activity level vector $\mathbf{p}$ and show that the average delivery time $\overline{T}(\mathbf{G})$ scales as

$$\overline{T}(\mathbf{G}) = O\left(\left(\frac{K_{\mathbf{p}}}{\Lambda} + \sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i - \mu)^2)}\right)\frac{\Lambda - t}{1+t}\right), \tag{2.8}$$

  and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{K_{\mathbf{p}}}{\Lambda}\frac{\Lambda - t}{1+t}\right) \tag{2.9}$$

  where $t = \Lambda\gamma$, $K_{\mathbf{p}} = \sum_{k=1}^{K} p_k$, $\mu_i = \sum_{k=1}^{K} g_{i,k}p_k$, $\sigma_i^2 = \sum_{k=1}^{K} g_{i,k}p_k(1-p_k)$, and $\mu = \frac{K_{\mathbf{p}}}{\Lambda}$. The scaling laws of the performance reveal interesting insights about the characteristics of the order-optimal association strategy $\mathbf{G}$ that leads to the minimum average delivery time. In particular, any association strategy $\mathbf{G}$ for which the factor $\sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i - \mu)^2)}$ scales as $O\left(\frac{K_{\mathbf{p}}}{\Lambda}\right)$ would be order-optimal as the scaling order of (2.8) yields $O\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right)$, thus, giving the exact scaling law of $\overline{T}(\mathbf{G}) = \Theta\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right)$. Based on the insights from our analysis, we propose a heuristic algorithm for the user-to-cache state association $\mathbf{G}$, which aims to minimize the average delay for any given activity level vector $\mathbf{p}$.

- In Section 5.3.2, we analyze the special case of identical user activity statistics (i.e., $p_1 = p_2 \cdots = p_K = p$) and uniform user-to-cache state association $\mathbf{G}$. For this setting, we provide tighter analytical upper and lower bounds on the performance, and show that the bounds have a bounded gap between them. We provide the asymptotic analysis of the performance for this special case of identical user activity level p and show that the average delay $\overline{T}(\mathbf{G})$ corresponding to the uniform association strategy $\mathbf{G}$ scales as

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+t}\right) & \textit{if } Ip = \Omega\left(\log \Lambda\right) \\ \Theta\left(\frac{Kp(1-\gamma)\log \Lambda}{(1+t)Ip \log \frac{\log \Lambda}{Ip}}\right) & \textit{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log \Lambda)\right], \end{cases} \tag{2.10}$$

where $I = \frac{K}{\Lambda}$.

Data-Driven Approach

In Section 5.4, we extend our analysis of subpacketization-constrained coded caching networks to the data-driven setting. In this setting instead of using the predetermined set of statistics $\mathbf{p}$, we define the user activity levels based on the past S different demand vectors (i.e., users' content request histories) span over the time horizon equal to the time it takes between two user-to-cache associations. The motivation of data-driven formulation of the subpacketization-constrained coded caching networks is that it inherits a crucial property of exploiting users' activity correlation in time. For example, users with similar request patterns should be associated with different cache states as this would guarantee more multicasting opportunities during the delivery phase. On the other hand, users that rarely request files at the same time can be allocated the same cache state without causing any performance deterioration. Using the bounded-depth user-request history, we propose a heuristic user-to-cache state association algorithm which is simple to implement and which we prove to be at most at a factor of $\frac{\log S}{\log \log S}$ from the optimal.
This work on coded caching in networks with heterogeneous user activity has been submitted for the following publication:

[59] *A. Malik, B. Serbetci, and P. Elia, "Coded Caching in Networks with Heterogeneous User Activity," Submitted to IEEE/ACM Transactions on Networking.*

Chapter 3

Coded Caching in Stochastic Shared-Cache Networks

In this chapter, we establish the exact performance limits of coded caching when users share a bounded number of cache states, and when the association between users and caches, is random. Under the premise that more balanced user-to-cache associations perform better than unbalanced ones, this chapter provides a statistical analysis of the average performance of such networks, identifying in closed form, the exact optimal average delivery time. We also present a detailed analysis on the multiplicative performance deterioration experienced in this random setting compared to the performance of the well-known deterministic uniform user-to-cache association case.

3.1 Network Setting

We consider the shared-cache coded-caching setting where a transmitter having access to a library of N equisized files, delivers content via a broadcast link to K receiving users, with the assistance of Λ cache-enabled helper nodes. Each helper node $\lambda \in [\Lambda]$ is equipped with a cache of storage capacity equal to the size of M files, thus being able to store a fraction $\gamma \triangleq \frac{M}{N} \in \left[\frac{1}{\Lambda}, \frac{2}{\Lambda}, \ldots, 1\right]$ of the library. Each such helper node, which will be henceforth referred to as a 'cache', can assist in the delivery of content to any number of receiving users. The communication process consists of three phases; the *content placement phase*, the *user-to-cache association phase*, and the *delivery phase*.

Content placement phase: The first phase involves the placement of library-content in the caches, and it is oblivious to the outcome of the next two phases.

User-to-cache association phase: The second phase is when each user is assigned – independently of the placement phase – to exactly one cache from which it can download content at zero cost. This second phase is also oblivious of the other two phases[1]. For any cache $\lambda \in [\Lambda]$, we denote by v_λ the number of users that are assisted by it, and we consider the *cache population vector* $\mathbf{V} = [v_1, \ldots, v_\Lambda]$. Additionally we consider the sorted version $\mathbf{L} = [l_1, \ldots, l_\Lambda] = sort(\mathbf{V})$, where $sort(\mathbf{V})$ denotes the sorting of vector $\mathbf{V}$ in descending order. We refer to $\mathbf{L}$ as a *profile vector*, and we note that each entry l_λ is simply the number of users assisted by the λ-th most populous (most heavily loaded) cache. Figure 3.1 depicts an instance of our shared-caches setting where $\mathbf{L} = [5, 4, 3, 2]$. In this work, we assume stochastic cache population intensities. For any cache $\lambda \in [\Lambda]$, let p_λ be the probability that a user can appear in the coverage area of λth cache-enabled helper node such that $\mathbf{p} = [p_1, p_2, \cdots, p_\Lambda]$, where $\sum_{\lambda \in [\Lambda]} p_\lambda = 1$, denotes the cache population intensities vector.

Delivery phase: The delivery phase commences with each user $k \in [K]$ requesting a single library file that is indexed by $d_k \in [N]$. As is common in coded caching works, we assume that each user requests a different file. Once the transmitter is notified of the demand vector $\mathbf{d} = [d_1, d_2, \ldots, d_K]$, it commences delivery over an error-free broadcast link of bounded capacity per unit of time. Naturally this phase is aware of the content of the caches, as well as aware of which cache assists each user.

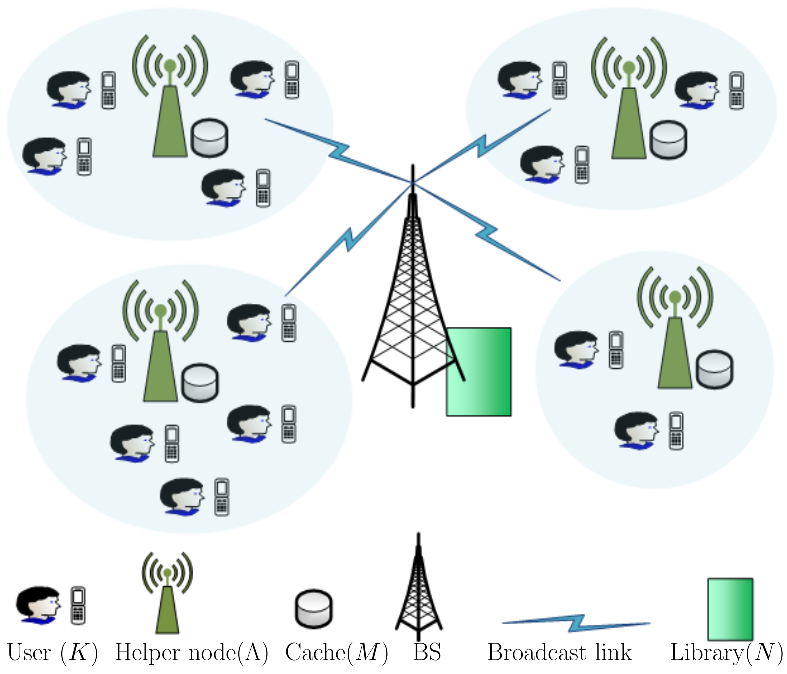


Figure 3.1: An instance of a shared-cache network for $\mathbf{L} = [5, 4, 3, 2]$.

[1]This assumption is directly motivated by the time-scales of the problem, as well as by the fact that in the heterogeneous setting, the user-to-cache association is a function of the geographical location of the user. Note that users can only be associated to caches when users are within the coverage of caches, and a dynamic user-to-cache association that requires continuous communication between the users and the server may not be desirable as one seeks to minimize the network load overhead and avoid the handover.

One can see, this setting implies that during the content delivery that follows the association of caches to each user, different broadcast sessions would experience user populations that differently span the spectrum of caches. In the most fortunate of scenarios, a transmitter would have to deliver to a set of K users that uniformly span the Λ caches (such that each cache is associated to exactly K/Λ users), while in the most unfortunate of scenarios, a transmitter would encounter K users that happen to be associated to the identical cache. Both cases are rare instances of a stochastic process, which we explore in this chapter in order to identify the exact optimal performance of such systems.

3.2 Metric of Interest

As one can imagine, any given instance of the problem, experiences a different user-to-cache association, and thus[2] a different $\mathbf{V}$. Our measure of interest is thus the average delay

$$\overline{T}(\gamma) \triangleq E_{\mathbf{V}}[T(\mathbf{V})] = \sum_{\mathbf{V}} P(\mathbf{V})T(\mathbf{V}), \tag{3.1}$$

where $T(\mathbf{V})$ is the worst-case delivery time[3] corresponding to any specific cache population vector $\mathbf{V}$, and where $P(\mathbf{V})$ is the probability that the user-to-cache association corresponds to vector $\mathbf{V}$.

More precisely, we use $T(\mathbf{V}, \mathbf{d}, \mathcal{X})$ to define the delivery time required by some generic caching-and-delivery scheme $\mathcal{X}$ to satisfy demand vector $\mathbf{d}$ when the user-to-cache association is described by the vector $\mathbf{V}$. Our aim here is to characterize the optimal average delay

$$\overline{T}^*(\gamma) = \min_{\mathcal{X}} E_{\mathbf{V}}\left[\max_{\mathbf{d}} T(\mathbf{V}, \mathbf{d}, \mathcal{X})\right] = \min_{\mathcal{X}} E_{\mathbf{L}}\left[E_{\mathbf{V}_\mathbf{L}}\left[\max_{\mathbf{d}} T(\mathbf{V}, \mathbf{d}, \mathcal{X})\right]\right], \tag{3.2}$$

where the minimization is over all possible caching and delivery schemes $\mathcal{X}$, and where $E_{\mathbf{V}_\mathbf{L}}$ denotes the expectation over all vectors $\mathbf{V}$ whose sorted version is equal to some fixed $sort(\mathbf{V}) = \mathbf{L}$. Consequently the metric of interest takes the form

$$\overline{T}(\gamma) = E_{\mathbf{L}}[T(\mathbf{L})] = \sum_{\mathbf{L}} P(\mathbf{L})T(\mathbf{L}), \tag{3.3}$$

[2]We briefly note that focusing on $\mathbf{V}$ rather than the sets of users connected to each cache, maintains all the pertinent information, as what matters for the analysis is the number of users connected to each cache and not the index (identity) of the users connected to that cache.

[3]This delay corresponds to the time needed to complete the delivery of any file-demand 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.

where $T(\mathbf{L}) \triangleq E_{\mathbf{V}_\mathbf{L}}[\max_\mathbf{d} T(\mathbf{V},\mathbf{d})]$, and where

$$P(\mathbf{L}) \triangleq \sum_{\mathbf{V}:sort(\mathbf{V})=\mathbf{L}} P(\mathbf{V}),$$

is simply the cumulative probability over all $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$.

We will consider here the uncoded cache placement scheme in [1] (see Section 1.2.1), and the delivery scheme in [31] (see Section 1.3.1), which will prove to be optimal for our setting under the common assumption of uncoded cache placement. This multi-round delivery scheme introduces – for any $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$ – a worst-case delivery time of

$$T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{3.4}$$

where $t = \Lambda\gamma$.

From (3.4) we can see that the minimum delay corresponds to the case when $\mathbf{L}$ is uniform. When Λ divides K, this minimum (uniform) delay takes the well-known form

$$T_{min} = \frac{K(1-\gamma)}{1+\Lambda\gamma}, \tag{3.5}$$

while for general K, Λ, it takes the form[4]

$$T_{min} = \frac{\Lambda - t}{1+t}\left(\left\lfloor\frac{K}{\Lambda}\right\rfloor + 1 - f(\hat{K})\right), \tag{3.6}$$

where $\hat{K} = K - \left\lfloor\frac{K}{\Lambda}\right\rfloor\Lambda$, $f(\hat{K}) = 1$ when $\hat{K} = 0$, $f(\hat{K}) = 0$ when $\hat{K} \geq \Lambda - t$, and $f(\hat{K}) = \frac{\prod_{i=t+1}^{\hat{K}+t}(\Lambda-i)}{\prod_{j=0}^{\hat{K}-1}(\Lambda-j)}$ when $0 < \hat{K} < \Lambda - t$. The proof of this is straightforward, but for completeness it can also be found in Appendix A.4. The above T_{min} is optimal under the assumption of uncoded placement (cf. [31]).

On the other hand, for any other (now non-uniform) $\mathbf{L}$, the associated delay $T(\mathbf{L})$ will exceed T_{min} (see [31] for the proof, and see Figure 3.2 for a few characteristic examples), and thus so will the average delay

$$E_\mathbf{L}[T(\mathbf{L})] = \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L})T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t}\sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} = \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda]\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{3.7}$$

where $\mathcal{L}$ describes the set of all possible profile vectors $\mathbf{L}$ (where naturally $\sum_{\lambda=1}^{\Lambda} l_\lambda = K$), and where $E[l_\lambda]$ is the expected number of users in the λ-th most populous cache[5].

[4]When $K/\Lambda \notin \mathbb{Z}^+$, the best-case delay corresponds to having $l_\lambda = \lfloor K/\Lambda \rfloor + 1$ for $\lambda \in \left[1, 2, \cdots, \hat{K}\right]$ and $l_\lambda = \lfloor K/\Lambda \rfloor$ for $\lambda \in \left[\hat{K}+1, \hat{K}+2, \cdots, \Lambda\right]$, where $\hat{K} = K - \lfloor K/\Lambda \rfloor \Lambda$.

[5]It is straightforward to see that $\sum_{\mathbf{L}\in\mathcal{L}} l_\lambda P(\mathbf{L})$ is equivalent to $\sum_{j=0}^{K} jP(l_\lambda = j) = E[l_\lambda]$, where $P(l_\lambda = j) = \sum_{\mathbf{L}\in\mathcal{L}:\mathbf{L}(\lambda)=j} P(\mathbf{L})$.

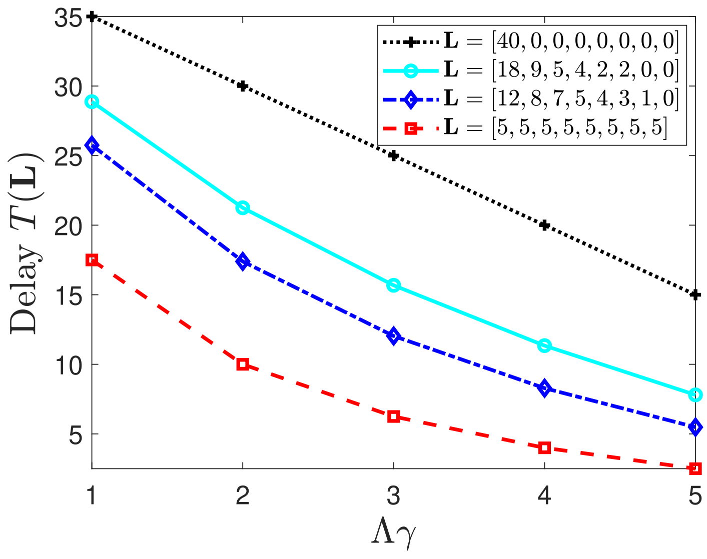


Figure 3.2: Delay $T(\mathbf{L})$ for different profile vectors $\mathbf{L}$, for K=40 and Λ=8.

In addition to the average delay $\overline{T}(\gamma)$, our second metric of interest is the multiplicative performance deterioration

$$G(\gamma) \triangleq \frac{\overline{T}(\gamma)}{T_{min}} \tag{3.8}$$

experienced in this random setting compared to the performance T_{min} of the well-known deterministic uniform case.

3.3 Main Results: Uniform Cache Population Intensities

In this section, we present our main results on the performance of the K-user broadcast channel with Λ shared-caches, each of normalized size γ, and a uniformly random user-to-cache association process, where each user can appear in the coverage area of any particular cache with equal probability (i.e., $p_\lambda = \frac{1}{\Lambda}$, $\forall \lambda \in [\Lambda]$).

3.3.1 Exact Characterization of the Optimal Average Delay

We proceed to characterize the exact optimal average delay $\overline{T}^*(\gamma)$. Crucial in this characterization will be the vector $\mathbf{B_L} = \left[b_1, b_2, \dots, b_{|B_\mathbf{L}|}\right]$, where each element $b_j \in \mathbf{B_L}$ indicates the number of caches in a distinct group of caches in which each cache has the same load[6]. Under the assumption that each

[6]For example, for a profile vector $\mathbf{L} = [5, 5, 3, 3, 3, 2, 1, 0, 0]$, there are five distinct groups in terms of having the same load, then the corresponding vector $\mathbf{B_L} = [2, 3, 1, 1, 2]$, because

user can be associated to any particular cache with equal probability, the optimal average delay $\overline{T}^*(\gamma)$ — optimized over all coded caching strategies with uncoded placement — is given by the following theorem.

Theorem 3.1. *In the K-user, Λ-caches setting with normalized cache size γ and a uniformly random user-to-cache association, the average delay*

$$\overline{T}^*(\gamma) = \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{K!\ t!\ (\Lambda-t)!\ l_\lambda \binom{\Lambda-\lambda}{t}}{\Lambda^K \prod_{i=1}^{\Lambda} l_i!\ \prod_{j=1}^{|\mathbf{B_L}|} b_j!} \tag{3.9}$$

is exactly optimal under the assumption of uncoded placement.

Proof. The proof can be found in Appendix A.1. □

One can now easily see that when $\frac{K}{\Lambda} \in \mathbb{Z}^+$, the optimal multiplicative deterioration $G(\gamma) = \frac{\overline{T}^*(\gamma)}{T_{min}}$ takes the form

$$G(\gamma) = \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{(K-1)!\ (\Lambda-t-1)!\ (t+1)!\ l_\lambda \binom{\Lambda-\lambda}{t}}{\Lambda^{K-1} \prod_{i=1}^{\Lambda} l_i!\ \prod_{j=1}^{|\mathbf{B_L}|} b_j!}. \tag{3.10}$$

Remark 3.1. *The worst-case computational time complexity for calculating the exact optimal average delay $\overline{T}^*(\gamma)$ is $O(\max(K, |\mathcal{L}|\,\Lambda))$. This complexity does not include the cost of creating the sets $\mathcal{L}$ which is an integer partition problem [60].*

We know from Theorem 3.1 that it is computationally expensive to numerically evaluate the exact average delay even for small system parameters. This is due to the fact that the cardinality of $\mathcal{L}$ is known to grow exponentially with system parameters K and Λ (see Table 3.1). This motivates our derivation of much-faster to evaluate analytical bounds on $\overline{T}^*(\gamma)$, which we provide next.

	$K = 10$	$K = 20$	$K = 30$	$K = 40$	$K = 50$
$\lvert\mathcal{L}\rvert$	42	530	3590	16928	62740

Table 3.1: Size of $\mathcal{L}$ ($\Lambda = 10$)

3.3.2 Computationally Efficient Bounds on the Optimal Performance

The following theorem bounds the optimal average delay $\overline{T}^*(\gamma)$.

two caches have a similar load of five users, three caches have a similar load of three users, two caches have a similar load of zero and all other caches have distinct number of users.

Theorem 3.2. *In the K-user, Λ-cache setting with normalized cache size γ and a uniformly random user-to-cache association, the optimal average delay $\overline{T}^*(\gamma)$ is bounded by*

$$\overline{T}^*(\gamma) \leq K\frac{\Lambda - t}{t+1} - \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right), \tag{3.11}$$

and

$$\overline{T}^*(\gamma) \geq \frac{\Lambda - t}{1+t}\left(\frac{K}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1} + \frac{t}{\Lambda-1}\left(K - \sum_{j=\lceil\frac{K}{\Lambda}\rceil}^{K-1} P_j\right)\right), \tag{3.12}$$

where

$$P_j = \sum_{i=0}^{j}\binom{K}{i}\left(\frac{1}{\Lambda}\right)^i\left(1-\frac{1}{\Lambda}\right)^{K-i}. \tag{3.13}$$

Proof. The proof is deferred to Appendix A.2. □

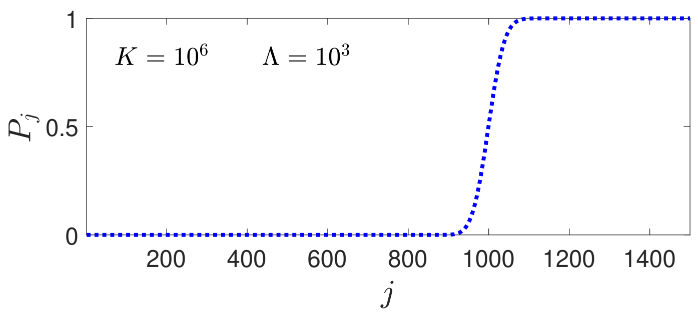


Figure 3.3: Behavior of P_j for $K = 10^6$ and $\Lambda = 10^3$.

Remark 3.2. *The worst-case computational time complexity for calculating the analytical bounds on the optimal performance $\overline{T}^*(\gamma)$ based on Theorem 3.2 is $O(\max(K\log K, K\Lambda))$. This is significantly better compared to the the complexity of $O(\max(K, |\mathcal{L}|\,\Lambda))$ for the exact calculation (cf. Theorem 3.1) The above bound is computationally efficient due to its dependence only on the P_j (cf. (3.13)), which is the cumulative distribution function (cdf) of a random variable that follows the binomial distribution with K independent trials and $\frac{1}{\Lambda}$ success probability. To compute bounds, the value of P_j needs to be calculated for all values of $j \in [0, K-1]$, which can be computationally expensive (i.e., $O(K\log K)$). However, as is known, there exists a $\tilde{j} \in [0, K-1]$, where $P_j \approx 1$. Since the cdf is a non-decreasing function in j, it is clear[7] that $P_j \approx 1$ for $j > \tilde{j}$. An illustration for $K = 10^6$, and $\Lambda = 10^3$ is shown in Figure 3.3, where it is evident that $\tilde{j} << K$.*

[7]The well-known De Moivre-Laplace Theorem can help us gain some intuition as to why the above method is computationally efficient and precise. In our case here, our binomial distribution — which according to the aforementioned theorem can be approximated by

Directly from Theorem 3.2 and (3.5), we can conclude that for $\frac{K}{\Lambda} \in \mathbb{Z}^+$, the performance deterioration $G(\gamma)$ as compared to the deterministic uniform case, is bounded as

$$G(\gamma) \leq \Lambda - \frac{t+1}{K - K\gamma} \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right), \tag{3.14}$$

and

$$G(\gamma) \geq \frac{\Lambda - t - 1}{\Lambda - 1} + \frac{\Lambda}{K} \frac{t}{\Lambda - 1} \left(K - \sum_{j=\lceil \frac{K}{\Lambda} \rceil}^{K-1} P_j \right), \tag{3.15}$$

where P_j is given in Theorem 3.2.

We now proceed to provide the exact scaling laws of the fundamental limits of the performance in a simple and insightful form.

3.3.3 Scaling Laws of Coded Caching with Random Association

The following theorem provides the asymptotic analysis of the optimal $\overline{T}^*(\gamma)$, in the limit of large Λ.

Theorem 3.3. *In the K-user, Λ-caches setting with normalized cache size γ and a uniformly random user-to-cache association, the optimal delay scales as*

$$\overline{T}^*(\gamma) = \begin{cases} \Theta\left(\frac{T_{min}\Lambda \log \Lambda}{K \log \frac{\Lambda \log \Lambda}{K}}\right) & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda \log \Lambda\right)\right] \\ \Theta\left(T_{min}\right) & \text{if } K = \Omega\left(\Lambda \log \Lambda\right). \end{cases} \tag{3.16}$$

Proof. Deferred to Appendix A.3. □

Directly from the above, we now know that the performance deterioration due to user-to-cache association randomness, scales as

$$G(\gamma) = \begin{cases} \Theta\left(\frac{\Lambda \log \Lambda}{K \log \frac{\Lambda \log \Lambda}{K}}\right) & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda \log \Lambda\right)\right] \\ \Theta\left(1\right) & \text{if } K = \Omega\left(\Lambda \log \Lambda\right), \end{cases} \tag{3.17}$$

which in turn leads to the following corollary.

Corollary 3.1. *The performance deterioration $G(\gamma)$ due to association randomness, scales as $\Theta\left(\frac{\log \Lambda}{\log \log \Lambda}\right)$ at $K = \Theta\left(\Lambda\right)$, and as K increases, this deterioration gradually reduces, and ceases to scale when $K = \Omega\left(\Lambda \log \Lambda\right)$.*

the normal distribution in the limit of large K — has mean K/Λ and standard deviation $\sqrt{K(\Lambda-1)/\Lambda^2}$. This simply means that the values within three standard deviations of the mean account for about $99.7\% \approx 100\%$ of the set. This in turn means that $P_{\tilde{j}} \approx 1$ as early on as $\tilde{j} = K/\Lambda + 3\sqrt{K(\Lambda-1)/\Lambda^2} << K$. Since $P_j \approx 1$ for $j \geq \tilde{j}$, implies that (3.13) can be rapidly evaluated with high precision.

Proof. The proof is straightforward from Theorem 3.3. □

In identifying the exact scaling laws of the problem, Theorem 3.3 nicely captures the following points.

- It describes the extent to which the performance deterioration increases with Λ and decreases with $\frac{K}{\Lambda}$.
- It reveals that the performance deterioration can in fact be unbounded.
- It shows how in certain cases, increasing Λ may yield diminishing returns due to the associated exacerbation of the random association problem. For example, to avoid a scaling $G(\gamma)$, one must approximately keep Λ below $e^{W(K)}$ ($W(.)$ is the Lambert W-function) such that $\Lambda \log \Lambda \leq K$.

At this point, we would like to highlight the utility of the results of this section to the subpacketization-constrained decentralized coded caching setting studied in [2].

Furthering the State of the Art on the Subpacketization-Constrained Decentralized Coded Caching [2]

Recall from our discussion in Section 1.3.2, where we showed that the shared-cache setting is isomorphic to the unavoidable subpacketization bottleneck of coded caching. This bottleneck forces the use of a reduced number of distinct cache states that must be inevitably shared among the many users [48]. In this context, the work in [2] proposed a decentralized coded caching in this subpacketization-constrained setting, where each cache-enabled user stores the content from one of Λ cache states with equal probability, which is exactly equivalent to our stochastic shared-cache setting with uniform cache population intensities, where each user appears in the coverage area of any particular cache-enabled helper node with equal probability. Therefore, our results of this section apply to the aforementioned related subpacketization-constrained setting which was studied in [2]. This interesting work in [2] introduced the main problem, and after providing upper bounds[8], introduced the challenge of identifying the fundamental limits of this same problem. This is indeed the challenge that is resolved in this section, where we are able to derive these fundamental limits of performance, in their exact form. We briefly mention below the utility of our results in this latter context.

- Theorem 3.1 provides the exact optimal performance in the random association setting, as well as a more efficient way to evaluate this performance compared to the state of the art (SoA) (cf. [2, Theorem 1]). The worst-case computational time complexity for calculating the exact optimal average delay $\overline{T}^*(\gamma)$ is $O\left(\max(K, |\mathcal{L}|\,\Lambda)\right)$ as compared to the

[8]It is worth noting that the provided upper bounds in [2] can have a large gap from the here-derived optimal average delay $\overline{T}^*(\gamma)$.

$O(\max(K, |\mathcal{V}|\,\Lambda \log \Lambda))$ for the case of [2, Theorem 1]. This speedup is due to the averaging being over the much smaller set $\mathcal{L}$ of all $\mathbf{L}$, rather than over the set $\mathcal{V}$ of all $\mathbf{V}$ (see Table 3.2 for a brief comparison). The time complexities mentioned above do not include the cost of creating the sets $\mathcal{L}$ and $\mathcal{V}$. However, we note that the creation of $\mathcal{V}$ is a so-called weak composition problem, whereas the creation of $\mathcal{L}$ is an integer partition problem [60]. It is easy to verify that the complexities of the algorithms for the integer partition problem are significantly lower than the ones for the weak composition problem [61–64].

	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{V}\rvert$
$K = 10$	42	92378
$K = 20$	530	10015005
$K = 30$	3590	211915132
$K = 40$	16928	2.054455634×10^9
$K = 50$	62740	$1.2565671261 \times 10^{10}$

Table 3.2: Size of $\mathcal{L}$ and $\mathcal{V}$ ($\Lambda = 10$)

- Theorem 3.2 offers a new tighter upper bound on $\overline{T}^*(\gamma)$ (see Figure 3.4) and the only known lower bound on $\overline{T}^*(\gamma)$.

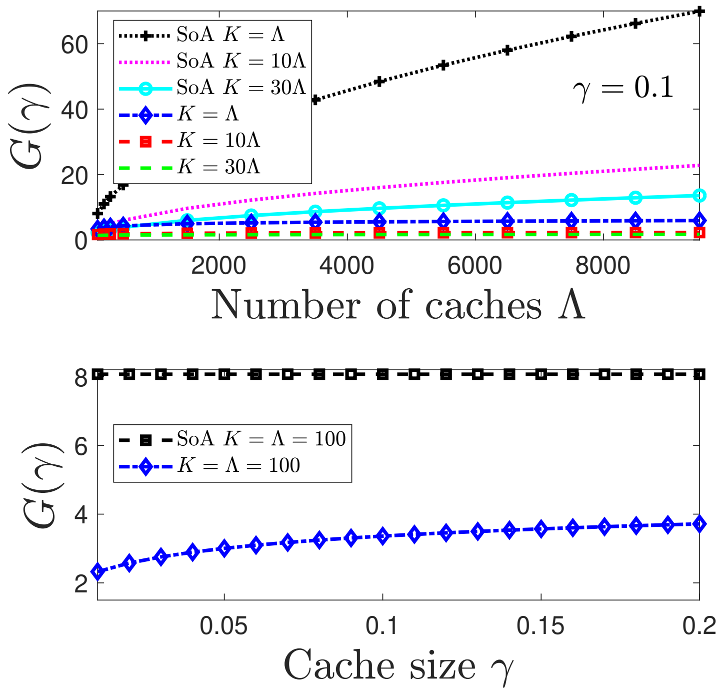


Figure 3.4: Upper bound comparison with SoA [2].

- Finally Theorem 3.3 completes our understanding of the scaling laws of the random association setting. For example, for the case where

$K = \Theta(\Lambda)$, prior to our work, $G(\gamma)$ was known to be $O\left(\sqrt{\Lambda}\right)$, whereas now we know that this deterioration scales exactly as $\Theta\left(\frac{\log \Lambda}{\log\log \Lambda}\right)$. Please refer to Table 3.3 for a detailed comparison of the known upper bounds and our exact scaling results.

	$\overline{T}^*(\gamma)$ in [2]	$\overline{T}^*(\gamma)$ in our work
$K = \Theta(\Lambda)$	$O\left(\sqrt{\Lambda}\right)$	$\Theta\left(\frac{\log \Lambda}{\log\log \Lambda}\right)$
$K = \Theta(\Lambda^a)$ for $1 < a < 2$ and $K = \Omega(\Lambda \log \Lambda)$	$O\left(\Lambda^{a/2}\right)$	$\Theta(T_{min}) = \Theta\left(\frac{K}{\Lambda}\right)$ $= \Theta(\Lambda^{a-1})$
$K = \Omega(\Lambda^2)$	$O\left(\frac{K}{\Lambda}\right)$	$\Theta(T_{min}) = \Theta\left(\frac{K}{\Lambda}\right)$

Table 3.3: SoA comparison of scaling laws.

Numerical Validation

We proceed to numerically validate our results, using two basic numerical evaluation approaches. The first is the *sampling-based numerical* (SBN) approximation method, where we generate a sufficiently large set $\mathcal{L}_1$ of randomly generated profile vectors $\mathbf{L}$, and approximate $E_{\mathbf{L}}[T(\mathbf{L})]$ as

$$E_{\mathbf{L}}[T(\mathbf{L})] \approx \frac{1}{|\mathcal{L}_1|} \sum_{\mathbf{L}\in\mathcal{L}_1} T(\mathbf{L}), \tag{3.18}$$

where we recall that $T(\mathbf{L})$ is defined in (3.4). The corresponding approximate performance deterioration is then evaluated by dividing the above by T_{min}.

The second is a *threshold-based numerical* method, whose first step is to generate a set $\mathcal{L}_2 \subseteq \mathcal{L}$ of profile vectors $\mathbf{L}$ such that $\sum_{\mathbf{L}\in\mathcal{L}_2} P(\mathbf{L}) \approx \rho$, for some chosen threshold value $\rho \in [0,1]$. Recall that the closed form expression for $P(\mathbf{L})$ is given in equation (A.1). Subsequently, with this subset $\mathcal{L}_2$ at hand, we simply have the numerical lower bound (NLB)

$$E_{\mathbf{L}}[T(\mathbf{L})] \geq \sum_{\mathbf{L}\in\mathcal{L}_2} P(\mathbf{L})T(\mathbf{L}) + (1-\rho)\, T_{min}, \tag{3.19}$$

by considering the best-case delay for each $\mathbf{L} \in \mathcal{L}/\mathcal{L}_2$, and similarly have the numerical upper bound (NUB)

$$E_{\mathbf{L}}[T(\mathbf{L})] \leq \sum_{\mathbf{L}\in\mathcal{L}_2} P(\mathbf{L})T(\mathbf{L}) + (1-\rho)\, K(1-\gamma), \tag{3.20}$$

by considering the worst possible delay $K(1-\gamma)$ for every $\mathbf{L} \in \mathcal{L}/\mathcal{L}_2$. The bounding of $G(\gamma)$ is direct by dividing the above with T_{min}.

Naturally the larger the threshold ρ, the tighter the bounds, the higher the computational cost. The additive gap between the bounds on $G(\gamma)$, takes the form $(1-\rho)\left(\frac{K(1-\gamma)}{T_{min}} - 1\right) \approx (1-\rho)\, t$, revealing the benefit of increasing ρ.

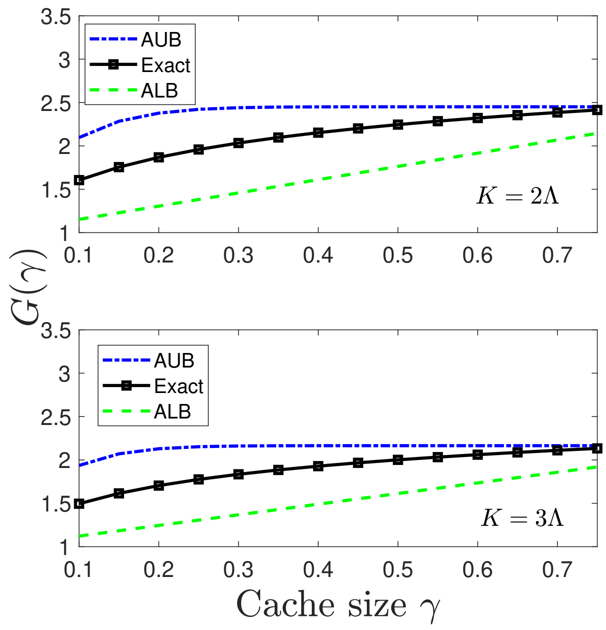


Figure 3.5: Analytical upper bound (AUB) from (3.14) vs. analytical lower bound (ALB) from (3.15) vs. exact $G(\gamma)$ from (3.10) ($\Lambda = 20$).

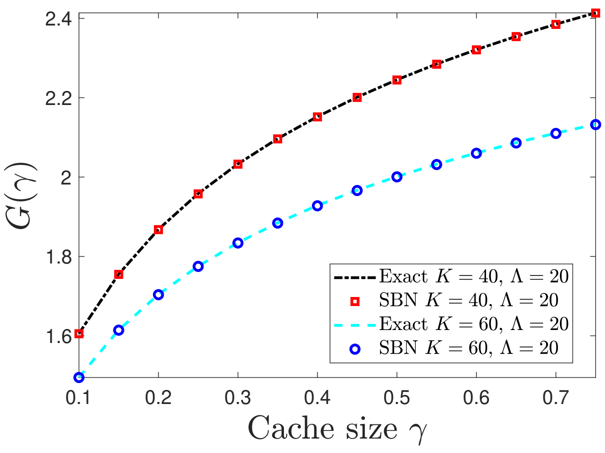


Figure 3.6: Exact $G(\gamma)$ from (3.10) vs. sampling-based numerical (SBN) approximation from (3.18) ($|\mathcal{L}_1| = 10000$).

First, Figures 3.5-3.7 include comparisons that involve the *exact* $G(\gamma)$ from (3.10), and thus – due to the computational cost – the number of caches remains at a modest $\Lambda = 20$ (and a relatively larger $\Lambda = 30$ for Figure 3.7). In particular, Figure 3.5 compares the exact $G(\gamma)$ with the analytical bounds in (3.14) and (3.15), where it is clear that both AUB and ALB yield sensible bounds, and AUB becomes much tighter as γ increases. Figure 3.6 compares the exact $G(\gamma)$ with the sampling-based numerical (SBN) approximation in (3.18) (for $|\mathcal{L}_1| = 10000$), where it is evident that the SBN approximation is consistent with the exact performance.

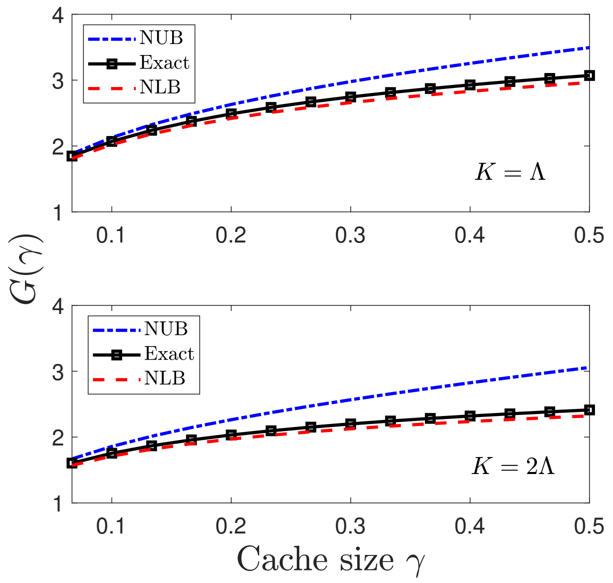


Figure 3.7: Threshold-based numerical upper bound (NUB) from (3.20) vs. threshold-based numerical lower bound (NLB) from (3.19) vs. exact $G(\gamma)$ from (3.10) ($\Lambda = 30$ and $\rho = 0.95$).

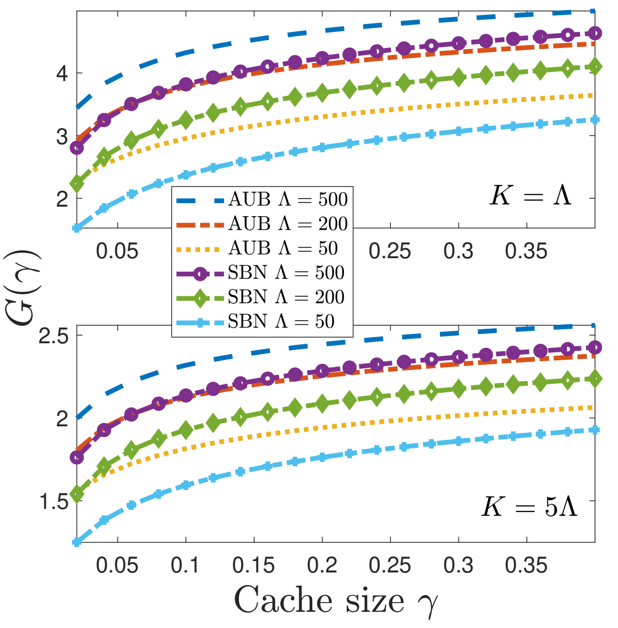


Figure 3.8: Analytical upper bound (AUB) from (3.14) vs. sampling-based numerical (SBN) approximation from (3.18) ($|\mathcal{L}_1| = 10000$).

Finally, Figure 3.7 compares the exact $G(\gamma)$ (for $\Lambda = 30$) with the threshold-based numerical bounds that are based on (3.19) and (3.20), using $\rho = 0.95$.

Interestingly, the threshold-based NLB turns out to be very tight in the entire range of γ, whereas the NUB tends to move away from the exact performance as γ increases.

Subsequently, for much larger dimensionalities, Figure 3.8 compares the AUB from (3.14) with the SBN approximation from (3.18) for $|\mathcal{L}_1| = 10000$. The figures highlight the extent to which the ratio $\frac{K}{\Lambda}$ affects the performance deterioration.

3.4 Main Results: Non-Uniform Cache Population Intensities

In this section, we extend our study to the scenario where cache population intensities $\mathbf{p}$ (i.e, probability that a user can appear in the coverage area of any particular cache-enabled cell) are following a non-uniform distribution[9].

3.4.1 Computationally Efficient Analytical Bounds

Considering the uncoded cache placement scheme in [1], and the delivery scheme in [31], the following theorem bounds the average delay $\overline{T}(\gamma)$, when cache population intensities are following a non-uniform distribution.

Theorem 3.4. *In the K-user, Λ-cache setting with normalized cache size γ and a random user-to-cache association with cache population intensities $\mathbf{p}$, the average delay $\overline{T}(\gamma)$ is bounded by*

$$\overline{T}(\gamma) \leq K\frac{\Lambda - t}{1+t} - \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(0, 1 - \frac{\Lambda - F(j)}{\lambda}\right), \tag{3.21}$$

and

$$\overline{T}(\gamma) \geq \frac{\Lambda - t}{1+t}\left(\frac{Kt\max(\mathbf{p})}{(\Lambda-1)} + \frac{K}{\Lambda}\frac{(\Lambda-t-1)}{(\Lambda-1)}\right), \tag{3.22}$$

where

$$F(j) = \sum_{k=1}^{\Lambda}\sum_{i=0}^{j} \binom{K}{i} (p_k)^i (1-p_k)^{K-i}. \tag{3.23}$$

Proof. The proof is deferred to Appendix A.5. □

Directly from Theorem 3.4 and equation (3.5), we can conclude that for $\frac{K}{\Lambda} \in \mathbb{Z}^+$, the performance deterioration $G(\gamma)$ as compared to the deterministic uniform case, is bounded as

$$G(\gamma) \leq \Lambda - \frac{1+t}{K - K\gamma} \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(0, 1 - \frac{\Lambda - F(j)}{\lambda}\right), \tag{3.24}$$

[9]All the results presented in this section are optimal for the case when the cache population intensities are not known during the cache placement phase.

and

$$G(\gamma) \geq \frac{\Lambda t \max(\mathbf{p})}{(\Lambda-1)} + \frac{(\Lambda-t-1)}{(\Lambda-1)}, \tag{3.25}$$

where $F(j)$ is given in Theorem 3.4. It is fast to numerically evaluate the analytical bound proposed in Theorem 3.4 for any given distribution of cache population intensities $\mathbf{p}$. However, in order to gain some simple and insightful form of the performance in the presence of non-uniform cache population intensities, we proceed with the asymptotic analysis of the $\overline{T}(\gamma)$ under the assumption that cache population intensities $\mathbf{p}$ follows the Zipf distribution[10]. For the Zipf distribution, cache population intensities $\mathbf{p}$ are given by[11]

$$p_\lambda = \frac{\lambda^{-\alpha}}{H_\alpha(\Lambda)}, \quad \forall \ \lambda \in [\Lambda], \tag{3.26}$$

where $\alpha > 0$ is the Zipf exponent, and $H_\alpha(\Lambda) = \sum_{i=1}^{\Lambda} i^{-\alpha}$ is a normalization constant formed as the generalized harmonic number.

3.4.2 Scaling Laws of the Performance

The following theorem provides the asymptotic analysis of the $\overline{T}(\gamma)$, in the limit of large Λ.

Theorem 3.5. *In the K-user, Λ-caches setting with normalized cache size γ and random user-to-cache association with cache population intensities $\mathbf{p}$ following the Zipf distribution with the Zipf exponent α, the delay scales as*

$$\overline{T}(\gamma) = \begin{cases} \Theta\left(T_{min}\Lambda\right) & \alpha > 1 \\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K} + \frac{\Lambda^2}{(\log\Lambda)^2}}\right) \quad \textit{and} \quad \Omega\left(T_{min}\frac{\Lambda}{\log\Lambda}\right) & \alpha = 1 \\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K} + \Lambda^{2\alpha}}\right) \quad \textit{and} \quad \Omega\left(T_{min}\Lambda^{\alpha}\right) & 0.5 < \alpha < 1 \\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K} + \Lambda\log\Lambda}\right) \quad \textit{and} \quad \Omega\left(T_{min}\sqrt{\Lambda}\right) & \alpha = 0.5 \\ O\left(T_{min}\left(\sqrt{\Lambda + \frac{\Lambda^2}{K}}\right)\right) \quad \textit{and} \quad \Omega\left(T_{min}\Lambda^{\alpha}\right) & \alpha < 0.5. \end{cases} \tag{3.27}$$

Proof. The proof is deferred to Appendix A.6. □

Directly from the above, we now know that when the cache population intensities $\mathbf{p}$ follows the Zipf distribution, the performance deterioration due

[10]There are several studies that propose different user distribution models (i.e., distribution of cache population intensities) for wireless networks [65–67]. We use the Zipf distribution as it nicely covers a wide range of non-uniform patterns by only tuning one parameter.

[11]Without loss of generality, we assume a descending order between cache population intensities of the Λ caches.

to user-to-cache association randomness, scales as

$$G(\gamma) = \begin{cases} \Theta(\Lambda) & \alpha > 1 \\ O\left(\sqrt{\frac{\Lambda^2}{K} + \frac{\Lambda^2}{(\log\Lambda)^2}}\right) \text{ and } \Omega\left(\frac{\Lambda}{\log\Lambda}\right) & \alpha = 1 \\ O\left(\sqrt{\frac{\Lambda^2}{K} + \Lambda^{2\alpha}}\right) \text{ and } \Omega(\Lambda^{\alpha}) & 0.5 < \alpha < 1 \\ O\left(\sqrt{\frac{\Lambda^2}{K} + \Lambda\log\Lambda}\right) \text{ and } \Omega\left(\sqrt{\Lambda}\right) & \alpha = 0.5 \\ O\left(\sqrt{\Lambda + \frac{\Lambda^2}{K}}\right) \text{ and } \Omega(\Lambda^{\alpha}) & \alpha < 0.5. \end{cases} \tag{3.28}$$

In identifying the scaling laws of the problem, Theorem 3.5 nicely captures the following points:

- It describes to what extent the performance deterioration increases with α (i.e., the skewness in cache population intensities).
- It shows that, in some cases there is no global caching gain, e.g., the performance deterioration scales as $\Theta(\Lambda)$ for $\alpha > 1$.
- It reveals that unlike the case of uniform cache population intensities – where the deterioration can be avoided as long as $K = \Omega(\Lambda\log\Lambda)$ – the existence of skewness in cache population intensities can lead to an unbounded deterioration irrespective of the relation between K and Λ.
- It highlights the importance of incorporating the knowledge of the cache population intensities vector while designing the placement and delivery scheme. As pointed out earlier, being unaware of the severeness of this non-uniformity may lead to the vanishing of the coding gain, and the system may eventually need to confine itself to the local caching gain.

Numerical Validation

We now numerically validate our results for the case of non-uniform cache population intensities. For the numerical analysis, we assume that cache population intensities $\mathbf{p}$ follows the Zipf distribution. Figure 3.9 compares the AUB from (3.24) with the SBN approximation from (3.18) for $|\mathcal{L}_1| = 10000$. Note that $\mathcal{L}_1$ is generated based on cache population intensities $\mathbf{p}$. The figure highlight the extent to which the Zipf exponent α (i.e., the skewness in cache population intensities) affects the performance deterioration.

3.5 Summary of Chapter

In this chapter we identified the exact optimal performance of coded caching with random user-to-cache association when users can appear in the coverage area of any particular cache-enabled cell with equal probability. In our

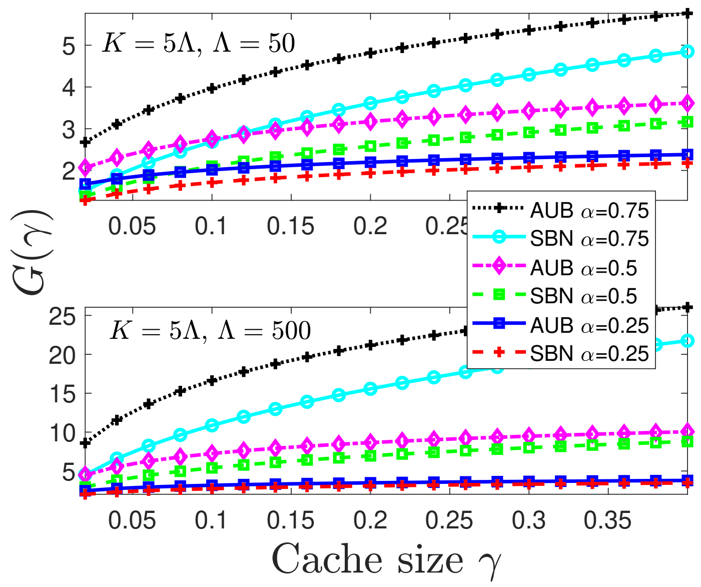


Figure 3.9: Analytical upper bound (AUB) from (3.24) vs. sampling-based numerical (SBN) approximation from (3.18) ($|\mathcal{L}_1| = 10000$).

opinion, the random association problem has direct practical ramifications, as it captures promising scenarios such as the heterogeneous network scenario as well as operational realities namely, the subpacketization constraint. The problem becomes even more pertinent as we now know that its effect can in fact scale indefinitely.

Key to our effort to identify the effect of association randomness, has been the need to provide expressions that can either be evaluated in a numerically tractable way, or that can be rigorously approximated in order to yield clear insight. The first part was achieved by deriving exact expressions as well as new analytical bounds that can be evaluated directly, while the second part was achieved by studying the asymptotics of the problem which yielded simple performance expressions and direct operational guidelines. Finally, we extended our analysis for the case where cache population intensities are following a non-uniform distribution. We provided analytical bounds and studied the asymptotics of the problem.

The detrimental impact of the user-to-cache association's randomness on the delivery time motivates the need for techniques to mitigate this impact. In the following chapter, we show how incorporating additional capabilities in the shared-cache setting can play a vital role in mitigating this impact.

Chapter 4

Mitigating the Impact of Cache-Load Imbalance Bottleneck of Coded Caching in Stochastic Shared-Cache Networks

In the previous chapter, our stochastic analyses of the coded caching in shared-cache networks revealed the impact of randomness in the cache population. We highlighted the importance of incorporating the knowledge of the cache population intensities while designing the placement and delivery scheme for this setting. In the context of known deterministic user-to-cache association setting (also known as a topology-aware scenario), the work in [68] proposed a novel coded placement that exploits knowledge of the user-to-cache association, while the work in [69] used this same knowledge, to modulate cache-sizes across the different helper nodes as a function of how many users they serve. Similarly, the work in [57] optimized over the cache sizes, again as a function of the load of each cache, and then proceeded to design a novel coded caching scheme which substantially outperforms the optimal scheme in [31]; the latter designed for the scenario where the cache placement is oblivious to the user-to-cache association phase. However, the above-mentioned studies [57, 68, 69] are limited to the deterministic user-to-cache association setting. In this chapter, we present three techniques that can mitigate the impact of the cache-load imbalance bottleneck of coded caching in stochastic shared-cache networks.

4.1 Load-Balancing in Stochastic Shared-Cache Networks

We know from our analyses in Chapter 3 that performance generally suffers when the different helper nodes (caches) are unevenly loaded. For this, it is only natural that we look at basic load-balancing approaches, which have long played a pivotal role in improving the statistical behavior of wireless networks. This role was highlighted in the survey found in [70], which discussed why long-standing assumptions about cellular networks need to be rethought in the context of load-balanced heterogeneous networks, and showed that natural user association metrics like signal-to-interference-plus-noise ratio (SINR) or received signal strength indication (RSSI) can lead to a major imbalance. This work has gathered together the earlier works on load-balancing in HetNets and compared the primary technical approaches – such as optimization, game theory and Markov decision processes – to HetNet load-balancing. In the same context, various algorithms have also been proposed to optimize the traffic load by analyzing user association to servers for cellular networks [71], by providing a distributed $\alpha-$optimal user association and cell load-balancing algorithm for wireless networks [72], by developing SINR-based flexible cell association policies in heterogeneous networks [73], and even investigating traffic load-balancing in backhaul-constrained cache-enabled small cell networks powered by hybrid energy sources [74].

In this section, we build a bridge between load-balancing and coded caching, with the aim of improving the network performance by balancing the user load placed on each cache. We will show that the effect of load-balancing can in fact be unbounded in the limit of many caches.

4.1.1 Load-Balancing Under Uniform Cache Population Intensities

In Section 3.3, we explored the performance of coded caching when each user is associated, at random and with equal probability, to one of Λ caches. Our aim now is to reduce the detrimental impact of the user-to-cache association's randomness on the delivery time, by using load-balancing methods that introduce a certain element of choice in this association, and thus allow for *better* profile vectors. Such choice can exist naturally in different scenarios, like for example in the wireless cache-aided heterogeneous network setting, where each user can be within the communication range of more than one cache helper node.

We define a generic load-balancing method ϕ to be a function that maps the set of users $[K]$ into a cache population vector $\mathbf{V} = \phi([K])$ as a result of the load-balancing choice. Similarly as in (3.2), the optimal delay, given a certain load-balancing policy ϕ, is defined as

$$\overline{T}^*_{\phi}(\gamma) = \min_{\mathcal{X}} E_{\mathbf{V}} \left[\max_{\mathbf{d}} T\left(\phi([K]), \mathbf{d}, \mathcal{X}\right) \right]. \tag{4.1}$$

The above definition is the same as the one in (3.2), with the only difference that the random variable representing the cache population vector $\mathbf{V}$ is now following a different probability distribution that depends on the load-balancing method ϕ. Employing the optimal scheme $\mathcal{X}$ from Theorem 3.1, the average delivery time takes the form (cf. equation (3.7))

$$\overline{T}_{\phi}(\gamma) = \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda] \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{4.2}$$

where $[l_1, l_2, \ldots, l_\Lambda] = sort(\phi([K]))$. It is important to point out that the choice of the load-balancing method can be in general limited by some practical constraints, such as geographical constraints and operational constraints[1]. We will focus on analyzing the above, for two load-balancing methods which will prove to allow for unbounded gains.

Randomized Load-Balancing with Multiple Choices

In the original scenario, for any given user, one cache is randomly picked to assist this user. Now we consider a load-balancing method ϕ_r which, for any given user, picks $h \geq 2$ candidate caches at random, and then associates each such user with the least loaded cache among these h caches. This static method is referred to as *randomized load-balancing with multiple choices* [75], and is considered in a variety of settings (see for example [76]). The performance of this method is presented in the following result, for the limiting case of large Λ.

Theorem 4.1. *In the K-user, Λ-cell heterogeneous network with normalized cache size γ, where each user benefits from the least loaded cache among h randomly chosen caches, the limiting optimal delay converges to*

$$\overline{T}^*_{\phi_r}(\gamma) = \begin{cases} \Theta\left(T_{min} \frac{\Lambda \log\log \Lambda}{K \log h}\right) & \text{if } K = o\left(\frac{\Lambda \log\log\Lambda}{\log h}\right) \\ \Theta\left(T_{min}\right) & \text{if } K = \Omega\left(\frac{\Lambda \log\log\Lambda}{\log h}\right) \end{cases}. \tag{4.3}$$

Proof. The achievability part of the theorem is deferred to Appendix B.1. After noticing that the definition of the optimal delay in (4.1) is equal to (3.2), optimality is proven the same way as for the optimality of Theorem 3.1 by following the same steps as in equations (A.3)-(A.4). Following those steps requires (cf. [31]) that $P(\mathbf{V})$ remains fixed for any $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$; which is true also for the considered load-balancing method ϕ_r because the method is not biased to any specific cache, i.e. ϕ_r assigns each user to one of the available caches only based on the load of the caches and independently from the cache identity. Therefore, the proof follows the same steps as for the case where there is no load-balancing. □

The above theorem naturally implies that the performance deterioration, due to random association, scales as

$$G_r(\gamma) = \begin{cases} \Theta\left(\frac{\Lambda \log\log \Lambda}{K \log h}\right) & \text{if } K = o\left(\frac{\Lambda \log\log\Lambda}{\log h}\right) \\ \Theta\left(1\right) & \text{if } K = \Omega\left(\frac{\Lambda \log\log\Lambda}{\log h}\right) \end{cases}, \tag{4.4}$$

[1]Removal of all these constraints naturally brings us back to the ideal user-to-cache association where each cache is associated to an equal number of users.

as well as implies the following corollary.

Corollary 4.1. *In the K-user, Λ-cell heterogeneous network with random-selection load-balancing, the performance deterioration due to random association, scales as $\Theta\left(\frac{\log\log\Lambda}{\log h}\right)$ when $K=\Theta(\Lambda)$, and then as K increases, this deterioration gradually reduces, and ceases to scale when $K=\Omega\left(\frac{\Lambda\log\log\Lambda}{\log h}\right)$.*

Proof. The proof is direct from (4.4). □

We can see that the above method can dramatically ameliorate the random association effect, where (for example when K is in the same order as Λ) even a small choice among $h=2$ caches, can tilt the scaling of $G(\gamma)$, from the original $\Theta\left(\frac{\log\Lambda}{\log\log\Lambda}\right)$ to a much slower $\Theta(\log\log\Lambda)$.

Load-Balancing Via Proximity-Based Cache Selection

The aforementioned randomized load-balancing method, despite its substantial impact, may not apply when the choice is limited by geographical proximity. To capture this limitation, we consider the load-balancing approach ϕ_p where the set of Λ caches is divided into Λ/h disjoint groups $\left[\mathbf{X}_1,\mathbf{X}_2,\dots,\mathbf{X}_{\Lambda/h}\right]$ of h caches each[2]. Once a user is associated at random, with uniform probability, to one of these groups, then we choose to associate this user to the least loaded cache from that group.

The performance of this method is presented in the following result, for the limiting case of large Λ.

Theorem 4.2. *In the K-user, Λ-cell heterogeneous network with normalized cache size γ, where each user benefits from the least loaded cache among h neighboring caches, then the limiting optimal delay converges to*

$$\overline{T}^*_{\phi_p}(\gamma)=\begin{cases}\Theta\left(\frac{T_{min}\Lambda\log\frac{\Lambda}{h}}{hK\log\frac{\Lambda\log\frac{\Lambda}{h}}{hK}}\right) & \text{if } K\in\left[\frac{\Lambda}{h\,\text{polylog}(\frac{\Lambda}{h})},o\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)\right]\\ \Theta\left(T_{min}\right) & \text{if } K=\Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right).\end{cases} \tag{4.5}$$

Proof. The achievability proof is deferred to Appendix B.2, while the optimality part of the theorem follows the same argument as the proof of Theorem 4.1. □

The above implies a performance deterioration of

$$G_p(\gamma)=\begin{cases}\Theta\left(\frac{\frac{\Lambda}{hK}\log\frac{\Lambda}{h}}{\log\frac{\Lambda\log\frac{\Lambda}{h}}{hK}}\right) & \text{if } K\in\left[\frac{\Lambda/h}{\text{polylog}(\frac{\Lambda}{h})},o\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)\right]\\ \Theta(1) & \text{if } K=\Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right),\end{cases} \tag{4.6}$$

which in turn implies the following.

[2]In this method, our focus is in the asymptotic setting, thus we do not need to assume that h divides Λ.

Corollary 4.2. *In the K-user, Λ-cell heterogeneous network with proximity-bounded load-balancing, the performance deterioration due to random association scales as $\Theta\left(\frac{\log(\Lambda/h)}{\log\log(\Lambda/h)}\right)$ when $K = \Theta\left(\frac{\Lambda}{h}\right)$, and as K increases, this deterioration gradually reduces, and ceases to scale when $K = \Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)$.*

Proof. The proof is straightforward from Theorem 4.2. □

We can see that proximity-bounded load-balancing significantly ameliorate the random association effect, where now deterioration ceases to scale when $K = \Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)$ compared to the original $K = \Omega(\Lambda\log\Lambda)$.

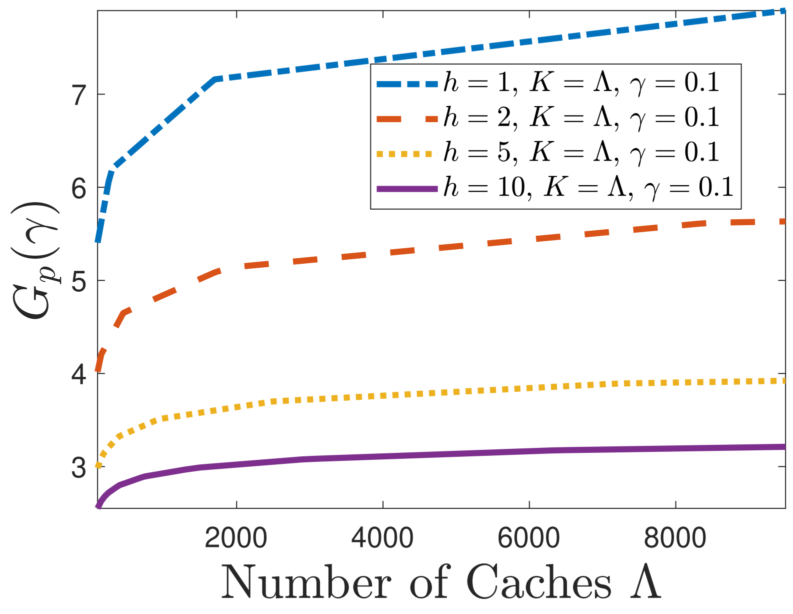


Figure 4.1: Analytical upper bound (AUB) from (4.8) without ($h = 1$) and with ($h > 1$) proximity-bounded load-balancing.

Numerical Validation

Figure 4.1 uses a suitably modified analytical upper bound to explore the effect of h when applying proximity-bounded load-balancing. We know from (A.8) that the expected number of users in the most populous cache group (i.e, $E[l_1^h]$), when each user can be associated to any cache group with equal probability $\frac{h}{\Lambda}$ is bounded as

$$E[l_1^h] \le K - \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{h}(1 - P_j^h), 0\right), \tag{4.7}$$

where $P_j^h = \sum_{i=0}^{j}\binom{K}{i}\left(\frac{h}{\Lambda}\right)^i\left(1-\frac{h}{\Lambda}\right)^{K-i}$. Also, from (B.4), the expected number of users in the most populous cache (i.e., $E[l_1]$) under proximity-bounded load-balancing is bounded as $E[l_1] < \frac{E[l_1^h]}{h} + 1$. Thus using (A.16), the analytical upper bound on the $\overline{T}^*_{\phi_p}(\gamma)$ is given by

$$\begin{aligned}\overline{T}^*_{\phi_p}(\gamma) &\leq \frac{\Lambda-t}{1+t}E[l_1] < \frac{\Lambda-t}{1+t}\left(\frac{E[l_1^h]}{h}+1\right)\\ &= \frac{\Lambda-t}{1+t}\left(1+\frac{K}{h}-\frac{1}{h}\sum_{j=0}^{K-1}\max\left(1-\frac{\Lambda}{h}(1-P_j^h),0\right)\right). \qquad (4.8)\end{aligned}$$

From Figure 4.1, we can see that, as expected, the performance deterioration decreases as h increases.

4.1.2 Load-Balancing Under Non-Uniform Cache Population Intensities

We consider a load-balancing method ϕ_n which, for any given user, picks $h \geq 2$ candidate caches at random based on the cache population intensities $\mathbf{p}$ following the Zipf distribution, and then associates each such user to the least loaded cache among these h caches. The performance of this method is presented in the following result, for the limiting case of large Λ.

Theorem 4.3. *In the K-user, Λ-cell heterogeneous network with normalized cache size γ, where each user benefits from the least loaded cache among h randomly chosen caches based on the cache population intensities $\mathbf{p}$ with the Zipf exponent α, the limiting delay converges to*

$$\overline{T}_{\phi_n}(\gamma)=\begin{cases}O\left(T_{min}\frac{\Lambda\log\log\Lambda}{K}\right) & \text{if } K=o\left(\Lambda\log\log\Lambda\right)\\ O\left(T_{min}\right) & \text{if } K=\Omega\left(\Lambda\log\log\Lambda\right),\end{cases} \qquad (4.9)$$

when $h=\Theta\left(\log\Lambda\right)$.

Proof. The proof is deferred to Appendix B.3. □

The above theorem naturally implies that, if h is in the same order as $\log\Lambda$ then the performance deterioration, due to random association, scales as

$$G_n(\gamma)=\begin{cases}O\left(\frac{\Lambda\log\log\Lambda}{K}\right) & \textit{if } K=o\left(\Lambda\log\log\Lambda\right)\\ O\left(1\right) & \textit{if } K=\Omega\left(\Lambda\log\log\Lambda\right).\end{cases} \qquad (4.10)$$

We can see that load-balancing can dramatically ameliorate the random association effect. For example, when $\alpha > 1$, picking any $\log\Lambda$ candidate caches is sufficient to tilt the scaling of $G(\gamma)$ from $\Theta\left(\Lambda\right)$ (refer to (3.28)) to a much slower $O\left(\log\log\Lambda\right)$ and $O\left(1\right)$, when $K=\Theta(\Lambda)$ and $K=\Omega\left(\Lambda\log\log\Lambda\right)$ respectively. As long as h is in the same order as $\log\Lambda$, significant improvements can be achieved irrespective of the level of skewness of the cache population intensities. In conclusion, even for the non-uniform cache population intensities, load-balancing can still be impactful. However, for non-uniform cache population intensities setting $h = 2$ may not bring significant gains which were observed for the case of uniform cache population intensities case (cf. Corollary 4.1) as now h must be in the order of $\log \Lambda$.

To summarize, in this section, we showed that in the practical scenario where we are given a choice to associate a user to the least loaded cache from a randomly chosen group of h neighboring helper nodes, the performance deterioration can be significantly reduced. An even more dramatic reduction in performance deterioration can be seen when the aforementioned neighboring/proximity constraint is lifted. The above analysis reveals that load-balancing, when applicable, can play a crucial role in significantly reducing the performance deterioration due to random user-to-cache association.

4.2 Optimizing Cache Size in Stochastic Shared-Cache Networks

In this section, 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. In this context of deterministic user-to-cache association, the work in [69] used the knowledge of user-to-cache association to adjust cache sizes as a function of cache populations. In a similar context, the work in [57] 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 [31]. Different from [31], the new scheme in [57] designed for the case when the cache placement is aware of the deterministic user-to-cache association phase, thus yielding extra gains. In this chapter, 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.

4.2.1 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 $\mathcal{F} = \{F^1, F^2, \ldots, F^N\}$, 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 $\mathbf{V} = [v_1, v_2, \dots, v_\Lambda]$ 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 $\bar{\mathbf{V}} = K\mathbf{p} = [\bar{v}_1, \bar{v}_2, \dots, \bar{v}_\Lambda]$ 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 $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \dots, \gamma_\Lambda]$. Consequently, the normalized cumulative cache-size constraint takes the form

$$\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t \triangleq \frac{M_\Sigma}{N}. \tag{4.11}$$

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 (4.11), 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 $\mathcal{Z} = [\mathcal{Z}_1, \mathcal{Z}_2, \dots, \mathcal{Z}_\Lambda]$. 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 $k \in [K]$. Then for any demand vector $\mathbf{d} = [d_1, \dots, d_{v_1}, d_{v_1+1}, \dots, d_K]$, 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.

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 time needed to complete the delivery of any demand vector $\mathbf{d}$. 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

$$\overline{T}(t) = E_{\mathbf{V}}[T(\mathbf{V})] = \sum_{\mathbf{V} \in \mathcal{V}} P(\mathbf{V}) T(\mathbf{V}), \tag{4.12}$$

where $T(\mathbf{V})$ is the worst-case time needed to complete the delivery of any demand 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 $\mathbf{V}$.

4.2.2 Main Results: Cache Size Optimization

In this section, we first present our main results on the performance of the stochastic network setting described in Section 4.2.1. 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 [57] 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 ψ, and the partition of $\mathbf{V}$ into a set $\mathcal{B}_V = [\mathbf{V}^1, \mathbf{V}^2, \dots, \mathbf{V}^{\beta_\mathbf{V}}]$ of $\beta_\mathbf{V} = \left\lceil \max_{i\in[\Lambda]} \frac{\psi v_i}{\bar{v}_i} \right\rceil$ vectors according to the partition algorithm presented in Algorithm 4.1. Each resulting partition vector $\mathbf{V}^j = [v_1^j, v_2^j, \dots, v_\Lambda^j]$ will then satisfy $v_\lambda^j \leq \frac{\bar{v}_\lambda}{\psi}$, 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 4.4. *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 $\mathbf{V}$*

$$T(\mathbf{V}) = \sum_{j\in[\beta_\mathbf{V}]} \frac{\sum\limits_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod\limits_{i=1}^{t+1} \frac{\bar{v}_{\tau(i)}}{\psi} - \sum\limits_{\tau\in\mathcal{X}_{t+1}^{\mathcal{A}_j}} \prod\limits_{i=1}^{t+1} \left(\frac{\bar{v}_{\tau(i)}}{\psi} - v_{\tau(i)}^j \right)}{\sum\limits_{\tau\in\mathcal{X}_t^{[\Lambda]}} \prod\limits_{i=1}^{t} \frac{\bar{v}_{\tau(i)}}{\psi}} \tag{4.13}$$

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 $\lambda \in \mathcal{A}_j$, $\frac{\bar{v}_\lambda}{\psi} > v_\lambda^j$.

Proof. The proof is deferred to Section 4.2.3. □

With Theorem 4.4 in hand, we present our next result, which is the average delay $\overline{T}(t)$ corresponding to our stochastic network setting.

Theorem 4.5. *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*

$$\overline{T}(t) = \sum_{\mathbf{V}\in\mathcal{V}} \frac{T(\mathbf{V})K!}{\prod_{\lambda\in[\Lambda]} v_\lambda!} \prod_{\lambda\in[\Lambda]} p_\lambda^{v_\lambda} \tag{4.14}$$

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

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

$$P(\mathbf{V}) = \frac{K!}{\prod_{\lambda\in[\Lambda]} v_\lambda!} \prod_{\lambda\in[\Lambda]} p_\lambda^{v_\lambda}. \tag{4.15}$$

Combining (4.13) with (4.15) allows us to obtain (4.14), 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 [57].

Corollary 4.3. *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 4.2.3 achieves the delivery time of*

$$T(\bar{\mathbf{V}}) = \psi \frac{\sum\limits_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod\limits_{i=1}^{t+1} \frac{\bar{v}_{\tau(i)}}{\psi}}{\sum\limits_{\tau\in\mathcal{X}_{t}^{[\Lambda]}} \prod\limits_{i=1}^{t} \frac{\bar{v}_{\tau(i)}}{\psi}}, \tag{4.16}$$

which is same as of [57, equation (11)] and it requires the subpacketization rate of

$$P = \sum_{\tau\in\mathcal{X}_{t}^{[\Lambda]}} \prod_{j=1}^{t} \frac{\bar{v}_{\tau(j)}}{\psi}, \tag{4.17}$$

which is ψ^t times less than the subpacketization rate of [57, equation (7)].

Proof. The proof is straightforward from (4.20) and (4.23). □

Corollary 4.3 reveals the substantial benefits of our scheme compared to the SoA [57] as it achieves the same delivery time with a significantly reduced – with a factor of ψ^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 [21] especially in the finite file size regimes. To illustrate this gain, in Figure 4.2, we compare the required subpacketization of our scheme with the scheme in [57] 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 4.1. *An interesting observation of our scheme is that ψ 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 $\psi = 1$. However, if we associate one virtual user to the Λ-th cache, ψ 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.*

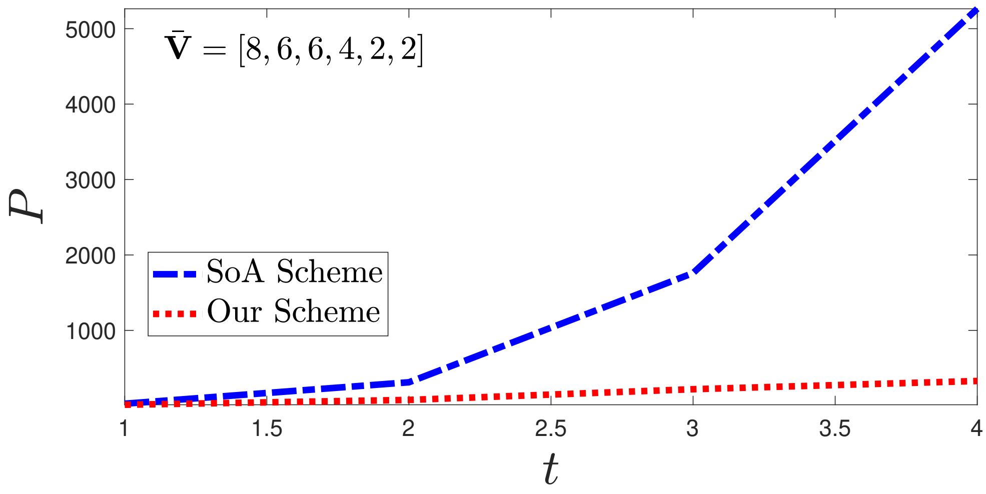


Figure 4.2: Comparison of the required subpacketization.

4.2.3 Placement and Delivery

In this section, we present the proof of Theorem 4.4. We describe the content placement and delivery strategies that can achieve the delay of (4.13).

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 $\hat{\mathbf{V}} \triangleq \frac{\bar{\mathbf{V}}}{\psi}$, where ψ 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 $F^i = \{F_\tau^i\}_{\tau \in \mathcal{Q}_t^{\mathcal{C}}}$. Then, the set of contents to be cached at each cache $\lambda \in [\Lambda]$ is given by

$$\mathcal{Z}_\lambda = \left\{ \mathcal{Z}_{\hat{\lambda}} : \hat{\lambda} \in \mathcal{C}_\lambda \right\}, \tag{4.18}$$

where

$$\mathcal{Z}_{\hat{\lambda}} = \left\{ F_\tau^i : F_\tau^i \in F^i, \tau \in \mathcal{Q}_t^{\mathcal{C}}, \hat{\lambda} \in \tau, F^i \in \mathcal{F} \right\}. \tag{4.19}$$

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

$$\left|\mathcal{Q}_t^{\mathcal{C}}\right| = \sum_{\tau \in \mathcal{X}_t^{[\Lambda]}} \prod_{j=1}^{t} \hat{v}_{\tau(j)}, \tag{4.20}$$

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

$$\gamma_\lambda = \frac{|\mathcal{Z}_\lambda|}{N\left|\mathcal{Q}_t^{\mathcal{C}}\right|} = \frac{\sum\limits_{\tau \in \mathcal{X}_t^{[\Lambda]} : \tau \ni \lambda} \prod\limits_{j=1}^{t} \hat{v}_{\tau(j)}}{\sum\limits_{\tau \in \mathcal{X}_t^{[\Lambda]}} \prod\limits_{j=1}^{t} \hat{v}_{\tau(j)}}. \tag{4.21}$$

Thus, (4.21) 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 $\{F_\tau^i : F^i \in \mathcal{F}\}$ are placed in t caches. Thus, the content placement strategy satisfies the total caching budget constraint (4.11).

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

Algorithm 4.1: Cache Population Vector Partition

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 demand vector $\mathbf{d}$, it commences delivery. We propose a delivery scheme that is completed in $\beta_{\mathbf{V}} = \left\lceil \max_{i \in [\Lambda]} \frac{v_i}{\hat{v}_i} \right\rceil$ 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 4.1, 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 4.1.

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 $\mathbf{U} = [u_1, u_2, \dots, u_{\hat{\Lambda}}]$ 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 demand vector is $\mathbf{d}^{\hat{\mathbf{V}}} = \left[d_{u_1}, d_{u_2}, \dots, d_{u_{\hat{\Lambda}}}\right]$. Let $\mathcal{Q}_{t+1}^{\mathcal{C}} \subseteq \mathcal{X}_{t+1}^{\mathcal{C}}$ be the set of all possible $(t+1)$-tuples of $\mathcal{C}$ such that for each tuple $\tau \in \mathcal{Q}_{t+1}^{\mathcal{C}}$, no two virtual caches i, $j \in \tau$ belong to the same physical cache. Then, for each $(t+1)$-tuple $\tau \in \mathcal{Q}_{t+1}^{\mathcal{C}}$, the BS transmits the following XOR:

$$\mathcal{Y}_\tau = \oplus_{\hat{\lambda}\in\tau} F_{\tau\backslash\hat{\lambda}}^{d_{u_{\hat{\lambda}}}}. \tag{4.22}$$

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 $\mathcal{Y} = \left\{\mathcal{Y}_\tau : \tau \in \mathcal{Q}_{t+1}^{\mathcal{C}}\right\}$ denote the set of all transmissions. In order to completely serve the demand vector $\mathbf{d}^{\hat{\mathbf{V}}}$ corresponding to cache population vector $\hat{\mathbf{V}}$, the BS transmits $|\mathcal{Y}| = \left|\mathcal{Q}_{t+1}^{\mathcal{C}}\right| = \sum_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod_{j=1}^{t+1} \hat{v}_{\tau(j)}$ XORs in the set $\mathcal{Y}$. Thus the corresponding transmission delay is given as

$$T(\hat{\mathbf{V}}) = \frac{\sum\limits_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod\limits_{i=1}^{t+1} \hat{v}_{\tau(i)}}{\sum\limits_{\tau\in\mathcal{X}_{t}^{[\Lambda]}} \prod\limits_{i=1}^{t} \hat{v}_{\tau(i)}}. \tag{4.23}$$

Case 2: $\mathbf{V}^j \ni v_\lambda^j < \hat{v}_\lambda$ **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 $\lambda \in \mathcal{A}_j$, $v_\lambda^j < \hat{v}_\lambda$ 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 $\sum_{\tau\in\mathcal{X}_{t+1}^{\mathcal{A}_j}} \prod_{i=1}^{t+1} (\hat{v}_{\tau(i)} - v_{\tau(i)}^j)$. Consequently, for any cache population vector $\mathbf{V}^j \ni v_\lambda^j \leq \hat{v}_\lambda \ \forall \ \lambda \in [\Lambda]$, the transmission delay $T(\mathbf{V}^j)$ is given as

$$T(\mathbf{V}^j) = \frac{\sum\limits_{\tau\in\mathcal{X}_{t+1}^{[\Lambda]}} \prod\limits_{i=1}^{t+1} \hat{v}_{\tau(i)} - \sum\limits_{\tau\in\mathcal{X}_{t+1}^{\mathcal{A}_j}} \prod\limits_{i=1}^{t+1} (\hat{v}_{\tau(i)} - v_{\tau(i)}^j)}{\sum\limits_{\tau\in\mathcal{X}_{t}^{[\Lambda]}} \prod\limits_{i=1}^{t} \hat{v}_{\tau(i)}}. \tag{4.24}$$

Hence, the transmission delay $T(\mathbf{V})$ corresponding to the cache population vector $\mathbf{V}$ is equal to (4.13).

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 $\hat{\mathbf{V}} = [2, 1, 1, 1]$ ($\psi = 2$ 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 $\mathcal{Q}_t^{\mathcal{C}} = [(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)]$ and each file is divided into nine subpackets. The content placement at each cache is given as

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

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 4.1 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 $\mathcal{Q}_{t+1}^{\mathcal{C}} = [(1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5)]$,

$$\begin{aligned}
\mathcal{Y}_{1,3,4} &= F_{3,4}^1 \oplus F_{1,4}^7 \oplus F_{1,3}^9, \\
\mathcal{Y}_{1,3,5} &= F_{3,5}^1 \oplus F_{1,5}^7 \oplus F_{1,3}^{10}, \\
\mathcal{Y}_{1,4,5} &= F_{4,5}^1 \oplus F_{1,5}^9 \oplus F_{1,4}^{10}, \\
\mathcal{Y}_{2,3,4} &= F_{3,4}^2 \oplus F_{2,4}^7 \oplus F_{2,3}^9, \\
\mathcal{Y}_{2,3,5} &= F_{3,5}^2 \oplus F_{2,5}^7 \oplus F_{2,3}^{10}, \\
\mathcal{Y}_{2,4,5} &= F_{4,5}^2 \oplus F_{2,5}^9 \oplus F_{2,4}^{10}, \\
\mathcal{Y}_{3,4,5} &= F_{4,5}^7 \oplus F_{3,5}^9 \oplus F_{3,4}^{10}.
\end{aligned}$$

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 $\mathcal{Q}_{t+1}^{\mathcal{C}}$,

$$\mathcal{Y}_{1,3,4} = F_{3,4}^3 \oplus F_{1,4}^8,$$

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

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 $\mathcal{Q}^{\mathcal{C}}_{t+1} \backslash (3,4,5)$,

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

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 $T(\mathbf{V}) = \frac{20}{9}$.

4.2.4 Numerical Evaluation

We know from Theorem 4.5 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 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

$$\overline{T}(t) \approx \frac{1}{|\mathcal{V}_1|} \sum_{\mathbf{V} \in \mathcal{V}_1} T(\mathbf{V}), \tag{4.25}$$

where $T(\mathbf{V})$ is given in (4.13). Then, the corresponding approximate performance is evaluated by comparing it with the achievable approximate average delay for the uniform cache size from Chapter 3, which as we recall is given as

$$\overline{T}(\gamma) \approx \frac{1}{|\mathcal{V}_1|} \sum_{\mathbf{V}\in\mathcal{V}_1} \sum_{\lambda=1}^{\Lambda-t} \mathbf{L}(\lambda) \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{4.26}$$

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

Figure 4.3 compares the SBN approximation from (4.25) with the SBN approximation from (4.26) 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 this scheme significantly outperforms the uniform cache size based coded caching scheme. For the same parameter setup, Figure 4.4 compares the cache capacity allocations of this scheme (N-UCS) with uniform cache capacity allocations (UCS) for various capacity budgets t. This new figure illustrates how this scheme allocates cache capacity in proportion with cache population intensities.

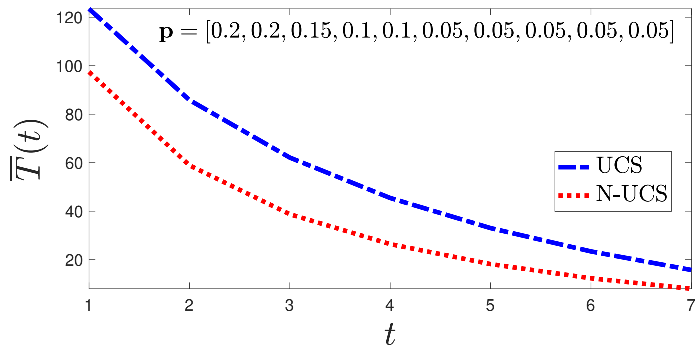


Figure 4.3: $\overline{T}(t)$ from (4.25) (non-uniform cache size N-UCS) vs $\overline{T}(\gamma)$ from (4.26) (uniform cache size UCS).

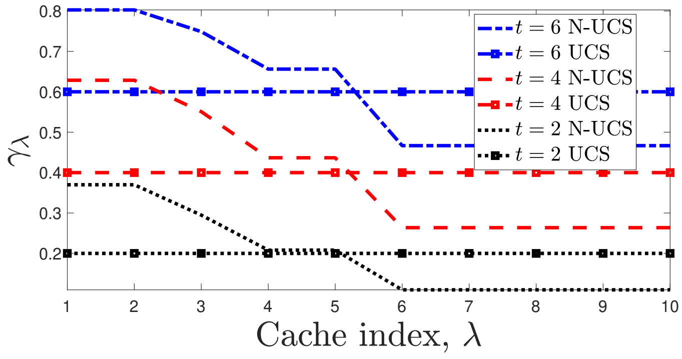


Figure 4.4: Cache capacity allocations of this scheme (N-UCS) vs uniform cache capacity allocations (UCS).

To summarize, in this section, we exploited cache-size differences and optimized the individual cache sizes to mitigate the cache-load imbalance bottleneck in stochastic shared-cache networks. We proposed a coded caching scheme that optimizes the individual cache sizes based on each cache's load statistics, subject to a given cumulative cache capacity. Our analysis revealed that the novel scheme alleviates the adverse effect of cache-load-imbalance by significantly ameliorating the detrimental performance deterioration due to randomness. We also showed that for a deterministic user-to-cache association setting, our scheme achieves the same state-of-the-art (SoA) delivery time as of [57] with a significant (exponential) reduction in subpacketization, thus making our scheme more suitable than [57] to apply in the finite file size regime.

4.3 Two-Layered Shared-Cache Networks: Adding Layers to Mitigate Randomness

In this section, we aim to mitigate the adverse effect of the cache-load imbalance bottleneck by adding an additional layer of cache in the shared-cache setting. The main idea is that in addition to the cache-enabled helper nodes, each user is also equipped with its own storage capacity. This will enable the overpopulated cell (i.e., the coverage area of a helper node) to have higher collective storage capacity compared to the less populated cells, and eventually alleviate the detrimental performance deterioration due to randomness.

4.3.1 Network Setting

We consider a shared-cache network setting which consists of a base station (BS) with a content library of N unit-sized files $\mathcal{F} = [F^1, F^2, \ldots, F^N]$, Λ cache-enabled helper nodes $\mathcal{H} = [\Lambda]$, and K cache-enabled users $\mathcal{U} = [K]$. Each helper node $\lambda \in \mathcal{H}$ is equipped with a normalized storage capacity of $\gamma \triangleq \frac{M_h}{N} \in \left[\frac{1}{\Lambda}, \frac{2}{\Lambda}, \ldots, 1\right]$ (i.e., can store the content equal to the size of M_h files) and each user $k \in \mathcal{U}$ is equipped with a normalized storage capacity of $\gamma_u \triangleq \frac{M_u}{N} \in \left[\frac{1}{\Lambda}, \frac{2}{\Lambda}, \ldots, 1\right]$ (i.e., can store the content equal to the size of M_u files). 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 in addition to its own cache, each user within the coverage area of any helper node $\lambda \in \mathcal{H}$ can download the content of helper node's cache at zero cost. In this setting, the storage regime of $\gamma_u + \gamma \geq 1$ becomes trivial, therefore, in this work, we are only interested in the storage regime such that $\gamma_u + \gamma < 1$. Figure 4.5 depicts an instance of our two-layered shared-cache setting.

The communication process consists of three phases; the *content placement phase*, the *user-to-cell association phase*[3], and the *content delivery phase*.

[3]In this setting, we have two types of caches (cache at users and cache at helper nodes),

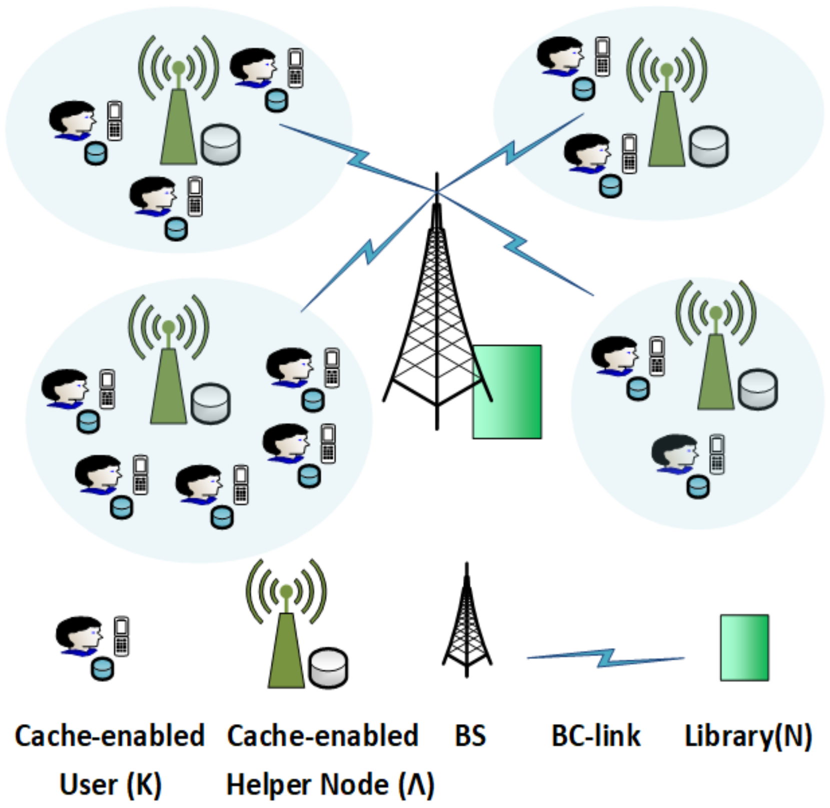


Figure 4.5: An instance of a two-layered shared-cache network.

The first phase involves the placement of library-content in each of the user's and helper node's cache, and we assume that this phase is oblivious to the outcome of the next two phases. The second phase is the association phase where each user appears within the coverage area of any particular helper node $\lambda \in \mathcal{H}$ (i.e., cell) from which it can download content at zero cost. We assume that this phase is also oblivious to the other two phases. The final phase involves the process of BS delivering the content to K users, where users simultaneously requesting one of the files from the library. This phase is aware of the outcome of the first two phases.

Content Placement We consider a library partition parameter $0 \leq \xi \leq 1$, which divides each file $F^i \in \mathcal{F}$ into two parts F^i_1 and F^i_2, such that $F^i = [F^i_1, F^i_2]$, where $|F^i_1| = \xi$ unit-sized, and $|F^i_2| = (1-\xi)$ unit-sized. We adopt the uncoded content placement scheme of [1] (see Section 1.2.1), where we use the first part of the library $\mathcal{F}_1 = [F^1_1, F^2_1, \ldots, F^N_1]$ for the content placement in the cells' caches and we use the second part of the library $\mathcal{F}_2 = [F^1_2, F^2_2, \ldots, F^N_2]$ for the content placement in the users' caches.

therefore, for better readability as well as to avoid confusion, we use the term user-to-cell association instead of the term user-to-cache association that we used in all previous section.

User-to-Cell Association We assume a uniformly random user-to-cell association process, where each user can appear in any particular cell with equal probability. Then, for any given instance, we observe a *cell population vector* $\mathbf{V} = [v_1, v_2, \ldots, v_\Lambda]$, where v_λ denotes the number of users that are within the coverage area of cell $\lambda \in \mathcal{H}$. In addition, we denote the profile vector $\mathbf{L} = [l_1, l_2, \ldots, l_\Lambda] = sort(\mathbf{V})$ as the sorted version of $\mathbf{V}$, where $sort(\mathbf{V})$ denotes the sorting of vector $\mathbf{V}$ in descending order.

Content Delivery The delivery phase begins when BS receives the request for a single file F^{d_k} that is indexed by $d_k \in [N]$ from each user $k \in \mathcal{U}$. We denote $\mathbf{d} = [d_1, d_2, \ldots, d_K]$ as the *demand vector* of K users. We assume that each user requests a different file, which is a common assumption in coded caching works [1, 31] as it leads to the worst-case delivery time. Once the BS is aware of users' demand vector $\mathbf{d}$, it commences delivery over an error-free broadcast link of bounded capacity of one unit-sized file per time slot.

4.3.2 Problem Formulation

The stochastic nature of the user-to-cell association leads to randomness in cell population vector $\mathbf{V}$. Thus, our measure of interest is the average delay given by

$$\overline{T}(\gamma, \gamma_u) = \min_{\xi} E_{\mathbf{V}}[T(\mathbf{V}, \xi)] = \min_{\xi} \sum_{\mathbf{V}} P(\mathbf{V}) T(\mathbf{V}, \xi), \tag{4.27}$$

where $T(\mathbf{V}, \xi)$ is the worst-case time needed to complete the delivery of any demand vector $\mathbf{d}$ corresponding to a specific cell population vector $\mathbf{V}$, and $P(\mathbf{V})$ is the probability of observing the cell population vector $\mathbf{V}$. We consider that the BS delivers the content in two phases. In the first phase, the BS adopts the delivery scheme proposed in [31] (see Section 1.3), where each user $k \in \mathcal{U}$ retrieves the first part of its request $F_1^{d_k}$ using the content received from the BS and the content stored in the cell's cache which it is associated with. We know from (1.4) that for any $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$, the BS needs to transmit the data equivalent to a total of

$$T_h(\mathbf{L}, \xi) = \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \tag{4.28}$$

partitioned files, where $t = \frac{\Lambda\gamma}{\xi}$, and the size of each partitioned file is $|F_1^i| = \xi$. Then, in the second phase, the BS adopts the delivery scheme proposed in [1] (see Section 1.2.1), where each user $k \in \mathcal{U}$ retrieves the second part of its request $F_2^{d_k}$ using the content received from the BS and the content stored in its own cache. This phase of delivery is independent of the user-to-cell association. We know from (1.2) that in this phase, the BS has to transmit the data equivalent to a total of

$$T_u(K, \xi) = \frac{K(1 - \frac{\gamma_u}{1-\xi})}{1 + \frac{K\gamma_u}{1-\xi}} \tag{4.29}$$

partitioned files, where the size of each partitioned file is $|F_2^i| = (1-\xi)$. Consequently, for a fixed partition parameter ξ, the worst-case delivery time for any $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$ is given as

$$T(\mathbf{L},\xi) = \xi T_h(\mathbf{L},\xi) + (1-\xi)T_u(K,\xi), \tag{4.30}$$

and the average delay takes the form of

$$\begin{aligned}\overline{T}(\gamma,\gamma_u,\xi) &= \xi E_{\mathbf{L}}[T_h(\mathbf{L},\xi)] + (1-\xi)E_{\mathbf{L}}[T_u(K,\xi)] \\ &= \xi \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + (1-\xi)\frac{K(1-\frac{\gamma_u}{1-\xi})}{1+\frac{K\gamma_u}{1-\xi}},\end{aligned} \tag{4.31}$$

where $\mathcal{L}$ is the set of all possible profile vectors $\mathbf{L}$ and

$$P(\mathbf{L}) \triangleq \sum_{\mathbf{V}:sort(\mathbf{V})=\mathbf{L}} P(\mathbf{V}), \tag{4.32}$$

is simply the cumulative probability over all $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$. Then, our metric of interest takes the form of

$$\overline{T}(\gamma,\gamma_u) = \min_{\xi} \overline{T}(\gamma,\gamma_u,\xi). \tag{4.33}$$

In this work, we focus on the asymptotic analysis of the performance of the proposed two-layered shared-cache setting. In the next section, we provide the scaling laws of the performance as well as the order-optimal partition parameter ξ which leads to the minimum average delay in a simple and insightful form.

4.3.3 Main Results: Two-Layered Shared-Cache Networks

In this section we present our main results on the performance of our proposed two-layered shared-cache setting. Our first result provides the scaling law of the average delivery time $\overline{T}(\gamma,\gamma_u,\xi)$ for any given partition parameter ξ, in the limit of large Λ.

Theorem 4.6. *In a Λ-cell, K-users shared-cache setting with each user equipped with a normalized storage capacity γ_u and each helper node equipped with normalized storage capacity of γ, the average delay for any given partition parameter ξ and a random user-to-cell association scales as*

$$\overline{T}(\gamma,\gamma_u,\xi) = \Theta\left(\frac{\xi^2(c\gamma_u+\gamma)}{\gamma\gamma_u} + \frac{\xi(-c\gamma\gamma_u+\gamma_u\gamma-2\gamma)+\gamma-\gamma\gamma_u}{\gamma\gamma_u}\right). \tag{4.34}$$

where

$$c = \begin{cases} \frac{\log\Lambda}{\log\left(\frac{\Lambda}{K}\log\Lambda\right)} & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o(\Lambda\log\Lambda)\right] \\ \frac{K}{\Lambda} & \text{if } K = \Omega(\Lambda\log\Lambda). \end{cases} \tag{4.35}$$

Proof. The proof can be found in Appendix B.4. □

Now, we proceed with the following lemma that provides the order-optimal $\hat{\xi}$ which leads to minimum average delay $\overline{T}(\gamma, \gamma_u, \xi)$.

Lemma 4.1. *The order-optimal partition parameter $\hat{\xi}$ that minimizes the average delay $\overline{T}(\gamma, \gamma_u, \xi)$ is given as*

$$\hat{\xi} = \begin{cases} \gamma & \text{if } \gamma_u \geq \frac{2-2\gamma}{1+c} \\ \frac{c\gamma\gamma_u - \gamma_u\gamma + 2\gamma}{2(c\gamma_u+\gamma)} & \text{otherwise,} \end{cases} \tag{4.36}$$

where

$$c = \begin{cases} \frac{\log\Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)} & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda\log\Lambda\right)\right] \\ \frac{K}{\Lambda} & \text{if } K = \Omega\left(\Lambda\log\Lambda\right). \end{cases} \tag{4.37}$$

Proof. The proof can be found in Appendix B.5. □

With Lemma 4.1 at hand, we are ready to to present our final result which is the scaling law of our performance metric.

Theorem 4.7. *In a Λ-cell, K-users shared-cache setting with each user equipped with a normalized storage capacity γ_u and each helper node equipped with normalized storage capacity of γ, the minimum average delay corresponding to the order-optimal partition parameter $\hat{\xi}$ and a random user-to-cell association scales as*

$$\overline{T}(\gamma, \gamma_u) = \begin{cases} \Theta\left((1-\gamma)\frac{1-\gamma_u-\gamma}{\gamma_u}\right) & \text{if } \gamma_u \geq \frac{2-2\gamma}{1+c} \\ \Theta\left(\frac{c(1-\gamma-\gamma_u)}{c\gamma_u+\gamma} - \frac{\gamma\gamma_u(c-1)^2}{4c\gamma_u+4\gamma}\right) & \text{otherwise} \end{cases} \tag{4.38}$$

where

$$c = \begin{cases} \frac{\log\Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)} & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda\log\Lambda\right)\right] \\ \frac{K}{\Lambda} & \text{if } K = \Omega\left(\Lambda\log\Lambda\right). \end{cases} \tag{4.39}$$

Proof. The proof can be found in Appendix B.6. □

Remark 4.2. *For the case of $\gamma_u = 0$, our proposed two-layered shared-cache setting is exactly equal to the original stochastic shared-cache setting studied in Section 3.3, and the scaling of average delay from Theorem 4.7 is exactly equal to the one presented in Theorem 3.3.*

Theorem 4.7 provides different memory regimes, and it hints out that adding an additional layer of cache in the shared-cache setting can be effective in removing the cache-load imbalance bottleneck. In order to better understand the impact of the additional layer, we proceed to present the following lemma which characterizes the performance of our proposed shared-cache setting for the well-known deterministic uniform user-to-cache association.

Lemma 4.2. *In a Λ-cell, K-users shared-cache setting with each user equipped with a normalized storage capacity of γ_u and each helper node equipped with normalized storage capacity of γ, the best-case delivery time for a uniform user-to-cell association evaluated at corresponding order-optimal partition parameter $\hat{\xi}$ scales as*

$$T_{min}(\gamma,\gamma_u) = \begin{cases} \Theta\left((1-\gamma)\frac{1-\gamma_u-\gamma}{\gamma_u}\right) & \text{if } \gamma_u \geq \frac{2-2\gamma}{1+c} \\ \Theta\left(\frac{c(1-\gamma-\gamma_u)}{c\gamma_u+\gamma} - \frac{\gamma\gamma_u(c-1)^2}{4c\gamma_u+4\gamma}\right) & \text{otherwise,} \end{cases} \tag{4.40}$$

where $c = \frac{K}{\Lambda}$.

Proof. The proof can be found in Appendix B.7. □

In identifying the exact scaling laws of the problem, the Theorem 4.7 nicely captures the following points.

- It identifies the system parameter regime $K = \Omega(\Lambda \log \Lambda)$ which is consistent with the observation of Theorem 3.3, where there is no performance deterioration due to randomness as the multiplicative gap between the average delivery time and the best-case delivery time is equal to one i.e., $\frac{\overline{T}(\gamma,\gamma_u)}{T_{min}(\gamma,\gamma_u)} = 1$.
- For the system parameter regimes where the performance deterioration due to the cache-load imbalance is unbounded, it identifies a memory regime of $\gamma_u \geq \frac{2-2\gamma}{1+\frac{\log\Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)}}$ for the additional cache layer that can completely nullify the impact of the cache-load imbalance such that multiplicative gap between average delivery time and best-case delivery time is equal to one.

4.3.4 Numerical Validation

We know from (4.31) that it is computationally expensive to numerically evaluate the exact average delay even for small system parameters. Therefore, we proceed to numerically evaluate the effectiveness of our proposed approach in mitigating the effect of the cache-load imbalance bottleneck by using the *sampling-based numerical* (SBN) approximation method, where we generate a sufficiently large set $\mathcal{L}_1$ of randomly generated profile vectors $\mathbf{L}$, and approximate $\overline{T}(\gamma,\gamma_u,\xi)$ for a fixed ξ as

$$\overline{T}(\gamma,\gamma_u,\xi) \approx \frac{1}{|\mathcal{L}_1|}\sum_{\mathbf{L}\in\mathcal{L}_1} T(\mathbf{L},\xi), \tag{4.41}$$

where we recall that $T(\mathbf{L},\xi)$ is defined in (4.30). Then, we numerically find the approximate optimal partition parameter $\hat{\xi}$ that minimizes the $\overline{T}(\gamma,\gamma_u,\xi)$ in (4.41) and approximate $\overline{T}(\gamma,\gamma_u)$ as

$$\overline{T}(\gamma,\gamma_u) \approx \frac{1}{|\mathcal{L}_1|}\sum_{\mathbf{L}\in\mathcal{L}_1} T(\mathbf{L},\hat{\xi}) \tag{4.42}$$

Following the same approach, for the uniform user-to-cell association, we numerically find the approximate optimal partition parameter $\hat{\xi}$ that minimizes the $T(\mathbf{L}_{uni}, \hat{\xi})$ in (4.30), where $\mathbf{L}_{uni}$ is a uniform profile vector, and approximate $T_{min}(\gamma, \gamma_u)$ as.

$$T_{min}(\gamma, \gamma_u) \approx T(\mathbf{L}_{uni}, \hat{\xi}) \tag{4.43}$$

The corresponding approximate performance gap is then evaluated by dividing $\overline{T}(\gamma, \gamma_u)$ by $T_{min}(\gamma, \gamma_u)$. For $|\mathcal{L}_1| = 10000$, and $K = \Lambda = 500$, Figure 4.6 compares the SBN approximation of multiplicative performance gap $G(\gamma, \gamma_u) = \frac{\overline{T}(\gamma,\gamma_u)}{T_{min}(\gamma,\gamma_u)}$ between the average delivery time $\overline{T}(\gamma, \gamma_u)$ for a random user-to-cell association and the best-case delivery time $T_{min}(\gamma, \gamma_u)$ for a uniform user-to-cell association. This figure highlights the significance of our proposed two-layered shared-cache setting in mitigating the impact of the cache-load imbalance. We see that even when γ_u is very small, the performance gap $G(\gamma, \gamma_u)$ is significantly less than the case of $\gamma_u = 0$ i.e., the original stochastic shared-cache setting of Section 3.3.

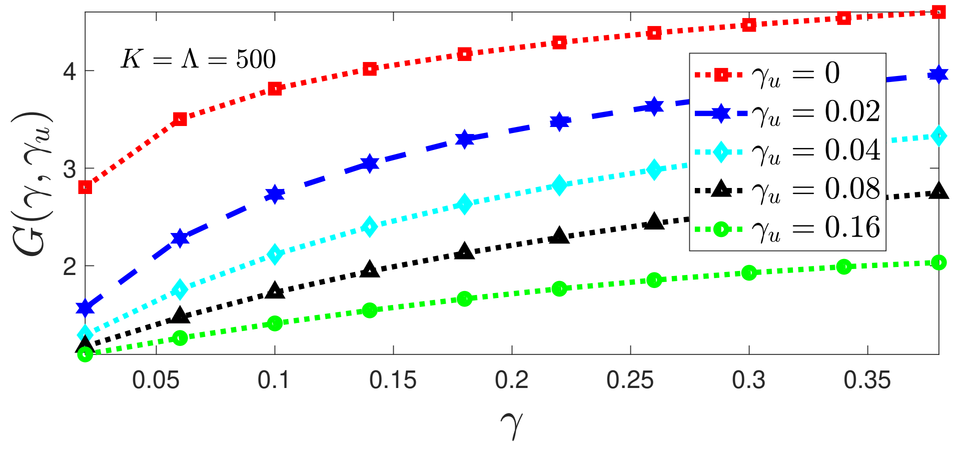


Figure 4.6: Multiplicative gap $G(\gamma, \gamma_u)$ between $\overline{T}(\gamma, \gamma_u)$ from (4.42) and $T_{min}(\gamma, \gamma_u)$ from (4.43).

To summarize, in this section, we explored the effectiveness of the two-layered coded caching scheme in resolving the cache-load imbalance bottleneck of coded caching in a stochastic shared-cache network. Our analysis revealed that empowering users with an additional layer –even a modest amount– of caching in stochastic shared-cache networks can be a viable solution to resolve the cache-load imbalance bottleneck.

4.4 Summary of Chapter

In this chapter, we studied three different techniques to resolve the cache-load imbalance bottleneck of coded caching in a stochastic shared-cache network. In the first technique, we used two load-balancing approaches to alleviate the effect of randomness. We showed that in scenarios where we are given a choice to associate a user to the least loaded cache from a randomly chosen group of h helper nodes (neighboring in case of proximity-bounded), the performance deterioration due to the random user-to-cache association can be significantly reducing. In the second technique, we 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. We also showed that 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. Finally, in the third technique, we proposed a two-layered coded caching scheme to resolve the cache-load imbalance bottleneck. We identified the exact scaling laws of the average delivery time of our proposed two-layered coded caching scheme and showed that this scheme significantly mitigates, and in certain memory regimes completely nullify the detrimental performance deterioration due to randomness. In a nutshell, empowering users with an additional layer of caching can be a viable solution to resolve the cache-load imbalance bottleneck.

Chapter 5

Subpacketization-Constrained Coded Caching Networks with Heterogeneous User Activity

In this chapter, we analyze the subpacketization-constrained coded caching networks when users have different activity levels. Let us quickly recall from our discussion in Section 1.3.2 about the serious Achilles' heel of coded caching. In order to benefit from the exceptional gain (i.e., $K\gamma + 1$) of coded caching, it requires that each user be allocated their own specifically-designed cache state (cache content), which — without delving into the esoteric details of coded caching — effectively requires the partitioning of each library-file into $\binom{K}{K\gamma}$ subpackets. This number scales exponentially in K, and thus requires files to be of truly astronomical sizes. Thus given any reasonable constraint on the file sizes, the number of cache states is effectively forced to be reduced, and the coding gains are indeed diminished to gains that are considerably less than $K\gamma{+}1$. What this file-size constraint (also known as the subpacketization bottleneck) effectively forces is the reduction of the number of cache states[1] to some $\Lambda \ll K$, which — under the basic principles of the clique-based cache-placement in [1] (see Section 1.2.1) — allows for a smaller subpacketization level $\binom{\Lambda}{\Lambda\gamma} \ll \binom{K}{K\gamma}$ at the expense of a much-reduced coding gain $\Lambda\gamma + 1 \ll K\gamma + 1$ and a much larger delay $\frac{K(1-\gamma)}{1+\Lambda\gamma}$ which is now unbounded.

The Connection Between Coded Caching, Complementary Cache States, User Activity Levels and User Activity Correlations

The performance of coded caching in the presence of an inevitably reduced number of cache states, has been explored in various works that include the work in [2] which introduced a new scheme for this setting, and the work in [31] which established the fundamental limits of the state-limited coded

[1]This simply means that even though there are K different users, each with their own physical cache, in essence, there can only exist Λ distinct caches, that must be shared among the users. This effectively means that groups of users are forced to have identical, rather than complementary, cache contents.

caching setting, by deriving the exact optimal worst-case delivery time as a function of the user-to-cache state association profile that represents the number of users served by each cache.

As we witness in the above works, in order to maintain the ability to jointly exploit multicasting opportunities, users must be associated to complementary cache states that are carefully designed and which cannot be identical. The above findings reveal that a basic problem with the state-limited scenario (where $\Lambda \ll K$) in coded caching is simply the fact that if two or more users are forced to share the same cache state (i.e., the same content in their caches), then these users generally do not have the ability to jointly receive a multicasting message that can be useful to all. Such state-limited scenario results in the aforementioned large deterioration in performance, irrespective of the user-to-cache association policy. What we additionally learn from our study in Chapter 3 is that if the users are assigned states at random, then this randomness imposes an additional *unbounded* performance deterioration that is a result of 'unfortunate' associations where too many users share the same cache state. That is why the task of user-to-cache state association is important.

At the same time though, coded caching experiences a certain synchronization aspect, which is a direct outcome of the fact that users are expected to be partially asynchronous in their timing of requesting files. Hence, the notion of time is of essence. This asynchronicity has a negative aspect, but also a positive one; both of which we explore in this chapter. On the one hand, having only a fraction of the users appear simultaneously, implies a smaller number of users that can simultaneously participate in coded caching and thus implies potentially fewer multicasting opportunities and thus a smaller coding gain. On the other hand, such asynchronicity implies less instantaneous interference. This is where user activity levels come into the picture, and this is where user activity correlations can be exploited. In essence — as it will become clearer later on — any users that are correlated in terms of their activity in time, should be associated to different cache states, as this is essential in using caches for handling their mutual interference. On the other hand, knowing that some users rarely request data at the same time, allows us to give them the same cache state resource. In essence, users that overlap more, interfere more, and thus have higher priority to secure complementary cache states. This optimization effort is particularly important because, as we recall, these resources are indeed scarce. By exploring user activities and learning from their history, we are able to predict interference patterns, and then we are able to assign cache states accordingly.

In this chapter, we study the K-user cache-aided broadcast channel with a limited number of cache states and explore the effect of user-to-cache state association strategies in the presence of arbitrary user activity levels. We first present a statistical analysis of the average worst-case delay performance of such subpacketization-constrained (state-constrained) coded caching networks, and provide computationally efficient performance bounds as well as scaling laws for any arbitrary probability distribution of the user activity levels. Next, we follow a data-driven approach that exploits the prior history on user activity levels and correlations, in order to predict interference patterns, and thus better design the caching algorithm. This is, to the best of our understanding, the first work that seeks to exploit user activity levels and correlations, in order to map future interference and provide optimized caching algorithms that better handle this interference.

5.1 Network Setting

We consider a cache-aided wireless network, which consists of a base station and K cache-enabled receiving users. The base station (BS) has access to a library of N equisized files $\mathcal{F} = [F_1, F_2, \dots, F_N]$ and delivers content via a broadcast link to K receiving users. Each user $k \in [1, 2, \dots, K]$ is equipped with a cache of normalized storage capacity of $\gamma \triangleq \frac{M}{N} \in [0, 1]$, and requests a file from content library with probability p_k. We use $\mathbf{p} = [p_1, p_2, \dots, p_K]$ to denote the users *activity level vector*. At any instance, if a user k is requesting a file, then we say that the user $k \in [K]$ is an *active user*. Naturally $K_{\mathbf{p}} = \sum_{k=1}^{K} p_k$ is the expected number of active users. Figure 5.1 depicts an instance of our cache-aided wireless network.

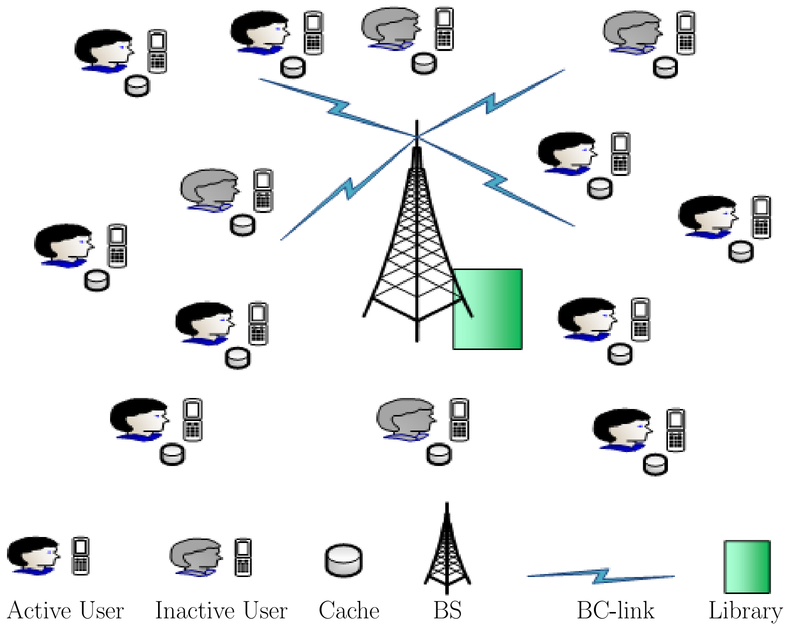


Figure 5.1: An instance of a cache-aided wireless network.

The communication process consists of two phases; the *placement phase* and the *delivery phase*. During the placement phase, each user's cache is filled with the content from the library, and this phase is oblivious to the upcoming number of users in the delivery phase, as well as is oblivious to the upcoming file demands. The delivery phase begins with *the active users* simultaneously requesting one file each, and continues with the BS delivering this content to the users. This phase is naturally aware of the demands of the active users, as well as is aware of the content cached at each user.

Placement phase: We consider the subpacketization-constrained uncoded cache placement scheme based on [1] (refer to 1.2.1). Let P_{max} denote the maximum allowable subpacketization of a file, which defines the maximum number of cache states as follows

$$\Lambda = \arg\max_{k\leq K} \left\{ \binom{k}{k\gamma} \leq P_{max} \right\}.$$

Each file $F_i \in \mathcal{F}$ is partitioned into $\binom{\Lambda}{t}$ distinct equisized subpackets, where $t \triangleq \Lambda\gamma$ for some $t \in [1, \ldots, \Lambda]$. Then we index each subpacket of a file by a distinct subset $\tau \subseteq [1, \ldots, \Lambda]$ of size t. The set of indexed subpackets corresponding to file $F_i \in \mathcal{F}$ is given by $\{F_{i,\tau} : \tau \subseteq [1, \ldots, \Lambda], |\tau| = t\}$. The content corresponding to each cache state $\lambda \in [1, \ldots, \Lambda]$ is then given by

$$C_\lambda = \{F_{i,\tau} : i \in [1, \ldots, N], \lambda \in \tau, \tau \subseteq [1, \ldots, \Lambda], |\tau| = t\},$$

where each cache state consists of $|C_\lambda| = N\binom{\Lambda-1}{t-1}$ subpackets, which abides by the cache-size constraint since $N\frac{\binom{\Lambda-1}{t-1}}{\binom{\Lambda}{t}} = M$.

During the placement phase, each user's cache is filled with the content of one of the cache states $\lambda \in [1, \ldots, \Lambda]$. The employed user-to-cache state association is defined by a matrix $\mathbf{G} = [0, 1]^{\Lambda \times K}$, of which the (λ, k) element $g_{\lambda,k}$ takes the value 1 if user k is storing the content of cache state $\lambda \in [\Lambda]$, else $g_{\lambda,k} = 0$. We denote by $\mathbf{G}_\lambda$ the set of users caching the content of cache state $\lambda \in [1, \ldots, \Lambda]$.

Delivery phase: The delivery phase commences with each active user requesting a single file from the content library. In line with the common assumptions in coded caching [1,21,31,57], we assume that requests are generated simultaneously by active users, and that each active user requests a different file. During this phase, the BS is aware of the user-to-cache state association matrix $\mathbf{G}$. Once the BS receives the users' requests, it commences delivery of the coded subpackets over a unit-capacity[2] error-free broadcast link. Here, together with the aforementioned optimal placement, we also consider the optimal[3] multi-round delivery scheme of [31] (refer to Section 1.3). At any instance of the problem, the *cache load vector* is denoted by $\mathbf{V} = [v_1, \ldots, v_\Lambda]$, where v_λ represents the number of active users that are associated with cache state $\lambda \in [\Lambda]$. Additionally, we use $\mathbf{L} = [l_1, \ldots, l_\Lambda] = sort(\mathbf{V})$ to be the *profile vector*, which is the sorted (in descending order) version of the cache load vector $\mathbf{V}$.

[2]Here the capacity is measured in units of file.

[3]Optimality here refers to the performance of the scheme over the traditional (deterministic) coded caching problem with *constant* user activity.

5.2 Metrics of Interest

To capture the randomness in user activity, we consider — for any given user-to-cache state association matrix $\mathbf{G}$ — the averaging metric

$$\overline{T}(\mathbf{G}) \triangleq E_{\mathbf{V}}[T(\mathbf{V})] = \sum_{\mathbf{V}} P(\mathbf{V})T(\mathbf{V}), \tag{5.1}$$

where $P(\mathbf{V})$ is the probability of $\mathbf{V}$, and where $T(\mathbf{V})$ is the worst-case delivery time[4] needed to complete the delivery of requested files given a certain cache load vector $\mathbf{V}$ associated to matrix $\mathbf{G}$. For any cache load vector $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$, the information-theoretically optimal delivery time — achieved with the multi-round delivery scheme [31] (refer to Section 1.3) — takes the form

$$T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}. \tag{5.2}$$

Thus, the average delay takes the form

$$\overline{T}(\mathbf{G}) = \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L})T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t}\sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} = \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda] \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{5.3}$$

where $\mathcal{L}$ describes the set of all possible profile vectors $\mathbf{L}$, where $P(\mathbf{L})$ is the probability of a profile vector $\mathbf{L}$ *given the user-to-cache state association* $\mathbf{G}$, and where $E[l_\lambda]$ is the expected number of active users in the λ-th most loaded cache, again given $\mathbf{G}$.

Our interest is in finding the optimal user-to-cache association that minimizes the average delay. This corresponds to the following optimization problem.

Problem 5.1.

$$\min_{\mathbf{G}} \quad \overline{T}(\mathbf{G}) \tag{5.4}$$

subject to

$$\sum_{i=1}^{\Lambda} g_{i,k} = 1 \;\; \forall k \in [K]. \tag{5.5}$$

5.3 Main Results: Statistical Approach

In this section, we present our main results on the performance of a coded caching network of K cache-aided users and Λ cache states, where the user

[4]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 BS to the user, had there been no caching and no interference.

activity levels follow an arbitrary probability distribution $\mathbf{p}$ and where the association between users and cache states is subject to an arbitrary association strategy $\mathbf{G}$.

We can see from (5.3) that for any given user-to-cache state association $\mathbf{G}$ and user activity statistics $\mathbf{p}$, the exact evaluation of (5.3) is computationally expensive especially for large system parameters, as the creation of $\mathcal{L}$ is an integer partition problem, and the cardinality of $\mathcal{L}$ is known to be growing exponentially with system parameters K and Λ [61]. Motivated by this complexity, we here proceed to provide computationally efficient bounds on the performance. After doing so, we resort to asymptotic analysis of the impact of $\mathbf{G}$ and $\mathbf{p}$ on the performance, and provide an insightful characterization of the scaling laws of this performance. Finally, based on the insights from these scaling laws, we propose a heuristic user-to-cache state association algorithm that aims to minimize the worst-case delivery time.

5.3.1 Performance Analysis with Arbitrary Activity Levels

In this subsection, we present the statistical analysis of our problem for the general setting of an arbitrary user-to-cache state association strategy $\mathbf{G}$ and an arbitrary activity level vector $\mathbf{p}$. Crucial to our analysis for this setting will be the mean $\mu_\lambda = \sum_{k\in G_\lambda} p_k$ and the variance $\sigma_\lambda^2 = \sum_{k\in G_\lambda} p_k(1-p_k)$ of the number of active users that are caching the content of cache state $\lambda \in [\Lambda]$. Now we proceed to present our first result which is the characterization of faster-to-compute analytical bounds on the performance.

Theorem 5.1. *In a state-constrained coded caching network of Λ cache states, K cache-aided users with normalized cache capacity γ and activity level vector $\mathbf{p}$, the average delay $\overline{T}(\mathbf{G})$ for a given user-to-cache state association strategy $\mathbf{G}$ is bounded as follows*

$$\overline{T}(\mathbf{G}) \leq \frac{\Lambda - t}{1+t}\left(A - \sum_{x=0}^{A-1} \max\left(0, 1-\Lambda + \sum_{\lambda=1}^{\Lambda} F_1(\lambda, x)\right)\right) \tag{5.6}$$

$$\overline{T}(\mathbf{G}) \geq \frac{\Lambda - t}{1+t}\frac{t}{\Lambda-1}\left(A - \sum_{x=0}^{A-1} \frac{\sum_{\lambda=1}^{\Lambda} F_2(\lambda, x)}{\Lambda}\right) + \frac{\Lambda - t}{1+t}\frac{K_{\mathbf{p}}}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}, \tag{5.7}$$

where $t = \Lambda\gamma$, $A = \max\left(\{|\mathbf{G}_\lambda|\}_{\lambda=1}^{\Lambda}\right)$, where $\mathbf{G}_\lambda$ is the set of users caching the content of cache state λ,

$$F_1(\lambda, x) = \begin{cases} 0 & \text{if } 0 \leq x \leq \mu_\lambda - 1 \\ F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{if } \mu_\lambda \leq x \leq |\mathbf{G}_\lambda| \\ 1 & \text{if } x > |\mathbf{G}_\lambda|, \end{cases} \tag{5.8}$$

$$F_2(\lambda, x) = \begin{cases} F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{if } 0 \leq x \leq \mu_\lambda - 1 \\ 1 & \text{if } x > \mu_\lambda - 1, \end{cases} \tag{5.9}$$

where $F_{bin}(n, q, x) = \sum_{i=0}^{x} \binom{n}{i} q^i (1-q)^{n-i}$ *and where* $\mu_\lambda = \sum_{k \in \mathbf{G}_\lambda} p_k$.

Proof. The proof is deferred to Appendix C.1. $\square$

Remark 5.1. *The bounds in Theorem 5.1 can be computed in a computationally-efficient manner, as for each* $\lambda \in [\Lambda]$, *their evaluation only requires to compute the binomial cumulative distribution function* $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right)$ *for all* $x \in [0, 1, 2, \cdots, |\mathbf{G}_\lambda|]$ *of a random variable with* $|\mathbf{G}_\lambda|$ *independent trials and* $\frac{\mu_\lambda}{|\mathbf{G}_\lambda|}$ *success probability*[5].

Next, we proceed to our next result, which provides the asymptotic analysis of the average delay $\overline{T}(\mathbf{G})$, in the limit of large Λ and K. Let us quickly recall that $\mu_\lambda = \sum_{k \in G_\lambda} p_k$ and $\sigma_\lambda^2 = \sum_{k \in G_\lambda} p_k(1-p_k)$ are respectively the mean and variance of the number of active users that are associated with cache state λ.

Theorem 5.2. *In a state-constrained coded caching network of* Λ *cache states,* K *cache-aided users with normalized cache capacity* γ *and activity level vector* $\mathbf{p}$, *the average delay* $\overline{T}(\mathbf{G})$ *for a given association strategy* $\mathbf{G}$ *scales as*

$$\overline{T}(\mathbf{G}) = O\left(\left(\frac{K_\mathbf{p}}{\Lambda} + \sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i - \mu)^2)}\right)\frac{\Lambda - t}{1+t}\right), \tag{5.10}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{K_\mathbf{p}}{\Lambda}\frac{\Lambda - t}{1+t}\right) \tag{5.11}$$

where $\mu = \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda = \frac{K_\mathbf{p}}{\Lambda}$.

Proof. This proof is deferred to Appendix C.2. $\square$

Furthermore we have the following.

Corollary 5.1. *Any association strategy* $\mathbf{G}$ *that satisfies* $\sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i - \mu)^2)} = O\left(\frac{K_\mathbf{p}}{\Lambda}\right)$ *is order-optimal.*

[5]In theory, $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right)$ needs to be calculated for all values of $x \in [0, |\mathbf{G}_\lambda|]$. However, it is known that there exists a $\tilde{x} \in [0, |\mathbf{G}_\lambda|]$, where $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, \tilde{x}\right) \approx 1$. By De Moivre-Laplace Theorem, it is known that binomial distribution can be approximated by the normal distribution in the limit of large $|\mathbf{G}_\lambda|$, and the well-known 68–95–99.7 rule states that $\tilde{x} << |\mathbf{G}_\lambda|$. Since $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) \approx 1$ for any $x \geq \tilde{x}$, both (5.8) and (5.9), and consequently (5.6) and (5.7) can be quickly evaluated with high accuracy.

Proof. Since the lower bound in (5.11) is independent of the association strategy $\mathbf{G}$, this implies that the optimal average delay $\overline{T}^*$ (corresponding to an optimal association $\hat{\mathbf{G}}$) is lower bounded by

$$\overline{T}^* = \Omega\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right). \tag{5.12}$$

Therefore any association $\mathbf{G}$ for which the gap factor $\sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2+(\mu_i-\mu)^2)}$ scales as $O\left(\frac{K_{\mathbf{p}}}{\Lambda}\right)$ would be order-optimal as the scaling order of (5.10) yields $O\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right)$, thus, giving the exact scaling law of $\overline{T}(\mathbf{G}) = \Theta\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right)$. □

Following the insights from Corollary 5.1, we now propose an algorithm that solves Problem 5.1.

Algorithm 5.1

The algorithm[6] aims to heuristically minimize $\sum_{i=1}^{\Lambda}(\sigma_i^2+(\mu_i-\mu)^2)$, and it works in K iterations, where for each iteration the algorithm finds a user and cache state pair $(\hat{k},\hat{\lambda})$ in accordance to step 02 of this algorithm. Consequently user $\hat{k}$ is assigned cache state $\hat{\lambda}$.

```
Input: $\mathbf{p}$, $K$, and $\Lambda$
Output: $\mathcal{G}$
Initialization: $\mathcal{G} \leftarrow \emptyset$; $\mathcal{K} \leftarrow [K]$; $f(\mathcal{G}) \leftarrow \sum_{i=1}^{\Lambda}(\sigma_i^2+(\mu_i-\mu)^2)$
Step 01: for $j$ from 1 to $K$ do
Step 02:    $[\hat{\lambda},\hat{k}] \leftarrow \underset{\lambda\in[\Lambda],k\in\mathcal{K}}{\operatorname{argmin}} f(\mathcal{G}\cup(\lambda,k))$
Step 03:    $\mathcal{G} \leftarrow \mathcal{G}\cup(\hat{\lambda},\hat{k})$
Step 04:    $\mathcal{K} \leftarrow \mathcal{K}\backslash\hat{k}$
Step 05: end for
```

Algorithm 5.1: Heuristic Algorithm for Problem 5.1

In Section 5.5, we will verify that the bounds presented in Theorem 5.1 are valid for any user-to-cache state association strategy. We will also show that Algorithm 5.1 provides an efficient user-to-cache state association that yields a performance very close to the performance of optimal user-to-cache state association.

5.3.2 Performance Analysis with Uniform Activity Level

In this subsection, we analyze a special setting where users have a uniform activity level p, corresponding to the equiprobable case of $p_1 = p_2 \cdots = p_K = p$. In the presence of uniform activity level probabilities p, the best user-to-cache

[6]$\mathcal{G}$ denotes the set form of the user-to-cache state association matrix $\mathbf{G}$, where $(\lambda,k)\in\mathcal{G}$ if $g_{\lambda,k}=1$.

state association policy is the uniform one where each cache state is allocated to $I \triangleq \frac{K}{\Lambda}$ users. We assume that I is an integer.

We now proceed to provide computationally efficient analytical bounds on the average delay $\overline{T}(\mathbf{G})$ achieved by the uniform association policy, and subsequently to provide the exact scaling laws of this policy.

Theorem 5.3. *In a state-constrained coded caching network of Λ cache states, K cache-aided users with normalized cache capacity γ and activity level of p, the average delay $\overline{T}(\mathbf{G})$ corresponding to the uniform user-to-cache state association strategy $\mathbf{G}$ is bounded by*

$$\overline{T}(\mathbf{G}) \leq \frac{\Lambda - t}{1+t} E[l_1] \tag{5.13}$$

and

$$\overline{T}(\mathbf{G}) \geq \frac{\Lambda - t}{1+t} \left(\frac{E[l_1]t}{\Lambda - 1} + \frac{Kp}{\Lambda} \frac{\Lambda - t - 1}{\Lambda - 1} \right) \tag{5.14}$$

where

$$E[l_1] = I - \sum_{j=0}^{I-1} \left(\sum_{i=0}^{j} \binom{I}{i} p^i (1-p)^{I-i} \right)^{\Lambda}. \tag{5.15}$$

Proof. The proof is deferred to Appendix C.3. □

Furthermore, the following shows that the bounds remain relatively close to the exact $\overline{T}(\mathbf{G})$.

Corollary 5.2. *For any fixed $\gamma \leq 1 - \frac{1}{\Lambda}$, the multiplicative gap between the analytical upper bound (AUB) in* (5.13) *and the analytical lower bound (ALB) in* (5.14)*, is at most $\frac{\Lambda-1}{t} < 1/\gamma$. This allows us to identify the exact $\overline{T}(\mathbf{G})$ within a factor that is independent of both Λ as well as K.*

Proof. The proof follows directly from the fact that $\frac{\Lambda-t}{1+t}\frac{E[l_1]t}{\Lambda-1} \leq \overline{T}(\mathbf{G}) \leq \frac{\Lambda-t}{1+t}E[l_1]$. □

Remark 5.2. *We note that the range of $\gamma \leq 1 - \frac{1}{\Lambda}$ covers in essence the entire range of γ and most certainly covers the range of pertinent γ values.*

We now proceed to exploit the bounds in Theorem 5.3, in order to provide in a simple and insightful form, the exact scaling laws of performance. The following theorem provides the asymptotic analysis of the average delay $\overline{T}(\mathbf{G})$, in the limit of large Λ and K.

Theorem 5.4. *In a coded caching setting with Λ cache states and K cache-aided users with equal cache size γ and activity level p, the average delay $\overline{T}(\mathbf{G})$ corresponding to the uniform association strategy $\mathbf{G}$ scales as*

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+t}\right) & \text{if } Ip = \Omega(\log \Lambda) \\ \Theta\left(\frac{Kp(1-\gamma)\log\Lambda}{(1+t)Ip\log\frac{\log\Lambda}{Ip}}\right) & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log\Lambda)\right]. \end{cases} \tag{5.16}$$

Proof. The proof is deferred to Appendix C.4. □

5.4 Main Results: Data-Driven Approach

In this section, we will extend our analysis to the data-driven setting. Unlike in the previous section where we used a predetermined set of statistics **p**, we will now exploit the users' content request histories to define the user activity levels as well as correlations. To proceed with our analysis we need to define the time scales involved. In our setting, the entire time horizon is equal to the time it takes between two user-to-cache associations. This time horizon will be here subdivided into S independent time slots, where one time slot corresponds to the amount of time that elapses from the appearance of one demand vector to the next demand vector. This dynamic time refinement captures the amount of memory of the system, and will capture how far back in history we can learn from regarding user activities.

Example 5.1. *In a scenario where users are assigned cache states once a week, then the time frame is equal to one week which is equal to 10080 minutes. In this same example, if we assume that independent demand vectors appear once every* 10 *minutes, then the number of independent time slots* S *is simply* $S = \frac{10080}{10} = 1008$.

In our setting, users' requests are served simultaneously, starting at the very beginning of each time slot. Any content request received during a time slot is put on hold, to be served in the beginning of the next time slot. This justifies the use of the term *dynamic duration* of each time slot $s \in [S]$, where this duration will be equal to the time needed to transmit all files that were requested during the previous time slot from the BS to the users. We can now proceed with the details of our data-driven approach.

Let $\mathbf{D} \in [0,1]^{S\times K}$ denote the *user activity matrix*, of which the (s,k) element $d_{s,k}$ is equal to 1 if user k requests content at time slot s, else $d_{s,k} = 0$. Then, for a given user-to-cache state association $\mathbf{G}$, the cache load vector for time slot $s \in [S]$ is denoted as $\mathbf{V}_s = [v_{s,1}, \dots, v_{s,\Lambda}]$, where $v_{s,\lambda} = \sum_{k=1}^{K} g_{\lambda,k} d_{s,k}$ is the number of active users at time slot s that are storing the content of cache state $\lambda \in [\Lambda]$. The profile vector at time slot s is denoted as $\mathbf{L}_s = [l_{s,1}, \dots, l_{s,\Lambda}]$, which is the sorted version of the cache load vector $\mathbf{V}_s$ in descending order. The average delay for a given user-to-cache state association $\mathbf{G}$ and a given user activity matrix $\mathbf{D}$, is given by

$$\overline{T}(\mathbf{G}) \triangleq \frac{1}{S}\sum_{s=1}^{S}\sum_{\lambda=1}^{\Lambda-t} l_{s,\lambda}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}. \tag{5.17}$$

Unlike in the statistical approach of Section 5.3, where an enormous number of possible profile vectors rendered the exact calculation of $\overline{T}(\mathbf{G})$ computationally intractable, in this current data-driven setting, the calculation of $\overline{T}(\mathbf{G})$ is direct even for large system parameters. This will allow us to design an algorithm that will find a user-to-cache association policy that is provably order-optimal.

An additional difference of the proposed data-driven problem formulation is that now this formulation inherits a crucial property of exploiting users' activity correlation in time. As previously discussed, users with similar request patterns will be associated with different cache states as this would guarantee more multicasting opportunities during the delivery phase. On the other hand, users that rarely request files at the same time, can be allocated the same cache state without any performance deterioration.

We now proceed to find an order-optimal user-to-cache state association $\hat{\mathbf{G}}$ corresponding to Problem 5.1. At this point we note that it is computationally intractable to brute-force solve Problem 5.1 for large system parameters K, Λ and S, since there are Λ^K possible user-to-cache state associations, corresponding to an exhaustive-search computational complexity of $O\left(S\Lambda^{K+1}\right)$. Under these circumstances, the most common approach is to use computationally efficient algorithms to obtain an approximate solution that is away from the optimal solution within provable gaps. In the following subsection, we will present two such computationally efficient algorithms.

5.4.1 Computationally Efficient Algorithms & Bounds on the Performance

We start with the following lemma which lower bounds the optimal average delay $\overline{T}^*$, optimized over all policies $\mathbf{G}$.

Lemma 5.1. *The optimal average delay, optimized over all association policies, is lower bounded by*

$$\overline{T}^* \geq \frac{1}{S}\sum_{s\in[S]}\left(\left\lfloor\frac{d_s}{\Lambda}\right\rfloor+1\right)\frac{\Lambda-t}{1+t}-\frac{1}{S}\sum_{s\in\mathbf{S}_2}\frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}, \tag{5.18}$$

where $d_s=\sum_{k\in[K]}d_{s,k}$, $A_s=d_s-\Lambda\left\lfloor\frac{d_s}{\Lambda}\right\rfloor$, *and where* $\mathbf{S}_2\subseteq[S]$ *is the set of time slots for which* $A_s<\Lambda-t$

Proof. The proof is deferred to Appendix C.5 □

The bound provided in Lemma 5.1 will serve as a benchmark for numerical performance evaluation of various user-to-cache state association algorithms. We now proceed to present our computationally efficient algorithms. In the following, $\mathcal{G}$ will denote the set form of the user-to-cache state association matrix $\mathbf{G}$, where $(\lambda,k)\in\mathcal{G}$ if $g_{\lambda,k}=1$. Similarly, $\mathcal{G}^{(\lambda)}=\{k:(\lambda,k)\in\mathcal{G}\}$ will denote the set of users that are storing the content of cache state $\lambda\in[\Lambda]$. Note that there is a direct correspondence between $\mathbf{G}$ and $\mathcal{G}$, and the two terms can be used interchangeably.

Algorithm 5.2

In the context of data-driven approach, problem 5.1 belongs to the family of well-known vector scheduling problems [77, 78], whose aim is to optimally assign each of the S-dimensional K jobs (i.e., the S-dimensional K vectors that are drawn from the columns of the user activity matrix $\mathbf{D}$) to one of the machines $\lambda \in [\Lambda]$ (i.e., cache states) with the objective of minimizing the maximum machine load (i.e., $\max_{s\in[S]} l_{s,1}$), or with the objective of minimizing the norm of the machine loads. One can see that the vector scheduling problem is the generalization of a classical load-balancing problem, where each job has a vector load instead of a scalar load.

We adopt the vector scheduling algorithm of [78, Section II-B3] to find the optimal user-to-cache state association within provable gaps. **Algorithm 5.2** consists of three parts. The first part is the data transformation, where the user activity matrix $\mathbf{D}$ is scaled according to step 00. The second part (steps 01 to 08) is the deterministic user-to-cache state association, where for each user $k \in [K]$, we find the cache state $\hat{\lambda} \in [\Lambda]$ according to step 02. If the scaled load (cf. step 03) of cache $\hat{\lambda}$ after the assignment of user k is less than $\frac{30 \log S}{\log \log S} + 1$ for all time slots $s \in [S]$, then user k is assigned to cache state $\hat{\lambda}$. Otherwise user k is not assigned to any of the cache states, and is instead added to a set of residual users denoted by $\mathcal{K}_r$, and will be associated to a cache state later in the third part of **Algorithm 5.2**. The outcome of the second part is the user-to-cache state association $\mathcal{G}_1$ for users in $[K]\backslash\mathcal{K}_r$. Next, the third part (steps 09 to 13) completes the association of the residual users in $\mathcal{K}_r$. Each user $k \in \mathcal{K}_r$ is assigned to cache $\hat{\lambda} \in [\Lambda]$ according to step 11. The outcome of this part is the user-to-cache state association $\mathcal{G}_2$ for users in $\mathcal{K}_r$. The final user-to-cache state association strategy for all users is then given by $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$.

Theorem 5.5. *When there are at least Λ requests at each time slot $s \in [S]$, the average delay $\overline{T}(\mathbf{G})$ corresponding to the user-to-cache state association $\mathbf{G}$ obtained from Algorithm 5.2 is bounded by*

$$\overline{T}(\mathbf{G}) = O\left(\frac{\log S}{\log \log S}\overline{T}^*\right), \tag{5.19}$$

which proves that Algorithm 5.2 is at most a factor $O\left(\frac{\log S}{\log\log S}\right)$ from the optimal.

Proof. The proof is deferred to Appendix C.6. □

Proposition 5.1. *The time complexity of Algorithm 5.2 is $O(\Lambda^2 KS)$.*

Proof. The first part of **Algorithm 5.2** runs for K iterations and in each iteration, the evaluation at step 02 takes at most $\Lambda^2 S$ basic operations. Then, the second part of **Algorithm 5.2** runs for at most K iterations and in each iteration, the evaluation at step 11 takes at most ΛS basic operations. Thus the time complexity of **Algorithm 5.2** is $O(\Lambda^2 KS)$. □

Directly from above, we can see that **Algorithm 5.2** is significantly faster than the exhaustive search algorithm for which as we recall the time complexity was $O\left(S\Lambda^{K+1}\right)$.

```
Input: $\mathbf{D}$, $K$, $\Lambda$, and $S$
Output: $\mathcal{G}$
Initialization: $\mathcal{G}_1 \leftarrow \emptyset$; $\mathcal{G}_2 \leftarrow \emptyset$; $\mathcal{K}_r \leftarrow \emptyset$; $\alpha = \frac{10 \log S}{\log \log S}$
Step 00: $\bar{d}_{s,k} \leftarrow \min\left(\frac{\Lambda \ d_{s,k}}{\sum_{i\in[K]} d_{s,i}}, 1\right) \quad \forall\ s \in [S], k \in [K]$
Step 01: for $k$ from 1 to $K$ do
Step 02:   $\hat{\lambda} \leftarrow \arg\min_{\lambda\in[\Lambda]} \left( \sum_{s=1}^{S} \sum_{\lambda=1}^{\Lambda} \left(\frac{1}{\alpha}\right)^{\frac{\alpha}{\Lambda} \sum_{i\in(\mathcal{G}_1\cup(\lambda,k))} \bar{d}_{s,i} - \sum_{j\in(\mathcal{G}_1\cup(\lambda,k))^{\lambda}} \bar{d}_{s,j}} \right)$
Step 03:   if $\sum_{k\in(\mathcal{G}_1\cup(\hat{\lambda},k))^{\hat{\lambda}}} \bar{d}_{s,k} < 3\alpha + 1 \ \forall\ s \in [S]$
Step 04:     $\mathcal{G}_1 \leftarrow \mathcal{G}_1 \cup (\hat{\lambda}, k)$
Step 05:   else
Step 06:     $\mathcal{K}_r \leftarrow \mathcal{K}_r \cup k$
Step 07:   end if
Step 08: end for
Step 09: for $c$ from 1 to $|\mathcal{K}_r|$ do
Step 10:   $k = \mathcal{K}_r(c)$
Step 11:   $\hat{\lambda} \leftarrow \arg\min_{\lambda\in[\Lambda]} \left( \max_{s\in[S]} \sum_{j\in(\mathcal{G}_2\cup(\lambda,k))^{\lambda}} \bar{d}_{s,j} \right)$
Step 12:   $\mathcal{G}_2 \leftarrow \mathcal{G}_2 \cup (\hat{\lambda}, k)$
Step 13: end for
Step 14: $\mathcal{G} \leftarrow \mathcal{G}_1 \cup \mathcal{G}_2$
```

Algorithm 5.2: Heuristic Algorithm for Problem 5.1

Algorithm 5.3

The main intuition behind **Algorithm 5.3** is to exploit the fact that both $\binom{\Lambda-\lambda}{t}$ and $l_{s,\lambda}$ are non-increasing with λ; a fact that directly follows from (5.17). Thus, the optimal user-to-cache state association strategy is the one that minimizes the variances of the cache load vectors $\mathbf{V}_s$ over all time slots. **Algorithm 5.3** aims to heuristically minimize the sum of squares of cache populations over all time slots, which is equivalent to minimizing the sum of variances of the cache load vectors over all time slots. **Algorithm 5.3** works in K iterations. At each iteration, it finds a pair of a user $\hat{k}$ and a cache state $\hat{\lambda}$ according to step 02 of **Algorithm 5.3** and assigns user $\hat{k}$ to cache state $\hat{\lambda}$.

Proposition 5.2. *The time complexity of Algorithm 5.3 is $O(\Lambda^2 K^2 S)$.*

Proof. **Algorithm 5.3** runs for K iterations and in each iteration, the evaluation at step 02 takes at most $K\Lambda^2 S$ basic operations. Thus the time complexity of **Algorithm 5.3** is $O(\Lambda^2 K^2 S)$. □

We can see that the time complexity of **Algorithm 5.3** is K times higher than the time complexity of **Algorithm 5.2**. However, in Section 5.5 we numerically show that **Algorithm 5.3** performs better than **Algorithm 5.2**.

```
**Input:** $\mathbf{D}$, $K$, and $\Lambda$
**Output:** $\mathcal{G}$
**Initialization:** $\mathcal{G} \leftarrow \emptyset$; $\mathcal{K} \leftarrow [K]$
Step 01: for $i$ from 1 to $K$ do
Step 02:     $[\hat{\lambda}, \hat{k}] \leftarrow \underset{\lambda \in [\Lambda], k \in \mathcal{K}}{\operatorname{argmin}} \sum_{s \in [S]} \sum_{i \in [\Lambda]} \left( \sum_{j \in (\mathcal{G} \cup (\lambda, k))^{(i)}} d_{s,j} \right)^2$
Step 03:     $\mathcal{G} \leftarrow \mathcal{G} \cup (\hat{\lambda}, \hat{k})$
Step 04:     $\mathcal{K} \leftarrow \mathcal{K} \backslash \hat{k}$
Step 05: end for
```

Algorithm 5.3: Heuristic Algorithm for Problem 5.1

5.5 Numerical Validation

In this section, we numerically validate our analytical bounds, and evaluate the performance of the different proposed user-to-cache state association algorithms.

5.5.1 Statistical Approach

We first evaluate our proposed analytical bounds in Theorem 5.1 and Theorem 5.3 for the statistical setting using the *sampling-based numerical* (SBN) approximation method, where for any given $\mathbf{G}$, we generate a sufficiently large set $\mathcal{L}_1$ of randomly generated profile vectors $\mathbf{L}$ based on user activity vector $\mathbf{p}$ and where we subsequently approximate $\overline{T}(\mathbf{G})$ as

$$\overline{T}(\mathbf{G}) \approx \frac{1}{|\mathcal{L}_1|} \sum_{\mathbf{L} \in \mathcal{L}_1} T(\mathbf{L}), \tag{5.20}$$

where $T(\mathbf{L})$ is defined in (5.2).

For our evaluations involving an arbitrary user activity level vector $\mathbf{p}$, we adopt the Pareto principle to generate the synthetic user activity level vector $\mathbf{p}$. According to the Pareto principle, 80% of consequences (content requests) come from 20% of causes (users). To be exact, each user $k \in [K]$ has a request with probability

$$p_k = \begin{cases} \frac{1}{\sum_{i=1}^{5} i^{-2.7}} & \textit{if} \quad k = [1, 2, \cdots, 0.2K] \\ \frac{2^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \textit{if} \quad k = [0.2K+1, 0.2K+2, \cdots, 0.4K] \\ \frac{3^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \textit{if} \quad k = [0.4K+1, 0.4K+2, \cdots, 0.6K] \\ \frac{4^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \textit{if} \quad k = [0.6K+1, 0.6K+2, \cdots, 0.8K] \\ \frac{5^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \textit{if} \quad k = [0.8K+1, 0.8K+2, \cdots, K]. \end{cases} \tag{5.21}$$

The intuition behind (5.21) is that users are divided into 5 equipopulated groups, and the users that belong to the same group have the same activity levels. The activity levels corresponding to these 5 groups then follow the Power law with parameter $\alpha = 2.7$, and with these carefully selected parameters, the user activity pattern satisfies the Pareto principle (80/20 rule) [79].

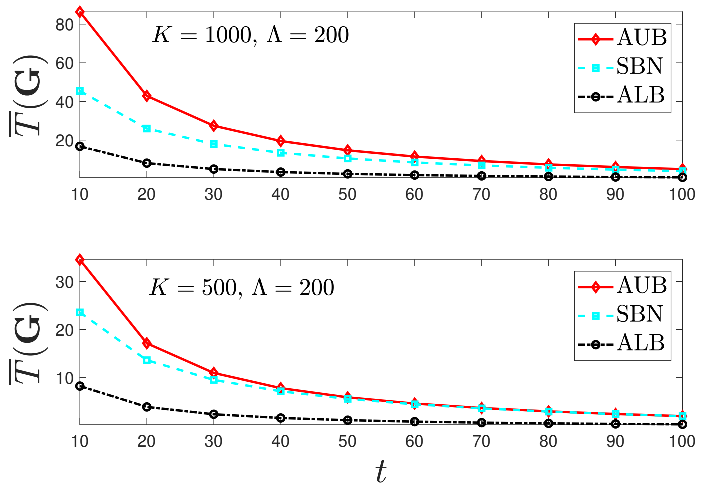


Figure 5.2: Analytical upper bound (AUB) from (5.6) vs. analytical lower bound (ALB) from (5.7) vs. sampling-based numerical (SBN) approximation in (5.20) (for $|\mathcal{L}_1| = 20000$, $\mathbf{p}$ in (5.21), and random user-to-cache state association).

In Figure 5.2, we compare the analytical bounds in (5.6) and (5.7) for an arbitrary activity level vector $\mathbf{p}$, where this comparison uses the sampling-based numerical (SBN) approximation which is done for $|\mathcal{L}_1| = 20000$ and random user-to-cache state association. Subsequently, Figure 5.3 compares the analytical bounds in (5.13) and (5.14) for uniform user activity level, where again the comparison is with sampling-based numerical (SBN) approximation which is done for $|\mathcal{L}_1| = 20000$ and uniform user-to-cache state association. Both figures reveal the proposed analytical bounds to be very tight, where in particular, analytical upper bounds are indeed very close to the exact performance.

Next, we evaluate the performance of our first proposed user-to-cache state association algorithm (**Algorithm 5.1**) by comparing it with the numerical lower bound (NLB) on the delay $\overline{T}^*$ corresponding to the optimal user-to-cache state association $\hat{\mathbf{G}}$ of Lemma 5.1. Figure 5.4 compares SBN approximation (once again done for $|\mathcal{L}_1| = 20000$) for the user-to-cache state association obtained from **Algorithm 5.1** with the numerical lower bound (NLB) on $\overline{T}^*$ in (5.18). Again we observe that the performance corresponding to the user-to-cache state association $\mathbf{G}$ obtained from **Algorithm 5.1** is very close to NLB for $\overline{T}^*$.

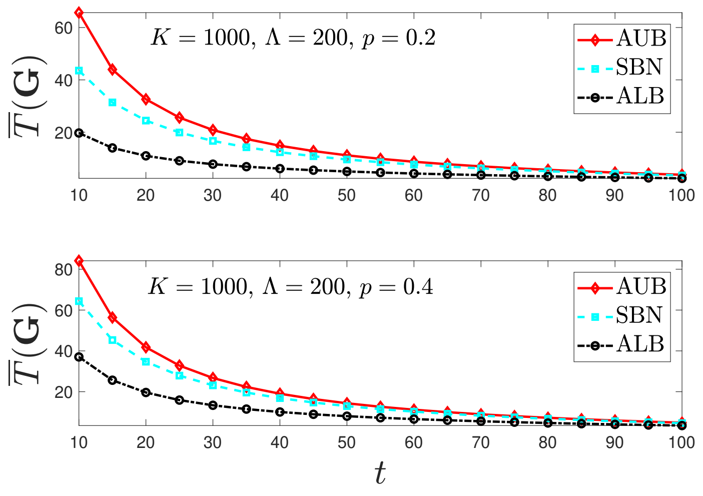


Figure 5.3: Analytical upper bound (AUB) from (5.13) vs. analytical lower bound (ALB) from (5.14) vs. sampling-based numerical (SBN) approximation in (5.20) (for $|\mathcal{L}_1| = 20000$ and uniform user-to-cache state association).

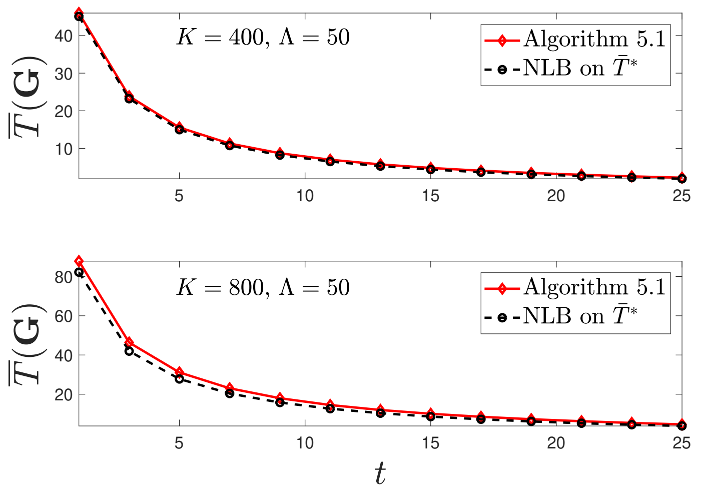


Figure 5.4: SBN from (5.20) of **Algorithm 5.1** vs. Numerical lower bound (NLB) on $\overline{T}^*$ from (5.18) (for $|\mathcal{L}_1| = 20000$ and $\mathbf{p}$ in (5.21)).

5.5.2 Data-Driven Approach

For the data-driven approach, we synthetically generate a user activity matrix $\mathbf{D}$ following the Pareto principle. To be exact, we assume that user $k \in [K]$ develops a request (i.e., is active) with probability p_k as in (5.21) at each time slot $s \in [S]$. Then, for each time slot $s \in [S]$, we pick a random number r_k between 0 and 1 for each user $k \in [K]$, and set $d_{s,k} = 1$ if $r_k \leq p_k$, and $d_{s,k} = 0$ if $r_k > p_k$, which yields a user activity matrix $\mathbf{D}$ satisfying the Pareto principle [79].

In Figure 5.5, we compare the average delay $\overline{T}(\mathbf{G})$ in (5.17) corresponding to the user-to-cache state association obtained from **Algorithm 5.2** and **Algorithm 5.3** with the lower bound (LB) on $\overline{T}^*$ in (5.18). It turns out that both algorithms yield performances that are over close to the optimal LB on $\overline{T}^*$, with **Algorithm 5.3** having a slight advantage over **Algorithm 5.2**.

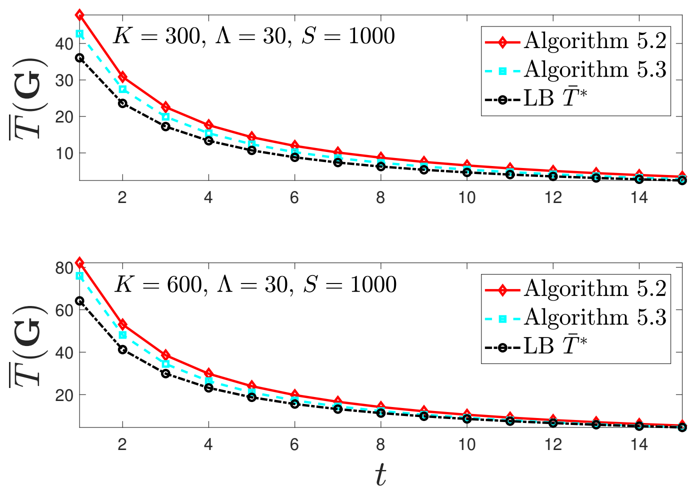


Figure 5.5: $\overline{T}(\mathbf{G})$ of **Algorithm 5.2** and **Algorithm 5.3** from (5.17) vs. lower bound (LB) on $\overline{T}^*$ from (5.18).

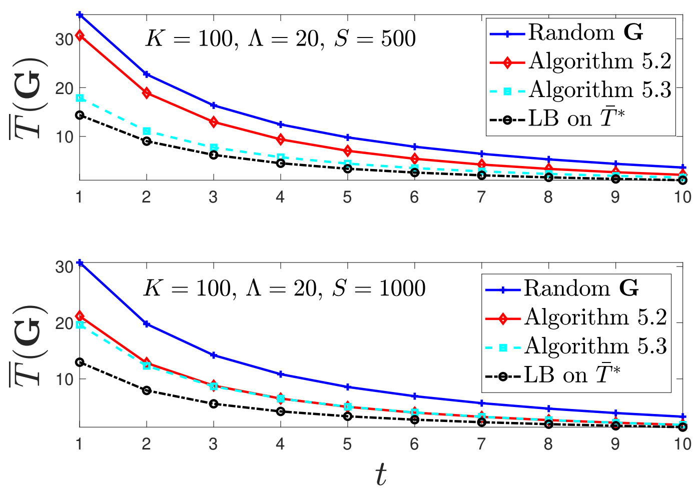


Figure 5.6: $\overline{T}(\mathbf{G})$ of random user-to-cache state association, the user-to-cache state associations obtained from **Algorithm 5.2**, and **Algorithm 5.3** from (5.17), and the lower bound (LB) of (5.18).

In our final evaluation, we highlight the importance of exploiting the user activity patterns and finding an efficient user-to-cache state association. Figure 5.6 compares $\overline{T}(\mathbf{G})$ values for random user-to-cache state association, and the user-to-cache state associations obtained from **Algorithm 5.2** and

Algorithm 5.3, where the lower bound (LB) of (5.18) serves as a benchmark. It turns out that both algorithms outperform random user-to-cache state association, and the corresponding delay performances perform very close to the optimal LB on $\overline{T}^*$, with once again **Algorithm 5.3** having a slight advantage over **Algorithm 5.2**.

5.6 Summary of Chapter

In this chapter, we analyzed the subpacketization-constrained coded caching networks when users have different activity levels. We highlighted the essence of exploiting the users' activity levels while developing the user-to-cache association strategies for such networks. We first presented a statistical analysis of the performance of coded caching under such networks and provided bounds and scaling laws under the assumption of probabilistic user activity levels. We proposed a heuristic user-to-cache state association algorithm with the ultimate goal of minimizing the average delay. Next, we extended our analysis to the data-driven setting and proposed two algorithms for user-to-cache state association strategies with the ability to learn and exploit the correlations between users' activity from the past demand vectors.

Chapter 6

Conclusion

In this thesis, we studied the actual gains of coded caching in stochastic networks, where the stochasticity of the networks originates from the heterogeneity in users' request behaviors. We divided our study into two parts. In the first part, we studied the impacts of stochastic cache population intensities on coded caching gains in the shared-cache settings. We identified the cache-load imbalance bottleneck of coded caching in such settings and proposed techniques that can play a crucial role in resolving this bottleneck. Then, in the second part, we studied the impacts of heterogeneous user activity in subpacketization-constrained coded caching networks, where at any given instance of the time a user may or may not request a file from the content library. It is known that, in the context of coded caching, the shared-cache setting is related to the subpacketization-constrained setting as the subpacketization constraint forces users to store the same content in their cache (i.e., share the same cache state). However, these two settings differ in the context of the extent of freedom in deciding the user-to-cache association strategies, which determines the extent of coding gain we can achieve by exploiting the multicasting opportunities. Unlike the shared-cache setting, the user-to-cache association strategies for the subpacketization-constrained setting are not limited by the geographical proximity constraints. Therefore, in this part of the thesis, we focused on identifying the optimal user-to-cache state association in the presence of arbitrary user activity levels in the subpacketization-constrained coded caching setting. Let us revisit the main contributions of this thesis and highlight the lessons we have learned from them.

6.1 Coded Caching in Stochastic Shared-Cache Networks

We identified the exact optimal performance of coded caching in the shared-cache setting with K users, Λ cache-enabled helper nodes, and random user-to-cache association when each user can appear in the coverage area of any particular cache-enabled helper node based on a given probability distribution.

Key to our effort to identify the effect of association randomness was the need to provide expressions that can either be evaluated in a numerically tractable way, or that can be rigorously approximated in order to yield clear insight. The first part was achieved by deriving exact expressions as well as new analytical bounds that can be evaluated directly, while the second part was achieved by studying the asymptotics of the problem which yielded simple performance expressions and direct operational guidelines.

The scaling laws revealed to what extent the performance deterioration experienced in this random setting compared to the deterministic uniform user-to-cache association setting increases with the skewness in cache population intensities. We now know that the performance deterioration can be unbounded. For example, under uniform cache population intensities, when $K = \Theta(\Lambda)$, the performance deterioration is unbounded and scales exactly as $\Theta\left(\frac{\log \Lambda}{\log\log \Lambda}\right)$. We learned that the impact of association randomness becomes prominent under non-uniform cache population intensities as in some cases the performance deterioration scales as $\Theta(\Lambda)$ (i.e., there is no coded caching gain). What we additionally learned is that under uniform cache population intensities the performance deterioration can be avoided as long as $K = \Omega(\Lambda \log \Lambda)$. However, the same objective cannot be achieved under non-uniform cache population intensities, and the deterioration remains unbounded irrespective of the relation between K and Λ.

In our opinion, these results are of the utmost importance as the random association problem has direct practical ramifications, as it captures promising scenarios (such as the heterogeneous network scenario) as well as operational realities (namely, the subpacketization constraint). The problem becomes even more pertinent as we now know that its effect can in fact scale indefinitely.

6.1.1 Resolving Cache-Load Imbalance Bottleneck

Our stochastic analysis highlighted the importance of designing the placement and delivery schemes that are based on the cache population intensities as being unaware of it may lead to the vanishing of the coding gain, and the system may eventually need to confine itself to the local caching gain. Therefore, we proposed three techniques to mitigate the impact of the cache-load imbalance bottleneck of coded caching in stochastic shared-cache networks.

In the first technique, we used load-balancing approaches to mitigate the effects of randomness and showed that in scenarios where we are given a choice to associate a user to the least loaded cache from a randomly chosen group of h helper nodes (neighboring in case of proximity-bounded), the performance deterioration due to the random user-to-cache association can be reduced significantly. An even more dramatic reduction in performance deterioration can be achieved when the aforementioned neighboring/proximity constraint is lifted. The main advantage of this technique is that it mitigates the effects of randomness by exploiting the freedom in deciding the user-to-cache association, therefore, it does not require the perfect knowledge of the cache population intensities. However, when applicable, load-balancing can play a crucial role in significantly reducing the performance deterioration due to random user-to-cache association.

In the second technique, we proposed a novel coded caching scheme that optimizes the cache sizes based on caches' load statistics 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. The main advantage of this technique is that it does not require optimizing the user-to-cache association, which makes it suitable for the shared-cache setting. The major disadvantage of this scheme is that the additional gains come at a cost of a higher subpacketization rate compared to the original shared-cache coded caching scheme. However, our scheme offers a trade-off between the subpacketization and the delivery time by just tuning a single parameter. This aspect has not been fully studied and would be considered in our future studies. At this point, we need to highlight that for deterministic user-to-cache association setting, we proved that our scheme achieves the same delivery time – which was proven to be close to optimal for bounded values of the cumulative cache-size constraint – with an exponential reduction in the subpacketization.

In the third technique, we proposed a two-layered coded caching scheme to resolve the cache-load imbalance bottleneck of coded caching in a stochastic shared-cache network. The main idea is that in addition to a cache-enabled helper node in each cell (i.e., the coverage area of a helper node), each user is also equipped with its own storage capacity. This will enable the overpopulated cell to have higher collective storage capacity compared to the less populated cells, and eventually alleviate the detrimental performance deterioration due to randomness. We identified the exact scaling laws of the average delivery time of our proposed two-layered coded caching scheme under the uniform cache population intensities and showed that this scheme significantly mitigates, and in certain memory regimes completely nullifies the detrimental performance deterioration due to randomness. The analysis for the case of non-uniform cache population intensities is still an open problem that we aim to solve in our future work. Moreover, this technique does not require optimizing the user-to-cache association, which makes it suitable for the shared-cache setting. However, similar to the previous technique, this technique requires a higher subpacketization rate compared to the original shared-cache setting.

6.2 Coded Caching in Networks with Heterogeneous User Activity

In the second part of the thesis, we analyzed coded caching networks with finite number of cache states and a user-to-cache state association subject to a grouping strategy in the presence of heterogeneous user activity. Even though coded caching techniques rely on the assumption of having enough number of users to provide its theoretically promised gains, all the earlier works ignored the fact of heterogeneity in user activities, which in our opinion has direct practical ramifications, as it captures practical wireless networks more accurately.

We first presented a statistical analysis of the average worst-case delivery performance of subpacketization-constrained coded caching networks, and provided bounds and scaling laws under the assumption of probabilistic user activity levels. Our analysis revealed interesting insights about the characteristics of the optimal association strategy that leads to the minimum average delivery time. Based on these insights, we proposed a heuristic algorithm for the user-to-cache association, which aims to minimize the average delay in the presence of heterogeneous user activity. Next, we extended our analysis to the data-driven setting, where we were able to learn from the past S different demand vectors in designing the caching policy. By exploiting this bounded-depth user request history, the emphasis then was placed on finding the optimal user-to-cache state association – as computing the average delay for any given data is trivial –. We proposed two algorithms to find the optimal user-to-cache state association strategy, with one providing the optimal within a constant gap, whereas the other one was numerically verified to outperform the other. For both aforementioned settings, the results highlighted the essence of exploiting the user activity level, and the importance of carefully associating users to cache states based on their activity patterns.

6.3 Final Remarks

In conclusion, our study highlighted that stochasticity in the network causes a deterioration in coded caching performance and leads to the vanishing of the coding gain. We determined the exact extent of the cache-load imbalance bottleneck of coded caching in stochastic networks, which was not known before, and this, in a nutshell, is the main contribution of this dissertation. For the scenarios where the user-to-cache state associations are restricted by proximity constraints between users and helper nodes (i.e., shared-cache setting), we proposed three effective techniques (load-balancing, cache size optimization, and two-layered cache network) to mitigate the impact of this bottleneck. Furthermore, for the scenario where user-to-cache state associations strategies are considered as a design parameter (subpacketization-constrained settings), we proposed several algorithms to find the user-to-cache state association strategy that exploits the prior history on user activity levels and correlations to mitigate the impact of cache-load imbalance, with one providing the optimal solution within a constant gap.

Appendix A

Proofs of Chapter 3

A.1 Proof of Theorem 3.1

We first note that the probability $P(\mathbf{L})$ of observing a specific profile vector $\mathbf{L} \in \mathcal{L}$ is simply the cumulative probability over all $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$. This probability takes the form

$$P(\mathbf{L}) = \overbrace{\frac{1}{\Lambda^K} \times \frac{K!}{\prod_{i=1}^{\Lambda} l_i!}}^{\text{term 1}} \times \overbrace{\frac{\Lambda!}{\prod_{j=1}^{|B_L|} b_j!}}^{\text{term 2}}. \tag{A.1}$$

To see this, we analyze the different terms of the above equation. The first term in (A.1) accounts for the fact that there are Λ^K different user-to-cache associations, i.e., there are Λ^K different ways that K users can be allocated to the Λ different caches. It also accounts for the fact that each user can be associated to any one particular cache, with equal probability $\frac{1}{\Lambda}$. The second term in (A.1) indicates the number of all user-to-cache associations that leads[1] to a specific $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$, for some fixed $\mathbf{L}$. Consequently term 1 in (A.1) is simply $P(\mathbf{V})$, which naturally remains fixed for any $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$, and which originates from the well-known probability mass function of the multinomial distribution. Consequently this implies that $P(\mathbf{L}) = |\{\mathbf{V}\colon sort(\mathbf{V}) = \mathbf{L}\}| \times P(\mathbf{V})$. Finally, term 2 describes the number of all possible cache population vectors $\mathbf{V}$ for which $sort(\mathbf{V})$ is equal to some fixed $\mathbf{L}$.

We now proceed to insert (A.1) into (3.7), which yields the average delay

$$E_{\mathbf{L}}[T(\mathbf{L})] = \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}$$

[1]Recall that different user-to-cache associations can lead to the same cache population vector $\mathbf{V}$. For example, when $K = \Lambda = 3$, the following 6 user-to-cache associations, $[1,2,3]$, $[1,3,2]$, $[2,1,3]$, $[2,3,1]$, $[3,2,1]$, and $[3,1,2]$ – each describing which user is associated to which cache – in fact all correspond to the same $\mathbf{V} = [1,1,1]$, because always each cache is associated to one user.

$$= \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{l_\lambda K!\Lambda!}{\Lambda^K \prod_{i=1}^{\Lambda} l_i! \prod_{j=1}^{|\mathbf{B_L}|} b_j!} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}$$
$$= \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{K!\ t!\ (\Lambda-t)!\ l_\lambda \binom{\Lambda-\lambda}{t}}{\Lambda^K \prod_{i=1}^{\Lambda} l_i! \prod_{j=1}^{|\mathbf{B_L}|} b_j!}, \tag{A.2}$$

which concludes the achievability part of the proof for the expression in Theorem 3.1.

Optimality of the aforementioned expression can be proved by means of the lower bound developed in [31]. We notice that the optimal delay $\overline{T}^*(\gamma)$ can be lower bounded as

$$\begin{aligned}\overline{T}^*(\gamma) &= \min_{\mathcal{X}} E_{\mathbf{L}}\left[E_{\mathbf{V_L}}\left[\max_{\mathbf{d}} T(\mathbf{V},\mathbf{d},\mathcal{X})\right]\right]\\ &\geq \min_{\mathcal{X}} E_{\mathbf{L}}\left[\max_{\mathbf{d}} E_{\mathbf{V_L}}\left[T(\mathbf{V},\mathbf{d},\mathcal{X})\right]\right]\\ &\geq E_{\mathbf{L}}\left[\min_{\mathcal{X}}\max_{\mathbf{d}} E_{\mathbf{V_L}}\left[T(\mathbf{V},\mathbf{d},\mathcal{X})\right]\right]\\ &\geq E_{\mathbf{L}}\Bigg[\underbrace{\min_{\mathcal{X}} E_{\mathbf{d}\in\mathcal{D}_{wc}} E_{\mathbf{V_L}}\left[T(\mathbf{V},\mathbf{d},\mathcal{X})\right]}_{T^*(\mathbf{L})}\Bigg],\end{aligned} \tag{A.3}$$

where $\mathcal{D}_{wc}$ denoted the set of demand vectors with distinct users' file-requests. Next, exploiting the fact that $P(\mathbf{V})$ is the same for any $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$, we notice that

$$T^*(\mathbf{L}) \triangleq \min_{\mathcal{X}} E_{\mathbf{d}\in\mathcal{D}_{wc}} E_{\mathbf{V_L}}\left[T(\mathbf{V},\mathbf{d},\mathcal{X})\right]$$

is lower bounded by equation (53) in [31], which then proves that $T^*(\mathbf{L})$ is bounded as

$$T^*(\mathbf{L}) \geq \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}. \tag{A.4}$$

This concludes the proof for the optimality of the delivery time in Theorem 3.1.

A.2 Proof of Theorem 3.2

We start our proof by deriving the expected number of users in the λ-th most populous cache (i.e., $E[l_\lambda]$), which is given by

$$E[l_\lambda] = \sum_{j=0}^{K-1} P[l_\lambda > j] = \sum_{j=0}^{K-1} (1 - P[l_\lambda \leq j]) = K - \sum_{j=0}^{K-1} P[l_\lambda \leq j]. \tag{A.5}$$

where $P[l_\lambda \leq j]$ is the probability that λ-th most populous cache is associated to no more than j requesting users. From [80, Proposition 2], we have

$$P[l_\lambda \leq j] \geq \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right), \tag{A.6}$$

where P_j is the probability that a cache is associated to no more than j requesting users. Recalling that each user can be assigned to any particular cache with equal probability, we can conclude that P_j is given as

$$P_j = \sum_{i=0}^{j} \binom{K}{i} \left(\frac{1}{\Lambda}\right)^i \left(1 - \frac{1}{\Lambda}\right)^{K-i}. \tag{A.7}$$

$E[l_\lambda]$ is upper bounded by

$$E[l_\lambda] \leq K - \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right). \tag{A.8}$$

Consequently the upper bound of $\overline{T}^*(\gamma)$ is given as

$$\begin{aligned}
\overline{T}^*(\gamma) &= \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda] \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\
&\leq \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \left(K - \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right)\right) \\
&= \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} K - \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right) \\
&\overset{(a)}{=} \frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} K - \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right) \\
&= K\frac{\Lambda-t}{t+1} - \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \sum_{j=0}^{K-1} \max\left(1 - \frac{\Lambda}{\lambda}(1-P_j), 0\right),
\end{aligned} \tag{A.9}$$

where in step (a), we used the column-sum property of Pascal's triangle, which is $\sum_{k=0}^{n} \binom{k}{t} = \binom{n+1}{t+1}$. This concludes the proof of the upper bound in (3.11).

Next, we prove the lower bound in (3.12). Crucial to this proof is the exploitation of the fact that $\sum_{\lambda=1}^{\Lambda} E[l_\lambda] = K$ and of the fact that both $E[l_\lambda]$ and $\binom{\Lambda-\lambda}{t}$ in (3.7) are non-increasing with λ. We first see that

$$\overline{T}^*(\gamma) = \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda] \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \geq \frac{E[l_1]\binom{\Lambda-1}{t} + \sum_{\lambda=2}^{\Lambda-t} B\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{A.10}$$

where $B = \frac{K-E[l_1]}{\Lambda-1}$. This can be simplified as

$$\overline{T}^*(\gamma) \geq \frac{E[l_1]\binom{\Lambda-1}{t} + \sum_{\lambda=2}^{\Lambda-t} B\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}$$

$$\begin{aligned}
&= E[l_1]\frac{\binom{\Lambda-1}{t}}{\binom{\Lambda}{t}} + B\frac{\binom{\Lambda-1}{t+1}}{\binom{\Lambda}{t}} \\
&= E[l_1]\frac{\Lambda-t}{\Lambda} + B\frac{(\Lambda-t)(\Lambda-t-1)}{(1+t)\Lambda} \\
&= (\Lambda-t)\left(\frac{E[l_1]}{\Lambda} + B\frac{\Lambda-t-1}{(1+t)\Lambda}\right) \\
&= (\Lambda-t)\left(\frac{E[l_1]}{\Lambda} + \frac{K-E[l_1]}{\Lambda-1}\frac{\Lambda-t-1}{(1+t)\Lambda}\right) \\
&= \frac{\Lambda-t}{1+t}\left(\frac{E[l_1]t}{\Lambda-1} + \frac{K}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}\right). \qquad \text{(A.11)}
\end{aligned}$$

To conclude the proof, we need to derive $E[l_1]$. It is straightforward that $l_1 \geq \lceil \frac{K}{\Lambda} \rceil$, thus for $j = \left[0, 1, 2, \cdots, \lceil \frac{K}{\Lambda} \rceil - 1\right]$ we have

$$P[l_1 \leq j] = 0, \qquad \text{(A.12)}$$

and for $j = \left[\lceil \frac{K}{\Lambda} \rceil, \lceil \frac{K}{\Lambda} \rceil + 1, \cdots, K\right]$, from [80, Proposition 1] we have

$$P[l_1 \leq j] \leq \min(P_j, 1) = P_j, \qquad \text{(A.13)}$$

where P_j is defined in (A.7). Therefore, using (A.5), $E[l_1]$ is lower bounded as

$$E[l_1] = K - \sum_{j=0}^{K-1} P[l_1 \leq j] \geq K - \sum_{j=\lceil \frac{K}{\Lambda} \rceil}^{K-1} P_j. \qquad \text{(A.14)}$$

Finally, combining (A.11) and (A.14), we obtain

$$\overline{T}^*(\gamma) \geq \frac{\Lambda-t}{1+t}\left(\frac{t}{\Lambda-1}\left(K - \sum_{j=\lceil \frac{K}{\Lambda} \rceil}^{K-1} P_j\right) + \frac{K}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}\right), \qquad \text{(A.15)}$$

which concludes the proof of Theorem 3.2.

A.3 Proof of Theorem 3.3

The fact that both $E[l_\lambda]$ and $\binom{\Lambda-\lambda}{t}$ in (3.7) are non-increasing with λ, we see that $\overline{T}^*(\gamma)$ is bounded by

$$\begin{aligned}
\overline{T}^*(\gamma) &= \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda]\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \leq \frac{1}{\binom{\Lambda}{t}} \sum_{\lambda=1}^{\Lambda-t} E[l_1]\binom{\Lambda-\lambda}{t} \\
&\overset{(a)}{=} \frac{E[l_1]}{\binom{\Lambda}{t}} \sum_{\lambda=1}^{\Lambda-t} \binom{\Lambda-\lambda}{t} = E[l_1]\frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} = E[l_1]\frac{\Lambda-t}{1+t}
\end{aligned}$$

$$= \frac{K(1-\gamma)}{1+t}\frac{\Lambda E[l_1]}{K}, \tag{A.16}$$

where in step (a), we used the column-sum property of Pascal's triangle, which is $\sum_{k=0}^{n}\binom{k}{t} = \binom{n+1}{t+1}$. Thus from (A.16), we get

$$\overline{T}^*(\gamma) = O\left(\frac{K(1-\gamma)}{1+t}\frac{\Lambda E[l_1]}{K}\right) \tag{A.17}$$

and from (A.11), we have

$$\overline{T}^*(\gamma) = \Omega\left(\frac{K(1-\gamma)}{1+t}\frac{\Lambda E[l_1]\gamma}{K}\right). \tag{A.18}$$

As γ is a constant, we can conclude that the expressions in (A.17) and (A.18) asymptotically match, and thus

$$\overline{T}^*(\gamma) = \Theta\left(\frac{K(1-\gamma)}{1+t}\frac{E[l_1]\Lambda}{K}\right). \tag{A.19}$$

Combining (A.19) and (3.6), we obtain

$$\overline{T}^*(\gamma) = \Theta\left(T_{min}\frac{E[l_1]\Lambda}{K}\right). \tag{A.20}$$

For the remaining part, which is to develop the asymptotics of $E[l_1]$, we proceed with the following lemma which is adopted and adapted here directly from the work of [81] on the *Balls into Bins problem.*

Lemma A.1 ([81, Theorem 1] - adaptation). *In a Λ-cell K-user setting where each user can be associated with equal probability to any of the caches, the tail of l_1 takes the form*

$$P[l_1 > k_\beta] = \begin{cases} o(1) & \text{if} \quad \beta > 1 \\ 1 - o(1) & \text{if} \quad 0 < \beta < 1, \end{cases} \tag{A.21}$$

for

$$k_\beta = \begin{cases} \frac{\log\Lambda}{\log\frac{\Lambda\log\Lambda}{K}}\left(1+\beta\frac{\log\log\frac{\Lambda\log\Lambda}{K}}{\log\frac{\Lambda\log\Lambda}{K}}\right) \text{ if } \frac{\Lambda}{\text{polylog}(\Lambda)} \leq K = o(\Lambda\log\Lambda) \\ \Theta(\beta\log\Lambda) \quad \text{if } K = \Theta(\Lambda\log(\Lambda)) \\ \frac{K}{\Lambda} + \beta\sqrt{\frac{K\log(\Lambda)}{0.5\Lambda}} \quad \text{if } \omega(\Lambda\log\Lambda) = K \leq \Lambda\,\text{polylog}(\Lambda) \\ \frac{K}{\Lambda} + \sqrt{\frac{K\log(\Lambda)}{0.5\Lambda}\left(1 - \frac{\log\log\Lambda}{2\beta\log\Lambda}\right)} \text{ if } K = \omega\left(\Lambda(\log\Lambda)^3\right). \end{cases} \tag{A.22}$$

Proof. The result comes directly from [81, Theorem 1]. □

With Lemma A.1 at hand, we consider the case of $\beta > 1$, for which we get that

$$E[l_1] = \sum_{j=0}^{k_\beta - 1} P[l_1 > j] + P[l_1 > k_\beta] + \sum_{j=k_\beta+1}^{K-1} P[l_1 > j] \overset{(a)}{\leq} k_\beta + o(1) + \sum_{j=k_\beta+1}^{K-1} P[l_1 > j]$$

$$\overset{(b)}{\leq} k_\beta + o(1) + (K - k_\beta - 1)o(1) = k_\beta(1 - o(1)) + Ko(1) = O(k_\beta), \quad \text{(A.23)}$$

where in step (a), we use the fact that $P[l_1 > j]$ is at most 1 for $j = [0, 1, \cdots k_\beta - 1]$ and in step (b), we use the fact if $P[l_1 > k_\beta] = o(1)$ then $P[l_1 > j]$ is at most $o(1)$ for $j = [k_\beta + 1, \cdots K - 1]$. Similarly, for $0 < \beta < 1$, we have

$$E[l_1] = \sum_{j=0}^{k_\beta - 1} P[l_1 > j] + P[l_1 > k_\beta] + \sum_{j=k_\beta+1}^{K-1} P[l_1 > j]$$

$$\overset{(a)}{\geq} k_\beta(1 - o(1)) + 1 - o(1) \geq k_\beta(1 - o(1)) = \Omega(k_\beta), \quad \text{(A.24)}$$

where in step (a), we use the fact that $\sum_{j=k_\beta+1}^{K-1} P[l_1 > j] \geq 0$ and if $P[l_1 > k_\beta] = 1 - o(1)$ then $P[l_1 > j]$ is at least $1 - o(1)$ for $j = [0, 1, \cdots k_\beta - 1]$. Combining (A.22), (A.23), and (A.24), we have

$$E[l_1] = \begin{cases} \Theta\left(\frac{\log \Lambda}{\log \frac{\Lambda \log \Lambda}{K}}\right) & \text{if } \frac{\Lambda}{\text{polylog}(\Lambda)} \leq K = o(\Lambda \log \Lambda) \\ \Theta(\log \Lambda) & \text{if } K = \Theta(\Lambda \log(\Lambda)) \\ \Theta\left(\frac{K}{\Lambda} + \sqrt{\frac{K \log(\Lambda)}{\Lambda}}\right) & \text{if } \omega(\Lambda \log \Lambda) = K \leq \Lambda\, \text{polylog}(\Lambda) \\ \Theta\left(\frac{K}{\Lambda} + \sqrt{\frac{K \log(\Lambda)}{\Lambda}}\right) & \text{if } K = \omega\left(\Lambda (\log \Lambda)^3\right), \end{cases} \quad \text{(A.25)}$$

which in turn implies that

$$E[l_1] = \begin{cases} \Theta\left(\frac{\log \Lambda}{\log \frac{\Lambda \log \Lambda}{K}}\right) & \text{if } K \in [\frac{\Lambda}{\text{polylog}(\Lambda)}, o(\Lambda \log \Lambda)] \\ \Theta\left(\frac{K}{\Lambda} + \sqrt{\frac{K \log(\Lambda)}{\Lambda}}\right) & \text{if } K = \Omega(\Lambda \log \Lambda). \end{cases} \quad \text{(A.26)}$$

Combining (A.20) with (A.26), allows us to directly conclude the proof of Theorem 3.3.

A.4 Characterization of T_{min}

From equation (3.4), we see the fact that l_λ and $\binom{\Lambda - \lambda}{t}$ are non-increasing with λ, which implies that the profile vector $\mathbf{L}$, which minimizes the delay has components of the form

$$l_\lambda = \begin{cases} \lfloor \frac{K}{\Lambda} \rfloor + 1 & \text{for } \lambda \in \left[1, 2, \ldots, \hat{K}\right] \\ \lfloor \frac{K}{\Lambda} \rfloor & \text{for } \lambda \in \left[\hat{K} + 1, \hat{K} + 2, \ldots, \Lambda\right], \end{cases} \quad \text{(A.27)}$$

where $\hat{K} = K - \left\lfloor \frac{K}{\Lambda} \right\rfloor \Lambda$. Consequently, when $\hat{K} \geq \Lambda - t$, the corresponding best-case delay T_{min} is given as

$$T_{min} = \sum_{\lambda=1}^{\Lambda-t} \left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 \right) \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} = \left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 \right) \frac{\Lambda - t}{1+t}, \tag{A.28}$$

while when $\hat{K} < \Lambda - t$, this is given as

$$\begin{aligned} T_{min} &= \sum_{\lambda=1}^{\hat{K}} \left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 \right) \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + \sum_{\lambda=\hat{K}+1}^{\Lambda-t} \left\lfloor \frac{K}{\Lambda} \right\rfloor \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left\lfloor \frac{K}{\Lambda} \right\rfloor \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + \sum_{\lambda=1}^{\hat{K}} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 \right) \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} - \sum_{\lambda=\hat{K}+1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 \right) \frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} - \frac{\binom{\Lambda-\hat{K}}{t+1}}{\binom{\Lambda}{t}} \\ &= \frac{\Lambda - t}{1+t} \left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 - \frac{\prod_{i=t+1}^{\hat{K}+t} (\Lambda - i)}{\prod_{j=0}^{\hat{K}-1} (\Lambda - j)} \right), \end{aligned} \tag{A.29}$$

which reverts back to the well-known delay

$$T_{min} = \frac{K}{\Lambda} \frac{\Lambda - t}{1+t}, \tag{A.30}$$

when $\hat{K} = 0$. This concludes the characterization of the best-case delay T_{min}.

A.5 Proof of Theorem 3.4

The proof is essentially the same as of Theorem 3.2's proof with some minor changes. Crucial to the proof of upper bound is the result from [80, Proposition 3], which is given as

$$P[l_\lambda \leq j] \geq \max \left(0, 1 - \frac{\Lambda - \sum_{k=1}^{\Lambda} F_{v_k}(j)}{\lambda} \right), \tag{A.31}$$

where $F_{v_k}(j)$ is the probability that kth cache is associated to no more than j requesting users (i.e., $P[v_k \leq j]$). Recalling that a user can be associated to kth cache with probability p_k, we can conclude that $F_{v_k}(j)$ is given as

$$F_{v_k}(j) = \sum_{i=0}^{j} \binom{K}{i} (p_k)^i (1 - p_k)^{K-i}. \tag{A.32}$$

We obtain the upper bound in (3.21) by simply replacing (A.6) with (A.31) and following the same procedure as in the proof of Theorem 3.2. This concludes the proof of the upper bound in (3.21). Next, we prove the lower bound in (3.22). We know from (A.11) that

$$\overline{T}(\gamma) \geq \frac{\Lambda - t}{1+t}\left(\frac{E[l_1]t}{\Lambda-1} + \frac{K}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}\right). \tag{A.33}$$

To conclude the proof, we need to derive $E[l_1]$. Let $m \in [\Lambda]$ is the cache with highest population intensity (i.e, $m = \underset{x\in[\Lambda]}{\text{argmax}}\, p_x$), then $E[l_1]$ is lower bounded as

$$E[l_1] = \sum_{\mathbf{V}\in\mathcal{V}} P(\mathbf{V})\max(\mathbf{V}) \geq \sum_{\mathbf{V}\in\mathcal{V}} P(\mathbf{V})v_m = E[v_m] = Kp_m. \tag{A.34}$$

Finally, we obtain

$$\overline{T}(\gamma) \geq \frac{\Lambda - t}{1+t}\left(\frac{Kp_m t}{\Lambda-1} + \frac{K}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}\right), \tag{A.35}$$

which concludes the proof of Theorem 3.4.

A.6 Proof of Theorem 3.5

We know from (A.20) that $\overline{T}(\gamma) = \Theta\left(T_{min}\frac{E[l_1]\Lambda}{K}\right)$, thus, we first characterize the scaling laws of $E[l_1]$. Let μ_i and σ_i^2 be the expectation and variance of random variable v_i's (i.e., the number of users associated to the ith cache) respectively, then from [82, Proposition 1], we have

$$E[l_1] \leq \bar{\mu} + \sqrt{\frac{\Lambda-1}{\Lambda}\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i - \bar{\mu})^2)}, \tag{A.36}$$

where $\bar{\mu} = \frac{1}{\Lambda}\sum_{i=1}^{\Lambda}\mu_i$. Since the random variable v_i follows the binomial distribution, we have $\mu_i = Kp_i$, $\sigma_i^2 = K(1-p_i)p_i$, and $\bar{\mu} = \frac{1}{\Lambda}\sum_{i=1}^{\Lambda}Kp_i = \frac{K}{\Lambda}$. Consequently the upper bound on the $E[l_1]$ is given as

$$\begin{aligned}
E[l_1] &\leq \frac{K}{\Lambda} + \sqrt{\frac{\Lambda-1}{\Lambda}\sum_{i=1}^{\Lambda}\left(K(1-p_i)p_i + \left(Kp_i - \frac{K}{\Lambda}\right)^2\right)} \\
&= \frac{K}{\Lambda} + \sqrt{\Lambda-1}\sqrt{\frac{K}{\Lambda}\sum_{i=1}^{\Lambda}\left(p_i - p_i^2 + K\left(p_i^2 - \frac{2p_i}{\Lambda} + \frac{1}{\Lambda^2}\right)\right)} \\
&= \frac{K}{\Lambda} + \sqrt{\Lambda-1}\sqrt{\frac{K}{\Lambda}\sum_{i=1}^{\Lambda}\left(p_i - p_i^2 + Kp_i^2 - \frac{2Kp_i}{\Lambda} + \frac{K}{\Lambda^2}\right)}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{K}{\Lambda}+\sqrt{\Lambda-1}\sqrt{\frac{K}{\Lambda}\sum_{i=1}^{\Lambda}\left(p_i\left(1-\frac{2K}{\Lambda}\right)+p_i^2(K-1)+\frac{K}{\Lambda^2}\right)}\\
&= \frac{K}{\Lambda}+\sqrt{\Lambda-1}\sqrt{\frac{K}{\Lambda}\left(1-\frac{2K}{\Lambda}+\frac{K}{\Lambda}+\sum_{i=1}^{\Lambda}p_i^2(K-1)\right)}\\
&= \frac{K}{\Lambda}+\sqrt{\Lambda-1}\sqrt{\frac{K}{\Lambda}\left(1-\frac{K}{\Lambda}+(K-1)\sum_{i=1}^{\Lambda}p_i^2\right)}\\
&= \frac{K}{\Lambda}+\sqrt{\Lambda-1}\sqrt{\frac{K(\Lambda-K)}{\Lambda^2}+\frac{K(K-1)}{\Lambda}\sum_{i=1}^{\Lambda}p_i^2}\\
&= \frac{K}{\Lambda}+\sqrt{\Lambda-1}\sqrt{\frac{K(\Lambda-K)+K(K-1)\Lambda\sum_{i=1}^{\Lambda}p_i^2}{\Lambda^2}}\\
&= \frac{K}{\Lambda}+\sqrt{\Lambda-1}\frac{K}{\Lambda}\sqrt{\frac{\Lambda-K+(K-1)\Lambda\sum_{i=1}^{\Lambda}p_i^2}{K}}\\
&= \frac{K}{\Lambda}\left(1+\sqrt{\Lambda-1}\sqrt{\frac{\Lambda}{K}-1+\frac{K-1}{K}\Lambda\sum_{i=1}^{\Lambda}p_i^2}\right).
\end{aligned}
\tag{A.37}
$$

When the cache population intensities $\mathbf{p}$ follows the Zipf distribution, we have $\sum_{i=1}^{\Lambda} p_i^2 = \sum_{i=1}^{\Lambda}\frac{i^{-2\alpha}}{(H_\alpha(\Lambda))^2} = \frac{H_{2\alpha}(\Lambda)}{(H_\alpha(\Lambda))^2}$, where the normalization constant $H_\alpha(\Lambda)$ known as the generalized harmonic number [83] scales as

$$
H_\alpha(\Lambda)=\begin{cases}\Theta(1) & \alpha>1\\ \Theta(\log\Lambda) & \alpha=1\\ \Theta(\Lambda^{1-\alpha}) & \alpha<1.\end{cases}
\tag{A.38}
$$

Similarly, the scaling of $H_{2\alpha}(\Lambda)$ is given as

$$
H_{2\alpha}(\Lambda)=\begin{cases}\Theta(1) & \alpha>0.5\\ \Theta(\log\Lambda) & \alpha=0.5\\ \Theta(\Lambda^{1-2\alpha}) & \alpha<0.5.\end{cases}
\tag{A.39}
$$

Consequently, we have

$$
\sum_{i=1}^{\Lambda}p_i^2=\begin{cases}\Theta(1) & \alpha>1\\ \Theta\left(\frac{1}{(\log\Lambda)^2}\right) & \alpha=1\\ \Theta\left(\frac{1}{\Lambda^{2-2\alpha}}\right) & 0.5<\alpha<1\\ \Theta\left(\frac{\log\Lambda}{\Lambda^{2-2\alpha}}\right)=\Theta\left(\frac{\log\Lambda}{\Lambda}\right) & \alpha=0.5\\ \Theta\left(\frac{\Lambda^{1-2\alpha}}{\Lambda^{2-2\alpha}}\right)=\Theta\left(\frac{1}{\Lambda}\right) & \alpha<0.5.\end{cases}
\tag{A.40}
$$

From (A.37) and (A.40), we obtain

$$E[l_1] = \begin{cases} O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\Lambda}\right)\right) & \alpha > 1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\frac{\Lambda}{(\log\Lambda)^2}}\right)\right) & \alpha = 1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\frac{\Lambda}{\Lambda^{2-2\alpha}}}\right)\right) & 0.5<\alpha<1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\frac{\Lambda\log\Lambda}{\Lambda}}\right)\right) & \alpha = 0.5 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\frac{\Lambda}{\Lambda}}\right)\right) & \alpha < 0.5 \end{cases}$$

$$= \begin{cases} O\left(\frac{K}{\Lambda}(\Lambda)\right) & \alpha > 1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\frac{\Lambda}{(\log\Lambda)^2}}\right)\right) & \alpha = 1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\frac{1}{\Lambda^{1-2\alpha}}}\right)\right) & 0.5<\alpha<1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+\log\Lambda}\right)\right) & \alpha = 0.5 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda}\sqrt{\frac{\Lambda}{K}+1}\right)\right) & \alpha < 0.5 \end{cases}$$

$$= \begin{cases} O(K) & \alpha > 1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\frac{\Lambda^2}{K}+\frac{\Lambda^2}{(\log\Lambda)^2}}\right)\right) & \alpha = 1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\frac{\Lambda^2}{K}+\Lambda^{2\alpha}}\right)\right) & 0.5<\alpha<1 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\frac{\Lambda^2}{K}+\Lambda\log\Lambda}\right)\right) & \alpha = 0.5 \\ O\left(\frac{K}{\Lambda}\left(1+\sqrt{\Lambda+\frac{\Lambda^2}{K}}\right)\right) & \alpha < 0.5 \end{cases}$$

$$= \begin{cases} O(K) & \alpha > 1 \\ O\left(\frac{K}{\Lambda}+\sqrt{K+\frac{K^2}{(\log\Lambda)^2}}\right) & \alpha = 1 \\ O\left(\frac{K}{\Lambda}+\sqrt{K+\frac{K^2}{\Lambda^{2-2\alpha}}}\right) & 0.5<\alpha<1 \\ O\left(\frac{K}{\Lambda}+\sqrt{K+\frac{K^2\log\Lambda}{\Lambda}}\right) & \alpha = 0.5 \\ O\left(\frac{K}{\Lambda}+\sqrt{\frac{K^2}{\Lambda}+K}\right) & \alpha < 0.5. \end{cases}$$

Consequently, the $E[l_1]$ is upper bounded as

$$E[l_1] = \begin{cases} O(K) & \alpha > 1 \\ O\left(\sqrt{K+\frac{K^2}{(\log\Lambda)^2}}\right) & \alpha = 1 \\ O\left(\sqrt{K+\frac{K^2}{\Lambda^{2-2\alpha}}}\right) & 0.5 < \alpha < 1 \\ O\left(\sqrt{K+\frac{K^2\log\Lambda}{\Lambda}}\right) & \alpha = 0.5 \\ O\left(\frac{K}{\Lambda}+\sqrt{\frac{K^2}{\Lambda}+K}\right) & \alpha < 0.5. \end{cases} \tag{A.41}$$

From (A.20) and (A.41), the upper bound on $\overline{T}(\gamma)$ is given by

$$\overline{T}(\gamma) = \begin{cases} O\left(T_{min}\Lambda\right) & \alpha > 1 \\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K} + \frac{\Lambda^2}{(\log\Lambda)^2}}\right) & \alpha = 1 \\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K} + \Lambda^{2\alpha}}\right) & 0.5 < \alpha < 1 \\ O\left(T_{min}\sqrt{\frac{\Lambda^2}{K} + \Lambda\log\Lambda}\right) & \alpha = 0.5 \\ O\left(T_{min}\left(1 + \sqrt{\Lambda + \frac{\Lambda^2}{K}}\right)\right) & \alpha < 0.5. \end{cases} \tag{A.42}$$

Next, we characterize the lower bound on $\overline{T}(\gamma)$. We know that

$$E[l_1] \geq K\,\mathsf{max}(\mathbf{p}) = Kp_1 = \frac{K}{H_\alpha(\Lambda)}. \tag{A.43}$$

Thus, from (A.38) we have

$$E[l_1] = \begin{cases} \Omega\left(K\right) & \alpha > 1 \\ \Omega\left(\frac{K}{\log\Lambda}\right) & \alpha = 1 \\ \Omega\left(\frac{K}{\Lambda^{1-\alpha}}\right) & \alpha < 1. \end{cases} \tag{A.44}$$

From (A.20) and (A.44), the lower bound on $\overline{T}(\gamma)$ is given by

$$\overline{T}(\gamma) = \begin{cases} \Omega\left(T_{min}\Lambda\right) & \alpha > 1 \\ \Omega\left(T_{min}\frac{\Lambda}{\log\Lambda}\right) & \alpha = 1 \\ \Omega\left(T_{min}\Lambda^{\alpha}\right) & \alpha < 1. \end{cases} \tag{A.45}$$

Combining (A.42) with (A.45), allows us to directly conclude the proof of Theorem 3.5.

Appendix B

Proofs of Chapter 4

B.1 Proof of Theorem 4.1

Directly from the result in [76, Corollary 1.4] on the *Balanced Allocations* problem, we can conclude that for $h > 1$, the $E[l_1]$ asymptotically converges to

$$E[l_1] = \frac{\log\log\Lambda}{\log h} + \frac{K}{\Lambda} \pm \Theta(1). \tag{B.1}$$

Consequently combining (A.20) and (B.1), directly yields (4.3) which concludes the proof of Theorem 4.1.

B.2 Proof of Theorem 4.2

We start our proof by deriving the expected number of users in the most populous cache (i.e., $E[l_1]$). Recall that under the proximity-based load-balancing technique, each user can be associated to any cache *group* with equal probability $\frac{h}{\Lambda}$. Once a user is associated to a group, then this user will be associated to the least loaded cache from that group. Let l_1^h be the number of users that are associated to the most populous group of caches, then the number of users in the most populous cache is given by $l_1 = \left\lceil \frac{l_1^h}{h} \right\rceil$. Thus, we have

$$E[l_1] = \sum_{i=1}^{K} P[l_1^h = i] \left\lceil \frac{l_1^h}{h} \right\rceil, \tag{B.2}$$

where $P[l_1^h = i]$ is the probability that i users are associated to the most populous group of caches. Let $S_1 \subseteq [K]$ be the set of elements such that for each element $i \in S_1$, $\frac{i}{h}$ is integer. Then, we have

$$\begin{aligned} E[l_1] =& \sum_{i\in S_1} P[l_1^h = i]\frac{l_1^h}{h} + \sum_{i\in S_1/[K]} P[l_1^h = i] \left\lceil \frac{l_1^h}{h} \right\rceil \\ =& \sum_{i\in S_1} P[l_1^h = i]\frac{l_1^h}{h} + \sum_{i\in S_1/[K]} P[l_1^h = i]\frac{l_1^h}{h} + \sum_{i\in S_1/[K]} P[l_1^h = i] \left(\left\lceil \frac{l_1^h}{h} \right\rceil - \frac{l_1^h}{h} \right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i\in[K]} P[l_1^h = i]\frac{l_1^h}{h} + \sum_{i\in S_1/[K]} P[l_1^h = i]\left(\left\lceil\frac{l_1^h}{h}\right\rceil - \frac{l_1^h}{h}\right)\\
&= \frac{E[l_1^h]}{h} + \sum_{i\in S_1/[K]} P[l_1^h = i]\left(\left\lceil\frac{l_1^h}{h}\right\rceil - \frac{l_1^h}{h}\right). \qquad \text{(B.3)}
\end{aligned}
$$

It is straightforward to see that $0 < \sum_{i\in S_1/[K]} P[l_1^h = i]\left(\left\lceil\frac{l_1^h}{h}\right\rceil - \frac{l_1^h}{h}\right) < 1$, Therefore, $E[l_1]$ is bounded as

$$\frac{E[l_1^h]}{h} < E[l_1] < \frac{E[l_1^h]}{h} + 1, \qquad \text{(B.4)}$$

and we can conclude that

$$E[l_1] = \Theta\left(\frac{E[l_1^h]}{h}\right). \qquad \text{(B.5)}$$

Evaluating $E[l_1^h]$ from (A.26) by treating each group as a single cache, we conclude that

$$E[l_1] = \begin{cases} \Theta\left(\frac{\log\frac{\Lambda}{h}}{h\log\frac{\Lambda\log\frac{\Lambda}{h}}{hK}}\right) & \text{if } K \in \left[\frac{\Lambda/h}{\text{polylog}(\frac{\Lambda}{h})}, o\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right)\right]\\ \Theta\left(\frac{K}{\Lambda} + \sqrt{\frac{K\log(\frac{\Lambda}{h})}{h\Lambda}}\right) & \text{if } K = \Omega\left(\frac{\Lambda}{h}\log\frac{\Lambda}{h}\right). \end{cases} \qquad \text{(B.6)}$$

Finally combining (A.20) and (B.6), directly yields (4.5), and thus concludes the proof of Theorem 4.2.

B.3 Proof of Theorem 4.3

We proceed with the following lemma which is adopted and adapted here directly from the work of [84] on the *Balls into Bins problem.*

Lemma B.1 ([84, Theorem 1.3] - adaptation). *In a Λ-cell, K-user setting where each ball is associated to the least loaded cache among $h \geq 2$ caches, independently sampled from Λ caches whose population intensities vector is following the distribution $\mathbf{p}$, the tail of l_1 takes the form*

$$P[l_1 > \delta] = o\left(\frac{1}{\Lambda}\right) \qquad \text{(B.7)}$$

for

$$\delta = \frac{K}{\Lambda} + \log\log\Lambda + O(1), \qquad \text{(B.8)}$$

when $h = \Theta\left(\frac{\log\left(\frac{ab-1}{a-1}\right)}{\log\left(\frac{ab-1}{ab-b}\right)}\right)$, where $a = \frac{1}{\Lambda\min(\mathbf{p})}$ and $b = \max(\mathbf{p})\Lambda$.

Proof. The result comes directly from [84, Theorem 1.3]. □

With Lemma B.1 at hand, we get that

$$\begin{aligned}
E[l_1] &= \sum_{j=0}^{\delta-1} P[l_1 > j] + P[l_1 > \delta] + \sum_{j=\delta+1}^{K-1} P[l_1 > j] \\
&\overset{(a)}{\leq} \delta + o\left(\frac{1}{\Lambda}\right) + \sum_{j=\delta+1}^{K-1} P[l_1 > j] \\
&\overset{(b)}{\leq} \delta + o\left(\frac{1}{\Lambda}\right) + (K-\delta-1)o\left(\frac{1}{\Lambda}\right) \\
&= \delta\left(1 - o\left(\frac{1}{\Lambda}\right)\right) + o\left(\frac{K}{\Lambda}\right) = O(\delta) + o\left(\frac{K}{\Lambda}\right),
\end{aligned} \tag{B.9}$$

where in step (a), we use the fact that $P[l_1 > j]$ is at most 1 for $j = [0, 1, \cdots, \delta - 1]$ and in step (b), we use the fact that if $P[l_1 > \delta] = o(1)$ then $P[l_1 > j]$ is at most $o(1)$ for $j = [\delta + 1, \delta + 2, \cdots K - 1]$. Combining (B.8) and (B.9), we have

$$E[l_1] = O\left(\frac{K}{\Lambda} + \log\log\Lambda\right), \tag{B.10}$$

when $h = \Theta\left(\frac{\log\left(\frac{ab-1}{a-1}\right)}{\log\left(\frac{ab-1}{ab-b}\right)}\right)$. We now proceed to simplify h. When the cache population intensities $\mathbf{p}$ follows the Zipf distribution, we know that $\min(\mathbf{p}) = p_\Lambda = \frac{\Lambda^{-\alpha}}{H_\alpha(\Lambda)}$ and $\max(\mathbf{p}) = p_1 = \frac{1}{H_\alpha(\Lambda)}$. We get the scaling of h as

$$\begin{aligned}
h &= \Theta\left(\frac{\log\left(\frac{\frac{H_\alpha(\Lambda)}{\Lambda^{1-\alpha}}\frac{\Lambda}{H_\alpha(\Lambda)} - 1}{\frac{H_\alpha(\Lambda)}{\Lambda^{1-\alpha}} - 1}\right)}{\log\left(\frac{\frac{H_\alpha(\Lambda)}{\Lambda^{1-\alpha}}\frac{\Lambda}{H_\alpha(\Lambda)} - 1}{\frac{H_\alpha(\Lambda)}{\Lambda^{1-\alpha}}\frac{\Lambda}{H_\alpha(\Lambda)} - \frac{\Lambda}{H_\alpha(\Lambda)}}\right)}\right) \\
&= \Theta\left(\frac{\log\left(\frac{\Lambda(1-\Lambda^{-\alpha})}{H_\alpha(\Lambda)-\Lambda^{1-\alpha}}\right)}{\log\left(\frac{(1-\Lambda^{-\alpha})H_\alpha(\Lambda)}{H_\alpha(\Lambda)-\Lambda^{1-\alpha}}\right)}\right) \\
&= \Theta\left(\frac{\log\left(\frac{\Lambda(\Lambda^{\alpha}-1)}{\Lambda^{\alpha}H_\alpha(\Lambda)-\Lambda}\right)}{\log\left(\frac{(\Lambda^{\alpha}-1)H_\alpha(\Lambda)}{\Lambda^{\alpha}H_\alpha(\Lambda)-\Lambda}\right)}\right) \\
&= \Theta\left(\frac{\log\Lambda + \log\left(\frac{\Lambda^{\alpha}-1}{\Lambda^{\alpha}H_\alpha(\Lambda)-\Lambda}\right)}{\log\left(\frac{(\Lambda^{\alpha}-1)H_\alpha(\Lambda)}{\Lambda^{\alpha}H_\alpha(\Lambda)-\Lambda}\right)}\right).
\end{aligned} \tag{B.11}$$

Let $X = \frac{\Lambda^{\alpha}-1}{\Lambda^{\alpha}H_\alpha(\Lambda)-\Lambda}$, from (A.38), we get

$$X = \begin{cases} \Theta(1) & \alpha > 1 \\ \Theta\left(\frac{1}{\log\Lambda}\right) & \alpha = 1 \\ \Theta\left(\Lambda^{\alpha-1}\right) & \alpha < 1. \end{cases} \tag{B.12}$$

As for $\alpha > 1$, $H_\alpha(\Lambda) = \Theta(1)$, thus $\Lambda^\alpha H_\alpha(\Lambda) - \Lambda = \Theta(\Lambda^\alpha)$ and we get $X = \Theta\left(\frac{\Lambda^\alpha}{\Lambda^\alpha}\right)$. When $\alpha = 1$, we have $H_\alpha(\Lambda) = \Theta(\log\Lambda)$, thus $\Lambda H_\alpha(\Lambda) - \Lambda = \Theta(\Lambda\log\Lambda)$, which gives $X = \Theta\left(\frac{\Lambda}{\Lambda\log\Lambda}\right)$. When $\alpha < 1$, we have $H_\alpha(\Lambda) = \Theta(\Lambda^{1-\alpha})$, thus $\Lambda^\alpha H_\alpha(\Lambda) - \Lambda = \Theta(\Lambda)$ and we get $X = \Theta\left(\frac{\Lambda^\alpha}{\Lambda}\right)$. Consequently, we have

$$\begin{aligned} h &= \Theta\left(\frac{\log\Lambda + \log X}{\log(XH_\alpha(\Lambda))}\right) \overset{(a)}{=} \Theta(\log\Lambda + \log X) \\ &= \begin{cases} \Theta(\log\Lambda) & \alpha > 1 \\ \Theta\left(\log\Lambda + \log\left(\frac{1}{\log\Lambda}\right)\right) = \Theta(\log\Lambda) & \alpha = 1 \\ \Theta(\log\Lambda + \log(\Lambda^{\alpha-1})) = \Theta(\log\Lambda) & \alpha < 1, \end{cases} \end{aligned} \tag{B.13}$$

where in step (a), we use the fact that $XH_\alpha(\Lambda) = \Theta(1)$, which is obtain by combining (A.38) and (B.12). Finally, combining (A.20) with (B.10), allows us to directly conclude the proof of Theorem 4.3.

B.4 Proof of Theorem 4.6

We know from Theorem 3.3 that $E_{\mathbf{L}}[T_h(\mathbf{L},\xi)]$ corresponding to the delivery of the first part of each requested file is given as

$$E_{\mathbf{L}}[T_h(\mathbf{L},\xi)] = \begin{cases} \Theta\left(K\frac{1-\frac{\gamma}{\xi}}{1+\Lambda\frac{\gamma}{\xi}}\frac{\Lambda\log\Lambda}{K\log\frac{\Lambda\log\Lambda}{K}}\right) & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o(\Lambda\log\Lambda)\right] \\ \Theta\left(K\frac{1-\frac{\gamma}{\xi}}{1+\Lambda\frac{\gamma}{\xi}}\right) & \text{if } K = \Omega(\Lambda\log\Lambda). \end{cases} \tag{B.14}$$

Then, for the case when $K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o(\Lambda\log\Lambda)\right]$, the average delay $\overline{T}(\gamma,\gamma_u,\xi)$ scales as

$$\begin{aligned} \overline{T}(\gamma,\gamma_u,\xi) &= \Theta\left(\xi K\frac{\xi-\gamma}{\xi+\Lambda\gamma}\frac{\Lambda}{K}\frac{\log\Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)} + (1-\xi)K\frac{1-\xi-\gamma_u}{1-\xi+K\gamma_u}\right) \\ &\overset{(a)}{=} \Theta\left(\xi\frac{\xi-\gamma}{\gamma}\frac{\log\Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)} + (1-\xi)\frac{1-\xi-\gamma_u}{\gamma_u}\right), \end{aligned} \tag{B.15}$$

where in step (a), we used the fact that $\Lambda\gamma = \omega(\xi)$ and $K\gamma_u = \omega(1-\xi)$. Let $c_1 = \frac{\log\Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)}$

$$\begin{aligned} \overline{T}(\gamma,\gamma_u,\xi) &= \Theta\left(\frac{c_1\xi^2 - c_1\xi\gamma}{\gamma} + \frac{\xi^2+\xi(\gamma_u-2)+1-\gamma_u}{\gamma_u}\right) \\ &= \Theta\left(\frac{c_1\xi^2\gamma_u - c_1\xi\gamma\gamma_u + \xi^2\gamma + \xi(\gamma_u-2)\gamma + \gamma - \gamma\gamma_u}{\gamma\gamma_u}\right) \\ &= \Theta\left(\frac{\xi^2(c_1\gamma_u+\gamma) + \xi(\gamma_u\gamma - c_1\gamma\gamma_u - 2\gamma) + \gamma - \gamma\gamma_u}{\gamma\gamma_u}\right). \end{aligned} \tag{B.16}$$

Similarly, for the case $K = \Omega(\Lambda \log \Lambda)$, the average delay $\overline{T}(\gamma, \gamma_u, \xi)$ is bounded by

$$\begin{aligned}\overline{T}(\gamma, \gamma_u, \xi) &= \Theta\left(\xi K \frac{\xi - \gamma}{\xi + \Lambda\gamma} + (1-\xi) K \frac{1 - \xi - \gamma_u}{1 - \xi + K\gamma_u}\right) \\ &\overset{(b)}{=} \Theta\left(\xi \frac{\xi - \gamma}{\gamma}\frac{K}{\Lambda} + (1-\xi)\frac{1 - \xi - \gamma_u}{\gamma_u}\right). \end{aligned} \tag{B.17}$$

where in step (b), we used the fact that $\Lambda\gamma = \omega(\xi)$ and $K\gamma_u = \omega(1-\xi)$. Let $c_2 = \frac{K}{\Lambda}$, then we have

$$\overline{T}(\gamma, \gamma_u, \xi) = \Theta\left(\frac{\xi^2(c_2\gamma_u + \gamma)}{\gamma\gamma_u} + \frac{\xi(-c_2\gamma\gamma_u + \gamma_u\gamma - 2\gamma) + \gamma - \gamma\gamma_u}{\gamma\gamma_u}\right). \tag{B.18}$$

This concludes the proof of Theorem 4.6.

B.5 Proof of Lemma 4.1

The order-optimal partition parameter $\hat{\xi}$ is the solution to the following optimization problem

$$\min_{\xi} \overline{T}(\gamma, \gamma_u, \xi) \tag{B.19a}$$

subject to

$$\gamma \leq \xi \leq 1 - \gamma_u. \tag{B.19b}$$

We use the Lagrangian method to solve the problem in (B.19). We start with the first case when $K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o(\Lambda \log \Lambda)\right]$, using (B.16), the corresponding Lagrange function $\mathcal{Y}(\xi, \delta, \sigma)$ is

$$\begin{aligned}\mathcal{Y}(\xi, \delta, \sigma) = &\frac{\xi^2(c_1\gamma_u + \gamma) + \xi(-c_1\gamma\gamma_u + \gamma_u\gamma - 2\gamma)}{\gamma\gamma_u} \\ &+ \frac{\gamma - \gamma\gamma_u}{\gamma\gamma_u} + \delta(\xi - 1 + \gamma_u) - \sigma(\xi - \gamma)\end{aligned} \tag{B.20}$$

where $\sigma, \delta \in \mathbb{R}$ and $c_1 = \frac{\log \Lambda}{log\left(\frac{\Lambda}{K}\log\Lambda\right)}$. The Karush-Kuhn-Tucker (KKT) conditions for (B.19) are then given by

$$\frac{\partial \mathcal{Y}(\hat{\xi}, \hat{\delta}, \hat{\sigma})}{\partial \xi} = 0, \tag{B.21}$$

$$\hat{\sigma}(\hat{\xi} - \gamma) = 0, \tag{B.22}$$

$$\hat{\delta}\left(\hat{\xi} - 1 + \gamma_u\right) = 0, \tag{B.23}$$

$$\hat{\delta} \geq 0, \tag{B.24}$$

$$\hat{\sigma} \geq 0, \tag{B.25}$$

where $\hat{\xi}, \hat{\delta}$, and $\hat{\sigma}$ represent the optimized values. Then from (B.21), we have

$$\frac{\partial \mathcal{Y}(\hat{\xi}, \hat{\delta}, \sigma)}{\partial \xi} = \frac{2\hat{\xi}(c_1\gamma_u + \gamma)}{\gamma\gamma_u} + \frac{(-c_1\gamma\gamma_u + \gamma_u\gamma - 2\gamma)}{\gamma\gamma_u} + \hat{\delta} - \hat{\sigma} = 0. \tag{B.26}$$

We obtain the solution by dividing the operating domain of the partition parameter into the following three regimes:

- Regime I: $\gamma < \hat{\xi} < 1 - \gamma_u$
- Regime II: $\hat{\xi} = 1 - \gamma_u$
- Regime III: $\hat{\xi} = \gamma$.

First, for Regime I, when $\gamma < \hat{\xi} < 1 - \gamma_u$, we have $\hat{\delta} = \hat{\sigma} = 0$ and from (B.26), we obtain

$$\hat{\xi} = \frac{(c_1\gamma\gamma_u - \gamma_u\gamma + 2\gamma)}{2(c_1\gamma_u + \gamma)} \tag{B.27}$$

Next, for Regime II, when $\hat{\xi} = 1 - \gamma_u$, we have $\hat{\sigma} = 0$ and from (B.26), we obtain

$$\begin{aligned}\hat{\delta} &= -\frac{2(1-\gamma_u)(c_1\gamma_u + \gamma) + (-c_1\gamma\gamma_u + \gamma_u\gamma - 2\gamma)}{\gamma\gamma_u} \\ &= -\frac{2c_1\gamma_u + 2\gamma - 2c_1\gamma_u^2 - 2\gamma\gamma_u - c_1\gamma\gamma_u + \gamma_u\gamma - 2\gamma}{\gamma\gamma_u} \\ &= -\frac{2c_1 - 2c_1\gamma_u - \gamma - c_1\gamma}{\gamma} \\ &= 1 - \frac{c_1(2 - 2\gamma_u - \gamma)}{\gamma}.\end{aligned} \tag{B.28}$$

We know from (B.24) that $\hat{\delta} \geq 0$, thus $\hat{\xi} = 1 - \gamma_u$ is a feasible solution only if

$$\begin{aligned} c_1(2 - 2\gamma_u - \gamma) &\leq \gamma \\ -\gamma_u - \gamma &\leq \frac{\gamma}{c_1} - 2 + \gamma_u \\ \gamma_u + \gamma &\geq 2 - \frac{\gamma}{c_1} - \gamma_u. \end{aligned} \tag{B.29}$$

However under our assumption that $\gamma_u + \gamma \leq 1$ it not possible as we know that when $K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda \log \Lambda\right)\right]$, $c_1 > 1$. Thus $\hat{\xi} = 1 - \gamma_u$ is not a feasible solution as it does not satisfy (B.24).

Finally, for Regime III, when $\hat{\xi} = \gamma$, we have $\hat{\delta} = 0$ and from (B.26), we obtain

$$\begin{aligned}\hat{\sigma} &= \frac{2\gamma(c_1\gamma_u + \gamma) - c_1\gamma\gamma_u + \gamma_u\gamma - 2\gamma}{\gamma\gamma_u} \\ &= \frac{2(c_1\gamma_u + \gamma) - c_1\gamma_u + \gamma_u - 2}{\gamma_u} \\ &= \frac{c_1\gamma_u + 2\gamma + \gamma_u - 2}{\gamma_u}. \end{aligned} \tag{B.30}$$

We know from (B.25) that $\hat{\sigma} \geq 0$, thus $\hat{\xi} = \gamma$ is a feasible solution only if $c_1\gamma_u + 2\gamma + \gamma_u - 2 \geq 0$. Finally, (B.27) and (B.30) concludes the proof of (4.36) for the case of $K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda \log \Lambda\right)\right]$.

Next, we solve the problem (B.19) for the case of $K = \Omega\left(\Lambda \log \Lambda\right)$. We can see that if we replace c_1 with c_2 in (B.16), we obtain (B.18), which is the average delay for the case of $K = \Omega\left(\Lambda \log \Lambda\right)$. Since both c_2 and c_1 are constants and does not depend on ξ, the optimal solution for the case of $K = \Omega\left(\Lambda \log \Lambda\right)$ is obtained using the same approach by simply replacing the c_1 with c_2. This concludes the proof of Lemma 4.1.

B.6 Proof of Theorem 4.7

We begin with the case when $\gamma_u \geq \frac{2-2\gamma}{1+c}$, from (4.36), we have $\hat{\xi} = \gamma$. Then, inserting $\hat{\xi}$ into (4.34) yields

$$\begin{aligned}\overline{T}\left(\gamma, \gamma_u\right) &= \overline{T}\left(\gamma, \gamma_u, \hat{\xi}\right) \\ &= \Theta\left(\frac{\gamma^2(c\gamma_u + \gamma) + \gamma(-c\gamma\gamma_u + \gamma_u\gamma - 2\gamma) + \gamma - \gamma\gamma_u}{\gamma\gamma_u}\right) \\ &= \Theta\left(\frac{c\gamma_u\gamma + \gamma^2 - c\gamma\gamma_u + \gamma_u\gamma - 2\gamma + 1 - \gamma_u}{\gamma_u}\right) \\ &= \Theta\left((1-\gamma)\frac{1 - \gamma_u - \gamma}{\gamma_u}\right). \end{aligned} \tag{B.31}$$

Next for the case when $\gamma_u < \frac{2-2\gamma}{1+c}$, from (4.36), we have $\hat{\xi} = \frac{c\gamma\gamma_u - \gamma_u\gamma + 2\gamma}{2(c\gamma_u + \gamma)}$. Then, inserting $\hat{\xi}$ into (4.34) yields

$$\begin{aligned}\overline{T}\left(\gamma, \gamma_u\right) &= \overline{T}\left(\gamma, \gamma_u, \hat{\xi}\right) \\ &= \Theta\left(\frac{\frac{(c\gamma\gamma_u - \gamma_u\gamma + 2\gamma)^2}{4(c\gamma_u + \gamma)} - \frac{(c\gamma\gamma_u - \gamma_u\gamma + 2\gamma)^2}{2(c\gamma_u + \gamma)} + \gamma - \gamma\gamma_u}{\gamma\gamma_u}\right) \\ &= \Theta\left(-\frac{(c\gamma\gamma_u - \gamma_u\gamma + 2\gamma)^2}{4(c\gamma_u + \gamma)\gamma\gamma_u} + \frac{1 - \gamma_u}{\gamma_u}\right) \\ &= \Theta\left(\frac{2c\gamma\gamma_u - c^2\gamma_u\gamma - \gamma\gamma_u}{4c\gamma_u + 4\gamma} + \frac{c(1 - \gamma - \gamma_u)}{c\gamma_u + \gamma}\right)\end{aligned}$$

$$
\begin{aligned}
&= \Theta\left(-\frac{\gamma\gamma_u(-2c+c^2+1)}{4c\gamma_u+4\gamma} + \frac{c(1-\gamma-\gamma_u)}{c\gamma_u+\gamma}\right) \\
&= \Theta\left(\frac{c(1-\gamma-\gamma_u)}{c\gamma_u+\gamma} - \frac{\gamma\gamma_u(c-1)^2}{4c\gamma_u+4\gamma}\right). \qquad \text{(B.32)}
\end{aligned}
$$

This concludes the proof of Theorem 4.7.

B.7 Proof of Lemma 4.2

We now from [55, Equation 5, 6] and (4.30) that for a fixed partition parameter ξ, the worst-case delivery time for a uniform user-to-cell association is

$$
T_{min}(\gamma,\gamma_u,\xi) = \Theta\left(\xi K \frac{1-\frac{\gamma}{\xi}}{1+\Lambda\frac{\gamma}{\xi}} + (1-\xi)\frac{K(1-\frac{\gamma_u}{1-\xi})}{1+\frac{K\gamma_u}{1-\xi}}\right). \qquad \text{(B.33)}
$$

From (B.17), we can see that $T_{min}(\gamma,\gamma_u,\xi)$ is exactly equal to the worst-case average delivery time for the case of $K = \Omega(\Lambda \log \Lambda)$. Therefore, the proof of Lemma 4.2 follows directly from the proof of Lemma 4.1 and proof of Theorem 4.7 for the case of $K = \Omega(\Lambda \log \Lambda)$.

Appendix C

Proofs of Chapter 5

C.1 Proof of Theorem 5.1

Exploiting the fact that in (5.3), both $\binom{\Lambda-\lambda}{t}$ and $E[l_\lambda]$ are non-increasing with λ, the average delay $\overline{T}(\mathbf{G})$ is bounded by

$$\overline{T}(\mathbf{G}) \leq \sum_{\lambda=1}^{\Lambda-t} E[l_1]\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \overset{(a)}{=} E[l_1]\frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} = E[l_1]\frac{\Lambda-t}{1+t}, \tag{C.1}$$

and

$$\begin{aligned}\overline{T}(\mathbf{G}) &\overset{(b)}{\geq} \frac{E[l_1]\binom{\Lambda-1}{t}+\sum_{\lambda=2}^{\Lambda-t}\frac{K_{\mathbf{p}}-E[l_1]}{\Lambda-1}\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}\\
&\overset{(c)}{=} E[l_1]\frac{\binom{\Lambda-1}{t}}{\binom{\Lambda}{t}} + \frac{K_{\mathbf{p}}-E[l_1]}{\Lambda-1}\frac{\binom{\Lambda-1}{t+1}}{\binom{\Lambda}{t}}\\
&= E[l_1]\frac{\Lambda-t}{\Lambda} + \frac{K_{\mathbf{p}}-E[l_1]}{\Lambda-1}\frac{(\Lambda-t)(\Lambda-t-1)}{(1+t)\Lambda}\\
&= (\Lambda-t)\left(\frac{E[l_1]}{\Lambda} + \frac{K_{\mathbf{p}}-E[l_1]}{\Lambda-1}\frac{\Lambda-t-1}{(1+t)\Lambda}\right)\\
&= \frac{\Lambda-t}{1+t}\left(\frac{E[l_1]t}{\Lambda-1} + \frac{K_{\mathbf{p}}}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}\right),\end{aligned} \tag{C.2}$$

where in steps (a) and (c), we inherit the the column-sum property of Pascal's triangle yielding $\sum_{k=0}^{n}\binom{k}{t} = \binom{n+1}{t+1}$, while in step (b), we have[1] $K_{\mathbf{p}} = \sum_{\lambda=1}^{\Lambda} E[l_\lambda]$, and the fact that uniformity in $\mathbf{L}$ leads to the minimum $\overline{T}(\mathbf{G})$.

Next, to complete the proof, we proceed to derive the expected number of active users that are storing the content of the most loaded cache state (i.e.,

[1]It straightforward to see that $\sum\limits_{\lambda\in[\Lambda]} E[l_\lambda] = \sum\limits_{i=0}^{K}\sum\limits_{\mathbf{L}\in\mathcal{L}:i=\sum\limits_{j\in[\Lambda]} l_j}\sum\limits_{\lambda\in[\Lambda]} l_\lambda P(\mathbf{L}) = \sum\limits_{i=0}^{K}\sum\limits_{\mathbf{L}\in\mathcal{L}:i=\sum\limits_{j\in[\Lambda]} l_j} iP(\mathbf{L}) = K_{\mathbf{p}}$.

$E[l_1]$), which is given by

$$E[l_1] = \sum_{x=0}^{A-1} P[l_1 > x] = \sum_{x=0}^{A-1} \left(1 - P[l_1 \leq x]\right), \tag{C.3}$$

where $A = \max\left(\{|\mathbf{G}_\lambda|\}_{\lambda=1}^{\Lambda}\right)$, $\mathbf{G}_\lambda$ is the set of users caching the content of cache state λ, and $P[l_1 \leq x]$ is the probability that number of active users storing the content of the most loaded cache state are less than or equal to x. From [80, Proposition 3], we have

$$P[l_1 \leq x] \geq \max\left(0, 1 - \Lambda + \sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)\right) \tag{C.4}$$

and

$$P[l_1 \leq x] \leq \frac{\sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)}{\Lambda}, \tag{C.5}$$

where $F_{v_\lambda}(x)$ is the probability that no more than x users that are caching the content of cache state $\lambda \in [\Lambda]$ are active (i.e., $P[v_\lambda \leq x]$). Then $E[l_1]$ is bounded by

$$E[l_1] \leq A - \sum_{x=0}^{A-1} \max\left(0, 1 - \Lambda + \sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)\right) \tag{C.6}$$

and

$$E[l_1] \geq A - \sum_{x=0}^{A-1} \frac{\sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)}{\Lambda}. \tag{C.7}$$

For each cache state $\lambda \in [\Lambda]$, the corresponding random variable v_λ follows the Poisson binomial distribution. Using Hoeffding's inequalities [85, Theorem 2.1], $F_{v_i}(x)$ is bounded by

$$F_{v_\lambda}(x) \geq \begin{cases} 0 & \text{for } 0 \leq x \leq \mu_\lambda - 1 \\ F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{for } \mu_\lambda \leq x \leq |\mathbf{G}_\lambda| \end{cases} \tag{C.8}$$

and

$$F_{v_\lambda}(x) \leq \begin{cases} F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{for } 0 \leq x \leq \mu_\lambda - 1 \\ 1. & \text{for } x > \mu_\lambda - 1, \end{cases} \tag{C.9}$$

where $\mu_\lambda = \sum_{k \in \mathbf{G}_\lambda} p_k$ is the expected number active users that are storing the content of cache state $\lambda \in [\Lambda]$ and $F_{bin}(n, q, x) = \sum_{i=0}^{x} \binom{n}{i} q^i (1-q)^{n-i}$ is the Binomial cumulative distribution function.

Finally, the upper bound in (5.6) can be obtained from (C.1), (C.6), and (C.8); and the lower bound in (5.7) can be obtained from (C.2), (C.7), and (C.9).

C.2 Proof of Theorem 5.2

From (C.1) and (C.2), we have,

$$\overline{T}(\mathbf{G}) = O\left(E[l_1]\frac{\Lambda - t}{1+t}\right), \tag{C.10}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{\Lambda - t}{1+t}\frac{E[l_1]t}{\Lambda - 1}\right). \tag{C.11}$$

As $\frac{t}{\Lambda-1} \approx \gamma$ is a constant, we get the exact scaling law of $\overline{T}(\mathbf{G})$, which is given by

$$\overline{T}(\mathbf{G}) = \Theta\left(E[l_1]\frac{\Lambda - t}{1+t}\right). \tag{C.12}$$

We know from [82, Proposition 1] that $E[l_1]$ is bounded by

$$\frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda \leq E[l_1] \leq \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda + \sqrt{\frac{\Lambda-1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\left(\sigma_\lambda^2 + \left(\mu_\lambda - \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda\right)^2\right)} \tag{C.13}$$

where $\mu_\lambda = \sum_{k\in G_\lambda} p_k$ and $\sigma_\lambda^2 = \sum_{k\in G_\lambda} p_k(1-p_k)$ are the mean and the variance of the number of active users that are caching the content of cache state $\lambda \in [\Lambda]$ respectively. After defining a new parameter $\mu = \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda$, we have

$$\overline{T}(\mathbf{G}) = O\left(\left(\mu + \sqrt{\sum_{i=1}^{\Lambda}[\sigma_i^2 + (\mu_i - \mu)^2]}\right)\frac{\Lambda - t}{1+t}\right), \tag{C.14}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\mu\frac{\Lambda - t}{1+t}\right). \tag{C.15}$$

This concludes the proof of Theorem 5.2.

C.3 Proof of Theorem 5.3

We start our proof by deriving the expected number of active users that are storing the content of the most loaded cache state (i.e., $E[l_1]$). Assuming that each user independently requests a content with probability p, the probability that no more than x out of I users are active and storing the content of cache state $\lambda \in [\Lambda]$ is given by

$$F_{v_\lambda}(x) = \sum_{i=0}^{x}\binom{I}{i}p^i(1-p)^{I-i}. \tag{C.16}$$

Then, the probability that l_1 (i.e., $\max(\mathbf{V})$, the maximum number of active users among all caches) is less than or equal to j, is equal to the probability of the event $v_\lambda \leq j$, $\forall\ \lambda \in [\Lambda]$, and given by

$$P[l_1 \leq x] = \prod_{\lambda=1}^{\Lambda} F_{v_\lambda}(x) = \left(\sum_{i=0}^{x} \binom{I}{i} p^i (1-p)^{I-i} \right)^{\Lambda}. \tag{C.17}$$

Now, we can characterize $E[l_1]$ as follows

$$E[l_1] = \sum_{x=0}^{I-1} (1 - P[l_1 \leq x]) = I - \sum_{x=0}^{I-1} \left(\sum_{i=0}^{x} \binom{I}{i} p^i (1-p)^{I-i} \right)^{\Lambda}. \tag{C.18}$$

Finally, we obtain the upper and lower bounds in Theorem 5.3 by combining (C.1) and (C.2) with (C.18), respectively.

C.4 Proof of Theorem 5.4

To prove Theorem 5.4, we will follow a similar approach as in [81]. For each cache state $\lambda \in [\Lambda]$, we denote Y_λ to be an indicator random variable, which is equal to 1 if $v_\lambda \geq k_\alpha$, and it is equal to 0 otherwise. It immediately follows that $E[Y_\lambda] = P[v_\lambda \geq k_\alpha]$, $\forall \lambda \in [\Lambda]$. Let $Y = \sum_{\lambda=1}^{\Lambda} Y_\lambda$ be the sum of the indicators over all cache states. Then, we have

$$E[Y] = E\left[\sum_{\lambda=1}^{\Lambda} Y_\lambda\right] = \sum_{\lambda=1}^{\Lambda} E[Y_\lambda] = \Lambda P[v_\lambda \geq k_\alpha]. \tag{C.19}$$

From [81, Section 2], we inherit the following properties that are drawn from the outcomes of Markov's inequality and Chebyshev's inequality

$$P[Y = 0] = \begin{cases} 1 - o(1) & \text{if } \log(E[Y]) \to -\infty \\ o(1) & \text{if } \log(E[Y]) \to \infty. \end{cases} \tag{C.20}$$

Consequently, the probability that there exists at least one cache state λ for which the number of active users v_λ is at least k_α is given by

$$P[Y \geq 1] = \begin{cases} o(1) & \text{if } \log(E[Y]) \to -\infty \\ 1 - o(1) & \text{if } \log(E[Y]) \to \infty. \end{cases} \tag{C.21}$$

We now proceed with the following results which are crucial for the derivation of the asymptotics of $E[l_1]$.

Lemma C.1 ([81, Lemma 2] - adaptation). *For a positive constant c, if $Ip + 1 \leq x \leq (\log I)^c$, then*

$$P[v_\lambda \geq x] = e^{x(\log Ip - \log x + 1) - Ip + O\left(log^{(2)} I\right)} \tag{C.22}$$

and if $x = Ip + o\left((p(1-p)I)^{\frac{2}{3}}\right)$ and $z = \frac{x - Ip}{\sqrt{p(1-p)I}}$ tends to infinity, then

$$P[v_\lambda \geq x] = e^{-\frac{z^2}{2} - \log z - \log\sqrt{2\pi} + o(1)} \tag{C.23}$$

Proof. The result comes directly from [81, Lemma 2]. □

Lemma C.2. *In a K-user Λ-cache state setting where each user requests a content with probability p, the probability that the maximum number of active users among all caches is less than or equal to k_α, takes the form*

$$P[l_1 \geq k_\alpha] = \begin{cases} o(1) & \text{if} \quad \alpha > 1 \\ 1 - o(1) & \text{if} \quad 0 < \alpha < 1, \end{cases} \tag{C.24}$$

for

$$k_\alpha = \begin{cases} Ip + \sqrt{2\alpha Ip(1-p)\log(\Lambda)} & \text{if } Ip = \omega\left((\log\Lambda)^3\right) \\ \left(1 + \alpha\sqrt{\frac{2\log\Lambda}{Ip}}\right) Ip & \text{if } Ip \in [\omega(\log\Lambda), O(\text{polylog}\,\Lambda)] \\ (\alpha + e - 1)\, Ip & \text{if } Ip = \Theta(\log\Lambda) \\ \frac{\log\Lambda}{\log\frac{\log\Lambda}{Ip}}\left(1 + \alpha\frac{\log^{(2)}\frac{\log\Lambda}{Ip}}{\log\frac{\log\Lambda}{Ip}}\right) & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log\Lambda)\right]. \end{cases} \tag{C.25}$$

Proof. We begin the proof for the case of $Ip = \omega\left((\log(\Lambda))^3\right)$. Let $k_\alpha = Ip + \sqrt{2\alpha Ip(1-p)\log(\Lambda)}$, then from (C.19) and (C.23), we have

$$\begin{aligned} \log(E[Y]) &= \log\Lambda - \frac{z^2}{2} - \log z - \log\sqrt{2\pi} + o(1) \\ &= \log\Lambda\left(1 - \alpha - \frac{\log 2\alpha + \log^{(2)}\Lambda}{2\log\Lambda}\right) - \log\sqrt{2\pi} + o(1) \end{aligned} \tag{C.26}$$

Using (C.21), we conclude the proof for this case as for $\Lambda \to \infty$, we have

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases} \tag{C.27}$$

Next, we proceed with the case of $Ip \in [\omega(\log\Lambda), O(\text{polylog}(\Lambda))]$. We first define $g \triangleq O(\text{polylog}(\Lambda))$. Then, assuming that $k_\alpha = \left(1 + \alpha\sqrt{\frac{2}{g}}\right) Ip$ and $Ip = g\log(\Lambda)$, from (C.19) and (C.22), we have

$$\begin{aligned} \log(E[Y]) &= \log\Lambda + k_\alpha(\log Ip - \log k_\alpha + 1) - Ip + O\left(\log^{(2)} I\right) \\ &= \log\Lambda - k_\alpha \log\left(1 + \alpha\sqrt{\frac{2}{g}}\right) + k_\alpha - Ip + O\left(\log^{(2)} I\right) \\ &\overset{(a)}{=} \log\Lambda - k_\alpha\alpha\sqrt{\frac{2}{g}}\left(1 - \alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right) + \alpha Ip\sqrt{\frac{2}{g}} + O\left(\log^{(2)} I\right) \\ &= \log\Lambda - \left(1 + \alpha\sqrt{\frac{2}{g}}\right) Ip\alpha\sqrt{\frac{2}{g}}\left(1 - \alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right) + \alpha Ip\sqrt{\frac{2}{g}} + O\left(\log^{(2)} I\right) \\ &= \log\Lambda - \left(\alpha\sqrt{\frac{2}{g}} + \alpha^2\frac{2}{g}\right) Ip\left(1 - \alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right) + \alpha Ip\sqrt{\frac{2}{g}} + O\left(\log^{(2)} I\right) \end{aligned}$$

$$= \log(\Lambda) - \left(\alpha\sqrt{\frac{2}{g}} + \alpha^2\frac{2}{g}\right) g\log(\Lambda)\left(1 - \alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right)$$
$$+ g\log(\Lambda)\alpha\sqrt{\frac{2}{g}} + O\left(\log^{(2)} I\right)$$
$$= \log\Lambda - \left(\alpha\sqrt{2g} + 2\alpha^2\right)\log(\Lambda)\left(1 - \alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right)$$
$$+ \log(\Lambda)\alpha\sqrt{2g} + O\left(\log^{(2)} I\right)$$
$$= \log\Lambda\left(1 - \left(\alpha\sqrt{2g} + 2\alpha^2\right)\left(1 - \alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right)\right)$$
$$+ \log(\Lambda)\alpha\sqrt{2g} + O\left(\log^{(2)} I\right)$$
$$= \log(\Lambda)\left(1 - \left(\alpha\sqrt{2g} - \alpha^2 + o\left(2\alpha^2\right)\right) - \left(2\alpha^2 - \alpha^3\sqrt{\frac{2}{g}} + o\left(\alpha^3\sqrt{\frac{8}{g}}\right)\right) + \alpha\sqrt{2g}\right)$$
$$+ O\left(\log^{(2)} I\right)$$
$$= \log(\Lambda)\left(1 - \alpha\sqrt{2g} + \alpha^2 - o\left(2\alpha^2\right) - 2\alpha^2 + \alpha^3\sqrt{\frac{2}{g}} - o\left(\alpha^3\sqrt{\frac{8}{g}}\right)\right)$$
$$+ \log(\Lambda)\alpha\sqrt{2g} + O\left(\log^{(2)} I\right)$$
$$= \log\Lambda\left(1 - \alpha^2 - o\left(2\alpha^2\right) + \alpha^3\sqrt{\frac{2}{g}} - o\left(\alpha^3\sqrt{\frac{8}{g}}\right) + O\left(\frac{\log^{(2)} I}{\log\Lambda}\right)\right)$$
$$= \log\Lambda\left(1 - \alpha^2(1 + o(1)) + ((1 - o(1))\,\alpha^3\sqrt{\frac{2}{g}} + O\left(\frac{\log^{(2)} I}{\log\Lambda}\right)\right) \tag{C.28}$$

where in step (a), we used the Maclaurin series expansion of the logarithm function, i.e., $\log(1 + x) = x - 0.5x^2 + o(x^2)$. Using (C.21), we conclude the proof for this case as for $\Lambda \to \infty$, $\log(E[Y])$ converges to $(1-\alpha^2)\log\Lambda$, and we obtain

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases} \tag{C.29}$$

Now, we proceed with the case of $Ip = \Theta(\log\Lambda)$. Assuming that $k_\alpha = (\alpha + e - 1)\, Ip$ and $Ip = \log\Lambda$, from (C.19) and (C.22), we obtain

$$\begin{aligned}\log(E[Y]) &= \log\Lambda + k_\alpha\left(\log Ip - \log k_\alpha + 1\right) - Ip + O\left(\log^{(2)} I\right)\\ &= k_\alpha\left(\log^{(2)}\Lambda - \log^{(2)}\Lambda - \log(\alpha + e - 1) + 1\right) + O\left(log^{(2)} I\right)\\ &= k_\alpha\left(1 - \log(\alpha + e - 1)\right) + O\left(log^{(2)} I\right)\\ &= \log\Lambda\left((\alpha + e - 1)\left(1 - \log(\alpha + e - 1)\right) + O\left(\frac{log^{(3)}\Lambda}{p\log\Lambda}\right)\right)\end{aligned} \tag{C.30}$$

Using (C.21), we conclude the proof for this case as for $\Lambda \to \infty$, we have

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases} \tag{C.31}$$

Finally, we consider the case of $Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log(\Lambda))\right]$. We first define $g \triangleq O\left(\text{polylog}(\Lambda)\right)$. Then, assuming that $k_\alpha = \frac{\log\Lambda}{\log g}\left(1 + \alpha\frac{\log^{(2)} g}{\log g}\right)$ and $Ip = \frac{\log(\Lambda)}{g}$, from (C.19) and (C.22), we obtain

$$\begin{aligned}
&\log(E[Y]) = \log\Lambda + k_\alpha\left(\log(Ip) - \log(k_\alpha) + 1\right) - Ip + O\left(\log^{(2)} I\right) \\
&= \log\Lambda + k_\alpha\left(\log^{(2)}\Lambda - \log g - \log^{(2)}\Lambda + \log^{(2)} g - \log\left(1 + \alpha\frac{\log^{(2)} g}{\log g}\right) + 1\right) \\
&- \frac{\log(\Lambda)}{g} + O\left(\log^{(2)} I\right) \\
&\overset{(a)}{=} \log\Lambda + k_\alpha\left(1 - \log g + \log^{(2)} g - \left(\alpha\frac{\log^{(2)} g}{\log g} - 0.5\left(\alpha\frac{\log^{(2)} g}{\log g}\right)^2 + o\left(\left(\alpha\frac{\log^{(2)} g}{\log g}\right)^2\right)\right)\right) \\
&- \frac{\log(\Lambda)}{g} + O\left(\log^{(2)} I\right) \\
&= \frac{\log\Lambda\log^{(2)} g}{\log g}\left(1 - \alpha + \alpha\frac{\log^{(2)} g}{\log g} + \alpha^2\frac{\log^{(2)} g}{(\log g)^2}\left(-0.5 + 0.5\left(\alpha\frac{\log^{(2)} g}{\log g}\right)\right.\right. \\
&\left.\left.-o\left(\alpha\frac{\log^{(2)} g}{\log g}\right) - o(1)\right) + \frac{1}{\log^{(2)} g} - \frac{\log g}{g\log^{(2)} g} + O\left(\frac{\log^{(2)} I\log g}{\log\Lambda\log^{(2)} g}\right)\right),
\end{aligned} \tag{C.32}$$

where in step (a), we used the Maclaurin series expansion of the logarithm function, i.e., $\log(1+x) = x - 0.5x^2 + o(x^2)$. Using (C.21), we conclude the proof for this case as for $\Lambda \to \infty$, $\log(E[Y])$ converges to $\frac{\log\Lambda\log^{(2)} g}{\log g}(1-\alpha)$, and we obtain

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases} \tag{C.33}$$

This concludes the proof of Lemma C.2. □

With Lemma C.2 at hand, we proceed to characterize $E[l_1]$. Let us first consider the case of $\alpha > 1$, for which we have

$$\begin{aligned}
E[l_1] &= \sum_{j=1}^{k_\alpha - 1} P[l_1 \geq j] + P[l_1 \geq k_\alpha] + \sum_{j=k_\alpha+1}^{I} P[l_1 \geq j] \\
&\overset{(a)}{\leq} k_\alpha - 1 + o(1) + (I - k_\alpha)o(1) = O\left(k_\alpha\right),
\end{aligned} \tag{C.34}$$

where in step (a), we use the fact that $P[l_1 \geq j]$ is at most 1 for $j = [1, \cdots k_\alpha - 1]$, and if $P[l_1 \geq k_\alpha] = o(1)$ then $P[l_1 \geq j]$ is at most $o(1)$ for $j = [k_\alpha + 1, \cdots I]$.

Similarly, for $0 < \alpha < 1$, we have

$$\begin{aligned} E[l_1] &= \sum_{j=1}^{k_\alpha - 1} P[l_1 \geq j] + P[l_1 > k_\alpha] + \sum_{j=k_\alpha+1}^{I} P[l_1 \geq j] \\ &\overset{(a)}{\geq} (k_\alpha - 1)(1 - o(1)) + 1 - o(1) = \Omega\left(k_\alpha\right), \end{aligned} \tag{C.35}$$

where in step (a), we use the fact that $\sum_{j=k_\alpha+1}^{I} P[l_1 \geq j] \geq 0$, and if $P[l_1 \geq k_\alpha] = 1 - o(1)$ then $P[l_1 \geq j]$ is at least $1 - o(1)$ for $j = [1, \cdots, k_\alpha - 1]$. Combining (C.25), (C.34), and (C.35), we have

$$E[l_1] = \begin{cases} \Theta\left(Ip + \sqrt{Ip(1-p)\log(\Lambda)}\right) & \text{if } Ip = \omega\left((\log \Lambda)^3\right) \\ \Theta\left(Ip + \sqrt{Ip \log \Lambda}\right) & \text{if } Ip \in [\Omega\left(\log \Lambda\right), O(\text{polylog}\,\Lambda)] \\ \Theta\left(\frac{\log \Lambda}{\log \frac{\log \Lambda}{Ip}}\right) & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log \Lambda)\right]. \end{cases} \tag{C.36}$$

From (5.13) and (5.14), we have,

$$\overline{T}(\mathbf{G}) = O\left(\frac{Kp(1-\gamma)}{1+t}\frac{E[l_1]}{Ip}\right) \tag{C.37}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{Kp(1-\gamma)}{1+t}\left(\frac{E[l_1]t}{Ip(\Lambda-1)}\right)\right). \tag{C.38}$$

As $\frac{t}{\Lambda - 1} \approx \gamma$ is a constant, we get the exact scaling law of $\overline{T}(\mathbf{G})$, which is given by

$$\overline{T}(\mathbf{G}) = \Theta\left(\frac{Kp(1-\gamma)}{1+t}\frac{E[l_1]}{Ip}\right) \tag{C.39}$$

Combining (C.39) with (C.36), we obtain

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+t}\left(1 + \sqrt{\frac{(1-p)\log \Lambda}{Ip}}\right)\right) & \text{if } Ip = \omega\left((\log \Lambda)^3\right) \\ \Theta\left(\frac{Kp(1-\gamma)}{1+t}\left(1 + \sqrt{\frac{\log \Lambda}{Ip}}\right)\right) & \text{if } Ip \in [\Omega\left(\log \Lambda\right), O(\text{polylog}\,\Lambda)] \\ \Theta\left(\frac{Kp(1-\gamma)}{1+t}\frac{\log \Lambda}{Ip \log \frac{\log \Lambda}{Ip}}\right) & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log \Lambda)\right], \end{cases} \tag{C.40}$$

which can be further simplified as

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+t}\right) & \text{if } Ip = \Omega\left(\log \Lambda\right) \\ \Theta\left(\frac{Kp(1-\gamma)}{1+t}\frac{\log \Lambda}{Ip \log \frac{\log \Lambda}{Ip}}\right) & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log \Lambda)\right]. \end{cases} \tag{C.41}$$

This concludes the proof of Theorem 5.4.

C.5 Proof of Lemma 5.1

From (5.17), we know that $l_{s,\lambda}$ and $\binom{\Lambda-\lambda}{t}$ are non-increasing with λ, which implies that for each time slot $s \in [S]$, the profile vector $\mathbf{L}_s$, which minimizes the delay has components of the form

$$l_{s,\lambda} = \begin{cases} \lfloor \frac{d_s}{\Lambda} \rfloor + 1 \text{ for } \lambda \in [1, 2, \dots, A_s] \\ \lfloor \frac{d_s}{\Lambda} \rfloor \quad\ \text{ for } \lambda \in [A_s + 1, A_s + 2, \dots, \Lambda], \end{cases} \tag{C.42}$$

where $d_s = \sum_{k\in[K]} d_{s,k}$, and $A_s \triangleq d_s - \Lambda \lfloor \frac{d_s}{\Lambda} \rfloor$. Consequently, when $A_s \geq \Lambda - t$, the corresponding best-case delay T_s for time slot $s \in [S]$ is given by

$$T_s = \sum_{\lambda=1}^{\Lambda-t} \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} = \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\Lambda - t}{1 + t}, \tag{C.43}$$

while when $A_s < \Lambda - t$, this is given as

$$\begin{aligned} T_s &= \sum_{\lambda=1}^{A_s} \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + \sum_{\lambda=A_s+1}^{\Lambda-t} \left\lfloor \frac{d_s}{\Lambda} \right\rfloor \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left\lfloor \frac{d_s}{\Lambda} \right\rfloor \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + \sum_{\lambda=1}^{A_s} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \sum_{\lambda=1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} - \sum_{\lambda=A+1}^{\Lambda-t} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} - \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\Lambda - t}{1 + t} - \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}. \end{aligned} \tag{C.44}$$

We denote $\mathbf{S}_2 \subseteq [S]$ to be the set of time slots for which $A_s < \Lambda - t$. Then, the average delay corresponding to the optimal user-to-cache state association $\hat{\mathbf{G}}$ is lower bounded by

$$\begin{aligned} \overline{T}^* &\geq \frac{1}{S} \sum_{s\in[S]/\mathbf{S}_2} \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\Lambda - t}{1 + t} + \frac{1}{S} \sum_{s\in[\mathbf{S}_2]} \left(\left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\Lambda - t}{1 + t} - \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}} \right) \\ &= \frac{1}{S} \sum_{s\in[S]} \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 \right) \frac{\Lambda - t}{1 + t} - \frac{1}{S} \sum_{s\in[\mathbf{S}_2]} \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}. \end{aligned} \tag{C.45}$$

This concludes the proof of Lemma 5.1.

C.6 Proof of Theorem 5.5

We denote $v_{s,\lambda}^{\mathcal{G}_1}$, $v_{s,\lambda}^{\mathcal{G}_2}$, and $v_{s,\lambda}^{\mathcal{G}}$ to be the scaled loads calculated using transformed user demand matrix $\mathbf{D}$ (Step 00 of **Algorithm 5.2**) of each cache state $\lambda \in [\Lambda]$ at time slot $s \in [S]$ following the user-to-cache state association given by $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}$ respectively. It is straightforward to see from step 03 of **Algorithm 5.2** that $v_{s,\lambda}^{\mathcal{G}_1} = O\left(\frac{\log S}{\log\log S}\right)$ $\forall s \in [S],\ \lambda \in [\Lambda]$. By combining Lemma 15 and Lemma 18 of [78], we have $v_{s,\lambda}^{\mathcal{G}_2} = O(1)$ $\forall s \in [S],\ \lambda \in [\Lambda]$. Thus, the combined scaled load of each cache $\lambda \in [\Lambda]$ at each time slot $s \in [S]$ is given by $v_{s,\lambda}^{\mathcal{G}} = O\left(\frac{\log S}{\log\log S}\right)$.

To complete the proof, we now proceed to convert the scaled load of each cache $\lambda \in [\Lambda]$ at time slot $s \in [S]$ to the actual load. Based on the assumption that for each time slot $s \in [S]$, $\sum_{k\in[K]} d_{s,k} \geq \Lambda$, we have $\bar{d}_{s,k} = \min\left(\frac{\Lambda\ d_{s,k}}{\sum_{i\in[K]} d_{s,i}}, 1\right) = \frac{\Lambda\ d_{s,k}}{\sum_{i\in[K]} d_{s,i}}$ $\forall\ s \in [S], k \in [K]$. Then, the actual load corresponding to user-to-cache state association $\mathcal{G}$ is given by

$$v_{s,\lambda} = \frac{\sum_{k\in[K]} d_{s,k}}{\Lambda} v_{s,\lambda}^{\mathcal{G}} = O\left(\frac{\sum_{k\in[K]} d_{s,k}}{\Lambda} \frac{\log S}{\log\log S}\right)$$

$\forall\ s \in [S],\ \lambda \in [\Lambda]$. Consequently, from (5.17), we have

$$\overline{T}(\mathbf{G}) = O\left(\frac{1}{S}\sum_{s=1}^{S}\sum_{\lambda=1}^{\Lambda-t} \frac{\sum_{k\in[K]} d_{s,k}}{\Lambda} \frac{\log S}{\log\log S} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}\right) \tag{C.46}$$

$$= O\left(\frac{\log S}{\log\log S}\frac{1}{S}\sum_{s=1}^{S} \frac{\sum_{k\in[K]} d_{s,k}}{\Lambda} \frac{\Lambda - t}{1+t}\right). \tag{C.47}$$

We also have from (C.45) that

$$\overline{T}^* = \Omega\left(\frac{1}{S}\sum_{s\in[S]} \frac{\sum_{k\in[K]} d_{s,k}}{\Lambda} \frac{\Lambda - t}{1+t}\right). \tag{C.48}$$

This concludes the proof of Theorem 5.5.

Bibliography

[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] S. Jin, Y. Cui, H. Liu, and G. Caire, "A new order-optimal decentralized coded caching scheme with good performance in the finite file size regime," *IEEE Trans. Commun.*, vol. 67, no. 8, pp. 5297–5310, Aug. 2019.

[3] "Cisco Annual Internet Report (2018-2023)," White paper, March 2020.

[4] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[5] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3665–3677, Oct. 2014.

[6] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting caching and multicast for 5G wireless networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2995–3007, Apr. 2016.

[7] N. Golrezaei, P. Mansourifard, A. F. Molisch, and A. G. Dimakis, "Base-station assisted device-to-device communications for high-throughput wireless video networks," *IEEE Trans. Wireless Commun.*, vol. 13, no. 7, pp. 3665–3676, 2014.

[8] M. Ji, G. Caire, and A. F. Molisch, "Wireless device-to-device caching networks: Basic principles and system performance," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 1, pp. 176–189, 2016.

[9] A. Malik, S. H. Lim, and W.-Y. Shin, "On the effects of subpacketization in content-centric mobile networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1721–1736, Aug. 2018.

[10] A. Malik, J. Kim, K. S. Kim, and W.-Y. Shin, "A personalized preference learning framework for caching in mobile networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 6, pp. 2124–2139, Jun. 2021.

[11] 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.

[12] 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.

[13] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," *IEEE Trans. Inf. Theory*, vol. 64, no. 2, pp. 1281–1296, 2018.

[14] 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.

[15] 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.

[16] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "On the average performance of caching and coded multicasting with random demands," in *Proc. 11th Int. Symp. Wireless Commun. Syst. (ISWCS)*, Barcelona, Aug. 2014, pp. 922–926.

[17] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. on Networking*, vol. 23, no. 4, pp. 1029–1040, Aug. 2015.

[18] S. S. Bidokhti, M. Wigger, and R. Timo, "Erasure broadcast networks with receiver caching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Barcelona, Jul. 2016, pp. 1819–1823.

[19] J. Zhang and P. Elia, "Wireless coded caching: A topological perspective," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Jun. 2017, pp. 401–405.

[20] E. Lampiris, J. Zhang, and P. Elia, "Cache-aided cooperation with no CSIT," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Jun. 2017, pp. 2960–2964.

[21] 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.

[22] J. Zhang and P. Elia, "Fundamental limits of cache-aided wireless BC: Interplay of Coded-Caching and CSIT feedback," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 3142–3160, May 2017.

[23] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-server coded caching," *IEEE Trans. Inf. Theory*, vol. 62, no. 12, pp. 7253–7271, 2016.

[24] A. Tölli, S. P. Shariatpanahi, J. Kaleva, and B. H. Khalaj, "Multi-antenna interference management for coded caching," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2091–2106, 2020.

[25] S. Shariatpanahi and B. H. Khalaj, "On multi-server coded caching in the low memory regime," *arXiv preprint arXiv:1803.07655*, 2018.

[26] J. Zhang, F. Engelmann, and P. Elia, "Coded caching for reducing CSIT-feedback in wireless communications," in *2015 53rd Annual Allerton Conf. on Commun., Cont., and Comput. (Allerton)*, 2015, pp. 1099–1105.

[27] E. Piovano, H. Joudeh, and B. Clerckx, "On coded caching in the over-loaded MISO broadcast channel," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2017, pp. 2795–2799.

[28] H. Joudeh, E. Lampiris, P. Elia, and G. Caire, "Fundamental limits of wireless caching under mixed cacheable and uncacheable traffic," *IEEE Trans. Inf. Theory*, vol. 67, no. 7, pp. 4747–4767, Jul. 2021.

[29] Y. Cao and M. Tao, "Treating content delivery in multi-antenna coded caching as general message sets transmission: A DoF region perspective," *IEEE Trans. Wireless Commun.*, vol. 18, no. 6, pp. 3129–3141, Jun. 2019.

[30] 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.

[31] 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.

[32] 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.

[33] C. Yapar, K. Wan, R. F. Schaefer, and G. Caire, "On the optimality of d2d coded caching with uncoded cache placement and one-shot delivery," *IEEE Trans. Commun.*, vol. 67, no. 12, pp. 8179–8192, 2019.

[34] A. Sengupta, R. Tandon, and O. Simeone, "Fog-aided wireless networks for content delivery: Fundamental latency tradeoffs," *IEEE Trans. Inf. Theory*, vol. 63, no. 10, pp. 6650–6678, Oct. 2017.

[35] Y. Cao, M. Tao, F. Xu, and K. Liu, "Fundamental storage-latency trade-off in cache-aided mimo interference networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 5061–5076, Aug. 2017.

[36] J. S. P. Roig, D. Gündüz, and F. Tosato, "Interference networks with caches at both ends," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, May 2017, pp. 1–6.

[37] E. Piovano, H. Joudeh, and B. Clerckx, “Robust cache-aided interference management under full transmitter cooperation,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO,, Jun. 2018, pp. 1540–1544.

[38] M. Bayat, R. K. Mungara, and G. Caire, “Achieving spatial scalability for coded caching via coded multipoint multicasting,” *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 227–240, Jan. 2019.

[39] E. Lampiris and P. Elia, “Achieving full multiplexing and unbounded caching gains with bounded feedback resources,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, Jun. 2018, pp. 1440–1444.

[40] H. Zhao, A. Bazco-Nogueras and P. Elia, “Wireless coded caching can overcome the worst-user bottleneck by exploiting finite file sizes,” *IEEE Trans. Wireless Commun.*, to be published, doi: 10.1109/TWC.2022.3140895.

[41] E. Lampiris and P. Elia, “Full coded caching gains for cache-less users,” *IEEE Trans. Inf. Theory*, vol. 66, no. 12, pp. 7635–7651, 2020.

[42] E. Lampiris, J. Zhang, O. Simeone, and P. Elia, “Fundamental limits of wireless caching under uneven-capacity channels,” in *Proc. Int. Zurich Seminar on Inf. and Commun. (IZS)*, Feb. 2020, pp. 120–124.

[43] M. J. Salehi, E. Parrinello, S. P. Shariatpanahi, P. Elia, and A. Tölli, “Low-complexity high-performance cyclic caching for large miso systems,” *IEEE Trans. Wireless Commun.*, pp. 1–1, 2021.

[44] N. S. Karat, S. Dey, A. Thomas, and B. S. Rajan, “An optimal linear error correcting delivery scheme for coded caching with shared caches,” 2019. [Online]. Available: http://arxiv.org/abs/1901.03188

[45] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, “Caching and coded multicasting: multiple groupcast index coding,” in *Proc. IEEE Global Conf. on Signal and Inf. Process. (GlobalSIP)*, Atlanta, GA,, Dec. 2014, pp. 881–885.

[46] A. Sengupta and R. Tandon, “Improved approximation of storage-rate tradeoff for caching with multiple demands,” *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 1940–1955, May 2017.

[47] Y. Wei and S. Ulukus, “Coded caching with multiple file requests,” in *Proc. 55th Annual Allerton Conf. on Commun., Cont., and Comput. (Allerton)*, Monticello, IL, Oct. 2017, pp. 437–442.

[48] 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.

[49] M. Ji, K. Shanmugam, G. Vettigli, J. Llorca, A. M. Tulino, and G. Caire, "An efficient multiple-groupcast coded multicasting scheme for finite fractional caching," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, Jun. 2015, pp. 3801–3806.

[50] 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.

[51] L. Tang and A. Ramamoorthy, "Low subpacketization schemes for coded caching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Jun. 2017, pp. 2790–2794.

[52] C. Shangguan, Y. Zhang, and G. Ge, "Centralized coded caching schemes: A hypergraph theoretical approach," *IEEE Trans. Inf. Theory*, vol. 64, no. 8, pp. 5755–5766, Aug. 2018.

[53] K. Shanmugam, A. M. Tulino, and A. G. Dimakis, "Coded caching with linear subpacketization is possible using Ruzsa-Szemeredi graphs," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Jun. 2017, pp. 1237–1241.

[54] E. Lampiris, A. Bazco-Nogueras, and P. Elia, "Resolving the feedback bottleneck of multi-antenna coded caching," *IEEE Trans. Inf. Theory*, pp. 1–1, 2021.

[55] 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.

[56] ——, "Stochastic analysis of coded multicasting for shared caches networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Taipei, Taiwan, Dec. 2020, pp. 1–6.

[57] 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.

[58] A. Malik, B. Serbetci, and P. Elia, "Stochastic coded caching with optimized shared-cache sizes and reduced subpacketization," 2021. [Online]. Available: https://arxiv.org/pdf/2112.14114.pdf

[59] ——, "Coded caching in networks with heterogeneous user activity," 2022. [Online]. Available: https://arxiv.org/pdf/2201.10250.pdf

[60] "*NIST Digital Library of Mathematical Functions*," http://dlmf.nist.gov/, Release 1.0.26 of 2020-03-15, f. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds. [Online]. Available: http://dlmf.nist.gov/26.9.iv

[61] I. Stojmenović and A. Zoghbi, "Fast algorithms for generating integer partitions," *Int. J. Comput. Math.*, vol. 70, no. 2, pp. 319–332, 1998.

[62] V. Vajnovszki, "Generating permutations with a given major index," 2013. [Online]. Available: https://arxiv.org/pdf/1302.6558.pdf

[63] M. Merca, "Fast algorithm for generating ascending compositions," *J. Math. Model. and Algorithms*, vol. 11, pp. 89–104, 2012.

[64] J. Kelleher and B. O'Sullivan, "Generating all partitions: A comparison of two encodings," 2009. [Online]. Available: http://arxiv.org/abs/0909.2331

[65] C. Li, A. Yongacoglu, and C. D'Amours, "Heterogeneous cellular network user distribution model," in *Proc. IEEE Latin-American Conference on Commun. (LATINCOM)*, Medellin, Nov. 2016, pp. 1–6.

[66] G. George, A. Lozano, and M. Haenggi, "Distribution of the number of users per base station in cellular networks," *IEEE Wireless Commun. Letters*, vol. 8, no. 2, pp. 520–523, Apr. 2019.

[67] S. Zhou, D. Lee, B. Leng, X. Zhou, H. Zhang, and Z. Niu, "On the spatial distribution of base stations and its relation to the traffic density in cellular networks," *IEEE Access*, vol. 3, pp. 998–1010, Apr. 2015.

[68] 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 Trans. Inf. Theory*, vol. 67, no. 4, pp. 2353–2378, 2021.

[69] T. X. Vu, S. Chatzinotas, and B. Ottersten, "Coded caching and storage planning in heterogeneous networks," in *Proc. IEEE Wireless Commun. and Netw. Conf. (WCNC)*, San Francisco, CA, Mar. 2017, pp. 1–6.

[70] J. G. Andrews, S. Singh, Q. Ye, X. Lin, and H. S. Dhillon, "An overview of load balancing in hetnets: old myths and open problems," *IEEE Wireless Commun.*, vol. 21, no. 2, pp. 18–25, 2014.

[71] Q. Ye, B. Rong, Y. Chen, M. Al-Shalash, C. Caramanis, and J. G. Andrews, "User association for load balancing in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 12, no. 6, pp. 2706–2716, 2013.

[72] H. Kim, G. de Veciana, X. Yang, and M. Venkatachalam, "Distributed α-optimal user association and cell load balancing in wireless networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, p. 177–190, Feb. 2012. [Online]. Available: https://doi.org/10.1109/TNET.2011.2157937

[73] H. Jo, Y. J. Sang, P. Xia, and J. G. Andrews, "Heterogeneous cellular networks with flexible cell association: A comprehensive downlink sinr analysis," *IEEE Trans. Wireless Commun.*, vol. 11, no. 10, pp. 3484–3495, 2012.

[74] T. Han and N. Ansari, “Network utility aware traffic load balancing in backhaul-constrained cache-enabled small cell networks with hybrid power supplies,” *IEEE Trans. Mobile Comput.*, vol. 16, no. 10, pp. 2819–2832, 2017.

[75] M. Mitzenmacher, “The power of two choices in randomized load balancing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, 2001.

[76] B. Petra, C. Artur, S. Angelika, and V. Berthold, “Balanced allocations: The heavily loaded case,” *SIAM J. Comput.*, vol. 35, no. 6, pp. 1350–1385, Jun. 2006.

[77] C. Chekuri and S. Khanna, “On multidimensional packing problems,” *SIAM J. Comput.*, vol. 33, no. 4, pp. 837–851, Apr. 2004.

[78] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi, “Tight bounds for online vector scheduling,” in *Proc. 56th Annu. Symp. Foundations Comput. Sci. (FOCS)*, Berkeley, CA, Oct. 2015, pp. 525–544.

[79] M. E. Newman, “Power laws, pareto distributions and zipf’s law,” *Contemporary physics*, vol. 46, no. 5, pp. 323–351, 2005.

[80] G. Caraux and O. Gascuel, “Bounds on distribution functions of order statistics for dependent variates,” *Statist. Probab. Lett.*, vol. 14, no. 2, pp. 103–105, May 1992.

[81] R. Martin and S. Angelika, “Balls into bins — A simple and tight analysis,” in *Proc. Int. Workshop Randomization Approx. Techn. Comput. Sci.*, 1998, pp. 159–170.

[82] O. Gascuel and G. Caraux, “Bounds on expectations of order statistics via extremal dependences,” *Statist. Probab. Lett.*, vol. 15, no. 2, pp. 143–148, Sep. 1992.

[83] S. Gitzenis, G. Paschos, and L. Tassiulas, “Asymptotic laws for joint content replication and delivery in wireless networks,” *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 2760–2776, May 2013.

[84] U. Wieder, “Balanced allocations with heterogenous bins,” in *Proc. of the nineteenth annual ACM symposium on Parallel algorithms and architectures (SPAA ’07)*, 2007, pp. 188–193.

[85] W. Tang and F. Tang, “The poisson binomial distribution – old & new,” 2019. [Online]. Available: https://arxiv.org/pdf/1908.10024.pdf